Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ασφάλεια Διαδικτυακών Εφαρμογών με Χρήση Προγραμματιστικού Πλαισίου Django** **Web Application Security using Django Framework**
Ονοματεπώνυμο Φοιτητή	**Χρήστος Κατόπης**
Πατρώνυμο	**Γεώργιος**
Αριθμός Μητρώου	**ΜΠΠΛ14028**
Επιβλέπων	**Παναγιώτης Κοτζανικολάου, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης: **Μάρτιος 2017**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Παναγιώτης Κοτζανικολάου Επίκουρος Καθηγητής	Κωνσταντίνος Πατσάκης Επίκουρος Καθηγητής	Μιχαήλ Ψαράκης Επίκουρος Καθηγητής

Επιτελική Σύνοψη

Η παρούσα εργασία εξετάζει τη χρήση του Django framework για τον σχεδιασμό ασφαλών διαδικτυακών εφαρμογών.Για το σκοπό αυτό, πραγματοποιείται ανάλυση των απαιτήσεων ασφαλείας για μια σύγχρονη διαδικτυακή εφαρμογή: παρουσιάζονται οι κύριες απειλές που μπορεί να αντιμετωπίσει σε διάφορα επίπεδα και οι ευπάθιες που αυτές εκμεταλλεύονται, όπως και οι απαιτήσεις ασφαλείας που πρέπει να πληροί μια διαδικτυακή εφαρμογή. Στη συνέχεια, αναλύονται οι βιβλιοθήκες και οι τεχνολογίες ασφάλειας που προσφέρει το πλαίσιο Django. Στην τελική φάση της εργασίας, με χρήση του προγραμματιστικού πλαισίου Django αναπτύσσεται διαδικτυακή εφαρμογή με αξιοποίηση ορισμένων τεχνολογιών ασφαλείας και ελέγχεται η ικανοποίηση των απαιτήσεων ασφαλείας αλλά και η επιτυχής αντιμετώπιση των απειλών. Προτείνεται μια στοιχειωδώς βελτιωμένη έκδοση της εφαρμογής με βάση την ανάλυση ευπαθειών ασφάλειας.

Abstract

The present thesis proposes the use of Django web framework for the purpose of designing secure web applications. In order to achieve that, the security vulnerabilities of the web applications, which may expose them to threats, the methodology of web attacks and the basic guidelines for designing a secure web application are presented. The Django web framework, its functionality and its security features are introduced. In the final chapters, we present a case study by designing a web application using Django and the security tools it provides. The web application is scanned for vulnerabilities by a security vulnerability scanner software. Based on the results of the vulnerability analysis, an improved version of the web application is presented.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

Κεφάλαιο 1: Εισαγωγή

1.1 Περιγραφή του Υπό Μελέτη Προβλήματος

1.1.1 Διαδικτυακή Εφαρμογή

Ως διαδικτυακή εφαρμογή ορίζεται μια εφαρμογή λογισμικού τύπου client – server (πελάτη – εξυπηρετητή), οι λειτουργίες του client της οποίας εκτελούνται μέσω web browser [1]. Γνωστά παραδείγματα διαδικτυακών εφαρμογών είναι οι υπηρεσίες webmail, τα ηλεκτρονικά καταστήματα (e – shop), οι ηλεκτρονικές εγκυκλοπαίδειες κ.α.

1.1.2 Ασφάλεια Διαδικτυακών Εφαρμογών

Έχοντας υπόψη τον παραπάνω ορισμό, γίνεται άμεσα αντιληπτό ότι το πλήθος των διαδικτυακών εφαρμογών, η συχνότητα χρήσης τους, καθώς και οι υπηρεσίες και λειτουργίες που εκτελούνται από αυτές (εισαγωγή και διαχείριση προσωπικών δεδομένων, οικονομικές συναλλαγές κ.α.) στη σύγχρονη καθημερινότητα τις καθιστούν συχνό στόχο επιθέσεων κακόβουλων χρηστών.

Στον πίνακα 1, αναφέρονται ενδεικτικά παραβιάσεις δεδομένων τα τελευταία χρόνια καθώς και τα αποτέλεσματά τους.

Έτος	Οργανισμός	Συνέπεια Παραβίασης
2015	Bitcoin Exchange [2]	5 εκ. δολλάρια
2015	Premiera Blue Cross [3]	Προσωπικά δεδομένα χρηστών
2014	Healthcare System [4]	Προσωπικά δεδομένα ασθενών
2014	eBay [5]	Προσωπικά δεδομένα ενεργών χρηστών
2012	Bitcoin Exchange [6]	24.000 Bitcoins
2011	Sony Corporation [7]	Login Credentials
2009	Heartland Payment System [8]	Πληροφορίες Πιστωτικών Καρτών

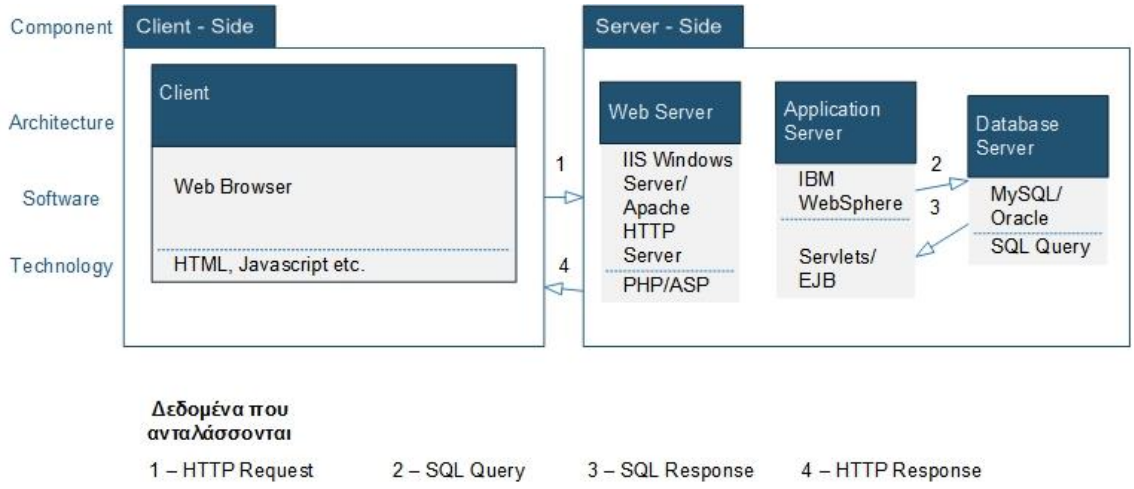
Πίνακας 1: Παραβιάσεις δεδομένων ανά είδος οργανισμού και συνέπεια τα τελευταία χρόνια [9]

Η προσπάθεια της προστασίας της διαθεσιμότητας, ακεραιότητας και εμπιστευτικότητας των δεδομένων και των υπηρεσιών μιας διαδικτυακής εφαρμογής είναι, λοιπόν, πρωταρχικής σημασίας [10].

Μάλιστα, η θεωρητική (και πρακτική) θεμελίωση της απάντησης στο ζήτημα της ασφάλειας έχει οδηγήσει στη ραγδαία ανάπτυξη της Ασφάλειας Διαδικτυακών Εφαρμογών: ενός κλάδου της Ασφάλειας Πληροφοριών που ασχολείται ειδικά με τις υπηρεσίες διαδικτύου και τις διαδικτυακές εφαρμογές [11].

1.1.3 Ασφαλής Σχεδιασμός Διαδικτυακών Εφαρμογών

Η αρχιτεκτονική και ο τρόπος λειτουργίας μιας διαδικτυακής εφαρμογής φαίνεται στην εικόνα 1.



Εικόνα 1: Αρχιτεκτονική Διαδικτυακής Εφαρμογής [9]

Ένας κακόβουλος χρήστης προσπαθεί να εκμεταλλευτεί οποιαδήποτε αδυναμία ασφάλειας σε κάθε επίπεδο της αρχιτεκτονικής μιας διαδικτυακής εφαρμογής, προκειμένου να πραγματοποιήσει διάφορων ειδών επιθέσεις.

Η πλειοψηφία όμως των επιθέσεων σε διαδικτυακές εφαρμογές γίνεται μέσω της εκμετάλλευσης κάποιας ευπάθειας ασφάλειας στο επίπεδο της εφαρμογής: δηλαδή μέσω αδυναμιών ασφαλείας που έχει μια εφαρμογή λόγω του τρόπου υλοποίησής της, του κώδικά της.

Σύμφωνα με την Έρευνα Ασφάλειας της εταιρείας Trustwave του 2016 (Trustwave Global Security Report 2016), η οποία εξέτασε εκατομμύρια περιπτώσεις επιθέσεων σε διαδικτυακές εφαρμογές σε 17 χώρες , διαπιστώθηκε ότι το 97% αυτών είχε κάποια ευπάθεια ασφάλειας [12].

Από τα παραπάνω, γίνεται σαφές το θεμελιώδες στάδιο στην ασφάλεια των διαδικτυακών εφαρμογών είναι η πρόληψη των επιθέσεων, μέσω του ασφαλούς σχεδιασμού μιας διαδικτυακής εφαρμογής. Δηλαδή η δημιουργία μιας εφαρμογής με τρόπο που να ικανοποιούνται βασικές αρχές ασφάλειας, ώστε να αποφεύγονται τυχόν ευπάθειες.

1.1.4 Ασφάλεια Διαδικτυακών Εφαρμογών με Χρήση Django

Ταυτόχρονα, οι αυξημένες απαιτήσεις στην ταχύτητα υλοποίησης μιας διαδικτυακής εφαρμογής οδηγούν στην χρήση προγραμματιστικών πλαισίων (frameworks) και CMSs (content management systems), αρκετά εκ των οποίων είναι γνωστά για τις ευπάθειές τους.

Ενδεικτικά αναφέρεται ότι η Trustwave Global Security Report 2016 αναφέρει ότι το 71% των επιθέσεων που εξετάστηκαν από την Trustwave την χρονιά του 2015 βασίστηκαν σε ευπάθειες που παρουσιάζει το CMS Worpress [12].

Η επιλογή ενός προγραμματιστικού πλαισίου το οποίο να επιτρέπει τον ασφαλή σχεδιασμό διαδικτυακής εφαρμογής είναι ζητούμενο για έναν σχεδιαστή και προγραμματιστή διαδικτυακών εφαρμογών.

Στη συγκεκριμένη εργασία, γίνεται απεύθυνση στο πρόβλημα που περιγράφηκε με χρήση του Django framework: ενός προγραμματιστικού πλαισίου τύπου Model – View - Controller (με διαφοροποιήσεις που θα αναλυθούν στη συνέχεια), που σύμφωνα με τους δημιουργούς του απευθύνεται σε «τελειομανείς με ασφυκτικά χρονικά περιθώρια»[13]. Το Django βασίζεται στη γλώσσα προγραμματισμού Python [14] και προσφέρει πλήθος βιβλιοθηκών και εργαλειών ασφάλειας, που αντιμετωπίζουν τις ευπάθειες ασφάλειας.

1.2 Στόχοι της Εργασίας

Βασικός στόχος της εργασίας είναι η συμβολή στην απάντηση στο μεγάλο πρόβλημα της ασφάλειας μιας διαδικτυακής εφαρμογής και συγκεκριμένα του ασφαλούς σχεδιασμού της. Για τον σκοπό αυτό χρησιμοποιείται το προγραμματιστικό πλαίσιο Django.

Προκειμένου να επιτευχθεί αυτό, θα γίνει:

1. Αναλυτική παρουσίαση του προγραμματιστικού πλαισίου Django, των βιβλιοθηκών και εργαλείων ασφάλειας που προσφέρει.

2. Μελέτη περίπτωσης χρήσης μέσω υλοποίησης διαδικτυακής εφαρμογής με χρήση του προγραμματιστικού πλαισίου Django και χρήση επιλεγμένων βιβλιοθηκών και εργαλείων ασφάλειας που διαθέτει.

3. Αξιολόγηση της ασφάλειας της διαδικτυακής εφαρμογής μέσω χρήσης εργαλείου ανάλυσης ευπαθειών (vulnerability scanner) και βελτίωσή της.

Βασικό βήμα για όλα τα παραπάνω είναι η κατανόηση των πιο κρίσιμων ειδών ευπαθειών και των απειλών που αντιμετωπίζει μια διαδικτυακή εφαρμογή. Η συνοπτική παρουσίασή τους θα πραγματοποιηθεί στο αμέσως επόμενο κεφάλαιο.

1.3 Μεθοδολογία της Εργασίας

Η μεθοδολογία που ακολουθήθηκε για την εκπόνηση της εν λόγω εργασίας είναι η εξής:

1. Καθορισμός των στόχων της εργασίας

2. Συλλογή, μελέτη και παρουσίαση του θεωρητικού υπόβαθρου σχετικά με την ασφάλεια διαδικτυακών εφαρμογών και το προγραμματιστικό πλαίσιο Django, τις βιβλιοθήκες και εργαλεία ασφάλειας που διαθέτει.
3. Δημιουργία εργαστηριακού περιβάλλοντος για τις ανάγκες της μελέτης περίπτωσης χρήσης (use case study): εγκατάσταση Python 2.7.12, Django 1.10.5, Sublime 3, Bootstrap 3.3.7,Font Awesome 4.7.0, Firefox 51.0.1, Virtualbox 5.1.4, Nessus Vulnerability Scanner 6.10 σε λειτουργικό σύστημα Windows 10 καθώς και εγκατάσταση Python 2.7.12, Django 1.10.5, apache2 web server, openssl 1.0.7 σε εικονικό λειτουργικό σύστημα Ubuntu 14.04.
4. Δημιουργία διαδικτυακής εφαρμογής με χρήση του προγραματιστικού πλαισίου Django και ενσωμάτωση επιλεγμένων βιβλιοθηκών και εργαλείων ασφαλείας που διαθέτει το Django. Η διαδικτυακή εφαρμογή θα εξυπηρετηθεί σε apache web server στο εικονικό λειτουργικό σύστημα Ubuntu 14.04, ενώ θα γίνει χρήση του Bootstrap και του Font - Awesome για την μορφοποίηση των σελίδων της διαδικτυακής εφαρμογής.
5. Έλεγχος της ασφάλειας της διαδικτυακής εφαρμογής με χρήση του εργαλείου ανάλυσης ευπαθειών Nessus Vulnerability Scanner.
6. Διεξαγωγή και ανάλυση συμπερασμάτων.

1.4 Δομή της Εργασίας

Η υπόλοιπη εργασία χωρίζεται στα παρακάτω κεφάλαια:

Στο **Κεφάλαιο 2 (Εισαγωγή στην Ασφάλεια των Διαδικτυακών Εφαρμογών)** γίνεται μια συνοπτική παρουσίαση βασικών ευπαθειών ασφάλειας, απειλών και βασικών αρχών ασφάλειας των διαδικτυακών εφαρμογών.

Στο **Κεφάλαιο 3 (Εισαγωγή στο Προγραμματιστικό Πλαίσιο Django)** γίνεται αναλυτική περιγραφή του τρόπου λειτουργίας του προγραμματιστικού πλαίσιου Django.

Στο **Κεφάλαιο 4 (Η Ασφάλεια στο Προγραμματιστικό Πλαίσιο Django)** γίνεται λεπτομερής περιγραφή των εργαλειών και βιβλιοθηκών ασφάλειας που προσφέρει το προγραμματιστικό πλαίσιο Django.

Στο **Κεφάλαιο 5 (Ασφαλής Σχεδιασμός Διαδικτυακής Εφαρμογής με Χρήση Προγραμματιστικού Πλαισίου Django)** γίνεται παρουσίαση της Διαδικτυακής Εφαρμογής που έχει δημιουργηθεί και των εργαλείων και βιβλιοθηκών ασφάλειας του Django που έχουν ενσωματωθεί.

Στο **Κεφάλαιο 6 (Έλεγχος Ασφάλειας της Διαδικτυακής Εφαρμογής με Χρήση του Εργαλείου Ανάλυσης Ευπαθειών Nessus)** περιγράφεται το εργαλείο ανάλυσης ευπαθειών Nessus και χρησιμοποιείται προκειμένου να πιστοποιηθεί η ασφάλεια της διαδικτυακής εφαρμογής. Γίνονται οι απαραίτητοι έλεγχοι και διορθώσεις.

Στο **Κεφάλαιο 7 (Συμπεράσματα)** περιγράφονται τα συμπεράσματα που διεξάχθηκαν από την εργασία.

1.5 Πλάνο Υλοποίησης της Εργασίας

Το πλάνο που ακολουθήθηκε για την εκπόνηση της εργασίας περιγράφεται στον παρακάτω πίνακα:

Παραδοτέο	Περιγραφή	Ημερομηνία παράδοσης
Π1	Πρόταση εκπόνησης μεταπτυχιακής εργασίας	17/05/2016
Π2 (Κεφάλαια 1,2)	Εισαγωγή και θεωρητική μελέτη της Ασφάλειας Διαδικτυακών Εφαρμογών	14/07/16
Π3.1 (Κεφάλαιο 3)	Θεωρητική μελέτη του πλαισίου εργασίας Django	13/12/16
Π3.2 (Πρώτη έκδοση της διαδικτυακής εφαρμογής)	Υλοποίηση των λειτουργικών απαιτήσεων της διαδικτυακής εφαρμογής	13/12/16
Π4 (Κεφάλαιο 4)	Θεωρητική μελέτη των βιβλιοθηκών και εργαλείων ασφάλειας του πλαισίου εργασίας Django	18/1/17
Π5.1(Κεφάλαιο 5)	Περιγραφή της διαδικτυακής εφαρμογής και της ενσωμάτωσης βιβλιοθηκών και εργαλείων ασφάλειας του Django	27/2/17
Π5.2 (2η έκδοση Διαδικτυακής Εφαρμογής)	Ολοκληρωμένη υλοποίηση της εφαρμογής με βάση τις απαιτήσεις ασφάλειας	27/2/17
Π6.1 (Κεφάλαιο 6)	Έλεγχος αδυναμιών ασφάλειας της διαδικτυακής εφαρμογής με επιλεγμένο εργαλείο ελέγχου αδυναμιών. Παρουσίαση των αποτελεσμάτων.	3/3/17
Π6.2 ΤΕΛΙΚΗ ΕΚΔΟΣΗ (Κεφάλαιο 7)	Διορθωμένη έκδοση της εργασίας με προσθήκη των συμπερασμάτων	3/3/17

Πίνακας 2: Συγκεντρωτικός πίνακας παραδοτέων

Κεφάλαιο 2: Εισαγωγή στην Ασφάλεια των Διαδικτυακών Εφαρμογών

2.1 Ευπάθειες Ασφάλειας

Όπως περιγράψαμε και στο προηγούμενο κεφάλαιο, η πλειοψηφία των επιθέσεων σε διαδικτυακές εφαρμογες βασίζεται σε υπαρκτές ευπάθειες λόγω του σχεδιασμού τους.

Μια ευπάθεια ασφάλειας μπορεί να επιτρέψει την εισαγωγή επιβλαβούς κώδικα από έναν κακόβουλο χρήστη. Λειτουργίες που επιτυγχάνονται με χρήση τέτοιου κώδικα είναι η αλλοίωση των δεδομένων που έχει εισάγει ένας έμπιστος χρήστης, η αποστολή δεδομένων από τη σύνοδο (session) ενός έμπιστου χρήστη στον κακόβουλο χρήστη κ.α.

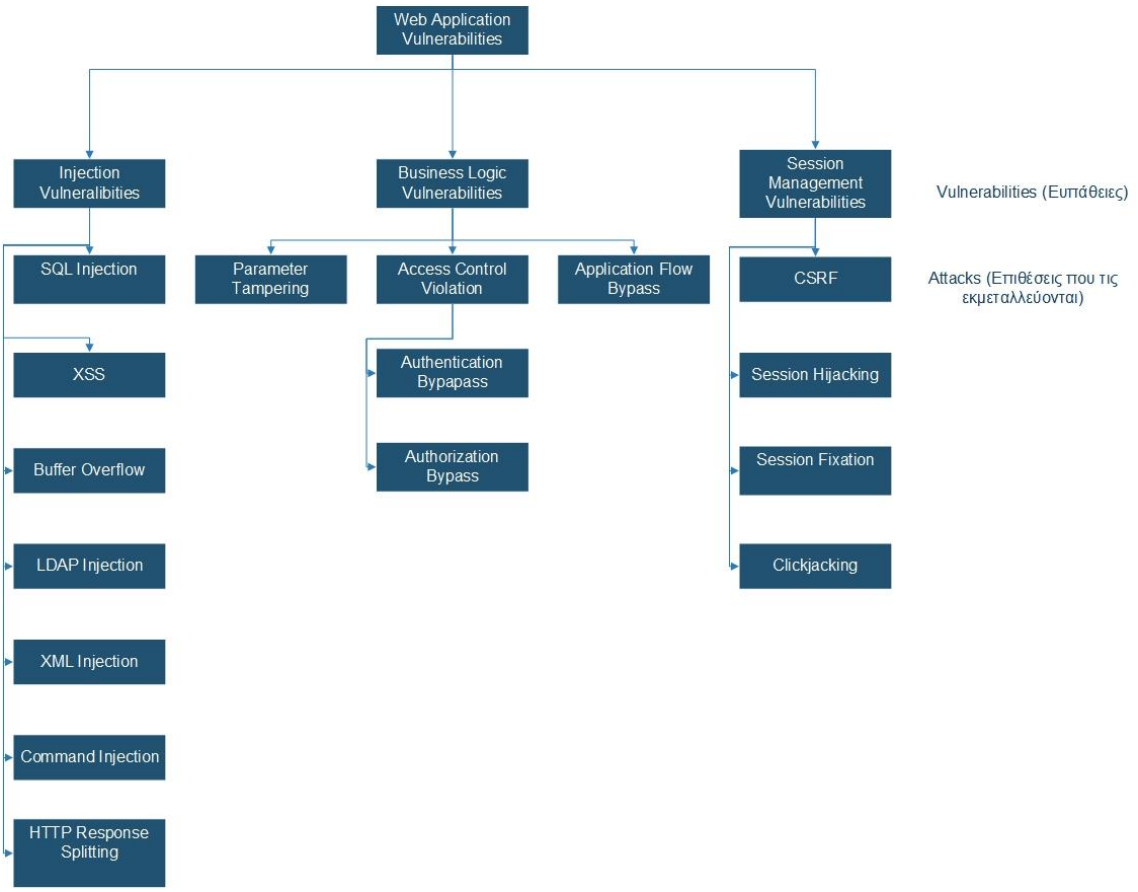
Οι ευπάθειες ασφάλειας προκύπτουν από σφάλματα στην υλοποίηση της εφαρμογής. Οι βασικές κατηγορίες των ευπαθειών με βάση τα σφάλματα στην υλοποίηση παρουσιάζονται στον πίνακα 3.

Ευπάθεια Ασφάλειας	Περιγραφή
Έλλειψη ή ανεπαρκής πιστοποίηση εισαγωγής δεδομένων	Έλλειψη ή ανεπαρκής έλεγχος στα σημεία που εισάγονται δεδομένα από τον χρήστη, που ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί για αλλοίωση των queries (SQL Injection, XML Injection κ.α.)
Εσφαλμένη αυθεντικοποίηση και εξουσιοδότηση	Έλλειψη συναρτήσεων αυθεντικοποίησης και πολιτικών ελέγχου πρόσβασης (access control policies) που μπορεί να επιτρέψει σε κακόβουλους χρήστες να αποκτήσουν πρόσβαση σε εμπιστευτικά δεδομένα πραγματικοποίησουν μη – πιστοποιημένες ενέργεις
Εσφαλμένη διαχείριση συνόδου	Αδυναμία στην δημιουργία και διαχείριση session tokes, τα οποία είναι απαραίτητα για την διατήρηση της ταυτότητας του χρήστη της εφαρμογής κατά την χρήση της, που μπορεί να επιτρέψουν σε κακόβουλο χρήση να παραβιάσει μία σύνοδο ενός χρήστη.

Πίνακας 3: Βασική κατηγοριοποίηση ευπαθεών με βάση τα σφάλματα κατά την υλοποίηση μιας διαδικτυακής εφαρμογής [9]

Ο αριθμός των ενδεχόμενων ευπαθειών ασφάλειας μιας διαδικτυακής εφαρμογής είναι μεγάλος. Ο διαχωρισμός τους σε είδη είναι απαραίτητος για την κατανόηση και την αντιμετώπισή τους.

Στην εικόνα 2 παρουσιάζεται σχηματικά η κατηγοριοποίηση των ευπαθειών, καθώς και τα είδη των επιθέσεων που τις εκμεταλλεύονται.



Εικόνα 2: Είδη ευπαθειών και επιθέσεων [9]

Στις επόμενες παραγράφους παρουσιάζονται αναλυτικότερα τα είδη των ευπαθειών και των επιθέσεων που τις εκμεταλλεύονται.

2.1.1 Injection Vulnerabilities (Ευπάθειες Έγχυσης Κώδικα)

Οι εν λόγω ευπάθειες επιτρέπουν σε έναν κακόβουλο χρήστη να τροποποιήσει τις παραμέτρους που εισάγονται σε μια εφαρμογή και χρησιμοποιούνται σε διάφορα queries ,αλλοιώνοντας έτσι τα queries αυτά καθαυτά. Η εισαγωγή κακόβουλων παραμέτρων στα queries που ανταλάσσονται, οδηγεί σε μη – επιθυμητή ροή δεδομένων από το θύμα προς τον κακόβουλο χρήστη και αντίστροφα. Τα είδη των Injection Vulnerabilites διαφέρουν ανάλογα με το είδος του query, της εντολής ή της γλώσσας που χρησιμοποιείται για injection.Τα πιο σημαντικά είδη παρουσιάζονται στις επόμενες παραγράφους [9].

2.1.1.1 SQL Injection Vulnerabilities (Ευπάθειες Έγχυσης Κώδικα SQL) [9]

Πρόκειται για σφάλματα τα οποία δίνουν την δυνατότητα σε κακόβουλους χρήστες να παραβιάσουν τη βάση δεδομένων μιας εφαρμογής, οδηγώντας σε ανεπιθύμητη εξαγωγή ή εισαγωγή δεδομένων σε αυτή. Οι επιθέσεις που εκμεταλλεύονται το συγκεκριμένο είδος ευπαθειών καλούνται **SQL Injection Attacks** [9]**.**

2.1.1.2 Cross Site Scripting ή XSS (Ευπάθειες τύπου XSS) [9]

Είναι είδος injection vulnerability που επιτρέπει στον επιτιθέμενο να εκτελέσει κακόβουλο κώδικα στον browser κάποιου χρήστη. Οι συγκεκριμένες ευπάθειες προκύπτουν σε σημεία που η διαδικτυακή εφαρμογή χρησιμοποιεί κώδικα που έχει εισαχθεί από χρήστη και δεν υπάρχει επαρκής έλεγχος. Οι επιθέσεις που εκμεταλλεύονται τις παραπάνω ευπάθεις καλούνται **XSS Attacks.** Μία επίθεση τύπου XSS πραγματοποιείτια ως εξής: η επίσκεψη σε τοποθεσία της εφαρμογής που έχει προσβληθεί από κακόβουλο κώδικα, οδηγεί στην εκτέλεσή του από τον browser του χρήστη. Οι XSS Attacks χωρίζονται σε τρία είδη**: Reflected XSS Attacks, Stored XSS Attacks και DOM XSS Attacks** [9].

2.1.1.3 Άλλες Ευπάθειες Έγχυσης Κώδικα [9]

Άλλες σημαντικές injection vulnerabilities είναι οι ευπάθειες τύπου **XML Injection**, **Command Injection, LDAP Injection, HTTP response splitting** και **Buffer Overflow.**

Οι πρώτες τρεις είναι παρεμφερείς με τις ευπάθειες τύπου SQL Injection. Οι ευπάθειες τύπου HTTP response splitting επιτρέπουν στον επιτιθέμενο να αλλοιώσει την HTTP header (κεφαλίδα HTTP) ώστε να αλλάξει τα HTTP responses, ενώ οι ευπάθειες τύπου Buffer Overflow επιτρέπουν την εκτέλεση κώδικα ο οποίος οδηγεί σε segmentation faults (σφάλματα μνήμης), denial of service (άρνηση παροχής εξυπηρέτησης) κ.α. [9].

2.1.2 Business Logic Vulnerabilities (Ευπάθειες Επιχειρησιακής Λογικής) [9]

Ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί τέτοιες ευπάθειες ώστε να παρακάμψει την ομαλή λειτουργία μιας εφαρμογής προς όφελός του. Για την καλύτερη κατανόηση του συγκεκριμένου είδους ευπαθειών, παρουσιάζουμε ένα σενάριο επίθεσης: Έστω ένα ηλεκτρονικό κατάστημα, το οποίο παρέχει στους χρήστες ηλεκτρονικούς κωδικούς για την πραγματοποίηση αγορών. Κάθε κωδικός μπορεί να χρησιμοποεί μία και μόνο φορά για την απόκτηση συγκεκριμένου προϊόντος. Η εκμετάλλευση μιας ευπάθειας Business Logic θα μπορούσε να επιτρέψει σε έναν κακόβουλο χρήστη να χρησιμοποιήσει τον ηλεκτρονικό κωδικό παραπάνω από μία φορές, αποκτώντας έτσι πάνω από ένα τεμάχια του προϊόντος [9].

2.1.2.1 Parameter Manipulation (Τροποποίηση Παραμέτρων)

Η ανεπιθύμητη τροποποίηση παραμέτρων οι οποίες επιτελούν σημαντικό ρόλο στη λειτουργία μιας διαδικτυακής εφαρμογής, επιτρέπουν σε έναν επιτιθέμενο να αλλάξει τη συμπεριφορά της εφαρμογής αυτής.

Η έλλειψη επαρκούς ελέγχου στην πλευρά του server μπορεί να επιτρέψει σε επιτιθέμενο να παρακάμψει τον έλεγχο στην πλευρά του client και να αλλοιώσει τις τιμές των παραμέτρων που εισάγονται από τον χρήστη στον server. Οι πιο κοινές επιθέσεις που βασίζονται στις συγκεκριμένες ευπάθειες είναι οι επιθέσεις τύπου Parameter Manipulation και Parameter Tampering [9].

2.1.2.2 Access Control Vulnerabilities (Ευπάθειες Ελέγχου Πρόσβασης)

Η διασφάλιση της ιδιωτικότητας των προσωπικών δεδομένων που διαχειρίζεται μια διαδικτυακή εφαρμογή επιτυγχάνεται μέσω του περιορισμού του αριθμού των χρηστών που έχουν το δικαίωμα πρόσβασης στα δεδομένα αυτά. Ο πιο συνήθης τρόπος υλοποίησης του παραπάνω είναι η παροχή διαπιστευτηρίων στους ενδιαφερόμενους χρήστες (π.χ. username, password).

Η έλλειψη ή η ανεπαρκής υλοποίηση του ελέγχου των διαπιστευτηρίων αυτών μπορει να οδηγήσει σε παραβίαση της ιδιωτικότητας των δεδομένων, δηλαδή τη δυνατότητα πρόσβασης σε αυτα από μη – διαπιστευμένους χρήστες. Τις συγκεκριμένες ευπάθειες τις εκμεταλλεύονται οι επιθέσεις Authentication Bypass και Authorization Bypass. Οι επιθέσεις αυτές συνήθως υλοποιούνται με αλλοίωση του URL ώστε να παρακαμθεί το στάδιο της εγγραφής (sign in) ή της σύνδεσης όπου αυτό (log in) απαιτείται [9].

2.1.2.3 Application Flow Vulnerabilities (Ευπάθειες Ροής της Εφαρμογής)

Οι συγκεκριμένες ευπάθειες επιτρέπουν στον επιτιθέμενο να παρακάμψει την προβλεπόμενη λογική/ροή μιας εφαρμογής. Ένα σενάριο τέτοιας επίθεσης είναι η επιβεβαίωση παραγγελίας ενός προϊόντος σε ένα ηλεκτρονικό κατάστημα, παρακάμπτωντας το στάδιο της πληρωμής.
Οι επιθέσεις που εκμεταλλεύονται τις εν λόγω ευπάθειες καλούνται Workflow Bypass Attacks [9].

2.1.3 Session Management Vulnerabilities (Ευπάθειες Διαχείρισης Συνόδου)

Οι Ευπάθειες Διαχείρισης Συνόδου προκύπτουν από την λανθασμένη διαχείριση των μεταβλητών συνόδου (session variables), οι οποίες χρησιμοποιούνται στο να καταγράφεται η κατάσταση της εφαρμογής και του διαπιστευμένου χρήστη. Ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί αυτού του τύπου τις ευπάθειες για να πραγματοποιήσει επιθέσεις τύπου **Session Hijacking, Session Fixation, Cross – Site Request Forgery, Clickjacking.**

Στις επιθέσεις τύπου **Session Hijacking** ο επιτιθέμενος υφαρπάζει το session token ενός διαπιστευμένου χρήστη προκειμένου να εκτελέσει ανεπιθύμητες ενέργεις.
Στις επιθέσεις τύπου **Session Fixation**, ο επιτιθέμενος μετατρέπει το session token του σε session token ενός διαπιστευμένου χρήστη, προκειμένου να υφαρπάξει τη σύνοδό του.

Οι επιθέσεις τύπου **Cross – Site Request Forgery** επιτρέπουν σε έναν κακόβουλο χρήστη να στείλει κακόβουλα αιτήματα χρησιμοποιώντας την ταυτότητα διαπιστευμένων χρηστών.

Οι επιθέσεις **Clickjacking** στοχεύουν στο να παραπλανήσουν χρήστες να κάνουν click σε κακόβουλα αντικείμενα που έχουν προστεθεί σε μια σελίδα, προκειμένου να υλοποιηθεί κάποια ανεπιθύμητη ενέργεια. Τα δύο πρώτα είδη επιθέσεων στοχεύουν την ταυτότητα της συνόδου

(session ID) ενός χρήστη, ενώ οι δύο τελευταίες στην εκτέλεση ανεπιθύμητων αιτημάτων στον browser [9].

2.2 Κατηγοριοποίηση Κινδύνων Ασφάλειας Σύμφωνα με τον OWASP

Είναι σαφές ότι αν η αντιμετώπιση του συνόλου των ευπαθειών μιας διαδικτυακής εφαρμογής φαντάζει δύσκολη ή και αδύνατη, η κατανόηση και η αντιμετώπιση των πιο «επικίνδυνων» ευπαθειών είναι ζωτικής ζημασίας για έναν προγραμματιστή διαδικτυακών εφαρμογών.

Για αυτό το λόγο θα πραγματοποιηθεί αναλυτικότερη παρουσίαση των σημαντικότερων κινδύνων ασφάλειας των διαδικτυακών εφαρμογών.

Η παρουσίαση θα βασιστεί στο **OWASP Top Ten 2013,** την κατηγοριοποίηση ευπαθειών βάσει του κινδύνου ασφάλειας που δημιουργούν, σύμφωνα με τον OWASP (Open Web Application Security Project) για το 2013.

Η κατηγοριοποίηση αυτή των κινδύνων ασφάλειας (που ανανεώνεται κάθε 3 χρόνια) προκύπτει μέσα από ανάλυση 8 βάσεων δεδομένων από 7 εταιρείες που ειδικεύονται στην Ασφάλεια Εφαρμογών. Τα δεδομένα αναφέρονται σε 500.000 ευπάθειες που εμφανίζονται σε σύνολο χιλιάδων εφαρμογών. Η έκδοση του 2013 είναι η πιο πρόσφατη [15].

Κίνδυνος Ασφάλειας	Περιγραφή
A1 - Injection (Έγχυση)	Αδυναμίες σε επιθέσεις έγχυσης κώδικα (π.χ. SQL Injection): αποστολή μη – έγκυρων δεδομένων ως είσοδο με αποτέλεσμα την εκτέλεση ανεπιθύμητων λειτουργιών
A2 - Broken Authentication & Session Management (Εσφαλμένη Υλοποίηση Αυθεντικοποίησης και Συνόδων)	Λανθασμένη υλοποίηση λειτουργιών που σχετίζονται με τις συνόδους που έχουν ως αποτέλεσμα τη δυνατότητα υποκλοπής συνθηματικών, κλειδιών ή άλλων στοιχείων συνόδων
A3 - Cross – Site Scripting (Ατέλειες τύπου XSS)	Αποδοχή μη – έγκυρων δεδομένων και αποστολή τους στον browser χωρίς έλεγχο
A4 - Insecure Direct Object References (Μη Ασφαλής Απευθείας Αναφορά σε Αντικείμενα)	Έκθεση ενός εσωτερικού αντικειμένου (πχ κλειδιού) χωρίς προστασία με αποτέλεσμα την δυνατότητα απόκτησής του από τρίτο
A5 - Security Misconfiguration (Σφάλματα Διαμόρφωσης Ασφάλειας)	Μη – ορθές ρυθμίσες ασφαλείας σε διάφορα επίπεδα (πχ servers, βάσεις δεδομένων)
A6 - Sensitive Data Exposure (Έκθεση Ευαίσθητων Δεδομένων)	Ανυπαρξία προστασίας προσωπικών δεδομένων (π.χ. στοιχεία αυθεντικοποίησης) με αποτέλεσμα τη δυνατότητα απόκτησής τους από τρίτο
A7 - Missing Function Level Access Control (Έλλειψη στη Λειτουργία	Έλλειψη ελέγχου πρόσβασης στην πλευρά του server με αποτέλεσμα τη δυνατότητα αποστολής αιτημάτων και τελικά πρόσβαση

Ελέγχου στο Επίπεδο Πρόσβασης)	στη λειτουργία του χωρίς εξουσιοδότηση από τρίτους.
A8 - Cross – Site Request Forgery (Πλαστογράφηση Αίτησης Μεταξύ Θέσεων)	Δυνατότητα εξαναγκασμού αποστολής ψευδών αιτημάτων από τον browser του θύματος (με διάφορες πληροφορίες), τα οποία μια εφαρμογή που έχει αδυναμία θα αντιληφθεί ως έγκυρα
A9 - Using Components with Known Vulnerabilities (Χρήση που έχουν γνωστές Αδυναμίες)	Χρήση βιβλιοθηκών, πλαισίων εργασίας με γνωστές αδυναμίες τις οποίες μπορεί να εκμεταλλευτεί ένας κακόβουλος χρήστης για να αποκτήσει πρόσβαση σε προσωπικά δεδομένα κ.α.
A10 - Unvalidated Redirects and Forwards (Μη πιστοποιημένες ανακατευθύνσεις και προωθήσεις)	Δυνατότητα ανακατεύθυνσης σε μη - επιθυμητούς ιστότοπους μέσω της διαδικτυακής εφαρμογής

Πίνακας 4: OWASP Top Ten 2013 [15]

Στις επόμενες παραγράφους, παρουσιάζονται παραδείγματα επίθεσης για κάθε έναν από τους 10 σημαντικότερους κίνδυνους ασφάλειας σύμφωνα με τον OWASP ώστε να γίνει κατανοητός ο τρόπος εμφάνισης και λειτουργίας τους.

2.2.1 A1 – Injection

Παραδείγματα Επίθεσης:

1) Μια εφαρμογή χρησιμοποιεί queries, χωρίς κάποιο έλεγχο, με αποτέλεσμα να είναι ευπαθή σε Injection.

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

2) Μια εφαρμογή χρησιμοποιεί frameworks που είναι ευπαθή σε Injection (π.χ. Hibernate Query Language) χωρίς επιπρόσθετο έλεγχο.

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

Και στις δύο περιπτώσεις ο επιτιθέμενος αλλάζει την παράμετρο ID στον browser του σε ' or '1'='1

```
http://example.com/app/accountView?id=' or '1'='1
```

Η παραπάνω αλλαγή, και στις δύο περιπτώσεις,επιστρέφει ως αποτέλεσμα όλες τις εγγραφές του table accounts [15].

2.2.2 A2 – Broken Authentication and Session Management

Παραδείγματα Επίθεσης:

1) Μια διαδικτυακή εφαρμογή αεροπορικής εταιρείας, εκθέτει το session ID στο URL της:

```
http://example.com/sale/saleitems?sessionid=268544541&dest=Hawaii
```

Ένας διαπιστευμένος χρήστης γνωστοποιεί τον παραπάνω σύνδεσμο σε τρίτους. Την ίδια στιγμή γνωστοποιεί και το session ID του και έτσι, όταν κάποιος χρησιμοποιήσει τον παραπάνω σύνδεσμο μπορεί να χρησιμοποιήσει στο session και την πιστωτική του εξουσιοδοτημένου χρήστη.

2) Ένας χρήστης χρησιμοποιεί υπολογιστή σε κοινόχρηστο χώρο για να μπει σε έναν ιστότοπο. Ο χρήστης κλείνει τον browser χωρίς να επιλέξει την έξοδο από την εφαρμογή. Μη – επαρκής υλοποίηση της διαχείρισης συνόδου μπορεί να επιτρέψει σε έναν κακόβουλο χρήστη να χρησιμοποιήσει τα διαπιστευτήρια του χρήστη, ανοίγοντας απλά ξανά τον browser.

3) Ένας επιτιθέμενος αποκτά πρόσβαση στον κωδικό της βάσης δεδομένων που χρησιμοποιεί η εφαρμογή. Τυχόν έλλειψη κρυπτογράφησης των κωδικών, εκθέτουν αμέσως τους κωδικούς όλων των εγγεγραμμένων χρηστών στον κακόβουλο χρήστη [15].

2.2.3 A3 – Cross – Site Scripting (XSS)

Παράδειγμα επίθεσης:

Η εφαρμογή χρησιμοποιεί μη – έμπιστα δεδομένα στη δομή του παρακάτω HTML snippet χωρίς κατάλληλη πιστοποίηση ή χαρακτήρες διαφυγής (escape characters):

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

Ο επιτιθέμενος τροποιοεί την παράμετρο ‘CC’ στον browser του σε:

```
'><script>document.location= 'http://www.attacker.com/cgi-
bin/cookie.cgi ?foo='+document.cookie</script>'.
```

Η παραπάνω επίθεση αποστέλλει το session ID του θύματος στον ιστότοπο του επιτιθέμενου, επιτρέποντας στον τελευταίο την υφαρπαγή της συνόδου του πρώτου.

2.2.4 A4 – Insecure Direct Object References

Παράδειγμα Επίθεσης:

Η εφαρμογή χρησιμοποιεί μη – πιστοποιημένα δεδομένα σε μια κλήση SQL κατά την οποία γίνεται πρόσβαση σε πληροφορίες λογαριασμού χρήστη:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt = connection.prepareStatement(query , … );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

Ο επιτιθέμενος απλώς εισάγει στη θέση της παραμέτρου 'acct' την τιμή του λογαριασμού την οποία επιθυμεί. Ο επιτιθέμενος μπορεί έτσι να αποκτήσει πρόσβαση στον λογαριασμό οποιουδήποτε χρήστη, όπως φαίνεται στο παρακάτω παράδειγμα [15]:

```
http://example.com/app/accountInfo?acct=notmyacct
```

2.2.5 A5 - Security Misconfiguration

Παραδείγματα Επίθεσης:

1) Οι προεπιλεγμένες σελίδες για τον διαχειριστή του server της εφαρμογής είναι εγκατεστημένες και τα στοιχεία του λογαριασμού του διαχειριστή δεν έχουν τροποποιηθεί (είναι αυτά των εργοστασιακών ρυθμίσεων). Ο επιτιθέμενος ανακαλύπτει τις σελίδες του διαχειριστή στον server και αποκτά πρόσβαση στον server χρησιμοποιώντας τον κωδικό των εργοστασιακών ρυθμίσεων.

2) Το directory listing δεν έχει απενεργοποιηθεί στον server. Ο επιτιθέμενος βρίσκει τυχόν κώδικα (πχ από κλάσεις που έχουν χρησιμοποιηθεί), και τις κάνει decompile προκειμένου να ανακαλύψει κάποιο σφάλμα στην υλοποιήση της εφαρμογής που θα μπορέσει να εκμεταλλευτεί.

3) Ο server της εφαρμογής περιέχει λειτουργίες και μικροεφαρμογές οι οποίες δεν έχουν απενεργοποιηθεί ή διαγραφεί. Ο επιτιθέμενος χρησιμοποιεί γνωστά σφάλματα υλοποίησης των εφαρμογών αυτών προκειμένου να παραβιάσει τον server [15].

2.2.6 A6 – Sensitive Data Exposure

Παραδείγματα Επίθεσης:

1) Μία διαδικτυακή εφαρμογή κρυπτογραφεί και αποθηκεύει τους αριθμούς πιστωτικών καρτών σε μία βάση δεδομένων, χρησιμοποιώντας την αυτόματη κρπτογράφηση της βάσης δεδομένων. Αυτό σημαίνει ότι τα δεδομένα αποκρυπτογραφούνται, προκειμένου να προβληθούν όταν ζητηθούν. Αυτό επιτρέπει την προβολή των κωδικών πιστωτικών καρτών από κακόβουλος χρήστες σε περίπτωση SQL Injection Attacks.

2) Ένας ιστότοπος δεν χρησιμοποιεί SSL για όλες τις σελίδες στις οποίες μπορεί να έχει πρόσβαση ένας διαπιστευμένος χρήστης. Ο επιτιθέμενος παρακολουθεί την κίνηση του

δικτύου και υφαρπάζει το session cookie ενός διαπιστευμένου χρήστη. Στη συνέχεια, αναπαράγει το session cookie και υφαρπάζει τη σύνοδο του χρήστη, αποκτώντας ταυτόχρονα πρόσβαση στα προσωπικά του δεδομένα.

3) Μία βάση δεδομένων χρησιμοποιεί για την κρυπτογράφηση των χρηστών απλά hashes (χωρίς να κάνει χρήση τυχαίων δεδομένων που ενισχύουν την πολυπλοκότητα). Σε περίπτωση απόκτησης του αρχείου της βάσης, ένας κακόβουλος χρήστης μπορεί να αποκρυπτογραφήσει τους κωδικούς χρησιμοποιώντας rainbow table με προϋπολογισμένα hashes [15].

2.2.7 A7 – Missing Function Level Access Control

Παραδείγματα Επίθεσης:

1) Ο επιτιθέμενος προσπαθεί να περιηγηθεί σε συγκεκριμένους συνδέσμους, όπως οι παρακάτω, πληκτρογώντας το URL τους στον browser:

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

Η δυνατότητα πρόσβασης ενός μη – διαπιστευμένου χρήστη (ή ενός διαπιστευμένου χρήστη που όμως δεν έχει τα δικαιώματα ενός διαχειριστή) σε οποιονδήποτε από τους παραπάνω συνδέσμους μπορεί να δώσει πληροφορίες που να χρησιμοποιηθούν για την υλοποίηση επίθεσης στην εφαρμογή,

2) Μια σελίδα χρησιμοποιεί την παράμετρο ‘action’ προκειμένου να διευκρινιστούν οι συναρτήσεις που καλούνται κάθε φορά. Η έλλειψη συγκεκριμένων διαπιστευτηρίων ως προϋπόθεση για την εκτέλεση συγκεκριμένων ενεργειών (‘actions’) μπορεί να γίνει αντικείμενο εκμετάλλευσης από έναν κακόβουλο χρήστη [15].

2.2.8 A8 – Cross – Site Request Forgery (CSRF)

Παράδειγμα Επίθεσης:

Μια εφαρμογή επιτρέπει σε έναν χρήστη να στείλει ένα αίτημα (request) που μεταβάλλει μία κατάσταση, χωρίς αυτό να περιέχει τίποτα κρυφό.

```
http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
```

Ο επιτιθέμενος κατασκευάζει ένα request το οποίο θα έχει ως αποτέλεσμα την μεταφορά χρημάτων από τον λογαριασμό του θύματος στον λογαριασμό του επιτιθέμενου και ύστερα ενσωματώνει τον κώδικα του παραπάνω request σε μια (οσοδήποτε μικρή) εικόνα ή iframe σε ιστότοπο που βρίσκεται υπό τον έλεγχό του, όπως φαίνεται παρακάτω:

```
 <img
src="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#" width="0" height="0" />
```

Αν το θύμα επισκεφτεί κάποιον τέτοιο ιστότοπο όσο είναι ταυτόχρονα συνδεμένος στο example. com, τα κακόβουλα requests θα περιέχουν τις πληροφορίες της συνόδου του, και άρα θα μπορούν να εκτελεστούν [15].

2.2.9 A9 – Using Components With Known Vulnerabilities

Παραδείγματα Επίθεσης:

Η χρήση των οποιονδήποτε από τις παρακάτω βιβλιοθήκες αποτελεί σημαντική ευπάθεια ασφάλειας για μια διαδικτυακή εφαρμογή, μιας και αυτές είναι προσβάσιμες στο σύνολο των χρηστών της εφαρμογής αυτής:

1) **Apache CXF Authentication Bypass**: Στο συγκεκριμένο framework υπηρεσιών διαδικτύου, οι επιτιθέμενοι μπορούν να καλέσουν οποιαδήποτε υπηρεσία διαδικτύου με πλήρη δικαιώματα.

2) **Spring Remote Code Execution:** Η εκμετάλλευση σφάλματος στον τρόπο υλοποίησης της Unified Expression Language στο συγκεκριμένο framework της Java επιτρέπει στους επιτιθέμενους την εκτέλεση κακόβουλου κώδικα που μπορεί να οδηγήσει στην πρόσβαση στον server [15].

2.2.10 A10 – Unvalidated Redirects and Forwards

Παραδείγματα Επίθεσης:

1) Μια εφαρμογή περιέχει μια σελίδα που ονομάζεται “redirect.sp’. Η σελίδα αυτή δέχεται μια μοναδική παράμετρο ‘url’.Ο επιτιθέμενος κατασκευάζει ένα κακόβουλο URL το οποίο ανακατευθύνει τον χρήστη σε κακόβουλο ιστότοπο:

```
http://www.example.com/redirect.jsp?url=evil.com
```

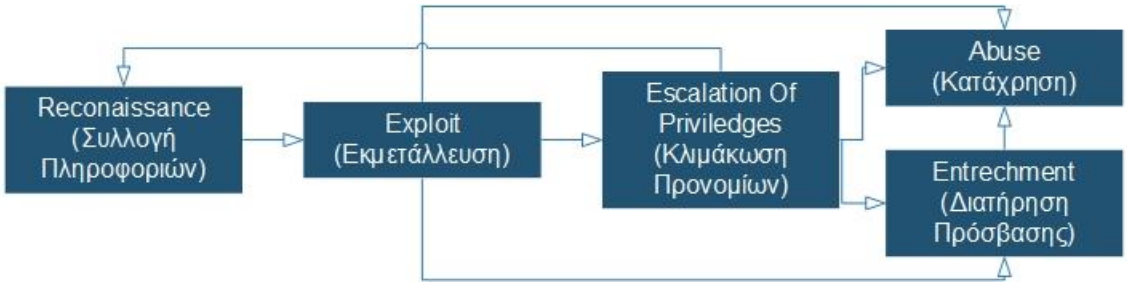
2) Μια εφαρμογή χρησιμοποιεί προωθήσεις για να δρομολογήσει αιτήματα μεταξύ των διαφορετικών μερών του ιστοτόπου. Για να υλοποιήσουν το παραπάνω, μερικές σελίδες χρησιμοποιούν μια παράμετορ η οποία δείχνει που θα πρέπει ν ασταλεί ο χρήστης αν μια συναλλαγη είναι επιτυχής. Ο επιτιθέμενος μπορεί να κατασκευάσει ένα URL το οποίο θα περάσει το επίπεδο ελέγχου πρόσβασης της εφαρμογής και θα επιτρέψει στον επιτιθέμενο να εκτελέσει λειτουργίες διαχειριστή για τις οποίες δεν έχει εξουσιοδότηση [15]:

```
http://www.example.com/boring.jsp?fwd=admin.jsp
```

2.3 Απειλές & Επιθέσεις

2.3.1 Μεθοδολογία Επίθεσης

Έχοντας περιγράψει τα είδη των ευπαθειών και τους σημαντικότερους κινδύνους ασφαλείας, το επόμενο βήμα για έναν επιτυχή ασφαλή σχεδιασμό διαδικτυακούς εφαρμογής είναι η κατανόηση του πώς αυτά χρησιμοποιούνται.Στην εικόνα 3 απεικονίζεται σχηματικά η συνήθης μεθοδολογία επίθεσης σε μια διαδικτυακή εφαρμογή.



Εικόνα 3: Μεθοδολογία Επίθεσης [16],[17]

2.3.1.1 Reconnaissance (Συλλογή πληροφοριών)

Το πρώτο στάδιο σε μια επίθεση είναι το στάδιο συλλογής πληροφοριών (Reconnaissance). Σε αυτό, ο επιτιθέμενος συλλέγει πληροφορίες σχετικά με τα πρωτόκολλα, τις τεχνολογίες που χρησιμοποιεί και τις λειτουργίες που επιτελεί μια διαδικτυακή εφαρμογή. Στοιχεία απαραίτητα για τον εντοπισμό μιας ευπάθειας που μπορεί να αποτέλεσει στόχο εκμετάλλευσης [16],[17].

2.3.1.2 Exploit (Εκμετάλλευση)

Το επόμενο στάδιο είναι η εκμετάλλευση της ευπάθειας που έχει εντοπιστεί σε οποιοδήποτε επίπεδο της αρχιτεκτονικής μιας διαδικτυακής εφαρμογής (Εικόνα 1), με σκοπό τη διείσδυση σε αυτή μέσω μη – εξουσιοδοτημένης πρόσβασης [16], [17].

2.3.1.3 Abuse (Κατάχρηση)

Η εκμετάλλευση της ευπάθειας επιτρέπει στον επιτιθέμενο να αποκτήσει προσωπικά δεδομένα ή να προκαλέσει κάποιου άλλου είδους ζημιά στη διαδικτυακή εφαρμογή, όπως μια επίθεση Denial Of Service. Το στάδιο αυτό, κατά το οποίο ο επιτιθέμενος διεξάγει τις κατάλληλες διαδικασίες προκειμένου να επιτύχει κακόβουλες ενέργειες καλείται Abuse (Κατάχρηση) [16],[17].

2.3.1.4 Escalation Of Privileges (Κλιμάκωση Προνομίων)

Εναλλακτικά, ο επιτιθέμενος μπορεί να εκμεταλλευτεί μια ευπάθεια για να κλιμακώσει τα προνόμιά του (Elevation Of Privileges) προκειμένου να αποκτήσει δικαίωμα πρόσβασης σε δεδομένα (Abuse) ή αντλώντας περισσότερες πληροφορίες για την εφαρμογή και τους χρήστες της (Information Gathering) [16],[17].

2.3.1.5 Entrenchment (Διατήρηση Πρόσβασης)

Η επιτυχής ανεπιθύμητη διείσδυση σε μια διαδικτυακή εφαρμογή (μέσω εκμετάλλευσης ή και κλιμάκωσης προνομίων) δεν οδηγεί απαιραίτητα σε μία και μοναδική παραβίαση δεδομένων. Ένας επιτιθέμενος μπορεί να θελήσει να διατηρήσει την πρόσβαση του στην εφαρμογή, διευκολύνοντας έτσι μελλοντικές επιθέσεις. Το στάδιο αυτό καλείται Entrenchment (διατήρηση πρόβαση). Ένας τρόπος υλοποίησης του είναι η εγκατάσταση προγράμματος πίσω πόρτας (backdoor) [16],[17].

2.3.2 Το Μοντέλο STRIDE

Είναι σαφές ότι οι επιθέσεις σε μια διαδικτυακή εφαρμογή διαφέρουν ανάλογα με το στάδιο της Εικόνας 3 στο οποίο ανήκουν: ανάλογα, δηλαδή, με τον σκοπό και τον τρόπο εκτέλεσής τους. Οι ενδεχόμενες επιθέσεις αποτελούν απειλή για την ασφάλεια μιας διαδικτυακής εφαρμογής από την πρώτη κιόλας μέρα διάθεσής της στο κοινό.

Μια γνωστή κατηγοριοποίηση των απειλών που αντιμετωπίζει μια διαδικτυακή εφαρμογή, ανάλογα με τον σκοπό τους και τον τρόπο υλοποίησης του, είναι το μοντέλο STRIDE (**S**poofing **T**ampering **R**epudiation Information **D**isclosure **D**enial Of Service **Elevation** of Privileges) που χρησιμοποιεί η Microsoft [18].

Απειλή	Περιγραφή
Spoofing Identity(Εξαπάτηση)	Προσπάθεια πρόσβασης με ψευδή ταυτότητα
Tampering (Αλλοίωση)	Τροποποίηση δεδομένων
Repudiation (Αποποίηση)	Άρνηση κακόβουλης ενέργειας από χρήστη που την πραγματοποίησε (μέσω κάλυψης των ιχνών του)
Information Disclosure (Αποκάλυψη Πληροφοριών)	Ανεπιθύμητη αποκάλυψη ιδιωτικών πληροφοριών (προσωπικών δεδομένων κ.α.)
Denial of Service (Άρνηση Παροχής Υπηρεσιών)	Διαδικασία μέσω της οποίας εφαρμογή ή σύστημα καθίσταται μη – διαθέσιμο στους χρήστες (μέσω αλλεπάλληλων αιτημάτων τα οποία δεν μπορούν να εξυπηρετηθούν)
Elevation of Privileges (Κλιμάκωση Δικαιωμάτων)	Ανεπιθύμητη αναβάθμιση των δικαιωμάτων ενός χρήστη

Πίνακας 5: το μοντέλο STRIDE [18]

2.4 Ασφαλής Σχεδιασμός Διαδικτυακών Εφαρμογών

2.4.1 Βασικές Αρχές Ασφάλειας

Βάσει των κατηγοριοποίησεων των ευπαθειών και απειλών που αναφέρθηκαν, προκύπτουν κάποιες γενικές αρχές ασφάλειας οι οποίες πρέπει να διαπνέουν τη λογική του σχεδιασμού και της υλοποιήσης μιας διαδικτυακής εφαρμογής.

Αρχή Ασφάλειας	Περιγραφή
Authentication (Αυθεντικοποίηση)	Διαδικασία ταυτοποίησης των χρηστών της εφαρμογής
Authorization (Εξουσιοδότηση)	Διαδικασία ελέγχου των δεδομένων που μπορεί να δει και των λειτουργιών που επιτρέπεται να χρησιμοποιήσει ένας αυθεντικοποιημένος χρήστης
Auditing (Καταγραφή)	Διαδικασία διασφάλισης της μη- αποποίησης ενεργειών από έναν χρήστη
Confidentiality (Εμπιστευτικότητα)	Διαδικασία διασφάλισης της απόκρυψης των δεδομένων από μη – εξουσιοδοτημένους χρήστες
Integrity (Ακεραιότητα)	Διαδικασία προστασίας των δεδομένων από αλλοιώσεις και τροποιήσεις
Availability (Διαθεσιμότητα)	Διαδικασία διασφάλισης απρόσκοπτης λειτουργίας της εφαρμογής

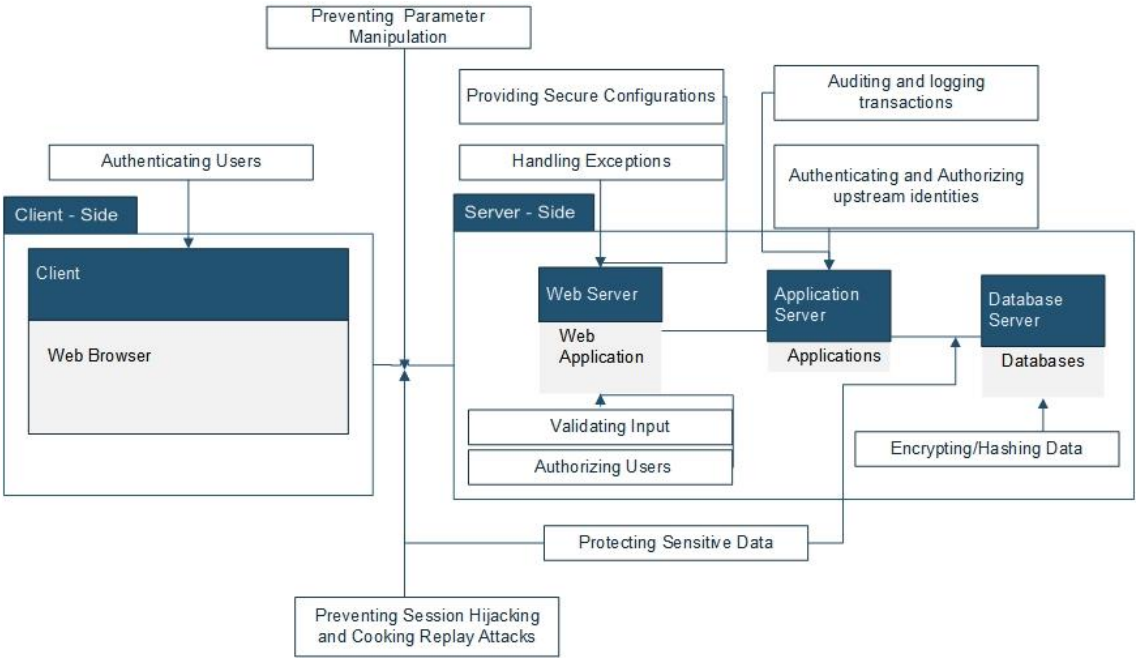
Πίνακας 6: Βασικές Αρχές Ασφάλειας [17]

2.4.2 Υλοποίηση Βασικών Αρχών Ασφάλειας

Οι βασικές αρχές ασφάλειας του Πίνακα 6 μπορούν να υλοποιηθούν με διάφορες μεθόδους, εργαλεία και τεχνολογίες.

Η σωστή μέθοδος αλλά και ο συγκεκριμένος τρόπος υλοποίησης τους κατά το σχεδιασμό μιας διαδικτυακής εφαρμογής είναι αυτός που διασφαλίζει την αντιμετώπιση των ευπαθειών και των απειλών ασφάλειας μιας διαδικτυακής εφαρμογής.

Στην εικόνα 4 παρουσιάζεται η υλοποίηση των βασικών αρχών ασφάλειας, και οι περιοχές εφαρμογής τους σε μια διαδικτυακή εφαρμογή σύμφωνα με τις Αρχές Σχεδιασμού Ασφαλών Διαδικτυακών Εφαρμογών της Microsoft [19].



Εικόνα 4: Αρχιτεκτονική Διαδικτυακής Εφαρμογής & Ασφάλεια [19]

Μια «διαφορετική», αναλυτικότερη παρουσίαση των αρχών ασφάλειας μιας διαδικτυακής εφαρμογής καθώς του τρόπου και μεθόδων υλοποίησής τους, πραγματοποιείται στον πίνακα 7.

Μέθοδος Υλοποίησης Αρχής Ασφάλειας	Περιγραφή
User authentication (Αυθεντικοποίηση Χρήστη)	Διαχωρισμός ιστοτόπου σε ανώνυμη και πιστοποιημένη περιοχή. Χρήση ισχυρών κωδικών. Εφαρμογή πολιτικής ανανέωσης κωδικών ανά τακτά χρονικά διαστήματα και δυνατότητα απενεργοποίησης λογαριασμών. Αποφυγή αποθήκευσης διαπιστευτηρίων (χρήση salted hashes). Κρυπτογράφηση διαύλων επικοινωνίας (με χρήση SSL) ώστε να προστατεύονται τα διαπιστευτήρια. Μεταφορά authentication cookies μόνο μέσω ασφαλών HTTPs συνδέσεων.
Parameter Manipulation Prevention (Προφύλαξη Από Αλλοίωση Παραμέτρων)	Κρυπτογράφηση των ευαίσθητων cookies. Μη εμπιστοσύνη των πεδίων που μπορεί να χειριστεί ένας χρήστης (queries, πεδία σε φόρμες, cookies, HTTP headers). Έλεγχος και πιστοποίηση όλων των σχετικών τιμών παραμέτρων που εισάγονται από χρήστες..
Session Management (Διαχείριση Συνόδου)	Εφαρμογή μικρής χρονικής διάρκειας συνόδων. Χρήση ασφαλών καναλιών επικοινωνίας. Κρυπτογράφηση του

	περιεχομένου των cookies αυθεντικοποίησης. Προστασία της συνόδου από μη – εξουσιοδοτημένη πρόσβαση.
Configuration Management (Διαχείριση Ρυθμίσεων)	Χρήση λογαριασμών ελαχίστων προνομίων. Αποφυγή αποθήκευσης διαπιστευτήριων χωρίς κρυπτογράφηση. Χρήση ισχυρών διαδικασιών αυθεντικοποίησης και εξουσιοδότησης στις σελίδες του διαχειριστή.
Sensitive Data Protection (Προστασία Ευαίσθητων Δεδομένων)	Κρυπτογράφηση των ευαίσθητων δεδομένων και χρήση ασφαλών διαύλων επικοινωνίας. Υλοποίηση ελέγχου πρόσβασης στα αποθηκευμένα δεδομένα. Αποφυγή αποθήκευσης ευαίσθητων δεδομένων σε persistent cookies. Αποφυγή μεταφοράς ευαίσθητων δεδομένων με τη χρήση του HTTP – GET Protocol.
Input Validation (Έλεγχος Εισαγωγής Δεδομένων)	Έλλειψη εμπιστοσύνης στα δεδομένα που εισάγονται από το χρήστη. Αποφυγή ελέγχου στην πλευρά του client. Περιορισμός της εισαγωγής. Διαδικασίες πιστοποίησης του τύπου και του μεγέθους των δεδομένων σε κάθε σημείο που εισάγονται δεδομένα.
User authorization (Εξουσιοδότηση Χρήστη)	Χρήση λογαριασμών ελάχιστων προνομίων. Χρήση λογαριασμών με διαφορετικά προνόμια. Περιορισμός της πρόσβασης των χρηστών σε πληροφορίες που σχετίζονται με την εφαρμογή.
Exception Management (Διαχείριση Εξαιρέσεων)	Χρήση δομημένης διαχείρισης εξαιρέσεων. Αποφυγή αποκάλυψης ευαίσθητων πληροφοριών σχετικά με την υλοποίηση της εφαρμογής. Αποφυγή αποθήκευσης ιδιωτικών δεδομένων όπως κωδικοί.
Auditing and Logging (Καταγραφή)	Αναγνώριση κακόβουλης συμπεριφοράς, σε αντίθεση με την φυσιολογική κίνηση. Καταγραφή δραστηριότητας σε κάθε επίπεδο της εφαρμογής. Διασφάλιση της πρόσβασης στα αρχεία καταγραφής.
Cryptography (Κρυπτογραφία)	Χρήση δοκιμασμένων δυνατοτήτων που παρέχονται από τις διάφορες πλατφόρμες. Χρήση των σωστών αλγόριθμων και μεγεθών κλειδιών.

Πίνακας 7: Τρόποι υλοποίησης βασικών αρχών ασφάλειας [18]

2.4.3 Αρχές, Απειλές & Κίνδυνοι Ασφάλειας

Συνοψίζοντας, όπως προκύπτει και από τις παραπάνω παραγράφους, ο ασφαλής σχεδιασμός διαδικτυακών εφαρμογών προϋποθέτει τη γνώση των ευπαθειών που μπορεί να έχει μια διαδικτυακή εφαρμογή, των επιθέσεων που τις εκμεταλλεύονται (ενδεχόμενες απειλές), και των τρόπων αντιμετώπισής τους (βασικές αρχές ασφάλειας και μέθοδοι υλοποίησης αυτών).

Η πλήρης κατανόηση και η λογική συσχέτιση όλων των παραπάνω είναι μία γνώση απαραίτητη για κάθε δημιουργό διαδικτυακών εφαρμογών.

Για αυτό το λόγο, στον πίνακα 8 επιχειρείται η παρουσίαση του τρόπου σύνδεσης των αρχών ασφάλειας, της υλοποίησής τους καθώς και των απειλών και κινδύνων ασφαλείας (σύμφωνα με το OWASP Top Ten 2013) που αντιμετωπίζουν.

Ο συγκεκριμένος πίνακας αποτελεί βασικό οδηγό για τον ασφαλή σχεδιασμό διαδικτυακών εφαρμογών, όπως και για την ανάπτυξη της υπόλοιπης εργασίας.

Αρχή Ασφάλειας	Μέθοδος Υλοποίησης	Απειλή	Κίνδυνος Ασφάλειας OWASP Top Ten 2013
Authentication	User Authentication, Input Validation	Spoofing Identity	Injection, Broken Authentication & Session Management, Cross – Site Request Forgery, Missing Function Level Access Control
Authorization	User Authorization, Input Validation	Escalation Of Privileges	Broken Authentication & Session Management, Missing Function Level Access Control
Auditing	Auditing & Logging, Session Management	Repudiation, Escalation Of Privileges	Broken Authentication & Session Management, Cross – Site Request Forgery
Confidentiality	Cryptography, Input Validation	Tampering, Information Disclosure	Sensitive Data Exposure, Insecure Data Direct Object References, Security Misconfiguration
Integrity	User authentication, User authorization,	Tampering	Cross – Site Scripting,

	Parameter Manipulation Prevention		Unvalidated Redirects & Forwards, Insecure Direct Object References
Availability	Parameter Manipulation Prevention, Exception Management	Denial Of Service	

Πίνακας 8: Συσχέτιση αρχών, απειλών και κίνδυνων ασφάλειας [15],[17],[18],[19]

2.4.4 Αντιμετώπιση των Κινδύνων Ασφάλειας του OWASP Top Ten 2013

Ο επαρκής ασφαλής σχεδιασμός μιας διαδικτυακής εφαρμογής, πέρα από γνώση των βασικών αρχών ασφαλείας και γενικών αρχών για την υλοποίησή τους απαιτεί και συγκεκριμένες ενέργειες για την πρόληψη όσο το δυνατόν περισσότερων ευπαθειών.

Η εμβάθυνση στις ενέργειες αυτές είναι απαραίτητη για τη συνέχεια της εργασίας. Για το λόγο αυτό χρησιμοποιούνται τα OWASP Prevention Cheat Sheets, οδηγίες αποτροπής του οργανισμού OWASP για τους δέκα σημαντικότερους κίνδυνους ασφάλειας που παρουσιάζονται στο OWASP Top Ten 2013.

2.4.4.1 Αντιμετώπιση του A1 - Injection

Η αποτροπή της συγκεκριμένης ευπάθειας απαιτεί τον διαχωρισμό των μη έμπιστων δεδομένων από τις εντολές και τα queries.

1. Η προτεινόμενη επιλογή είναι η χρήση μιας ασφαλούς προγραμματιστικής διεπαφής API η οποία αποφεύγει τελείως τη χρήση του διερμηνευτή (interpreter) και παρέχει μια παραμετροποιημένη διεπαφή (interface). Εδώ πρέπει να δοθεί προσοχή σε API's τύπου «αποθηκευμένες διαδικασίες» τα οποία είναι μεν παραμετροποιημένα αλλά μπορούν να παρουσιάσουν και αυτά ευπάθεια τύπου έγχυσης (injection), με όχι ορατό τρόπο. Η μη κατάλληλη χρήση των αποθηκευμένων διαδικασιών, όπως με την εισαγωγή δυναμικού SQL μέσα σε αυτές, τις καθιστούν τόσο επιβλαβείς όπως το δυναμικό SQL σε μια ιστοσελίδα. Όταν χρησιμοποιείται δυναμικό SQL μέσα σε αποθηκευμένες διαδικασίες θα πρέπει η εφαρμογή να έχει την δυνατότητα να ελέγχει τις εκχωρήσεις του χρήστη να ώστε να μειώνει τον κίνδυνο έγχυσης κώδικα. Αν δεν έχει αυτή την δυνατότητα τότε ο χρήστης μπορεί να εισάγει κακόβουλο SQL το οποίο θα εκτελεστεί με την αποθηκευμένη διαδικασία.

2. Αν δεν είναι διαθέσιμο ένα παραμετροποιημένο API, θα πρέπει προσεκτικά να αναιρεθεί η ειδική σημασία των ειδικών χαρακτήρων με την χρήση του ειδικώμ χαρακτήρων διαφυγής (escaping) για τον διερμηνευτή (interpreter).

3. Προτείνεται επίσης η θετική ή διαφορετικά «λευκή λίστα» (white-list) επικύρωσης εισόδου, π.χ. με χρήση κανονικών εκφράσεων – με άλλα λόγια η θεώρηση εισόδων που περιέχουν ειδικούς χαρακτήρες ως μη παραδεκτές. Η μέθοδος αυτή ωστόσο, δεν θεωρείται

πλήρης άμυνα, καθώς αρκετές εφαρμογές μπορεί να απαιτούν τη δυνατότητα χρήσης ειδικών χαρακτήρων στην είσοδό τους [15], [20]

2.4.4.2 Αντιμετώπιση του A2 - Broken Authentication and Session Management

Η κυριότερη σύσταση για την αποτροπή αυτής της ευπάθειας είναι η χρήση ισχυρών εργαλείων ελέγχου αυθεντικοποίησης της συνόδου. Αυτά τα εργαλεία ελέγχου θα πρέπει να: έχουν μια απλή διεπαφή για τους κατασκευαστές λογισμικού.

Μεγάλες προσπάθειες επίσης πρέπει να γίνουν για να αποφευχθούν ελαττώματα του XSS τα οποία μπορούν να οδηγήσουν σε κλοπή των αναγνωριστικών συνεδρίας (sessions id) [15],[21],[22].

2.4.4.3 Αντιμετώπιση του A3 - Cross Site Scripting (XSS)

Η αποτροπή της ευπάθειας XSS προϋποθέτει τον διαχωρισμό των μη έμπιστων δεδομένων από το ενεργό περιεχόμενο της εφαρμογής πλοήγησης (browser).

1. Η επιλογή που είναι προτιμότερη είναι να γίνει κατάλληλη χρήση διαφυγής (escaping) όλων των μη έμπιστων δεδομένων τα οποία βασίζονται στο πλαίσιο του προτύπου HTML (σώμα, ιδιότητες, Javascript, CSS ή URL) όπου τα δεδομένα μπορούν να τοποθετηθούν σε αυτό.

2. Η θετική ή λευκή λίστα (whitelist) επικύρωσης εισόδου προτείνεται επίσης, καθώς βοηθάει στην προστασία από την ευπάθεια XSS, αλλά αυτή δεν αποτελεί ολοκληρωμένη προστασία, καθώς αρκετές εφαρμογές μπορεί να απαιτούν τη δυνατότητα χρήσης ειδικών χαρακτήρων στην είσοδό τους. Μια τέτοια επικύρωση πρέπει να επικυρώνει το μήκος, τους χαρακτήρες και τη μορφή των δεδομένων εισόδου, προτού να κάνει αποδεκτή την είσοδο [15],[23].

2.4.4.4 Αντιμετώπιση του A4 - Insecure Direct Object References

Η αποτροπή της επισφαλούς άμεσης αναφοράς αντικειμένου ενός συστήματος απαιτεί την επιλογή μιας προσέγγισης για την προστασία κάθε αντικειμένου που είναι προσβάσιμο από τον χρήστη (π.χ. αριθμός, αντικείμενο, αρχείο):

1. Χρήση μόνο εμμέσων αναφορών αντικειμένου, με διαφορετικές τιμές ανά χρήστη ή ανά σύνοδο. Με τον τρόπο αυτό αποτρέπεται ο επιτιθέμενος να στοχεύει άμεσα σε πόρους στους οποίους δεν έχει εξουσιοδότηση. Για παράδειγμα, σε μια πτυσσόμενη λίστα (dropdown) με έξι αντικείμενα, αντί της χρησιμοποίησης του κλειδιού της βάσεως δεδομένων ως τιμή της επιλογής, είναι προτιμότερο είναι να χρησιμοποιούνται οι αριθμοί από το 1 έως το 6. Στην συνέχεια η εφαρμογή θα πρέπει να αντιστοιχήσει την τιμή που επέλεξε ο χρήστης με την πραγματική τιμή του κλειδιού της βάσης δεδομένων προκειμένου να εκτελεστούν οι υπόλοιπες διαδικασίες.

2. Έλεγχος πρόσβασης. Κάθε χρήση της άμεσης αναφοράς αντικειμένου από μια μη έμπιστη πηγή θα πρέπει να περιλαμβάνει έναν έλεγχο πρόσβασης για να διασφαλίσει ότι ο χρήστης είναι εξουσιοδοτημένος για το αντικείμενο που ζήτησε [15],[24].

2.4.4.5 Αντιμετώπιση του A5 - Security Misconfiguration

Οι κύριες συστάσεις για την αποτροπή της συγκεκριμένης ευπάθειας είναι να καθιερωθούν τα παρακάτω:

1. Μια επαναλαμβανόμενη διαδικασία ενίσχυσης της ασφάλειας που θα καθιστά γρήγορη και εύκολη την ανάπτυξη άλλου περιβάλλοντος το οποίο θα είναι κατάλληλα διασφαλισμένο. Το περιβάλλον ανάπτυξης, διασφάλισης ποιότητας και παραγωγής θα πρέπει να είναι συντονισμένα μεταξύ τους (με διαφορετικούς κωδικούς (passwords) το κάθε περιβάλλον). Η διαδικασία αυτή θα πρέπει να είναι αυτοματοποιημένη έτσι ώστε να ελαχιστοποιήσει την προσπάθεια που απαιτείται προκειμένου να εγκατασταθεί ένα νέο ασφαλές περιβάλλον.

2. Μια διαδικασία για την ενημέρωση και την εγκατάσταση των ενημερώσεων ασφαλείας και των επιδιορθώσεων του λογισμικού, η οποία θα πραγματοποιείται έγκαιρα σε όλα τα λειτουργούντα περιβάλλοντα. Η διαδικασία αυτή θα πρέπει να περιλαμβάνει οπωσδήποτε όλες τις βιβλιοθήκες κώδικα, οι οποίες συχνά παραλείπονται.

3. Μια ισχυρή αρχιτεκτονική της εφαρμογής η οποία παρέχει ασφαλή ασφάλεια ανάμεσα στα στοιχεία της.

4. Μια διαδικασία περιοδικής εκτέλεσης σαρώσεων και ελέγχων ασφάλειας βοηθά στον εντοπισμό των επισφαλών ρυθμίσεων ασφαλείας ή στον εντοπισμό μη εγκατεστημένων επιδιορθώσεων ασφαλείας [15].

2.4.4.6 Αντιμετώπιση του A6 - Sensitive Data Exposure

Για όλα τα ευαίσθητα δεδομένα χρειάζεται κατ' ελάχιστον να γίνουν τα παρακάτω:

1. Λαμβάνοντας υπόψη όλες τις απειλές από τις οποίες πρέπει να προστατευτούν τα δεδομένα (π.χ. εσωτερική επίθεση, εξωτερικός χρήστης), να εξασφαλιστεί ότι κρυπτογραφούνται όλα τα ευαίσθητα δεδομένα όταν είναι κάπου αποθηκευμένα όσο και όταν ανταλλάσσονται, με ένα τρόπο που να τα προστατεύει από αυτές τις απειλές.

2. Να μην αποθηκεύονται ευαίσθητα δεδομένα όταν δεν είναι απαραίτητο.

3. Να διασφαλιστεί ότι χρησιμοποιούνται ισχυροί αλγόριθμοι και δυνατά κλειδιά καθώς και ότι υπάρχει σωστή διαχείριση των κλειδιών.

4. Να διασφαλιστεί ότι οι κωδικοί πρόσβασης (passwords) είναι αποθηκευμένοι με την χρήση ενός αλγόριθμου που είναι σχεδιασμένος για την προστασία κωδικών πρόσβασης, όπως είναι ο bcrypt, PBKDF2, ή ο scrypt.

5. Να απενεργοποιηθεί η αυτόματη συμπλήρωση σε φόρμες που συγκεντρώνουν ευαίσθητα δεδομένα και να απενεργοποιηθεί το caching για σελίδες που περιέχουν ευαίσθητα δεδομένα.

2.4.4.7 Αντιμετώπιση του A7 - Missing Function Level Access Control

Η εφαρμογή θα πρέπει να έχει ένα συνεπές και εύκολο για ανάλυση στοιχείο εξουσιοδότησης που καλείται για χρήση από όλες τις επαγγελματικές λειτουργίες. Συχνά τέτοιου είδους προστασία παρέχεται από ένα ή περισσότερα εξωτερικά στοιχεία (components) σε σχέση με τον κώδικα της εφαρμογής.

1. Σκεφτείτε τη διαδικασία διαχείρισης των δικαιωμάτων και διασφαλίστε ότι μπορείτε να ανανεώσετε και να ελέγξετε εύκολα.

2. Οι μηχανισμοί ελέγχου πρόσβασης πρέπει να απαγορεύουν την πρόσβαση σε όλους εξ ορισμού, απαιτώντας ρητές εξουσιοδοτήσεις σε συγκεκριμένους χρήστες για την πρόσβαση σε κάθε λειτουργία.

3. Αν η λειτουργία εμπλέκεται σε μια ροη εργασίας (workflow), ελέγξτε για να βεβαιωθείτε ότι οι συνθήκες είναι σε κατάσταση τέτοια ώστε να επιτρέψουν την πρόσβαση [15],[25].

2.4.4.8 Αντιμετώπιση του A8 - Cross Site Request Forgery (CSRF)

Η αποτροπή του CSRF απαιτεί συνήθως να συμπεριλάβουμε ένα μη προβλέψιμο token στο σώμα κάθε HTTP αίτησης. Τέτοια διακριτικά πρέπει να είναι, τουλάχιστον μοναδικά ανά συνεδρία χρήστη.

1. Η προτιμόμενη επιλογή είναι να συμπεριλάβουμε το μοναδικό token σε ένα κρυφό πεδίο. Αυτό έχει ως συνέπεια, η τιμή να στέλνεται στο σώμα του αιτήματος HTTP, αποφεύγοντας την ενσωμάτωση του στο URL το οποίο εκτίθεται.

2. Το μοναδικό token μπορεί επίσης να συμπεριληφθεί στο ίδιο το URL ή σε μια παράμετρο αυτού. Μια τέτοια τοποθέτηση όμως, διατρέχει τον κίνδυνο της έκθεσης του URL στον επιτιθέμενο, με αποτέλεσμα να τίθεται σε κίνδυνο το μυστικό διακριτικό.

3. Η απαίτηση από τον χρήστη να επαναυθενικοποιείται (reauthenticate) ή το να αποδεικνύει το ότι είναι πραγματικό πρόσωπο (π.χ. με την χρήση του CAPTCHA) μπορούν επίσης να προστατεύσουν από ευπάθειες CSRF.

2.4.4.9 Αντιμετώπιση του A9 - Using Known Vulnerable Components

Μια λύση είναι να μην χρησιμοποιούνται εξωτερικά στοιχεία λογισμικού (components). Αλλά αυτό δεν είναι και τόσο ρεαλιστικό. Τα περισσότερα σύνολα στοιχείων λογισμικού (component projects) δεν επιδιορθώνουν τις ευπάθειες για τις παλιές εκδόσεις. Αντί αυτού επιδιορθώνουν το πρόβλημα στην επόμενη έκδοση. Συνεπώς η αναβάθμιση σε αυτές τις νέες εκδόσεις είναι κρίσιμη. Τα προγράμματα λογισμικού θα πρέπει να έχουν μια διαδικασία με σκοπό:

1. Να αναγνωρίζουν όλα τα στοιχεία λογισμικού και τις εκδόσεις που χρησιμοποιούνται συμπεριλαμβανομένων όλων των σχετικών (π.χ. τις εκδόσεις plug in).

2. Να παρακολουθούν την ασφάλεια αυτών των στοιχείων σε ελεύθερα προσπελάσιμες βάσεις δεδομένων, σε λίστες email και λίστες email ασφαλείας και να τις κρατάνε ενήμερες. 3. Να καθιερώνουν πολιτικές ασφάλειας που διέπουν την χρήση των στοιχείων (component),

όπως με το να απαιτούν ορισμένες πρακτικές ανάπτυξης λογισμικού, να περνάνε επιτυχώς τους ελέγχους ασφαλείας και να έχουν αποδεκτές άδειες.

3. Όπου είναι κατάλληλο, καλό είναι να προστίθενται περιτυλίγματα ασφαλείας γύρω από τα στοιχεία (components) για να απενεργοποιούν μη χρήσιμες λειτουργικότητες και/ή να ασφαλίζουν αδύναμα ή ευπαθή χαρακτηριστικά των στοιχείων (component).

2.4.4.10 Αντιμετώπιση του A10 - Unvalidated Redirects and Forwards

Ασφαλής χρήση των ανακατευθύνσεων ή των προωθήσεων σε μια εφαρμογή μπορεί να πραγματοποιηθεί με τους παρακάτω τρόπους:

1. Απλά, να σταματήσει η χρήση αυτών των μεθόδων.

2. Αν όμως, δεν μπορεί να σταματήσει η χρήση τους, τότε καλό είναι να μην γίνεται η χρήση των παραμέτρων των χρηστών στον υπολογισμό της σελίδας προορισμού.

3. Αν δεν μπορεί να αποφευχθεί η εισαγωγή των παραμέτρων των χρηστών, πρέπει να εξασφαλιστεί ότι οι τιμές που δίνουν οι χρήστες στο σύστημα είναι έγκυρες και εξουσιοδοτημένες για τον χρήστη. Συνίσταται δε, οι τιμές των παραμέτρων προορισμού να μην είναι το ίδιο το URL αλλά ένας κωδικός, ο οποίος μέσω απεικόνισης στην πλευρά του εξυπερετητή να οδηγεί στο URL στόχο. Οι εφαρμογές μπορούν να χρησιμοποιήσουν το ESAPI για να παρακάμψουν τη μέθοδο sendRedirect() και να εξασφαλίσουν ότι όλοι οι προορισμοί ανακατευθύνσεων είναι ασφαλείς.

Η αποφυγή τέτοιων λαθών είναι εξαιρετικά σημαντική δεδομένου ότι είναι ένας από τους αγαπημένους στόχους των επιτιθέμενων τέτοιου είδους (phishers), που προσπαθούν να κερδίσουν την εμπιστοσύνη των χρηστών [15],[27].

Κεφάλαιο 3: Εισαγωγή στο Προγραμματιστικό Πλαίσιο Django

3.1 Προγραμματιστικό Πλαίσιο (Framework)

Ο όρος προγραμματιστικό πλαίσιο (framework) στην Πληροφορική, αντλεί τη σημασία του από τον όρο σκελετό (επίσης framework) στην Μηχανική. Όπως και σε μια κτιριακή κατασκευή, έτσι και στην Πληροφορική, ένα framework αποτελεί την υποδομή στην οποία στηρίζεται ένα πρόγραμμα, μια εφαρμογή.

Αναλυτικά,ο όρος framework χρησιμοποιείται για ένα σύνολο κλάσεων και άλλων στοιχείων λογισμικού (συναρτήσεις, βιβλιοθήκες κ.α.) που αλληλοσχετίζονται, διευκολύνουν και επιταχύνουν τη διαδικασία ανάπτυξης μιας εφαρμογής, παρέχοντας ήδη έτοιμα τμήματα κώδικα που μπορούν να επαναχρησιμοποιηθούν [28].
Ένα framework, δηλαδή, είναι ένα επίπεδο αφαίρεσης κώδικα, τέτοιο ώστε να υλοποιεί γενικές λειτουργίες, οι οποίες μπορούν να συγκεκριμενοποιηθούν ανάλογα με τις ανάγκες της εφαρμογής που αναπτύσσεται [29].

Ένα σημαντικό χαρακτηριστικό των frameworks, που τα διακρίνει από τις απλές βιβλιοθήκες, τα APIs, κ.α., είναι ότι συμπεριλαμβάνουν λειτουργίες (συναρτήσεις,κλάσεις, και υπορουτίνες) που διαχειρίζονται τη ροή δεδομένων σε μια εφαρμογή ή πρόγραμμα.

Τα **web frameworks** (προγραμματιστικά πλαίσια εργασίας που χρησιμοποιούνται ειδικά για την ανάπτυξη διαδικτυακών εφαρμογών) διακρίνονται σε 2 βασικές κατηγορίες:

1) τα request – based web frameworks, που χρησιμοποιούν controllers και ενέργειες που χειρίζονται τα εισερχόμενα requests (αιτήματα).

2) τα component – based web frameworks, τα οποία δεν αφήνουν στον προγραμματιστή την αρμοδιότητα του χειρισμού των requests, και ενσωματώνουν τη λογική σε επαναχρησιμοποιούμενα components (συστατικά λογισμικού), που μπορεί να είναι και ανεξάρτητα από το μέσο που χρησιμοποιείται η εφαρμογή.

Τα πιο πολυχρησιμοποιημένα web frameworks ακολοθούν το Model – View – Controller Pattern (πρότυπο MVC), και έχουν ενσωματωμένα εργαλεία και κώδικα για την περιήγηση, την αυθεντικοποιήση/πιστοποίησητων χρηστών, τον χειρισμό της προβολής μιας σελίδας, τον χειρισμό ετικετών, την χρήση HTML κ.α [30].
Web frameworks υπάρχουν για το σύνολο των πιο γνωστών γλωσσών προγραμματισμού.
Μερικά παραδείγματα είναι το JSF (Java Server Faces) της Java, το Cake της PHP, το Rails της Ruby, και το Django της Python.

3.2 Model - View - Controller Pattern Design (Πρότυπο Σχεδιασμού MVC)

Το σύνολο των δυνατοτήτων και των υπηρεσιών που προσφέρει μια εφαρμογή, συνήθως στηρίζεται σε δύο βασικές λειτουργίες: α) την ανάκτηση δεδομένων, από μια αποθήκη

δεδομένων και η παρουσίασή τους στο χρήστη β) την δυνατότητα ενημέρωσης των δεδομένων από το χρήστη και αποθήκευσής των αλλαγών στην αποθήκη δεδομένων.

Είναι σαφές ότι η υλοποίηση των παραπάνω λειτουργιών σε ενιαίο κώδικα, δημιουργεί δυσκολίες και στην ανάπτυξη αλλά κυρίως στη συντήρηση μιας εφαρμογής, μιας και κάθε αλλαγή που θα προκύπτει σε ένα επίπεδο (π.χ. στο επίπεδο διεπαφής του χρήστη – user interface), θα απαιτούσε επέμβαση στο σύνολο του κώδικα και νέα έκδοση ολόκληρης της εφαρμογής.

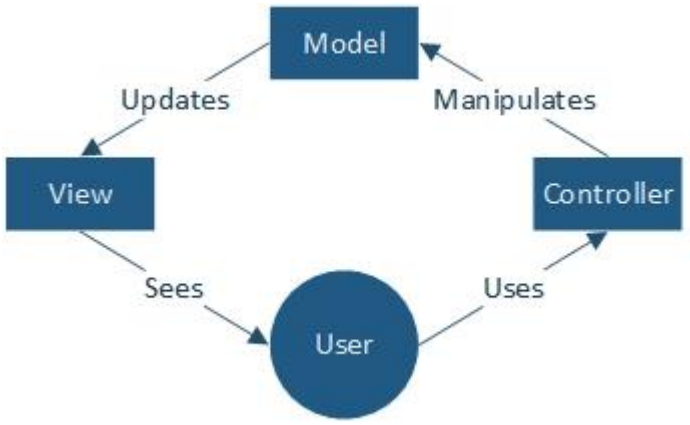
Μία πρόκληση που αντιμετωπίζει ένας προγραμματιστής κατά την ανάπτυξη μιας εφαρμογής, είναι η κατάλληλη τμηματοποίηση (modularity) του κώδικα, σε τμήματα κώδικα τέτοια ώστε η το κάθε ένα να υλοποιεί ξεχωριστή λειτουργία, καθιστώντας την ανάπτυξη του κάθε τμήματος διαδικασία ανεξάρτητη από την ανάπτυξη των υπολοίπων.
Ένας τέτοιος σχεδιασμός καθιστά την ανάπτυξη της εφαρμογής ταχύτερη (μιας και επιτρέπει τη συνεργασία πολλών προγραμματιστών), τη συντήρηση και τις μελλοντικές αλλαγές κολότερες και εν τέλει, την εφαρμογή πιο δυναμική [31].

Το **Model – View – Controller Design Pattern (πρότυπο σχεδιασμού MVC)** είναι μία φιλοσοφία σχεδιασμού λογισμικού,που εφευρέθηκε το 1970 στο Xerox Parc. και πλέον απευθύνεται σε αυτήν ακριβώς την πρόκληση, αν και αρχικά αφορούσε μόνο στα γραφικά περιβάλλοντα διεπαφής χρήστη (Graphical User Interfaces).

Σύμφωνα με τον δημιουργό του, Trygve Reenskaug, *«ο σκοπός του MVC είναι να γεφυρωθεί το κενό ανάμεσα στο μοντέλο σκέψης του χρήστη και το ψηφιακό μοντέλο που υπάρχει σε έναν υπολογιστή»* [32].Όπως αποκαλύπτει και το όνομά του, σύμφωνα με το MVC Design Pattern, μια εφαρμογή χωρίζεται σε τρία επίπεδα: το Model, το View και το Controller.Τα τρία αυτά επίπεδα θα περιγραφούν αναλυτικότερα στις επόμενες παραγράφους.

Όπως αναφέρθηκε προηγουμένως, το MVC Pattern δεν είναι κάποια συγκεκριμένη υλοποίηση κώδικα: αλλά μια λογική, ένα αφαιρετικό πρότυπο, το οποίο ακολουθεί η ανάπτυξη ενός προγράμματος.

Το παρακάτω σχήμα περιγράφει σχηματικά το MVC design Pattern:



Εικόνα 5: MVC Design Pattern [33]

3.2.1 Model

Το Model είναι το τμήμα του κώδικα που διαχειρίζεται όλες τις λειτουργίες που σχετίζονται με τα δεδομένα: την πιστοποίηση των δεδομένων (validation), κατάσταση συνόδου (session state), την δομή και τον έλεγχο της βάσης δεδομένων. Η ύπαρξή του απλοποιεί τον κώδικα που χρειάζεται να γράψει ο ίδιος ο προγραμματιστής [35]. Το επίπεδο Model (Model layer) αναλαμβάνει την υλοποίηση της επιχειρισιακής λογικής (business logic) μιας εφαρμογής. Ενσωματώνει μεθόδους πρόσβασης στα απαραίτητα δεδομένα (βάσεις δεδομένων αρχεία κ.α.

Το επίπεδο Model αποτελείται από τις κλάσεις που καθορίζουν το domain της εφαρμοργής. Τα αντικείμενα (objects) που ανήκουν στο domain, ενσωματώνουν δεδομένα που έχουν αποθηκευτεί στη βάση δεδομένων αλλά ταυτόχρονα περιέχουν κώδικα που χρησιμοποιείται για τη διαχείριση των δεδομένων αυτών και για την τήρηση των επιχειρησιακών κανόνων (business rules) [34],[36].

3.2.2 View

Το τμήμα του κώδικα του View είναι υπεύθυνο για την διαχείριση των λειτουργιών που σχετίζονται με την γραφικό περιβάλλον των διεπαφών του χρήστη. Αυτό σημαίνει όλα τα HTML elements (buttons, forms κ.α.). Η ύπαρξη του View επιτυγχάνει το διαχωρισμό του σχεδιασμού από τη λογική μιας εφαρμογής, πράγμα που ελαχιστοποιεί την πιθανότητα να προκύψουν σφάλματα στην λειτουργία μιας εφαρμογής όταν ο σχεδιαστής επιχειρήσει να αλλάγει το interface ή το logo της [35]

Συχνά το επίπεδο View αναφέρεται ως web design ή templates. Ελέγχει τον τρόπο παρουσίασης των δεδομένων καθώς και την αλληλεπίδραση του χρήστη με αυτά. Οι τεχνολογίες που χρησιμοποιούνται σε αυτό είναι κυρίως οι scripting γλώσσες HTML, CSS, JavaScript [34].

Σήμερα, πολλά web frameworks κάνουν χρήση template engine η οποία με χρήση κατάλληλου κώδικα παράγει η ίδια τα αναγκαία HTML elements, ελαχιστοποιώντας, έτσι, τη χρήση HTML από πλευράς προγραμματιστή [37].

3.2.3 Controller

Το Controller είναι εκείνο το τμήμα κώδικα που αναλαμβάνει το χειρισμό των events. Τα events ενεργοποιούνται είτε από την αλληλεπίδραση του χρήση με την εφαρμογή, είτε από μια εσωτερική διαδικασία της εφαρμογής [35].

Είναι αυτό που συνδέει την παρουσίαση των δεδομένων του View με τις λειτουργίες του Model. Ένα Controller δέχεται requests (αιτήματα), ανακτά τα δεδομένα από το Model, και δημιουργεί το κατάλληλο View το οποίο θα προβάλλει τα δεδομένα αυτά στον χρήστη [36].

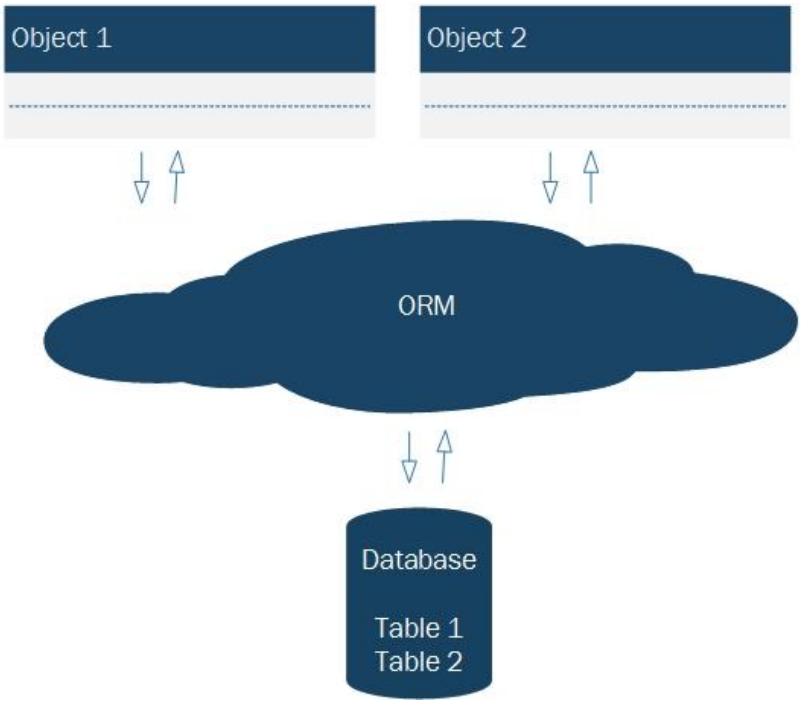
3.3 Object – Relational Mapping (ORM)

Το πρότυπο MVC περιγράφει εν πολλοίς μία φιλοσοφία χειρισμού και επεξεργασίας δεδομένων που εισάγονται ή προβάλλονται από έναν χρήστη. Ο τρόπος αποθήκευσής και ανάκτησης των δεδομένων από τη βάση,τα δεδομένα αυτά καθαυτά καθώς και ο χειρισμός και η δημιουργία της βάσης δεδομένων, είναι ένα ακόμη θέμα στο οποίο απευθύνονται πολλά frameworks.

Ο όρος **Object Relational Mapping (ORM)** χρησιμοποιείται για να ορίσει έναν μηχανισμό μέσω του οποίου λογισμικά που βασίζονται στον αντικειμενοστραφή προγραμματισμό μπορούν να αποθηκεύσουν δεδομένα με ασφαλή τρόπο για μεγάλο χρονικό διάστημα, έχοντας τη δυνατότητα διαχείρισης και προβολής τους ως αντικείμενα εντός τους [37]. Ένα σύστημα ORM επιτρέπει σε έναν προγραμματιστή να μην ασχολείται με τη δομή μιας βάσης δεδομένων. [38].

Ουσιαστικά, ένα σύστημα Object Relational Mapping είναι ένας τρόπος αντιστοίχισης των κλάσεων μιας γλώσσας προγραμματισμού με τα δεδομένα μιας βάσης. Μέσω αυτής της αντιστοίχισης, ένας προγραμματιστής έχει τη δυνατότητα να διαχειριστεί τα δεδομένα μέσω μεθόδων και χαρακτηριστικών των κλάσεων στις οποίες αντιστοιχίζονται.

Πολλά frameworks, μεταξύ των οποίων και το Django διαθέτουν σύστημα ORM ως έναν τρόπο για εύκολη και ασφαλή πρόσβαση στη βάση δεδομένων. Το ORM λειτουργεί στο παρασκήνιο του επιπέδου Model [39].



Εικόνα 6: Η τεχνική Object Relational Mapping [40]

3.4 Αρχιτεκτονική του Προγραμματιστικού Πλαίσιου Django

Το προγραμματιστικό πλαίσιο Django είναι ένα web framework, ανεπτυγμένο στην αντικειμενοστραφή γλώσσα Python, που ακολουθεί το πρότυπο MVC, με τη βοήθεια συστήματος ORM [13].

Όμως, αντί για τα Model – View – Controller, τα τρία βασικά συστατικά/επίπεδα (components/layers) του Django είναι το **Model, Template και View (MTV)**. Η ιδιαιτερότητα του Django σε σχέση με τα υπόλοιπα MVC frameworks, είναι οι δύο βασικές διαφορές που παρουσιάζει.

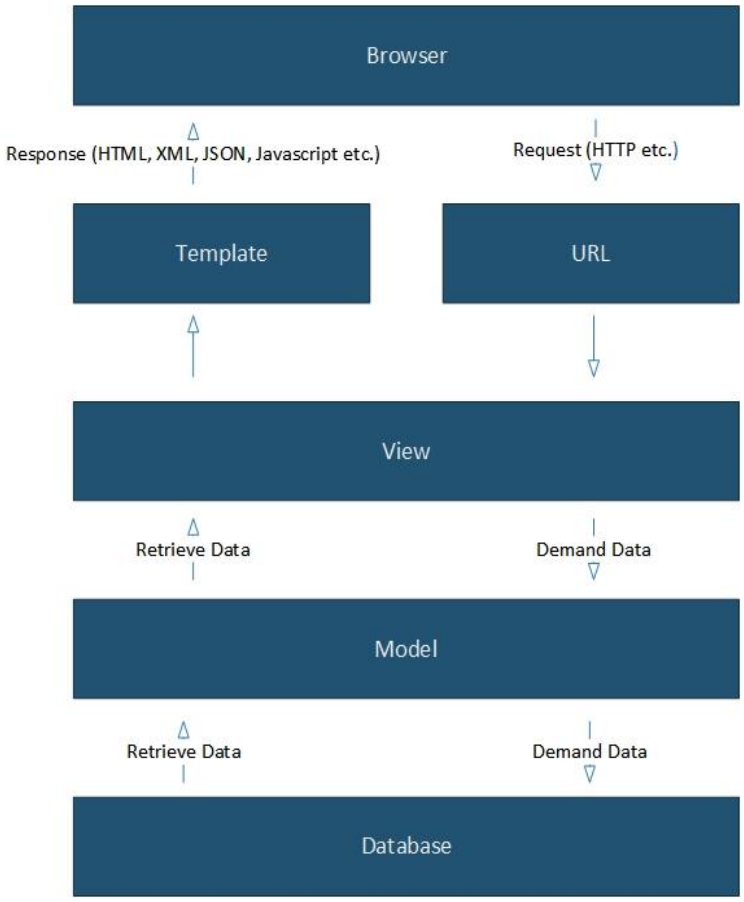
Η πρώτη βασική διαφορά του Django έγκειται στον τρόπο παρουσίασης των δεδομένων. Στα περισσότερα MVC frameworks το View είναι αυτό που διαχειρίζεται ποιά δεδομένα θα προβληθούν στον χρήστη αλλά και τον τρόπο που αυτά θα προβληθούν (με χρήση κώδικα HTML). Το Django χωρίζει αυτές τις δύο δραστηριότητες στα επίπεδα View και Template ως εξής: το View διαχειρίζεται ποιά δεδομένα θα προβληθούν στον χρήστη (επικοινονώντας με το Model) και ταυτόχρονα καλεί το template, το οποίο είναι υπεύθυνο για τον τρόπο με τον οποίο αυτά θα παρουσιαστούν (με χρήση κώδικα HTML).

Η δεύτερη βασική διαφορά του Django βρίσκεται στο επίπεδο Controller. Σε αντίθεση με το κλασσικό πρότυπο MVC, το Controller στο Django, υπό μία έννοια, είναι το ίδιο το framework: Ο μηχανισμός που αναλαμβάνει να συνδυάσει ένα request με το κατάλληλο view, σύμφωνα με ένα κομμάτι κώδικα που καλείται URL conf είναι ενσωματομένος στο framework. Το μόνο που χρειάζεται να υλοποιήσει ένας προγραμματιστής, είναι το εκάστοτε URLconf [41].

MVC Design Pattern	Django Web Framework
Model	Model
View	View + Template
Controller	Framework Itself (URLconf)

Πίνακας 9: Αντιστοιχία MVC Design Pattern και Django Architecture

Στην παρακάτω εικόνα περιγράφεται σχηματικά ο τρόπος λειτουργίας του Django framework:



Εικόνα 7: Τρόπος λειτουργίας του Django framework [42]

Όπως έχει ήδη ειπωθεί, το Django είναι ανεπτυγμένο εξ' ολοκλήρου σε γλώσσα Python. Ο κώδικάς κάθε component του, είναι γραμμένος σε αρχεία με την κατάληξη .**py**. Το ίδιο το framework αποτελείται κυρίως από Python κλάσεις και συναρτήσεις που αλληλεπιδρούν μεταξύ τους.

Ο τρόπος λειτουργίας του Django έχει σκιαγραφηθεί. Το επόμενο βήμα για την καλύτερη κατανόησή του είναι ο τρόπος υλοποίησης των components του, δηλαδή η αναλυτική περιγραφή των Model, Template, View, URL κ.α.

Στις επόμενες παραγράφους θα παρουσιαστούν τα κύρια components τα οποία περιέχει το Django και στα οποία βασίζονται οι διαδικτυακές εφαρμογές που αναπτύσσονται με αυτό.

3.5 The Model layer

3.5.1 Models

Ένα model είναι η μοναδική πηγή πληροφοριών για τα δεδομένα μιας εφαρμογής. Περιέχει τα απαραίτητα χαρακτηριστικά και τρόπο συμπεριφοράς των δεδομένων που αποθηκεύονται στη

βάση. Κάθε model που δημιουργείται αντιστοιχίζεται με ένα πίνακα (table) της βάσης δεδομένων.

Κάθε model είναι μια Python υποκλάση της κλάσης **django.db.models.Model**.
Κάθε attribute (χαρακτηριστικό) του model αναπαριστά ένα database field.

Παράδειγμα:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Σε αυτή την περίπτωση, το **class Person** (subclass του **django.db.models.Model**) είναι ένα model. Τα **first_name** και **last_name** είναι τα fields του model.

Ο παραπάνω κώδικας ισοδυναμεί με ένα query που δημιουργεί ένα database table ως εξής:

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Σημείωση: *Το όνομα **myapp_person** είναι αυτό το default όνομα το οποίο δίνει το Django (προέρχεται από το όνομα της εφαρμογής και το όνομα του class).*

3.5.1.1 Fields

Το πιο σημαντικό μέρος ενός model είναι η λίστα με τα database fields που περιέχονται σε αυτό. Κάθε field καθορίζεται από το αντίστοιχο class attribute.

Παράδειγμα:

```
class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

3.5.1.2 Field Type

Κάθε field σε ένα model είναι ένα instance της κατάλληλης κλάσης Field. Στο παραπάνω παράδειγμα, το **artist** είναι ένα instance της κλάσης ForeignKey, ενώ το **num_stars** ένα instance της κλάσης **CharField**.

Το Django χρησιμοποιεί τύπους της κλάσης Field (field class type) προκειμένου να

καθορίζσει τα εξής:

α) Τον τύπο του database column (που αντιστοιχεί στο field) , που υποδεικνύει στην βάση δεδομένων τί τύπο δεδομένων θα δεχθεί (**π.χ. INTEGER, VARCHAR, TEXT**).

β)Τί HTML ετικέτα να χρησιμοποιηθεί όταν θα προβληθεί μια φόρμα (π.χ. <input type="text">).

γ) Τα προαπαιτούμενα που χρειάζονται για να πιστοποιηθεί η ακεραιότητα των δεδομένων, που χρησιμοποιούνται στη σελίδα admin του Django και σε forms που δημιουργεί αυτόματα το framework.

Παραδείγματα field types είναι τα **ForeignKey, ManyToManyField** κ.α.

3.5.1.3 Field Options

Κάθε field, ανάλογα με τον τύπο του, δέχεται συγκεκριμένο σύνολο από arguments. Για παράδειγμα το **CharField** δέχεται το **max_length** argument. Επίσης συνθισμένα arguments τα οποία μπορούν να χρησιμοποιηθούν σε όλα τα field types είναι τα **null,blank,default, primary_key, unique κ.α.**

α) Primary Key Fields

Το Django δίνει αυτόματα σε κάθε model το παρακάτω field:

```
id = models.AutoField(primary_key=True)
```

το οποίο είναι ένα auto – incrementing primary key (πρωτεύον κλειδί του database table που αντιστοιχίζεται στο model).

β) Verbose Field Names

Κάθε field type εκτός από τα **ForeignKey, ManyToMany**, έχει τη δυνατότητα να δεχθεί στην πρώτη θέση των arguments ένα argument που καλείται verbose name. Σε περίπτωση που ο προγραμματιστής δεν εισάγει κάποιο verbose name, το Django θα το δημιουργήσει αυτόματα χρησιμοποιώντας το όνομα του attribute του Field, μετατρέποντας τα underscores σε κενά. Στο παρακάτω παράδειγμα, το verbose name είναι το "**person's first name**".

```
first_name = models.CharField("person's first name", max_length=30)
```

Στο παρακάτω παράδειγμα, το verbose name είναι το "fist name" (δεν έχουμε εισάγει κάποιο verbose name, οπότε το Django χρησιμοποιεί το όνομα του attribute ως τέτοιο):

```
first_name = models.CharField(max_length=30)
```

Τα ForeignKey, ManyToManyField και **OneToOneField** απαιτούν το πρώτο argument να είναι κλάση , προκειμένου να χρησιμοποιήσου το argument **verbose_name**.

Παράδειγμα:

```
poll = models.ForeignKey(
    Poll,
    on_delete=models.CASCADE,
    verbose_name="the related poll",
)
```

Στο παραπάνω παράδειγμα, το **Poll** είναι το όνομα της κλάσης **Poll**.

3.5.1.4 Relationships

Το Django προσφέρει τρόπους υλοποίησης των τριών πιο κοινών ειδών συσχετίσεων μεταξύ πινάκων μιας βάσης (database relationships): **many-to-one, many-to-many, one-to-one.**

α) Many – to – one Relationships:

Για να δημιουργηθεί μία συσχέτιση **many – to – one**, πρέπει να συμπεριληφθεί το field **django.db.models.ForeignKey** ως class attribute του model που θα λάβει μέρος στη συσχέτιση.

Το **ForeignKey** απαιτεί στην πρώτη θέση ένα argument που είναι το όνομα της κλάσης με την οποία το model θα συσχετιστεί.

Παράδειγμα:

Στο παρακάτω παράδειγμα ένα model με όνομα **Car** έχει έναν **Manufacture**r. Ο **Manufacturer** φτιάχνει πολλά cars (άρα συσχετίζεται με πολλές κλάσεις **Car**) όμως κάθε κλάση **Car** έχει μοναδικό Manufacturer.
Την παραπάνω περίπτωση υλοποιεί ο παρακάτω κώδικας:

```
from django.db import models

class Manufacturer(models.Model):
    # ...

class Car(models.Model):
    manufacturer          =          models.ForeignKey(Manufacturer,
on_delete=models.CASCADE)
    # ...
```

β) Many – to – Many:

Για την υλοποίηση **many – to – many** συσχετίσεων, χρησιμοποείται το **ManyToManyField**. Όπως και το **ForeignKey**, τοποθετείται ως class attribute του model το οποίο θέλουμε να συσχετίσουμε με άλλο model.
Για να δηλώσουμε το model με το οποίο θέλουμε να συσχετιστεί το επιθυμητό model, θέτουμε ως πρώτο argument το όνομά του.

Παράδειγμα:

Μία κλάση **Pizza** συσχετίζεται με πολλά αντικείμενα **Toppings**. Αντίστοιχα, ένα **Topping** συσχετίζεται με πολλές κλάσεις **Pizza**.
Ο κώδικας που θα υλοποιεί την παραπάνω περίπτωση είναι ο παρακάτω:

```
from django.db import models

class Manufacturer(models.Model):
    # ...
    pass

class Car(models.Model):
    manufacturer          =          models.ForeignKey(Manufacturer,
on_delete=models.CASCADE)
    # ...
```

γ) One – to – One:

Για να υλοποιηθεί μία **one – to – one** συσχέτιση, χρησιμοποιείται το **OneToOneField**. Το **OneToOneField** χρησιμοποιείται όπως κάθε άλλο Field: ως class attribute στο model που θα συμπεριληθφεί στη συσχέτιση. Το **OneToOneField** απαιτεί ως argument το όνομα του model με το οποίο το επιλεγμένο model σχετίζεται. Συνήθως, η χρήση του **OneToOneField** είναι χρήσιμη για αντικείμενα που αποτελούν επέκταση άλλων αντικειμένων.

Παράδειγμα:

Αν σε μία εφαρμογή έχει υλοποιηθεί η κλάση **Place** και υπάρχει ανάγκη να δημιουργηθεί μία κλάση **Restaurant**, τότε στην κλάση **Restaurant** θα γίνει χρήση του field type **OnetoOneField** με την μορφή **OnetoOneField** (**Place**).

3.5.1.5 Models Across Files

Στο Django υπάρχει η δυνατότητα χρήσης ενός model μιας διαδικτυακής εφαρμογής σε μια άλλη. Προκειμένου να συμβεί αυτό, χρειάζεται η κλάση του model να γίνει import στην επιθυμητή εφαρμογή, με κατάλληλη εντολή στην αρχή του αρχείου του κώδικα.

Παράδειγμα:

```
from django.db import models
from geography.models import ZipCode

class Restaurant(models.Model):
    # ...
    zip_code = models.ForeignKey(
        ZipCode,
        on_delete=models.SET_NULL,
        blank=True,
        null=True,
    )
```

3.5.1.6 Meta Options (metadata)

Σε ένα model, υπάρχει η δυνατότητα να δοθούν **metadata** (δεδομένα που να αφορούν το ίδιο το model) με τη χρήση μιας εσωτερικής κλάσης με τίτλο **Meta**. Τα model metadata είναι οτιδήποτε δεν αποτελεί instance κάποιου field, όπως επιλογές σχετικά με την κατάταξη των δεδομένων (ordering), το όνομα του πίνακα της βάσης δεδομένων που αντιστοιχεί στο model (db_table), τα verbose names κ.α.

Παράδειγμα:

```
from django.db import models

class Ox(models.Model):
    horn_length = models.IntegerField()

    class Meta:
        ordering = ["horn_length"]
        verbose_name_plural = "oxen"
```

3.5.1.7 Model Manager

Το πιο σημαντικό attribute (χαρακτηριστικό) ενός model είναι το **Manager**. Το **Manager** είναι μια κλάση, ξεχωριστή για κάθε model, η οποία διαχειρίζεται την αλληλεπίδραση των models με τον αντίστοιχο πίνακα της βάσης δεδομένων, δηλαδή τα queries στον αντίστοιχο πίνακα. Χρησιμοποιείται για να ανακτήσει αντικείμενα με πληροφορίες, δεδομένα που έχουν αποθηκευτεί στη βάση. Αν ένας προγραμματιστής δεν ορίσει κάποιον Manager, το Django παρέχει αυτόματα έναν με το όνομα **objects**.

Παράδειγμα:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    role = models.CharField(max_length=1, choices=(('A',
_('Author')), ('E', _('Editor'))))
    people = models.Manager()
    authors = AuthorManager()
    editors = EditorManager()

class AuthorManager(models.Manager):
    def get_queryset(self):
        return super(AuthorManager,
self).get_queryset().filter(role='A')
```

Στο παραπάνω παράδειγμα η κλάση **AuthorManager** είναι ο **Manager** που έχει ορίσει ένας προγραμματιστής για την κλάση **Person**. Γράφοντας στον κώδικα ή σε ένα command prompt το **Person.authors.all(),** η βάση δεδομένων θα επιστρέψει στον χρήστη όλες τις εγγραφές Person που έχουν role ‘A’ (δηλαδή Author).

3.5.1.8 Model Methods

Το Django παρέχει τη δυνατότητα να οριστούν **methods** (μέθοδοι) σε ένα model. Οι μέθοδοι ουσιαστικά χρησιμοποιούνται για να παρέχουν πληροφορίες σε «επίπεδο εγγραφής» μιας βάσης δεδομένων,προκειμένου δηλαδή να επιστρέφονται πληροφορίες που αφορούν ένα συγκεκριμένο αντικείμενο μιας κλάσης model.

Παράδειγμα:

Στο παρακάτω παράδειγμα το **baby_boomer_status()** είναι μια μέθοδος του model **Person**:

```
from django.db import models

class Person(models.Model):
birth_date = models.DateField()

    def baby_boomer_status(self):
        "Returns the person's baby-boomer status."
        import datetime
        if self.birth_date < datetime.date(1945, 8, 1):
            return "Pre-boomer"
        elif self.birth_date < datetime.date(1965, 1, 1):
            return "Baby boomer"
        else:
            return "Post-boomer"
```

Η κλήση της μεθόδου **baby_boomer_status ()** μπορεί να μας δώσει πληροφορίες για ένα αντικείμενο της κλάσης **Person**, ελέγχοντας την ημερομηνία γέννησής του (**birth_date**).
Το Django παρέχει κάποιες βασικές methods, όπως οι παρακάτω:

α) __str__() (στην Python 3.x ή __unicode__ στην Python 2.x)

Η συγκεκριμένη μέθοδος επιστρέφει μία **unicode** αναπαράσταση ενός αντικειμένου. Χρησιμοποιείται από το Django και την Python, όταν χρειάζεται να αναπαρασταθεί ένα αντικείμενο με την μορφή **string** (π.χ. στην σελίδα admin του Django ή στο command prompt).

β) get_absolute_url()

Η συγκεκριμένη μέθοδος χρησιμοποιείται για να υπολογίσει το URL ενός αντικειμένου. Το Django κάνει χρήση της μεθόδου στο interface της admin σελίδας του. Κάθε αντικείμενο που έχει μοναδικό URL χρειάζεται τη συγκεκριμένη μέθοδο.

Οι methods που παρέχει το Django μπορούν να γίνουν override (δηλαδή να αλλάξει η λειτουργία τους), αν ο προγραμματιστής επιλέξει, χρησιμοποιώντας το όνομα μιας μεθόδου, να υλοποιήσει με κατάλληλο κώδικα διαφορετική λειτουργία στο εσωτερικό της.

Παράδειγμα:

Στο παρακάτω παράδειγμα, γίνεται κατανοητό πώς μπορεί να αλλάξει η λειτουργία της μεθόδου **save()** που παρέχει το Django:

```
from django.db import models

class Blog(models.Model):
```

```
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def save(self, *args, **kwargs):
        do_something()
        super(Blog, self).save(*args, **kwargs) # Call the "real"
save() method.
        do_something_else()
```

3.5.1.9 Model Inheritance

Η κληρονομικότητα στο Django λειτουργεί ακριβώς όπως και η κληρονομικότητα σε κάθε άλλη κλάση της Python. Φυσικά, η κλάση – γονέας πρέπει να είναι υποκλάση της κλάσης **django.db.models.Model.**

Παράδειγμα:

```
from django.db import models

class CommonInfo(models.Model):
    name = models.CharField(max_length=100)
    age = models.PositiveIntegerField()

    class Meta:
        abstract = True

class Student(CommonInfo):
    home_group = models.CharField(max_length=5)
```

Στο παραπάνω παράδειγμα, το model Student έχει το field **home_group**, καθώς και fields **name** και **age** που κληρονομεί από το model – γονέα **CommonInfo** [43].

3.5.2 Database (Βάση Δεδομένων)

3.5.2.1 Εκτέλεση Queries

Το Django παρέχει ένα **API** το οποίο επιτρέπει την δημιουργία, ανάκτηση, ενημέρωση και διαγραφή αντικειμένων που βασίζονται σε κλάσεις model που έχει ήδη δημιουργήσει ο προγραμματιστής. Για τις ανάγκες της παρουσίασης των παραδειγμάτων στη συγκεκριμένη παράγραφο, θα γίνει χρήση των παρακάτω κλάσεων, οι οποίες είναι τα models ενός διαδικτυακού blog:

```
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):              # __unicode__ on Python 2
```

```
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

    def __str__(self):              # __unicode__ on Python 2
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):              # __unicode__ on Python 2
        return self.headline
```

3.5.2.2 Δημιουργία Αντικειμένων

Για τις ανάγκες αναπαράστασης των δεδομένων πινάκων δεδομένων με την μορφή αντικειμένων της Python, το Django χρησιμοποιεί τον εξής μηχανισμό ORM:
Κάθε κλάση model αναπαριστά έναν πίνακα βάσης δεδομένων, και κάθε αντικείμενο αυτής της κλάσης, αναπαριστά μια εγγραφή σε αυτό τον πίνακα.

Προκειμένου να δημιουργηθεί ένα αντικείμενο μιας κλάσης model, γίνεται χρήση keyword arguments εντός της κλάσης ώστε να οριστεί, και ύστερα χρήση της μεθόδου **save()** ώστε να αποθηκευτεί ως εγγραφή στον κατάλληλο πίνακα της βάσης δεδομένων.

Παράδειγμα:

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles
news.')
>>> b.save()
```

Στο παραπάνω παράδειγμα διεκρυνίζεται ο τρόπος δημιουργίας ενός αντικειμένου. Ο παραπάνω κώδικας υλοποιεί την κλήση του SQL statement **INSERT** στο παρασκήνιο. Το Django δεν επικοινωνεί με την βάση δεδομένων μέχρι την κλήση της μεθόδου **save()**.

3.5.2.3 Αποθήκευση Αλλαγών σε Αντικείμενα

Προκειμένου να αποθηκευτούν οι αλλαγές που πραγματοποιούνται σε αντικείμενα που έχουν ήδη αποθηκευτεί στη βάση δειδιμένων, γίνεται και πάλι, χρήση της μεθόδου save.

Παράδειγμα:

Αν **b5** ένα αντικείμενο της model κλάσης **Blog** που έχει ήδη αποθηκευτεί στη βάση, ο παρακάτω κώδικας αλλάζει το attribute "**name**" του **b5** σε «**New name**» και αποθηκεύει το αποτέλεσμα στη βάση:

```
>>> b5.name = 'New name'
>>> b5.save()
```

Ο παραπάνω κώδικας εκτελεί ένα SQL statement **UPDATE** στο παρασκήνιο.

Το Django παρέχει επίσης τη δυνατότητα ταυτόχρονης αποθήκευσης αλλαγών σε μια συλλογή αντικειμένων.

Παράδειγμα:

Στο παρακάτω παράδειγμα, όλα τα headlines των αντικειμένων του model **Entry** που έχουν χρονολογία δημοσίευσης το 2007 (**pub_date__year=2007**), αλλάζουν σε «**Everything is the same**».

```
Entry.objects.filter(pub_date__year=2007).update(headline='Everything
is the same')
```

3.5.2.4 Ανάκτηση Αντικειμένων

Προκειμένου να ανακτηθούν αντικείμενα από την βάση δεδομένων (για να προβληθούν στον χρήστη), απαιτείται η κατασκευή ενός **QuerySet**, μέσω του Manage**r** της model κλάσης στην οποία ανήκουν τα αντικείμενα αυτά.Όπως αναφέρθηκε παραπάνω, σε περίπτωση που δεν έχει οριστεί κάποιος Manager, ο default Manager που παρέχει το Django για κάθε model κλάση, είναι το objects.

Ένα QuerySet αποτελεί την αναπαράσταση ενός συνόλου αντικειμένων από την βάση δεδομένων. Έχει τη δυνατότητα να δεχθεί φίλτρα (filters) τα οποία περιορίζουν τα αποτελέσματα, βάσει των παραμέτρων που θα εισαχθούν σε αυτά.
Το QuerySet ουσιαστικά είναι το αντίστοιχο του SQL statement **SELECT** και το **filter** το αντίστοιχο της SQL φράσης **WHERE** ή **LIMIT**.

α) Ανάκτηση όλων των αντικειμένων

Ο απλούστερος τρόπος ανάκτησης των αντικειμένων ενός πίνακα της βάσης δεδομένων είναι η ανακτήση όλων των αντικειμένων. Προκειμένου να επιτευχθεί αυτό, χρησιμοποιείται η μέθοδος **all()** πάνω σε έναν Manager.

Παράδειγμα:

```
>>> all_entries = Entry.objects.all()
```

β) Ανάκτηση συγκεκριμένων αντικειμένων με χρήστη φίλτρων (filters):

Σε περίπτωση που χρειάζεται να ανακτηθεί ένα υποσύνολο των αντικειμένων που αναπαριστούν τις εγγραφές ενός πίνακα της βάσης δεδομένων,γίνεται χρήση των filters.
Οι δύο πιο συνήθεις τρόποι είναι οι εξής:

i) Με χρήση της μεθόδου filter(kwargs):**

Αυτή η μέθοδος επιστρέφει ένα **QuerySet** που αντιστοιχεί στις παραμέτρους που έχουν δοθεί.

Παράδειγμα:

```
Entry.objects.filter(pub_date__year=2006)
```

ή

```
Entry.objects.all().filter(pub_date__year=2006)
```

Και με τους δύο τρόπους, επιστρέφεται ένα **QuerySet** που περιέχει τα αντικείμενα της model κλάσης **Entry** που έχουν ως χρονολογία δημοσίευσης (**pub_date_year**) το 2006.

ii) Με χρήση της μεθόδου exclude(kwargs):**

Αυτή η μέθοδος επιστρέφει ένα σύνολο με αντικείμενα που δεν ικανοποιούν τις δοθείσες παραμέτρους.

Τα διάφορα φίλτρα μπορούν να συνδυαστούν ώστε να επιστραφεί το επιθυμητό QuerySet.

Παράδειγμα:

```
>>> Entry.objects.filter(
...     headline__startswith='What'
... ).exclude(
...     pub_date__gte=datetime.date.today()
... ).filter(
...     pub_date__gte=datetime(2005, 1, 30)
... )
```

Ο παραπάνω κώδικας έχει ως αποτέλεσμα την επιστροφή ενός συνόλου αντικειμένων του model **Entry** που ικανοποιούν της εξής παραμέτρους: η επικεφαλίδα τους αρχίζει με What και δεν έχουν ημερομηνία δημοσίευσης την ημερομηνία που πραγματοποιείται η εκτέλεση του κώδικα αλλά την 30/1/2005.

γ) Ανάκτηση συγκεκριμένου αντικειμένου με την εντολή get()

Η μέθοδος **filter()** επιστρέφει πάντοτε ένα QuerySet. Ακόμη και στην περίπτωση που μόνο ένα αντικείμενο ικανοποιεί τις παραμέτρους που έχουν εισαχθεί, θα επιστραφεί ένα QuerySet που θα περιέχει μόνο ένα στοιχείο.

Προκειμένου το αποτέλεσμα της εκτέλεσης ενός query να είναι ένα και μοναδικό αντικείμενο, γίνεται χρήση της μεθόδου **get()** στον Manager του επιθυμητού model.

Παράδειγμα:

```
>>>one_entry = Entry.objects.get(pk=1)
```

Ο παραπάνω κώδικας επιστρέφει το αντικείμενο του model **Entry** που έχει **primary key = 1**.

Σε περίπτωση που δεν υπάρχει καμία εγγραφή που να ικανοποιεί τις παραμέτρους, το Django επιστρέφει την εξαίρεση **DoesNotExist**. Σε περίπτωση που υπάρχουν παραπάνω από μία εγγραφές που ικανοποιούν τις παραμέτρους, το Django επιστρέφει την την εξαίρεση **MultipleObjectsReturned**.

Οι μέθοδοι **filter(), eclude(), get()** δέχονται παραπάνω από ένα arguments,δίνοντας αποτελέσματα αντίστοιχα με το SQL Statements **WHERE..AND.**

3.5.2.4 Αναζήτηση Αντικειμένων με βάση τα fields

Στις μεθόδους **filter(), exclude(), get()** τα fields που χρησιμοποιούνται κάθε φορά ως arguments, χρησιμοποιούνται με τον ίδιο τρόπο που θα τα χρησιμοποιούσε ένα SQL statement **WHERE.**

Αυτό σημαίνει ότι υπάρχει η δυνατότητα διαφορετικών τρόπων χρήσης της αναζήτησης με βάσει τα fields,όπως συμβαίνει και στην SQL.

Η βασική μορφή που έχουν αυτές οι μορφές αναζήτησης και ανάκτησης αντικειμένων είναι οι εξής:

field__lookuptype=value, όπου field του όνομα του χαρακτηριστικού και l**ookupvalue** ο τρόπος αναζήτησης.

Παράδειγμα:

```
>>> Entry.objects.filter(pub_date__lte='2006-01-01')
```

Ο παραπάνω Django κώδικας ισοδυναμεί με τον παρακάτω κώδικα σε SQL:

```
SELECT * FROM blog_entry WHERE pub_date <= '2006-01-01';
```

Παραδείγματα αναζήτησης με βάσει τα fields είναι τα **field__exact,field__iexact,field__contains, field__startswith,field__endswith κ.α.**

3.5.2.5 Πολύπλοκη Αναζήτηση Αντικειμένων με Χρήση Αντικειμένων Τύπου Q (Q – Objects)

Τα arguments σε μία μέθοδο όπως η **filter()**, μεταφράζονται ως ένα **WHERE…AND SQL** statement.

Σε περίπτωση που απαιτείται μια πολύπλοπη ανάκτηση αντικειμένων, όπως για παράδειγμα η αναζήτηση αντικειμένων που ισοδυναμεί με SQL statement **WHERE...OR**, απαιτείται η χρύση των **Q objects**.

Ένα **Q object (django.db.models.Q)** είναι ένα αντικείμενο που χρησιμοποιείται για να αναπαραστήσει ένα σύνολο από keyword arguments.

Παράδειγμα:

Το παρακάτω αντικείμενο χρησιμοποιείται για να αναπαραστήσει το αποτέλεσμα ενός **LIKE**

query:

```
from django.db.models import Q
Q(question__startswith='What')
```

Τα Q objects συνδυάζονται με χρήση των λογικών τελεστών **& (AND)** και **| (OR)**. Το αποτέλεσμα μιας λογικής πράξης σε δύο Q objects, είναι ένα νέο Q object.

Παράδειγμα:

Η παρακάτω γραμμή κώδικα έχει ως αποτέλεσμα ένα νέο Q object που είναι το αποτέλεσμα της πράξης “OR” ανάμεσα σε δύο “question__startswith” queries.

```
Q(question__startswith='Who') | Q(question__startswith='What')
```

Το παραπάνω query είναι ισοδύναμο του SQL statement :

```
WHERE question LIKE 'Who%' OR question LIKE 'What%'
```

Τα Q objects μπορούν να χρησιμοποιηθούν και σαν arguments των μεθόδων **filter(), eclude(), get().**

3.5.2.6 Σύγκριση Αντικειμένων

Προκειμένου να συγκριθούν δύο instances ενός model, χρησιμοποιείται ο κλασσικός τελεστής ισότητας της Python, δηλαδή το == (διπλό ίσον). Η διαδικασία που πραγματοποιείται είναι η σύγκριση του πρωτεύοντος κλειδιού των δύο αντικειμένων.

Παράδειγμα:

Αν το **some_entry** και το **other_entry** είναι δύο αντικείμενα του model **Entry** και **id** το πρωτεύον κλειδί του model Entry, οι δύο παρακάτω γραμμές είναι ισοδύναμες.

```
>>> some_entry == other_entry
>>> some_entry.id == other_entry.id
```

Με παρόμοιο τρόπο μπορούμε να συγκρίνουμε οποιοδήποτε attribute δύο αντικειμένων ενός model.

3.5.2.7 Διαγραφή Αντικειμένων

Ένα αντικείμενο μπορεί να διαγραφεί, με χρήση της μεθόδου **delete()**. Η μέθοδος αυτή είναι ισοδύναμη του SQL statement **DELETE** και λειτουργεί ως εξής: διαγράφει το αντικείμενο και επιστρέφει τον αριθμό των αντικειμένων που διαγράφηκαν, όπως και ένα **dictionary** με τον αριθμό των διαγραφών ανά τύπο αντικειμένου.

Παράδειγμα:

```
>>> e.delete()
(1, {'weblog.Entry': 1})
```

Με τον ίδιο τρόπο ακριβώς, μπορεί να διαγραφεί και ένα ολόκληρο QuerySet.

Παράδειγμα:

Στο παρακάτω παράδειγμα θα διαγραφούν όλα τα αντικείμενα του model κλάσης **Entry** που έχουν χρονολογία δημοσίευσης το 2005.

```
>>> Entry.objects.filter(pub_date__year=2005).delete()
(5, {'webapp.Entry': 5})
```

3.5.2.8 Raw SQL Queries

Το Django παρέχει τη δυνατότητα χρήσης raw SQL με δύο τρόπους:

α) με τη χρήση της μεθόδου raw() στον Manager του επιθυμητού model

β) με παράκαμψη του επιπέδου Model και απευθείας εκτέλεση SQL queries.

α) Manager.raw (raw_query, params=None, translations=None)

Αυτή η μέθοδος δέχεται ένα raw SQL query, το εκτελεί και επιστρέφει ένα instance της κλάσης **django.db.models.query.RawQuerySet**.Το **RawQuerySet** συμπεριφέρεται ακριβώς όπως το QuerySet.

Παράδειγμα:

Έστω το παρακάτω model:

```
class Person(models.Model):
    first_name = models.CharField(...)
    last_name = models.CharField(...)
    birth_date = models.DateField(...)
```

Το Django δίνει τη δυνατότητα εκτέλεσης SQL κώδικα ως εξής:

```
>> for p in Person.objects.raw('SELECT * FROM myapp_person'):
...     print(p)
John Smith
Jane Jones
```

β) Απευθείας Εκτέλεση SQL

Σε περίπτωση που η παράκαμψη του επιπέδου Model είναι επιθυμητή ή αναγκαία το Django δίνει τη δυνατότητα άμεσης πρόσβασης στη βάση δεοδμένων.

Η κλάση **django.db.connection** αναπαριστά τη σύνδεση με τη βάση δεδομένων. Προκειμένου να χρησιμοποιηθεί η σύνδεση με τη βάση, απαιτείταη η κλήση της μεθόδου **connection.cursor()** που επιστρέφει ένα αντικείμενο **cursor()**. Ύστερα, χρειάζεται η κλήση της μεθόδου και των μεθόδου **cursor.fetchone()** ή **cursor.fetchall()** για επιστροφή των γραμμών [44].

Παράδειγμα:

Το παρακάτω παράδειγμα περιγράφει τον ορισμό μιας συνάρτησης η οποία επιστρέφει αποτελέσματα βάσει συγκεκριμένων SQL queries:

```
from django.db import connection

def my_custom_sql(self):
    with connection.cursor() as cursor:
        cursor.execute("UPDATE bar SET foo = 1 WHERE baz = %s",
[self.baz])
        cursor.execute("SELECT foo FROM bar WHERE baz = %s",
[self.baz])
        row = cursor.fetchone()
```

Ο κώδικας μπορεί να χρησιμοποιηθεί στο command prompt ως εξής:

```
cursor.execute("SELECT foo FROM bar WHERE baz = '30%'")
cursor.execute("SELECT foo FROM bar WHERE baz = '30%%' AND id = %s",
[self.id])
```

3.6 The View Layer

3.6.1 URL

Για να υλοποιηθεί ο σχεδιασμός των URLs για μία διαδικτυακή εφαρμογή, απαιτείται η δημιουργία ενός Python module το οποίο καλείται **URLconf**. Το **URLconf** στην πραγματικότητα είναι ένα τμήμα ενός αρχείου κώδικα Python (το οποίο συνήθως έχει το όνομα **urls.py**) που αντιστοιχίζει τα URLs στα κατάλληλα views (που με τη σειρά τους είναι συναρτήσεις της Python).

Ο αλγόριθμος ο οποίος αποφασίζει ποιός κώδικας θα εκτελεστεί κάθε φορά που ένας χρήστης, πληκτρολογώντας ένα URL, στέλνει ένα αίτημα σε ένα site που έχει δημιουργηθεί με Django, περιγράφεται στα εξής βήματα:

1. Το Django εντοπίζει το βασικό **URLconf** που θα χρησιμοποιήσει.Συνήθως αυτό ορίζεται από τη ρύθμιση **ROOT_URLCONF** που βρίσκεται στο αρχείο **settings.p**y, εκτός από την περίπτωση που το εισερχόμενο αντικείμενο της κλάσης **HttpRequest** έχει ορισμένο attribute **urlconf**. Σε αυτή την περίπτωση, η τιμή του **urlconf** θα χρησιμοποιηθεί στη θέση της τιμής της **ROOT_URLCONF** ρύθμισης.

2. Το Django φορτώνει το **URLconf** και αναζητά την Python λίστα **urlpatterns** εντός του κώδικα του, η οποία είναι μία Python list από instances της κλάσης **django.conf.urls.url()**. Η κλάση **django.conf.urls.url()** δέχεται ως arguments κανονικές εκφράσεις (regular expressions), οι οποίες ουσιαστικά περιγράφουν με την μορφή string τα URLs που διαθέτει μια διαδικτυακή εφαρμογή.

3. Το Django διατρέχει κάθε στοιχείο της λίστας **urlpatterns** μέχρι να συναντήσει αυτό που ταιριάζει στο URL που έχει πληκτρολογήσει ο χρήστης.

4. Μόλις η αναζήτηση τερματιστεί επιτυχώς, το Django κάνει import και καλεί την κατάλληλη view, η οποία μπορεί να είναι είτε μία συνάρτηση Python κάποια κλάση (class – based view). Μία view, ως συνάρτηση, μπορεί να δεχθεί ως arguments:

α) Ένα αντικείμενο της κλάσης **HttpRequest** (το οποίο αναπαριστά το request του χρήστη).

β) Τις κανονικές εκφράσεις που ταιριάζουν με το URL που έχει πληκτρολογήσει ο χρήστης.

γ) Ως keyword arguments,τα **named groups** που είναι τμήμα της κανονικής έκφρασης που ικανοποιεί το αίτημα του χρήστη.

5. Αν, μετά το τέλος της διαδικασίας, καμία κανονική έκφραση δεν ταιριάζει, ή έχει προκληθεί η εκτέλεση κάποιας εξαίρεσης, τότε το Django καλεί αυτόματα την view που αντιστοιχεί στο σφάλμα που έχει προκύψει.

Παράδειγμα:

Ο παρακάτω κώδικας αποτελεί παράδειγμα ενός **URLconf**:

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^articles/2003/$', views.special_case_2003),
    url(r'^articles/([0-9]{4})/$', views.year_archive),
    url(r'^articles/([0-9]{4})/([0-9]{2})/$', views.month_archive),
    url(r'^articles/([0-9]{4})/([0-9]{2})/([0-9]+)/$',
views.article_detail),
]
```

Στο παραπάνω παράδειγμα, views είναι το όνομα του αρχείου **views.**py στο οποίο έχουν αποθηκευτεί οι συναρτήσεις views μιας διαδικτυακής εφαρμογής. Τα στοιχεία της λίστας urlpatterns,είναι instances της προαναφερθείσας κλάσης **url().** Οι κανονικές εκφράσεις που χρησιμοποιούνται ως arguments για να προσδιορίσουν τα instances της κλάσης, συντάσσονται σύμφωνα με τους κανόνες της Python.

Οι χαρακτήρες **r'** υποδεικνύουν στο Django ότι κανένας χαρακτήρας στο string της κανονικής έκφρασης που ακολουθεί, δεν πρέπει να αγνοηθεί.

Σύμφωνα με το παράδειγμα, ένα request για το URL **/articles/2005/03**, θα ταίριαζε με το τρίτο στοιχείο της λίστας. Έτσι, το Django θα καλούσε τη συνάρτηση **views.month_archive(request,'2005','03**).

Το request για το **/articles/2005/3/** δεν θα ταίριαζε με κανένα από τα URL patterns,μιας και το τρίτο στοιχείο της λίστας, με το οποίο ταυτοποιείτα, απαιτεί δύο ψηφία και όχι ένα, στο σημείο μετά το τρίτο **«/»** της κανονικής έκφρασης.

Το **/articles/2003/** θα ταίριαζε με την πρώτη κανονική έκφραση της λίστας, γιατί το Django διατρέχει την λίστα με τη σειρά, και η συγκεκριμένη κανονική έκφραση είναι το πρώτο αποτέλεσμα που ταιριάζει με το request.

3.6.1.1 Named Group

Στο παραπάνω παράδειγμα γίνεται χρήση ομάδων κανονικών εκφράσεων που δεν ανήκουν σε κάποιο **ορισμένο group (named group).** Σε όλες τις παραπάνω περιπτώσεις, τμήματα του URL χρησιμοποιούνται ως arguments στην αντίστοιχη συνάρτηση view. Το Django μπορεί να καλύψει και τις ανάγκες πιο εξειδικευμένης χρήσης, χρησιμοποιώντας **named groups κανονικών εκφράσεων**, προκειμένου να χρησιμοποιήσει κομμάτια ενός URL ως keyword arguments στην συνάρτηση view που θα καλεστεί.

Στα πλαίσια των κανόνων κατασκευής κανονικών εκφράσεων στην Python, ένα named group κανονικής έκφρασης συντάσσεται ως εξής**:**

```
(?P<name>pattern)
```

όπου το name είναι το όνομα ενός group και pattern το pattern με το οποίο συγκρίνεται το εκάστοτε request.

Παράδειγμα:

Το παρακάτω παράδειγμα περιγράφει την μορφή που θα είχε το αρχείο **URLconf**, ώστε να χρησιμοποιήσει named groups:

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^articles/2003/$', views.special_case_2003),
    url(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    url(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$',
views.month_archive),
    url(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-
9]{2})/(?P<day>[0-9]{2})/$', views.article_detail),
]
```

Ο παραπάνω κώδικας υλοποιεί ακριβώς την ίδια λειτουργία με τον προηγούμενο. Η διαφορά του έγκειται στο ότι τα τμήματα του URL χρησιμοποιούνται ως keyword arguments και όχι ως **positional arguments** (παράμετροι θέσης) στις κατάλληλες views.

Για παράδειγμα, ένα request για το URL **/articles/2005/03/** θα είχε ως αποτέλεσμα την κλήση της συνάρτησης **views.month_archive(request,year='2005',month='03')** αντί για την κλήση της συνάρτησης **views.month_archive(request,'2005','03')**.

Η χρήση named groups επιτρέπει την αποφυγή σφαλμάτων και προβλημάτων κατά την εκτέλεση του κώδικα, καθώς και την αλλαγή της σειράς των arguments κατά τη διάρκεια του ορισμού των view συναρτήσεων που θα χρησιμοποιηθούν.

3.6.1.2 Εισαγωγή Άλλων URLconfs στη Λίστα urlpatterns

Στην Python λίστα **urlpatterns** που βρίσκεται σε ένα **URLconf** module, υπάρχει η δυνατότητα να συμπεριληφθούν και άλλα **URLconf**, δημιουργώντας ενός είδους διάταξη και ιεραρχία ανάμεσα στα URLs. Αυτό επιτυγχάνεται με χρήση της μεθόδου **include()**, μέσω της οποίας, σε

μια λίστα urlpatterns μπορούν να συμπεριληφθούν patterns που ανήκουν στο ίδιο ή σε άλλο αρχείο **URLconf**.

Κατά τη διάρκεια της αναζήτησης σε μία λίστα urlpatterns, όταν το Django συναντήσει την φράση “include”, κρατάει το κομμάτι του URL που έχει ήδη ελέγξει και προωθεί το υπόλοιπο στο **URLconf** που περιέχεται ως argument στην μέθοδο **include()** για περαιτέρω επεξεργασία.

Παράδειγμα:

Έστω το παρακάτω URLconf:

```
from django.conf.urls import include, url

from apps.main import views as main_views
from credit import views as credit_views

extra_patterns = [
    url(r'^reports/$', credit_views.report),
    url(r'^reports/(?P<id>[0-9]+)/$', credit_views.report),
    url(r'^charge/$', credit_views.charge),
]

urlpatterns = [
    url(r'^$', main_views.homepage),
    url(r'^help/', include('apps.help.urls')),
    url(r'^credit/', include(extra_patterns)),
]
```

Στον παραπάνω κώδικα το URL **/credits/report** θα αντιστοιχηθεί στην view **credit_views.report()**.

Ένα URLconf που περιέχεται σε ένα άλλο URLconf, λαμβάνει ως παραμέτρους, όσες έχουν χρησιμοποιηθεί ως τέτοιες στο **URLconf** γονέα.

Παράδειγμα:

```
# In settings/urls/main.py
from django.conf.urls import include, url

urlpatterns = [
    url(r'^(?P<username>\w+)/blog/', include('foo.urls.blog')),
]
```

```
# In foo/urls/blog.py
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.blog.index),
    url(r'^archive/$', views.blog.archive),
]
```

Στο παραπάνω παράδειγμα, η τιμή της μεταβλητής variable μεταφέρεται στο **URLconf foo.urls.blog**.

3.6.1.3 Εμφωλευμένες Παράμετροι (Nested Arguments)

Οι κανονικές εκφράσεις στην Python επιτρέπουν τη χρήση εμφωλευμένων παραμέτρων (δηλαδή παράμετρο εντός παραμέτρου). Το ίδιο συμβαίνει και στον Django. Το Django θα τις λάβει υπόψη και θα τις προωθήσει στην αντίστοιχη view.

Παράδειγμα:

```
from django.conf.urls import url

urlpatterns = [

   url(r'comments/(?:page-(?P<page_number>\d+)/)?$', comments
]
```

3.6.1.4 Επιπλέον Παράμετροι (Extra Arguments)

Το Django επιτρέπει την μεταφορά επιπλέον παραμέτρων στις συναρτήσεις view, με την μορφή **Python dictionary**. Η συνάρτηση **django.conf.urls.url()** δέχεται δηλαδή, ένα προαιρετικό τρίτο argument, το οποίο έχει την μορφή dictionary οι τιμές του οποίου θα χρησιμοποιηθούν από την αντίστοιχη view ως keyword arguments.

Παράδειγμα:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^blog/(?P<year>[0-9]{4})/$',  views.year_archive,  {'foo':
'bar'}),
]
```

Βάσει του παραπάνω κώδικα, το Django θα πραγματοποιήσει κλήση της συνάρτησης **views.year_archive(request,year='2005', foo='bar'**).

Η μέθοδος **include()** δέχεται επίσης επιπλέον παραμέτρους με παρόμοιο τρόπο.

3.6.1.5 Reverse URL Resolution

Το Django διευκολύνει την υλοποίηση δυναμικού κώδικα και δίνει στον προγραμματιστή τη δυνατότητα να μην χρησιμοποιεί αναγκαστικά κάθε φορά ένα URL ολογράφως. Έτσι, η διαδικασία κατά την οποία μετά η αποδοχή ενός request οδηγεί στην κλήση συγκεκριμένης συνάρτησης view, μπορεί να γίνει και αντίστροφα. Μια συγκεκριμένη view συνάρτηση, μαζί με τα arguments που έχει μπορεί να οδηγήσει σε συγκεκριμένο URL. Η διαδικασία αυτή ονομάζεται

URL Reversing και υλοποιείται με τρεις τρόπους:

α) Σε ένα **template** (HTML κώδικας) με χρήση της template ετικέτας **{% url %}**
β) Σε οποιοδήποτε τμήμα του κώδικα Python με χρήση της συνάρτησης **reverse()**
γ) Σε κώδικα που σχετίζεται με τον χειρισμό των URLs που αντιστοιχούν σε αντικείμενα μιας model κλάσης, με χρήση της συνάρτησης **get_absolute_url()**.

Παράδειγμα:

Έστω το παρακάτω **URLconf**:

```
from django.conf.urls import url

from . import views

urlpatterns = [
    #...
    url(r'^articles/([0-9]{4})/$',  views.year_archive,  name='news-
year-archive'),
    #...
]
```

Στον παραπάνω κώδικα, η καταχώρηση στο urlpatterns έχει ένα keyword **name="news-years-archive"**.

Σε έναν κώδικα HTML, υπάρχει η δυνατότητα να προβάλλουμε το URL που αντιστοιχεί στην παραπάνω καταχώρηση ως εξής:

```
<a href="{% url 'news-year-archive' 2012 %}">2012 Archive</a>
{# Or with the year in a template context variable: #}
<ul>
{% for yearvar in year_list %}
<li><a href="{% url 'news-year-archive' yearvar %}">{{ yearvar }}
Archive</a></li>
{% endfor %}
</ul>
```

Κάνοντας δηλαδή χρήση της ετικέτας **{% url %}** μαζί με τις τιμές για τις παραμέτρους που δέχεται η κανονική έκφραση (στη συγκεκριμένη περίπτωση τον αριθμό «2002»)

Ταυτόχρονα, μπορεί να υπάρξει πρόσβαση στο ίδιο URL με τη βοήθεια της συνάρτησης **reverse()**, χρησιμοποιώντας την όπως παρακάτω:

```
from django.urls import reverse
from django.http import HttpResponseRedirect

def redirect_to_year(request):
    # ...
    year = 2006
    # ...
    return          HttpResponseRedirect(reverse('news-year-archive',
args=(year,)))
```

Στον συγκεκριμένο κώδικα ορίζεται η συνάρτηση **redirect_to_year(),** η οποία δέχεται request και ανακατευθύνει το χρήστη σε συγκεκριμένη σελίδα, επιστρέφοντας ένα αντικείμενο της κλάσης **HttpResponseRedirect** που προσδιορίζεται από το **reverse('new-year-archive,)**. Η ανακατεύθυνση, δηλαδή, οδηγεί στον χρήστη σε URL που περιγράφεται στο urlpatterns στην αρχή του παραδείγματος.

3.6.1.6 URL Pattern Names

Όπως φάνηκε στο προηγούμενο παράδειγμα, για να επιτευγχεί η διαδικασία **URL reversing**, στα στοιχεία της λίστας urlpatterns, χρειάζεται να προστεθεί ένα keyword argument που καλείται **name**.

3.6.1.7 URL Namespaces

Τα **namespaces** διασφαλίζουν την μοναδικότητα του αποτελέσματος της διαδικασίας URL Reversing, ακόμη και αν διαφορετικές εφαρμογές χρησιμοποιούν τα ίδια **URL names**. Σε περίπτωση, δηλαδή, που σε μία διαδικτυακή εφαρμογή ή δύο υποεφαρμογές μιας εφαρμογής, χρησιμοποιούνται ίδια ονόματα για διαφορετικά URL patterns, τα namespaces αποτελούν εργαλείο διαχωρισμού του ενός pattern από το άλλο.

Τα URLs, μετά από χρήση του namespace συγκεκριμενοποιούνται με χρήση του τελεστή ':'.

Έτσι, το όνομα ενός named URL που ανήκει σε ένα namespace, έχει πλέον την εξής μορφή:

'namespace_name:URL_name'.

Για παράδειγμα, το **'admin:index'** αναφέρεται σε URL με το όνομα index που ανήκει στο namespace admin.

Παράδειγμα:

Έστω το παρακάτω URLconf στο **urls.py**:

```
from django.conf.urls import include, url

urlpatterns = [
    url(r'^author-polls/',  include('polls.urls',  namespace='author-
polls'))]
```

Έστω ένα ακόμη URLconf που βρίσκεται στο **polls/urls.py**:

```
from django.conf.urls import url

from . import views

app_name = 'polls'
urlpatterns = [
    url(r'^$', views.IndexView.as_view(), name='index'),
    url(r'^(?P<pk>\d+)/$',                views.DetailView.as_view(),
name='detail'),
    ...
```

```
]
```

Βάσει του παραπάνω παραδείγματος, μέσω του HTML κώδικα ****, γίνεται κλήση της συνάρτησης **view.IndexView.as_view()**.

Έχοντας περιγράψει την διαδικασία αντιστοίχισης των URLs σε συγκεκριμένα views, στην επόμενη παράγραφο θα περιγραφεί αναλυτικότερα ο τρόπος υλοποίησης ενός view [45].

3.6.2 Views

Μία **συνάρτηση view** είναι μία συνάρτηση Python, η οποία δέχεται ως argument ένα **διαδικτυακό αίτημα (Web request)** και επιστρέφει μία **απάντηση (Web response)**,ανακατεύθυνση σε άλλη σελίδα, σφάλμα 404, κώδικα XML, μια εικόνα κ.α. Η ίδια η συνάρτηση περιέχει τον κώδικα ο οποίος υλοποιεί τον τρόπο με τον οποίο θα επιστραφεί το αποτέλεσμα. Το τμήμα του κώδικα που σχετίζεται με τις συναρτήσεις views, είθιστε να αποθηκεύεται σε αρχεία με το όνομα **views.py**.

Παράδειγμα:

Η παρακάτω συνάρτηση view επιστρέφει την ημερομηνία και την ώρα, με την μορφή HTML:

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

Όπως φαίνεται στο παραπάνω παράδειγμα, η view που υλοποιήσαμε είναι η **current_datetime()**. Η **current_datetime()** δέχεται ως argument ένα Web request, με την βοήθεια της βιβλιοθήκης της Python **datetime** υπολογίζει την παρούσα ημερομηνία και ώρα, και επιστρέφει ένα αντικείμενο **HttpResponse** της κλάσης **django.http.HttpResponse** που περιέχει κώδικα HTML (ο οποίος έχει εισχωρηθεί με την μορφή string στην μεταβλητή html).

3.6.1.1 Επιστροφή Σφαλμάτων

Στο Django η επιστροφή σφαλμάτων, όταν αυτά προκύπτουν κατά την εκτέλεση ενός view υλοποιείται με την βοήθεια υποκλάσεων της κλάσης **HttpRespose**. Υπάρχουν υποκλάσεις που αντιστοιχούν σε διάφορους **κωδικούς HTTP status**, όπως ο **404** που υποδηλώνει ότι την αποτυχία εύρεσης της σελίδας που ζητήθηκε από τον χρήστη.

Παράδειγμα:

```
from django.http import HttpResponse, HttpResponseNotFound

def my_view(request):
    if foo:
        return HttpResponseNotFound('<h1>Page not found</h1>')
    else:
```

```
        return HttpResponse('<h1>Page was found</h1>')
```

Στο παραπάνω παράδειγμα σε περίπτωση που δεν ισχύει η συνθήκη «foo» η συνάρτηση **my_view()** επιστρέφει ένα instance της κλάσης **HttpResponseNotFound** [46].

3.6.1.2 Class – Based Views

Η χρήση συναρτήσεων Python είναι ένας συνήθης τρόπος υλοποίησης views, αλλά δεν είναι ο μοναδικός. Τα **Class – Based Views** είναι ο τρόπος υλοποίησης των Views με κλάσεις αντί για συναρτήσεις Python.

Ένα class – based view δίνει τη δυνατότητα διαφορετικών **responses** σε διαφορετικές **HTTP methods** μέσω διαφορετικών μεθόδων του instance μιας κλάσης.

Παράδειγμα:

Ο κώδικας μιας συνάρτησης view για χειρισμό της HTTP μεθόδου **GET** έχει ως εξής:

```
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # <view logic>
        return HttpResponse('result')
```

Με χρήση class – based view ο παραπάνω κώδικας θα μετατρεπόταν σε:

```
from django.http import HttpResponse
from django.views import View

class MyView(View):
    def get(self, request):
        # <view logic>
    return HttpResponse('result')
```

Το Django παρέχει πολλές ενσωματωμένες κλάσεις τύπου class – based views (generic views) , όπως το **TemplateView, RedirectView** και **TemplateView**. Όλες είναι υποκλάσεις της βασικής κλάσης **View**.

Ο πιο απλός τρόπος χρήσης των generic views είναι η δημιουργία τους στο αρχείο URLconf, όπως φαίνεται παρακάτω:

Παράδειγμα:

```
from django.conf.urls import url
from django.views.generic import TemplateView

urlpatterns = [
    url(r'^about/$',
TemplateView.as_view(template_name="about.html")),
]
```

Η μέθοδος **as_view()** χρησιμοποιείται για να επανακαθορίσει τα attributes κάθε class – based view. Στο συγκεκριμένο παράδειγμα,δημιουργείται ένα instance του TemplateView με συγκεκριμένο **template_name** [47].

3.6.3 Αρχεία

Η διαχείριση αρχείων από το Django είναι απλή.Ο τρόπος περιγράφεται παρακάτω.

Παράδειγμα:

Έστω η φόρμα (αντικείμενο της κλάσης **forms.Form**, που θα περιγραφεί αναλυτικά αργότερα) που ορίζεται από τον παρακάτω κώδικα:

```
from django import forms

class UploadFileForm(forms.Form):
    title = forms.CharField(max_length=50)
    file = forms.FileField()
```

Μία view που χειρίζεται ένα instance της κλάσης **forms.Form** θα δεχθεί τα δεδομένα του αρχείου στο αντικείμενο **request.FILES**, το οποίο είναι ένα dictionary που περιέχει key για κάθε υποκλάση των κλάσεων **FileField, ImageField κ.α.** που περιέχεται στην form.

Το **request.FILES** θα περιέχει δεδομένα μόνο σε περίπτωση που η request method είναι **POST** και στον κώδικα HTML η ετικέτα **<form>** που αντιστοιχεί στο αντίστοιχο instance της κλάσης **UploadFileForm** περιέχει το attribute **enctype=”multipart/form-data”**.

Συνήθως, η μόνη διαδικασία που απαιτείται είναι η μεταφορά των δεδομένων ενός αρχείου από το request στην form.

Παράδειγμα:

Ο παρακάτω κώδικας, περιέχει τη συνάρτηση **upload_file()** που χειρίζεται το ανέβασμα αρχείων

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import UploadFileForm

# Imaginary function to handle an uploaded file

from somewhere import handle_uploaded_file

def upload_file(request):
    if request.method == 'POST':
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            handle_uploaded_file(request.FILES['file'])
            return HttpResponseRedirect('/success/url/')
    else:
        form = UploadFileForm()
    return render(request, 'upload.html', {'form': form})
```

Το ανέβασμα ενός αρχείου επίσης μπορεί να επιτευχθεί με τη δημιουργία μιας κλάσης Model η οποία περιέχει ένα **field** που είναι υποκλάση του **FileField()**. Σε αυτή την περίπτωση το αρχείο θα αποθηκευτεί στην τοποθεσία που αναφέρεται στο **upload_to** argument του FileField, κατά την κλήση της μεθόδου **form.save().**

Παράδειγμα:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import ModelFormWithFileField

def upload_file(request):
    if request.method == 'POST':
        form = ModelFormWithFileField(request.POST, request.FILES)
        if form.is_valid():
            # file is saved
            form.save()
            return HttpResponseRedirect('/success/url/')
    else:
        form = ModelFormWithFileField()
```

Σε περίπτωση που ένα αρχείο είναι μικρότερο από 2,5 megabytes το Django θα κρατήσει το περιεχόμενο αποθηκευμένο στην προσωρινή μνήμη του υπολογιστή. Αν όμως ένα αρχείο είναι πολύ μεγάλο, τότε το Django θα αποθηκεύεσει το περιεχόμενό του σε προσωρινό αρχείο στην τοποθεσία του **temporary directory** του υπολογιστή του χρήστη.

3.6.4 Decorators

Το Django παρέχει «decorators», τα οποία μπορούν να προστεθούν σε συναρτήσεις views ώστε να υλοποιήσουν διάφορα HTTP features.Για να χρησιμοποιηθεί ένα decorator, χρειάζεται να προστεθεί η γραμμή κώδικα **@decorator_name()** πάνω από το επιθυμητό view.

Παράδειγμα:

Τα decorators που είναι instances των κλάσεων **django.views.decorators.http** τα οποία περιορίζουν την πρόσβαση στα views μιας διαδικτυακής εφαρμογής βάσει της μεθόδου του request **(POST** ή **GET**).

```
from django.views.decorators.http import require http methods

@require_http_methods(["GET", "POST"])
def my_view(request):
    # I can assume now that only GET or POST requests make it this
far
    # ...
    Pass
```

Στο παραπάνω παράδειγμα, πρόσβαση στο **my_view** μπορεί να υπάρξει μόνο μέσω ενός request που είναι **GET** ή **POST**.

Decorators μπορούν να χρησιμοποιηθούν ώστε να ελεγχθεί η συμπεριφορά της μνήμης

cache σε συγκεκριμένα views (instances της κλάσης **django.views.decorators.http**), ώστε να ελεγχθεί το caching ανάλογα με τις κεφαλίδες (headers) των requests (instances της κλάσης **django.views.decorators.vary**) [49].

3.6.5 Shortcuts

Το Django, με το πακέτο **django.shortcuts** παρέχει πολλές βοηθητικές συναρτήσεις, οι οποίες μέσω συνδυασμού κλάσεων και συναρτήσεων επιτρέπουν την υλοποίηση λειτουργών με πολύ πιο σύντομο τρόπο.

Ακολουθούν τρία βασικά παραδείγματα συναρτήσεων shortcut.

α) render(request, template_name, context=None, content_type=None, status=None, using=None):

Η συγκεκριμένη συνάρτηση δέχεται ένα request, και συνδυάζει το όνομα ενός template μαζί με ένα Python dictionary με το όνομα context (στο οποίο μπορεί να καθοριστεί περιεχόμενο που θα μεταφερθεί στο template) και επιστρέφει ένα αντικείμενο **HttpRespons**e το οποίο περιέχει τις τιμές του dictionary με την μορφή του template το όνομα του οποίου χρησιμοποιείται.

Παράδειγμα:

```
from django.shortcuts import render

def my_view(request):
    # View code here...
    return render(request, 'myapp/index.html', {
        'foo': 'bar',
    }, content_type='application/xhtml+xml')

from django.http import HttpResponse
from django.template import loader

def my_view(request):
    # View code here...
    t = loader.get_template('myapp/index.html')
    c = {'foo': 'bar'}
    return             HttpResponse(t.render(c,             request),
content_type='application/xhtml+xml')
```

β) redirect(to, permanent=False, *args, **kwargs)

Η συγκεκριμένη συνάρτηση επιστρέφει ένα αντικείμενο **HttpResponseRedirect** πραγματοποιώντας μία ανακατεύθυνση στο URL που αντιστοιχεί στα arguments της συνάρτησης.

Παράδειγμα:

Η συνάρτηση **redirect()** μπορεί να χρησιμοποιηθεί με τους παρακάτω τρόπους:

i) Με χρήση ενός αντικειμένου ως argument (που θα προκαλέσει και την κλήση της αντίστοιχης μεθόδου **get_absolute_url()** για την κλάση του αντικειμένου ώστε να βρεθεί το URL)

```
from django.shortcuts import redirect

def my_view(request):
    ...
    object = MyModel.objects.get(...)
    return redirect(object)
```

ii) Με χρήση του ονόματος μιας view συνάρτησης και προαιρετικά arguments θέσης ή keyword arguments (με αυτόματη κλήση της συνάρτησης **reverse()** προκειμένου να βρεθεί το κατάλληλο URL για την ανακατεύθυνση)

```
def my_view(request):
    ...
    return redirect('some-view-name', foo='bar')
```

iii) Με απευθείας χρήση του URL της ανακατεύθυνσης

```
def my_view(request):
    ...
    return redirect('/some/url/')
```

γ) get_object_or_404(klass, *args, **kwargs)

Η συγκεκριμένη συνάρτηση καλεί την μέθοδο **get()** στον αντίστοιχο **Manager** του model. Σε περίπτωση που αποτύχει προκαλεί την εξαίρεση **Http404** αντί για την εξαίρεση **DoesNotExist** της κλάσης Model [50].

Παράδειγμα:

Ο παρακάτω κώδικας ανακτά το αντικείμενο της κλάσης **MyModel** που έχει **primary key = 1**:

```
from django.shortcuts import get_object_or_404

def my_view(request):
    my_object = get_object_or_404(MyModel, pk=1)
```

Ο παραπάνω κώδικας είναι ισοδύναμος με τον παρακάτω:

```
from django.http import Http404

def my_view(request):
    try:
        my_object = MyModel.objects.get(pk=1)
    except MyModel.DoesNotExist:
        raise Http404("No MyModel matches the given query.")
```

3.6.6 Middleware

Το **Middleware** είναι ένα τμήμα του framework (που ουσιαστικά είναι και το ίδιο framework)που παρεμβάλλεται στη διαδικασία request/response. Χρησιμοποιείται ως ένα ελαφρύ πρόσθετο που διαμορφώνει κατάλληλα την είσοδο ή την έξοδο δεδομένων και πληροφοριών στο Django.

Κάθε middleware χρησιμοποιείται για τις ανάγκες μιας συγκεκριμένης λειτουργίας. Χαρακτηριστικό παράδειγμα είναι το **AuthenticationMiddleware**, που χρησιμοποιείται για τις ανάγκες της αυθεντικοποίησης χρήστη, το οποίο θα περιγραφεί αναλυτικότερα σε επόμενο κεφάλαιο.

Τα middleware αντιμετωπίζονται από το Django ως επιπλέον «στρώσεις» ανάμεσα στην είσοδο και την έξοδο δεδομένων, και διατρέχονται top – down κατά τη διαδικασία request/response. Ένας προγραμματιστής έχει τη δυνατότητα δημιουργίας των δικών του middleware, προσθέτοντας τον κατάλληλο κώδικα σε οποιοδήποτε αρχείο .py.

Παράδειγμα:

Ένα middleware μπορεί να δημιουργηθεί με την μορφή συνάρτησης:

```
def simple_middleware(get_response):
    # One-time configuration and initialization.

    def middleware(request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.

        response = get_response(request)

        # Code to be executed for each request/response after
        # the view is called.

        return response

    return middleware
```

Επίσης, ένα middleware, μπορεί να έχει την μορφή αντικειμένου:

```
class SimpleMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.

    def __call__(self, request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.

        response = self.get_response(request)

        # Code to be executed for each request/response after
        # the view is called.

        return response
```

Στη θέση της φράσης «**get_response**» στο παραπάνω παράδειγμα μπορεί να είναι μια view ή το επόμενο middleware στη διαδικασία request/response.

Προκειμένου να ενεργοποιηθεί ένα middleware πρέπει να προστεθεί το όνομά του στο αρχείο ρυθμίσεων της διαδικτυακής εφαρμογής [51].

Παράδειγμα:

Ο παρακάτω κώδικας, που βρίσκεται στο αρχείο ρυθμίσεων μιας διαδικτυακής εφαρμογής δηλώνει λίστα των middleware που χρησιμοποιούνται στην εφαρμογή αυτή.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

3.7 The Template Layer

3.7.1 Templates

Το Django, όπως και τα υπόλοιπα web frameworks, παρέχει τρόπο για δημιουργία HTML κώδικα με δυναμικό τρόπο, που χρησιμοποιείται για την προβολή των επιθυμητών δεδομένων στον χρήστη. Αυτό επιτυγχάνεται μέσω των **templates.**

Ένα Django template ουσιαστικά είναι ένα αρχείο, το οποίο περιέχει στατικό HTML (ή XML κ.α.) κώδικα καθώς και εξειδικευμένο κώδικα για την προβολή του δυναμικού περιεχομένου (γλώσσα template). Ένα template ουσιαστικά επιστρέφεται μαζί με τα επιθυμητά δεδομένα ως αποτέλεσμα ενός request, μέσω ενός view.

Το Django παρέχει τη δικιά του template γλώσσα, που μπορεί να υποστηριχθεί από δύο διαφορετικές template engiens (τη Django Template Language και την Jinja2).

Η template engine που χρησιμοποιείται κάθε φορά, καθορίζεται στη ρύθμιση TEMPLATE του αρχείου ρυθμίσεων της διαδικτυακής εφαρμογής, όπως φαίνεται παρακάτω [52]:

Παράδειγμα:

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
```

```
        'OPTIONS': {
            # ... some options here ...
        },
    },
]
```

3.7.2 Γλώσσα Template

Παρακάτω μπορούμε να δούμε την μορφή ενός αρχείου template. Ο παρακάτω κώδικας μπορεί να αποτελεί αποτελεί τμήμα ενός αρχείου **.html, .xml** κ.α.

Παράδειγμα:

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

3.7.2.1 Μεταβλητές

Μία μεταβλητή στην γλώσσα template έχει την εξής μορφή: **{{variable}}.** Όταν η template engine συναντήσει μια μεταβλητή, την αντικαθιστά με την τιμή της. Η τιμή σε μια μεταβλητή της γλώσσας template, εκχωρείται από ένα view μέσω των context dictionary.

Παράδειγμα:

Έστω η παρακάτω συνάρτηση **my_view()** που ορίζεται στο αρχείο **views.py**:

```
from Django.shortcuts import import render

def my_view(request):
 variable=”This is a string”
 context={“rendered_variable”:variable}
 return render(request,”template.html”, context)
```

Έστω ο παρακάτω κώδικας στο **template.html**:

```
<html>
<body>
<h1>{{rendered_variable}}</h2>
</body>
</html>
```

Στο παραπάνω παράδειγμα η κλήση της **my_view()** θα επιστρέψει στον χρήστη τη φράση **“This is a string”.**

3.7.2.2 Attributes & Methods

Αν μια μεταβλητή αναπαριστά ένα αντικείμενο, τότε υπάρχει η δυνατότητα πρόσβασης σε attributes ή μεθόδους του αντικειμένου με την χρήση του τελεστή **«.»**.

Παράδειγμα:

Παρακάτω ορίζεται ο τρόπος γραφής ενός attribute και μιας μεθόδου μιας variable στην οποία έχει εκχωρηθεί ένα αντικείμενο μέσω context και συνάρτησης view.

```
{{variable.attribute}}

{% variable.method %}
```

3.7.2.3 Filters (Φίλτρα)

Τα **filters** στην γλώσσα template έχουν την εξής μορφή: **{{variable_name|filter_name}}** και χρησιμοποιούνται για παραμετροποίηση της προβολής των τιμών των μεταβλητών. Για παράδειγμα το **{{name|lower}}** μετατρέπει την string τιμή του **name** σε πεζά.

Παραδείγματα filters είναι το **{{text | linebreaks}}** που τοποθετεί παραγράφους σε κείμενο ή το **{{bio | truncatewords : 30}}** που προβάλει μόνο τις πρώτες 30 λέξεις της τιμής της μεταβλητής.

3.7.2.4 Tags (Ετικέτες)

Οι ετικέτες στην γλώσσα template έχουν την εξής μορφή**: {% tag %}**. Οι λειτουργίες τους μπορεί να είναι η δημιουργεία κειμένου ή η διαχείριση ροής δεδομένων μέσω της εκτέλεσης επαναλήψεων ή λογικών πράξεων.

Κάποιες ετικέτες απαιτούν ετικέτες που να οριοθετούν τον κώδικα που εκτελείται εντός τους (πχ. **{% tag %}…tag content…{% endtag %}).**
Παραδείγματα ετικετών είναι λέξεις που χρησιμοποιούνται στην Python όπως **{% if %} {% elif %} {% endif %}, {% for %}, {% endfor %} κ.α.**

Παράδειγμα:

```
{% if athlete_list|length > 1 %}
   Team: {% for athlete in athlete_list %} ... {% endfor %}
{% else %}
   Athlete: {{ athlete_list.0.n0ame }}
{% endif %}
```

3.7.3 Template Inheritance (Κληρονομικότητα των Templates)

Το πιο σημαντικό στοιχείο της template engine του Django είναι η δυνατότητα **κληρονομικότητας των templates**. Η κληρονομικότητα επιτρέπει σε έναν προγραμματιστή τη δημιουργία ενός βασικού template – σκελετού το οποίο μπορεί να επαναχρησιμοποιηθεί από άλλα templates. Αυτό το template – γονέας ονομάζεται base template και το αρχείο στο οποίο τοποθετείται συνήθως καλείται **base.html**.

Παράδειγμα:

Έστω το παρακάτω **base.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}My amazing site{% endblock %}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

Στο παραπάνω παράδειγμα, οι ετικέτες **{% block %}** ορίζουν τμήματα τα οποία μπορεί να συμπληρωθούν με διαφορετικό τρόπο από κάθε template που θα είναι παιδί του **base.html**. Κάθε block προσδιορίζεται με μοναδικό τρόπο από το όνομα που το ακολουθεί.
Ο παρακάτω κώδικας, που βρίσκεται εντός ενός άλλου template, διευκρινίζει πώς ακριβώς επιτυγχάνεται αυτό:

```
{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

Το tag **{% extends %}** υποδεικνύει στην template engine ότι αυτό το template κάνει extend ένα άλλο template. Έτσι, **το base.html** γίνεται γονέας και το παραπάνω template «παιδί».

Η κληρονομικότητα μπορεί να λειτουργήσει ακριβώς αντίστροφα με χρήση του tag **{% include %}**.

Σε περίπτωση που απαιτείται πρόσβαση στο περιεχόμενο ενός block του template γονέα, μπορεί να επιτευχθεί μέσω της χρήσης της ετικέτας **{{ block.super }}** [53].

3.7.4 Δημιουργία Template Tags & Filters

Το Django δίνει τη δυνατότητα στους προγραμματιστές να δημιουργήσουν τα δικά τους **tags** και **filters**. Για να γίνει αυτό, απαιτείται η δημιουργία ενός Python module σε έναν φάκελο με όνομα **templatetags** που θα βρίσκεται εντός του κύριου φάκελου της εφαρμογής. Ένα template filter έχει την μορφή μιας συνάρτησης που παίρνει ένα ή δύο arguments.

Παράδειγμα:

```
def cut(value, arg):
    """Removes all values of arg from the given string"""
    return value.replace(arg, '')
```

Στο παραπάνω παράδειγμα, το **value** αναπαριστά την τιμή της μεταβλητής στην οποία θα εφαρμοστεί το φίλτρο, ενώ το **arg** το argument που θα δεχθεί. Ο κώδικας ορίζει ένα φίλτρο το οποίο αφαιρεί όλες τους χαρακτήρες που έχουν καταχωρηθεί στο arg από το string.

Όπως αναφέρθηκε, το δεύτερο argument (arg) είναι προαιρετικό. Στο παρακάτω παράδειγμα ορίζεται το φίλτρο lower που δεν κάνει χρήση δεύτερου argument:

Παράδειγμα :

```
def lower(value): # Only one argument.
    """Converts a string into all lowercase"""
    return value.lower()
```

Ο παραπάνω κώδικας μετατρέπει όλα τα γράμματα του string που προηγείται του **| lower** σε πεζά.

Αφού έχουμε οκοκληρώσει τον κώδικα που υλοποιεί το επιθυμητό φίλτρο, πρέπει να το δηλώσουμε ως τέτοιο.

Προκειμένου να γίνει αυτό, συμπληρώνουμε τον παραπάνω κώδικα στο αρχείο που τον

έχουμε αποθηκεύσει ως εξής:

```
from django import template
from django import template

register = template.Library()

@register.filter(name='cut')
def cut(value, arg):
    return value.replace(arg, '')

@register.filter
def lower(value):
    return value.lower()
```

Αν ένα φίλτρο έχει αποθηκευτεί σε ένα module με τίτλο **my_filter.py**, τότε προκειμένου να χρησιμοποιηθεί σε κάποιο template χεριάζεται να προστεθεί η φράση **{% load my_filter %}. Εφόσον έχει συμβεί αυτό, τότε γράφοντας, για παράδειγμα, {{variable|cut:"0"}}**, όλοι οι χαρακτήρες **«0»** που βρίσκονται στο string που έχει καταχωρηθεί στην μεταβλητή variable αφαιρούνται. Αντίστοιχη είναι και η λειτουργία του φίλτρου **{{ variable | lower}** ή όποιου φίλτρου υλοποιήσει κάποιος προγραμματιστής.

Με παρόμοιο τρόπο μπορούν να δημιουργηθούν και να χρησιμοποιηθούν σε επιθυμητό template,διάφορα template tags.Ο πιο κοινός τρόπος δημιουργίας απλών ετικετών είναι μέσω χρήσης συναρτήσεων shortcuts που προσφέρει το Django.

Παράδειγμα:

Ο παρακάτω κώδικας δημιουργεί ένα tag με όνομα **current_time**, μέσω της χρήσης της συνάρτησης του Django **simple_tag**. Το tag που δημιουργείται παρακάτω δέχεται ένα string που περιγράφει την ημερομηνία και ώρα και το επιστρέφει με μορφή ημερομηνίας [54].

```
import datetime
from django import template

register = template.Library()

@register.simple_tag
def current_time(format_string):
    return datetime.datetime.now().strftime(format_string)
```

3.8 Forms (Φόρμες)

Συνήθως, με τον όρο **form** ορίζεται ένα σύνολο **HTML στοιχείων** (**π.χ. <input>,<button**>) που περικλείονται στις ετικέτες **<form>...</form>**. Οι φόρμες (forms) χρησιμοποιούνται για εισαγωγή στοιχείων από πλευράς του χρήστη και ύστερα κατάλληλο χειρισμό τους από μια διαδικτυακή εφαρμογή μέσω των **HTTP μεθόδων GET** και **POST**.

Το Django παρέχει τρόπους που διευκολύνουν την προετοιμασία των κατάλληλων δεδομένων για προβολή στον χρήστη, την δημιουργία των φορμών σε HTML, τον έλεγχο των δεδομένων εισάγονται μέσω φορμών από τον χρήστη και την προώθησή τους στο server.

Ακολουθώντας τη φιλοσοφία που το διαπνέει, το Django διαχωρίζει το σύνολο των λειτουργιών που σχετίζονται με τις φόρμες σε τρία επίπεδα: το **model, view, template**. Το πρώτο επίπεδο διαχειρίζεται τα δεδομένα και την επικοινωνία με την βάση, το δεύτερο τη ροή δεδομένων και το ποιά δεδομένα θα παρουσιαστούν/αποθηκευτούν ενώ το τρίτο, το πώς θα παρουσιαστούν αυτά.

3.8.1 Form Model

Στο Django, υπάρχουν δύο τρόποι δημιουργίας μιας κλάσης που σχετίζεται με τη λειτουργία μιας φόρμας. Παρακάτω περιγράφονται και οι δύο.

α) Η κλάση Form

Ο πρώτος τρόπος είναι η εκ νέου δημιουργία μιας κλάσης.
Η δημιουργία μιας φόρμας βασίζεται στην προεγκατεστιμένη κλάση **django.forms.Form** του Django. Για τη δημιουργία απαιτείται η δημιουργία κώδικα σε αρχείο με ένα όνομα όπως **forms.py**:

Παράδειγμα:

```
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

Η κλάση **NameForm** ορίζει μια φόρμα η οποία έχει ως μοναδικό attribute το **your_name**. Με το keyword argument **label** ορίζεται η HTML ετικέτα που θα εμφανίζεται στο πεδίο εισαγωγής δεδομένων που αντιστοιχεί **your_name**, όταν η κλάση **Form** παρουσιαστεί σε μορφή HTML. Το keyword argument **max_length** ορίζει τον μέγιστο αριθμό χαρακτήρων που επιτρέπεται να εισάγει ένας χρήστης στο αντίστοιχο πεδίο.

β) Η κλάση ModelForm

Ο δεύτερος τρόπος είναι χρήσιμος σε περίπτωση που απαιτείται η δημιουργία μιας φόρμας που χρησιμοποιείται για εισαγωγή δεδομένωνπου σχετίζονται με ήδη δημιουργημένα models της διαδικτυακής εφαρμογής. Σε αυτη την περίπωση, η δημιουργία μιας φόρμας βασίζεται στην κλάση **django.forms.ModelForm**.

Παράδειγμα:

Ο παρακάτω κώδικας περιγράφει τη δημιουργία μιας φόρμας που χρησιμοποιείται για την εισαγωγή άρθρων σε ένα διαδικτυακό Blog. Όπως και στην προηγούμενη περίπτωση, ο κώδικας τοποθετείται σε αρχείο **forms.py**.

```
from django.forms import ModelForm
from myapp.models import Article

class ArticleForm(ModelForm):
   class Meta:
       model = Article
       fields = ['pub_date', 'headline', 'content', 'reporter']
```

Η κλάση **ArticleForm** βασίζεται σε ήδη δημιουργημένη κλάση model με το ονομα **Article**. Η λίστα **fields** ορίζει ποιά attributes τη κλάσης Article θα μπορούν να εισαχθούν ή να αλλαχθούν μέσω πεδίων εισαγωγής δεδομένων όταν η φόρμα παρουσιαστεί σε HTML μορφή.

Σε αυτό το σημείο, αξίζει να σημειωθεί ότι κάθε instance των κλάσεων **django.forms.Form** και **django.forms.ModelForm** δέχεται την μέθοδο **is_valid()** η οποία πιστοποιεί όλα τα δεδομένα που έχουν εισαχθεί στα πεδία του. Κατά την κλήση αυτής της μεθόδου, επιστρέφεται η μεταβλητή True σε περίπτωση που όλα τα δεδομένα πληρούν τους κανόνες που έχουν τεθεί από τα validators που έχουν οριστεί στο αρχείο ρυθμίσεων της διαδικτυακής εφαρμογής.

3.8.2 Form View

Τα δεδομένα που εισάγονται μέσω μίας φόρμας σε μια διαδικτυακή εφαρμογή Django, επεξεργάζονται μέσω ενός view. Ο τρόπος υλοποίησης ενός τέτοιου view περιγράφεται στο παράδειγμα που ακολουθεί.

Παράδειγμα:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

from .forms import NameForm

def get_name(request):
    if request.method == 'POST':
        form = NameForm(request.POST)
         if form.is_valid():
            # process the data in form.cleaned_data as required
            # redirect in a URL
            return HttpResponseRedirect('/thanks/')
    # if a GET (or any other method) we'll create a blank form
    else:
        form = NameForm()
return render(request, 'name.html', {'form': form})
```

Ο παραπάνω κώδικας περιγράφει τον χειρισμό ενός instance της κλάσσης **NameForm** που δημιουργείται σε περίπτωση που γίνει request για το URL το οποίο αντιστοιχει στην view **get_name()**. Με χρήση της μεθόδου **is_valid()** ελέγχονται τα δεδομένα που εισάγονται από το χρήστη, και σε περίπτωση επιτυχίας, πραγματοποιείται ανακατεύθυνση στο επιθυμητό URL.

3.8.3 Form Template

Ο τρόπος παρουσίασης μιας φόρμας με HTML κώδικα είναι εξαιρετικά απλός και εύχρηστος.

Παράδειγμα:

Στο παρακάτω παράδειγμα αναφέρεται ο κώδικας που χρειάζεται να εισαχθεί στο επιθυμητό template (συγκεκριμένα το **name.html** βάσει του προηγούμενου παραδείγματος) προκειμένου η φόρμα να μπορεί να προβληθεί στον χρήστη.

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit" />
</form>
```

Στην ετικέτα **<form>** ορίζουμε τα attributes **action** που προσδιορίζει το URL στο οποίο ορίζεται η ενέργεια που πρέπει να ακολουθηθεί κατά την υποβολή μιας φόρμας και το **method** στο οποίο ορίζεται η **HTTP μέθοδος** που συνοδεύει την υποβολή αυτή [55].

3.9 Django Admin

Ένα ιδιαιτέρως εύχρηστο και σημαντικό component που παρέχει το Django είναι η ύπαρξη σελίδας **admin** η οποία χρησιμοποιείται για τη διαχείριση μιας διαδικτυακής εφαρμογής. Πρόκειται, ουσιαστικά, για μία εφαρμογή Django (της οποίας οι λειτουργίες υλοποιούνται στα επίπεδα **model,view, template**) . Με την έναρξη της ανάπτυξης μιας διαδικτυακής εφαρμογής σε Django, δημιουργείται αυτόματα ένα instance της εφαρμογής **admin** που αφορά στην συγκεκριμένη εφαρμογή.

3.9.1 Η Κλάση ModelAdmin

Για τις ανάγκες της διαχείρισης των δεδομένων μιας διαδικτυακής εφαρμογής, το Django παρέχει την κλάση **ModelAdmin**. Η συγκεκριμένη κλάση εξυπηρετεί την αναπαράσταση ενός model στο interface της διαδικτυακής εφαρμογής. Προκειμένου να υπάρχει η δυνατότητα προβολής του επιθυμητού model στο interface του admin, χρειάζεται να προσθέσουμε την κατάλληλη υποκλάση ModelAdmin που σχετίζεται με το συγκεκριμένο model, σε αρχείο που είθιστε να ονομάζεται **admin.py**

Παράδειγμα:

Ο παρακάτω κώδικας περιγράφει την δημιουργία μιας υποκλάσης του **ModelAdmin**, η οποία είναι υπεύθυνση για τη διαχείριση των δεδομένων των αντικειμένων της model κλάσης Author.

```
from django.contrib import admin
from myproject.myapp.models import Author

class AuthorAdmin(admin.ModelAdmin):
    pass
admin.site.register(Author, AuthorAdmin)
```

Με την γραμμή **admin.site.register**,συνδέεται η model κλάση με την αντίσχοιχη κλάση που είναι υπεύθυνη για την διαχείριση δεδομένων, και διασφαλίζεται ότι η κλάση **AuthorAdmin** θα προβληθεί στο interface της εφαρμογής admin.

Κάθε υποκλάση της κλάσης **ModelAdmin** έχει τη δυνατότητα να δεχθεί διάφορα attributes τα οποία καθορίζουν τον τρόπο παρουσίασης δεδομένων στο **admin site**.
Για παράδειγμα, το attribute **list_display** καθορίζει ποιά attributes του model που διαχειρίζεται η αντίστοιχη κλάση θα προβληθούν, το **fields** ποιά attributes του model θα μπορούν να

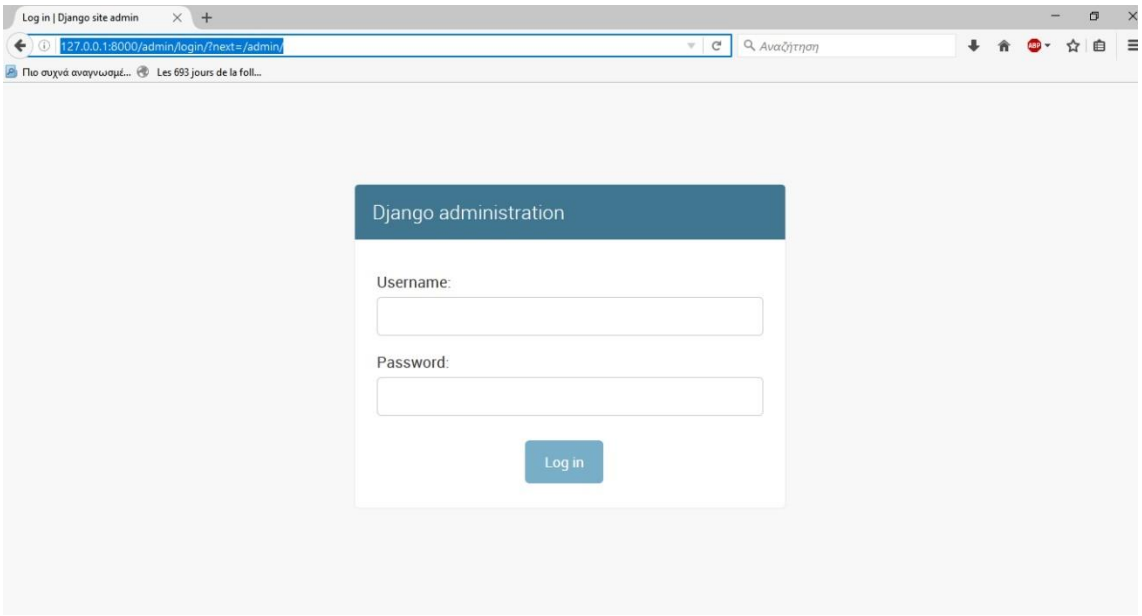
αλλάξουν μέσω φόρμας που υπάρχει στο interface του admin, το **actions** τη δυνατότητα προσθήκης άλλων ενεργειών (που υλοποιούνται μέσω συναρτήσεων του Django) κ.α [55].

3.9.2 Διαχείριση Δεδομένων Μέσω της Σελίδας admin

Η πρόβαση στη σελίδα admin γίνεται μέσω του URL **my_domain/admin/** όπου **my_domain** είναι το URL του domain της διαδικτυακής εφαρμογής.

Παράδειγμα:

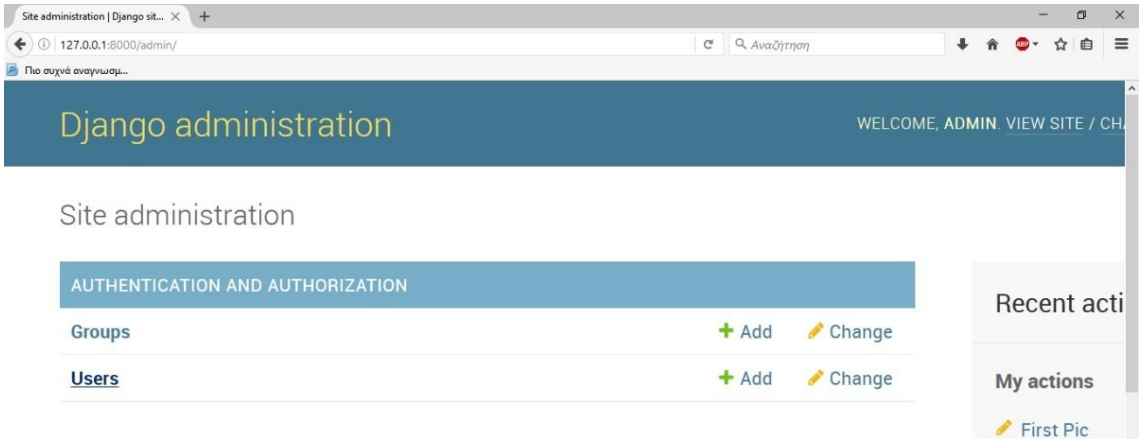
Πληκτρολογώντας το κατάλληλο URL, εμφανίζεται η οθόνη σύνδεσης χρήστη.



Εικόνα 8: Οθόνη σύνδεσης χρήστη στη σελίδα admin του Django

Εφόσον έχει δημιουργηθεί ένας χρήστης με ρόλο διαχειριστή (**superuser**), πληκτρολογώντας τα κατάλληλα στοιχεία η σύνδεση επιτυγχάνεται. Στην σελίδα που προβάλλεται, φαίνονται όλες οι model κλάσεις, οι οποίες έχουν γίνει **register** στο admin – site, μέσω της εντολής **admin.site.register()**.

Παράδειγμα:

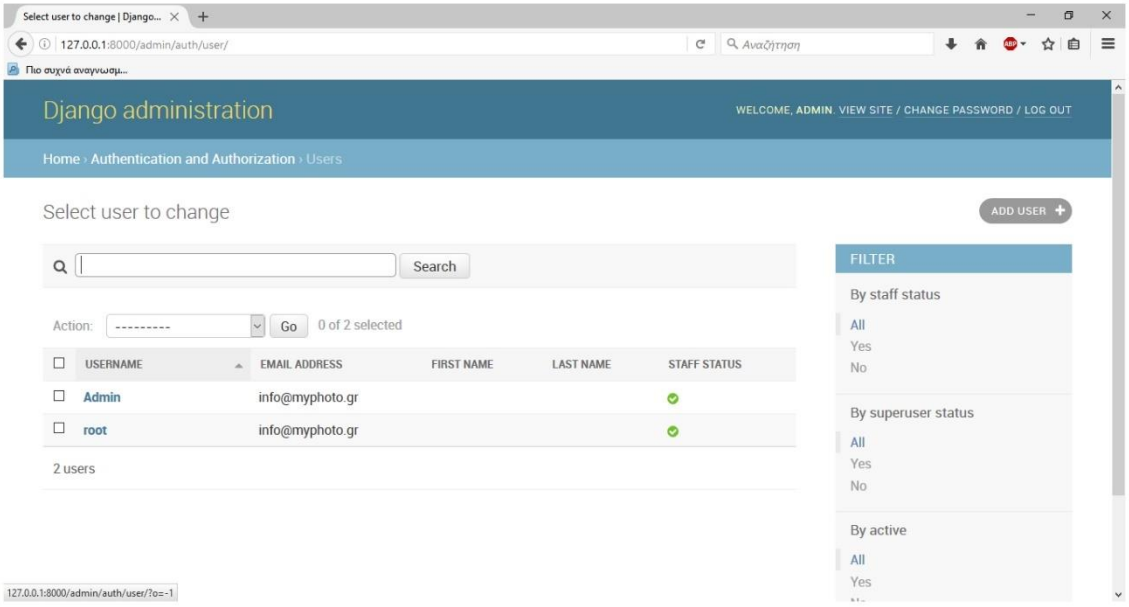


Εικόνα 9: Κεντρική οθόνη της σελίδας admin του Django

Στην παραπάνω εικόνα, παρατηρείται στο interface του admin, η υποκλάση του **ModelAdmin** που είναι ήδη ενσωματωμένη στο Django, (μέσω κώδικα στην εφαρμογή **django – admin**) και σχετίζεται με τη διαχείριση των χρηστών.

Το Django δίνει τη δυνατότητα της προβολής όλων των αντικειμένων των models που έχουν γίνει register στο admin site (μέσω εντολής του **admin.site.registe**r) [55].

Παράδειγμα:

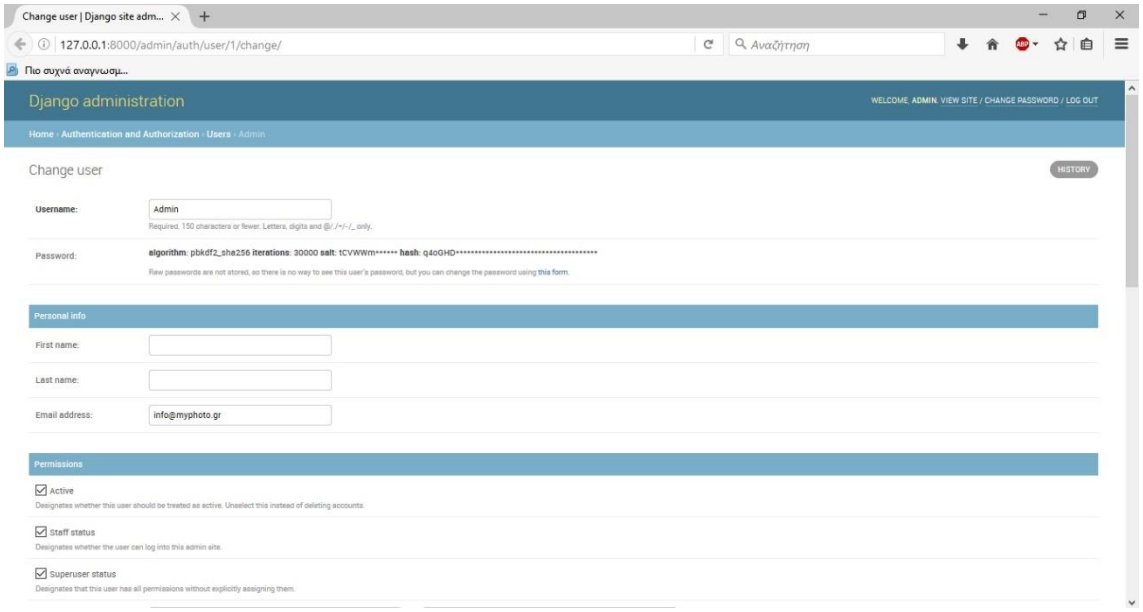


Εικόνα 10: Οθόνη διαχείρισης χρηστών στη σελίδα admin του Django

Στην παραπάνω εικόνα, παρατηρούμε 2 χρήστες (αντικείμενα της model κλάσης User) που έχουν ήδη δημιουργηθεί και αποθηκευτεί στη βάση: τον χρήστη root, και τον χρήστη admin.

Το **admin interface** παρέχει, για κάθε model, τη δυνατότητα πρόσθεσης ενός αντικειμένου της model κλάσης που θα αποθηκευτεί στην βάση, ή τη δυνατότητα αλλαγής των δεδομένων για ένα ήδη αποθηκευμένο αντικείμενο.

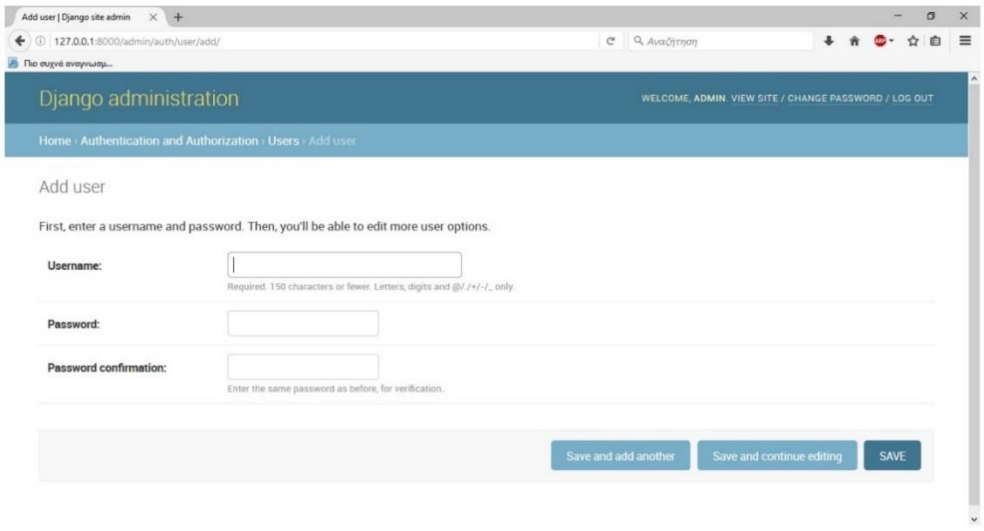
Παράδειγμα:



Εικόνα 11: Φόρμα αλλαγής στοιχείων χρήστη στη σελίδα admin του Django

Στο παραπάνω παράδειγμα, εμφανίζεται η φόρμα που χρησιμοποιείται για αλλαγή των στοιχείων του χρήστη admin. Τα στοιχεία που μπορούν να αλλάξουν, είναι όσα έχουν οριστεί ως **fields** στην αντίστοιχη υποκλάση του **ModelAdmin** που είναι υπεύθυνη για το model **User.**

Παράδειγμα:



Εικόνα 12: Φόρμα προσθήκης νέων χρηστών στη σελίδα admin του Django

Οι λειτουργίες που προσφέρονται από το admin site μπορούν να αλλάξουν με κατάλληλες παρεμβάσεις στον κώδικα.

3.10 Αρχιτεκτονική μιας Εφαρμογής σε Django Framework

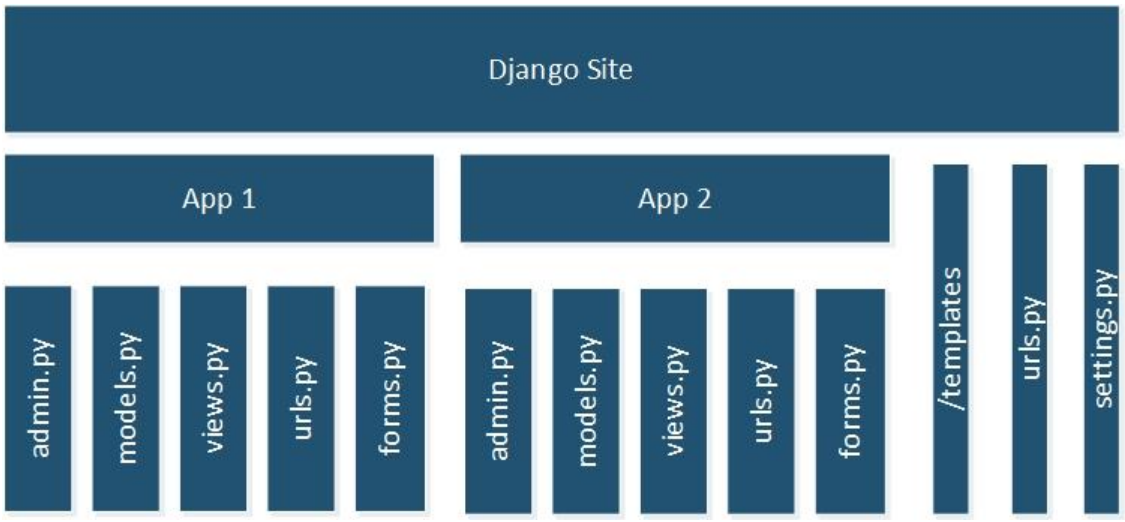
Τα components του Django έχουν περιγραφεί αναλυτικά στην προηγούμενη παράγραφο. Για την καλύτερη κατανόησή τους, σε αυτή την παράγραφο παρουσιάζεται ο τρόπος υλοποίησής τους για τις ανάγκες μιας διαδικτυακής εφαρμογής (**Django Web App**) γενικά και για την εφαρμογή που αναπτύσσεται για τις ανάγκες της εργασίας συγκεκριμένα. Μια διαδικτυακή εφαρμογή, που είναι ανεπτυγμένη με το Django framework, ακολουθεί το MTV πρότυπο του Django.

Όπως έχει ήδη αναφερθεί στην προηγούμενη παράγραφο, το επίπεδο Model (Model layer) ενός app, τα «models», καθορίζονται από τον κώδικα ο οποίος περιέχεται στο αρχείο **models.py**. Το επίπεδο View (View layer) ενός app καθορίζεται από τον κώδικα ο οποίος συνήθως περιέχεται στο αρχείο **views.py**, ενώ το επίπεδο Template (Template layer) από τα html αρχεία τα οποία βρίσκονται στον φάκελο templates. Το **URLconf** είναι αποθηκευμένο συνήθως σε αρχείο **urls.py**. Βασικής σημασίας για μια εφαρμογή είναι οι ρυθμίσεις που βρίσκονται στο αρχείο ρυθμίσεων **settings.py**, ενώ ο κώδικας που ορίζει τις υποκλάσεις **ModelAdmin** για τα επιθυμητά models βρίσκεται στο αρχείο **admin.py**. Τυχόν φόρμες τοποθετούνται σε αρχεία **forms.py**.

Σε αυτό το σημείο αξίζει να σημειωθεί ότι μια εφαρμογή Django μπορεί να συνεργάζεται ή ακόμη και να αποτελείται από άλλες υπο-εφαρμογές (οι οποίες με τη σειρά τους ακολουθούν το πρότυπο MTV).

Η εικόνα 13 απεικονίζει την αρχιτεκτονική μιας διαδικτυακής εφαρμογής που έχει αναπτυχθεί

σε Django.



Εικόνα 13: Αρχιτεκτονική Διαδικτυακής Εφαρμογής ανεπτυγμένης σε Django

Στην παραπάνω εικόνα, το Django Site συμβολίζει το κεντρικό **directory** του site. Μέσα σε αυτό, πρέπει να βρίσκονται τα βασικά αρχεία **urls.py, settings.py** και ο φάκελος **templates** που αφορούν σε όλο το site. Τα **app1** και **app2** είναι φάκελοι που περιέχουν τον κώδικα διαφορετικών στις οποίες βασίζονται οι λειτουργίες του site (για παράδειγμα η εφαρμογή **app1** μπορεί να υλοποιεί την ανάρτηση άρθρων σε ένα blog και η εφαρμογή **app2** να υλοποιεί την ανάρτηση και διαχείριση σχολίων στα άρθρα). Τα αρχεία που βρίσκονται μέσα σε κάθε φάκελο, αφορούν την αντίστοιχη εφαρμογή.

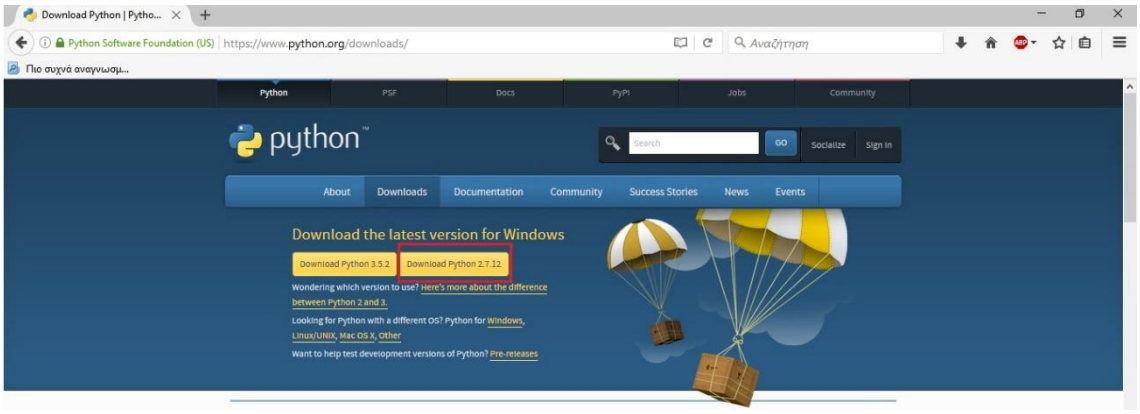
3.11 Παράδειγμα Ανάπτυξης Εφαρμογής με Django

Για την καλύτερη κατανόηση του τρόπου λειτουργίας του Django και της μεθόδου ανάπτυξης διαδικτυακών εφαρμογών με χρήση του, σε αυτή την παράγραφο παρουσιάζεται ένα παράδειγμα στοιχειώδους υλοποίησης τμημάτων μιας εφαρμογής (**models, views, templates, urls, admin**). Για τις ανάγκες του παραδείγματος, αναπτύσσεται μια εφαρμογή **newproject** και μια υποεφαρμογή **myapp**, η οποία προβάλλει αντικείμενα σε μία σελίδα του site **newproject**.

3.11.1 Εγκατάσταση Python 2.7 και Django 1.10

Το πρώτο βήμα είναι η εγκατάσταση της Python (που είναι απαραίτητη για να λειτουργήσει το Django) και του Django framework . Για τις ανάγκες του παραδείγματος της εργασίας, επιλέγεται η τελευταία έκδοση της **Python 2.7 (2.7.12)** και η τελευταία έκδοση του Django (**1.10**).

Μετά από πρόσβαση στο www.python.org/downloads/ επιλέγεται το **Download Python 2.7.12**, όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 14: Οθόνη λήψης της γλώσσας Python

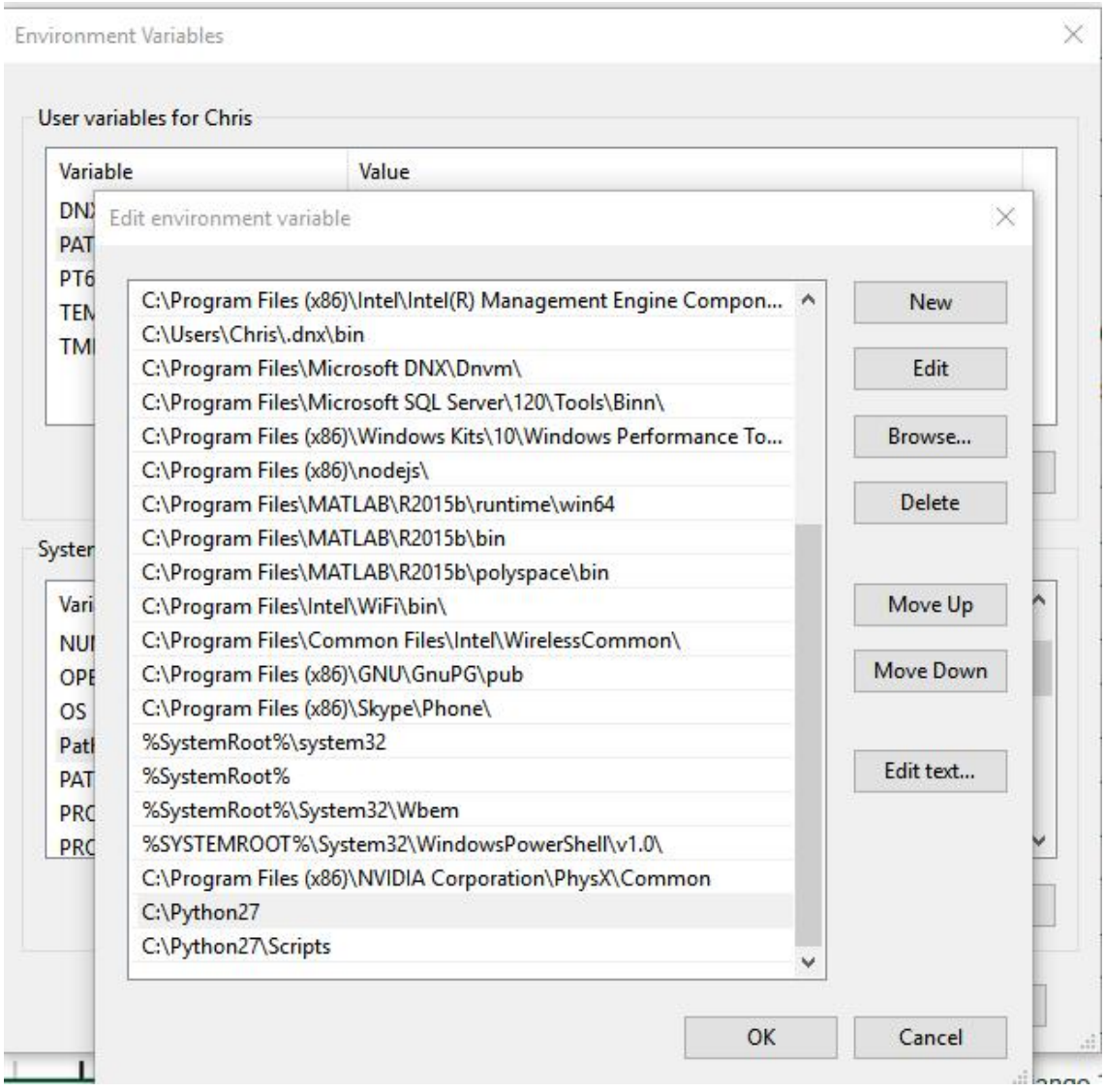
Μετά την λήψη του αρχείου του setup, πραγματοποιείται η εγκατάσταση.

Εικόνα 15: Οθόνη εγκατάστασης Python

Μετά την ολοκλήρωση της εγκατάστασης, σημαντικό βήμα για την διευκόλυνση της χρήσης της Python αλλά και του Django, είναι η τοποθέτηση της Python όπως και των Python Scripts

στα **Enviromental Variables** του συστήματός μας ώστε να υπάρχει η δυνατότητα εκτέλεσης εντολών Python μέσω command prompt από οποιοδήποτε. Προκειμένου να συμβεί αυτό, στην Αναζήτηση πρέπει να πληκτρολογηθεί το **"Enviromental Variables**".

Στο μενού που θα εμφανιστεί, επιλέγεται το **Variable Path**. Επιλέγοντας το **New** πάνω δεξιά, προσθέτουμε το **path** στο οποίο βρίσκεται εγκατεστημένη η Python (στη συγκεκριμένη περίπτωση **C:/Python27**) και τα Scripts της, τα οποία βρίσκονται εντός του directory στο οποίο βρίσκεται η Python (στη συγκεκριμένη περίπτωση **C:/Python27/Scripts**)

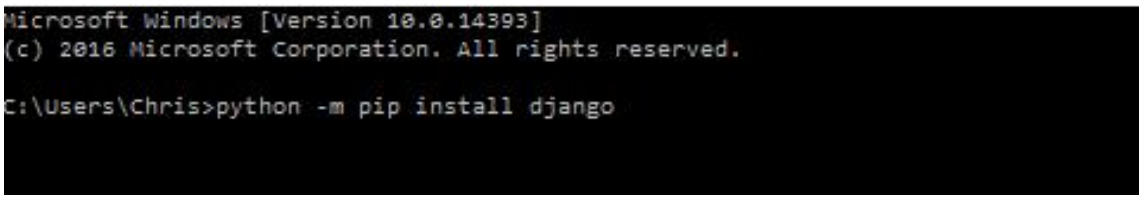


Εικόνα 16: Προσθήκη της Python και των Python Scripts στα Enviromental Variables

Τελευταίο βήμα είναι η εγκατάσταση του Django framework. Προκειμένου να συμβεί αυτό, χρησιμοποιούμε το module της Python **pip**, το οποίο είναι προεγκατεστημένο στις τελευταίες εκδόσεις της Python2 και της Python3 (αν δεν είναι, η εγκατάσταση μπορεί να πραγματοποιηθεί ακολουθώντας τα βήματα που περιγράφονται στον σύνδεσμο:

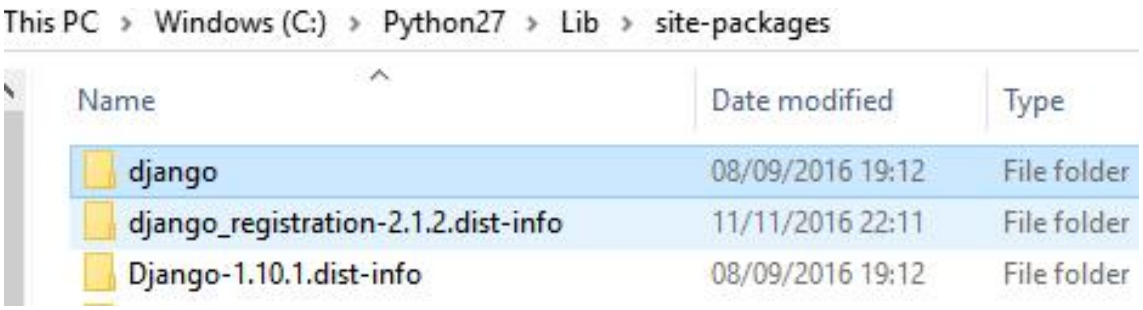
https://pip.pypa.io/en/stable/installing/)

Η εγκατάσταση της τελευταίας έκδοσης του Django μπορεί να πραγματοποιηθεί μέσω της πληκτρολόγησης της εντολής **python -m pip install django** στην κονσόλα. Εφόσον, η Python και τα scripts της έχουν τοποθετηθεί στα Enviromental Variables, δεν έχει σημασία η τοποθεσία από την οποία θα εκτελεστεί η εντολή. Σε διαφορετική περίπτωση, η εντολή είναι απαραίτητο να εκτελεστεί από την τοποθεσία στην οποία βρίσκεται εγκατεστημένο το **pip**.



Εικόνα 17: Εγκατάσταση της τελευταίας έκδοσης του Django

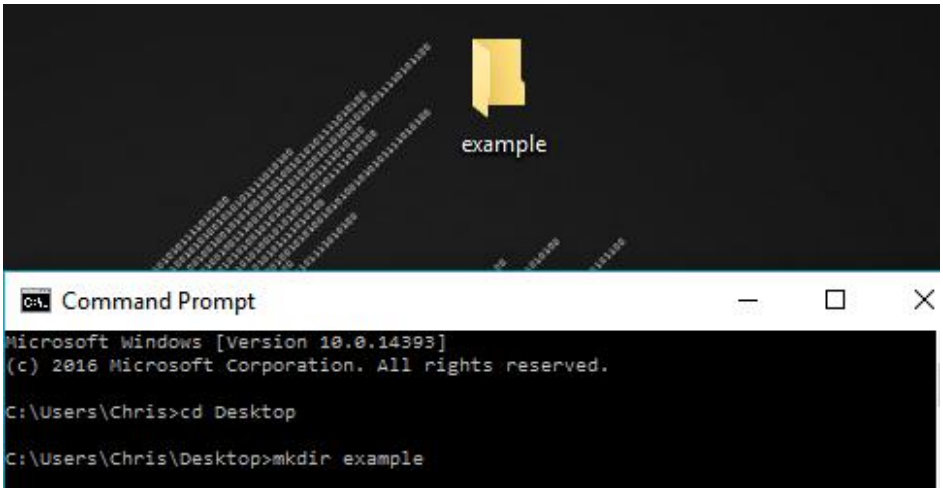
Το Django εγκαθίσταται αυτόματα στην τοποθεσία **Lib/site-packages** εντός της τοποθεσίας στην οποία βρίσκεται η Python.



Εικόνα 18: Αρχεία του Django framework

3.11.2 Δημιουργία Εφαρμογής

Το επόμενο βήμα είναι η έναρξη της ανάπτυξης ενός project σε Django. Για να γίνει αυτό, δημιουργείται ένας φάκελος **example** στην Επιφάνεια εργασίας, στον οποίο πρόκειται να αποθηκευτούν τα αρχεία του κώδικα του project.



Εικόνα 19: Δημιουργία φακέλου example στον οποίο θα αποθηκευτούν τα αρχεία του Django project

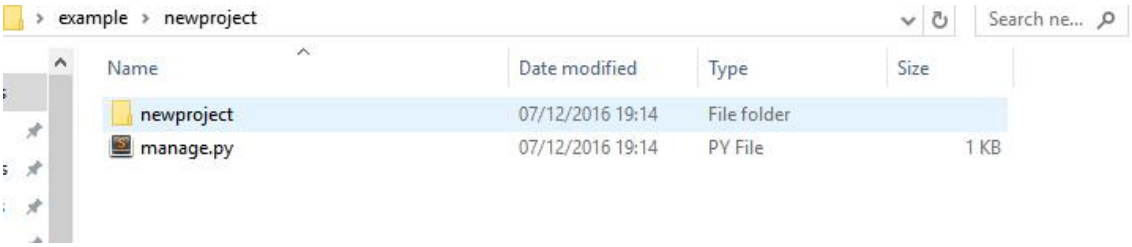
Εντός του φακέλου example που δημιουργήσαμε, σε μία κονσόλα πληκτρολογούμε την εντολή **django-admin startproject newproject**. Το newproject έχει τον ρόλο του τίτλου του project, ο οποίος μπορεί να είναι οποιοσδήποτε.

```
C:\Users\Chris\Desktop\example>django-admin startproject newproject

C:\Users\Chris\Desktop\example>
```

Εικόνα 20: Δημιουργία νέου Django project

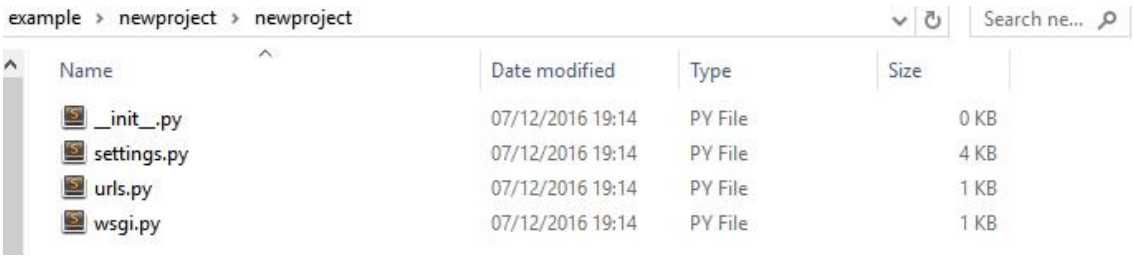
Η εκτέλεση της παραπάνω εντολής έχει ως αποτέλεσμα τη δημιουργία αρχείων εντός του φάκελου **example**.

Αρχικά, παρατηρείται ότι εντός του φακέλου example έχει δημιουργηθεί ένας φάκελος **newproject**. Αυτός ο φάκελος είναι το βασικό directory του site που θα αναπτυχθεί. Μέσα σε αυτό τον φάκελο, υπάρχει ένας ακόμη φάκελος **newproject** και ένα αρχείο **manage.py**. Το αρχείο **manage.py** είναι αυτό που διαχειρίζεται διαδικασίες απαραίτητες για τη λειτουργία της εφαρμογής (συγχρονισμό της βάσης δεδομένων με τα models που δημιουργούνται, λειτουργία server κ.α.), ενώ στον φάκελο **newproject** αποθηκεύονται όλα τα αρχεία κώδικα που σχετίζονται με το σύνολο της διαδικτυακής εφαρμογής.



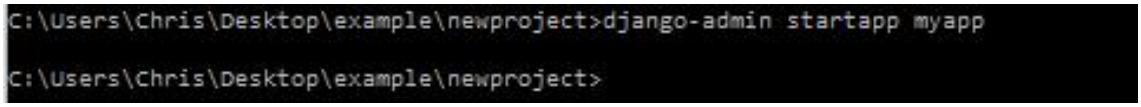
Εικόνα 21: Βασικό directory του Django project

Κατά την είσοδο στον φάκελο **newproject** παρατηρούνται τα αρχεία: **urls.py, settings.py, __init__.py, wsgi.py**. Το αρχείο **settings.py** περιέχει τις απαραίτητες ρυθμίσεις για το σύνολο της διαδικτυακής εφαρμογής, το αρχείο **urls.py** περιέχει το σύνολο των urlpatterns (δηλ. το **URL conf**), ενώ το αρχείο **wsgi.py** χρησιμοποιείται για το deployment της εφαρμογής σε κάποιον web server μετά την ολοκλήρωση της ανάπτυξής της. Το αρχείο **__init__py**, είναι ένα κενό αρχείο που υποδηλώνει ότι όλα τα αρχεία που βρίσκονται στον ίδιο φάκελο είναι Python modules.



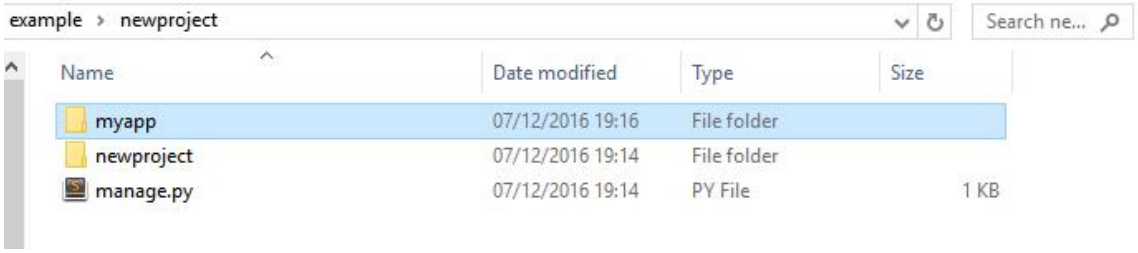
Εικόνα 22: Αρχεία κώδικα του Django project

Η κάθε λειτουργία ενός ιστότοπου είθισται να υλοποιείται από υποεφαρμογές, διευκολύνοντας έτσι την ανάπτυξη του κώδικα και την επανάχρησή του σε άλλα projects. Για να δημιουργηθεί μια τέτοια υποεφαρμογή, από την κύρια τοποθεσία **example/newproject**, σε command prompt πληκτρολογούμε την εντολή **django-admin startapp myapp**, όπου το **myapp** είναι το όνομα της εφαρμογής.



Εικόνα 23: Δημιουργία Django εφαρμογής

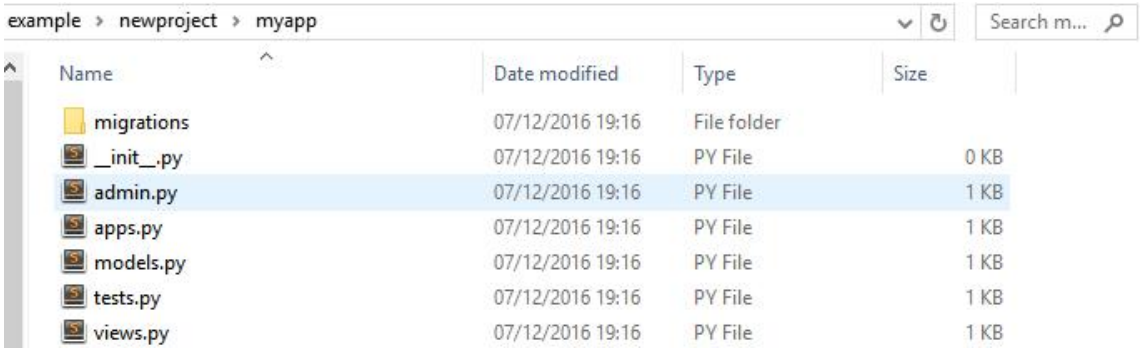
Το αποτέλεσμα είναι η δημιουργία ενός φακέλου **myapp** στο ίδιο επίπεδο με τον φάκελο **newproject** και το αρχείο **manage.py**.



Εικόνα 24: Ο φάκελος που περιέχει τα αρχεία της διαδικτυακής εφαρμογής myapp

Εντός του φακέλου **myapp,** περιέχονται αρχεία και ένας φάκελος που αφορούν στην υποεφαρμογή **myapp.** Τα αρχεία **admin.py, views.py, models.py** καθορίζουν τις λειτουργίες που σχετίζονται με το admin site της εφαρμογής, το επίπεδο Model και View αντίστοιχα, ενώ ο φάκελος **migrations** περιέρχει αρχεία που καταγράφουν τις αλλαγές στο επίπεδο Model και διασφαλίζει το συγχρονισμό του με τη βάση δεδομένων. Το αρχείο **tests.py** περιέχει κώδικα που χρησιμοποιείται για έλεγχο της εφαρμογής, ενώ το **apps.py** περιέχει κώδικα

χρησιμοποιείται για να δηλώσει ότι το σύνολο των αρχείων που βρίσκονται στο συγκεκριμένο φάκελο αποτελούν διαδικτυακή εφαρμογή, καθώς και το όνομα της εφαρμογής αυτής.



Εικόνα 25: Αρχεία της εφαρμογής myapp

Το τελευταίο βήμα, προκειμένου να ληφθεί υπό όψη η υποεφαρμογή myapp ως υποεφαρμογή του project, είναι να συμπεριληφθεί ως τέτοια. Για να πραγματοποιηθεί αυτό, στο αρχείο **settings.py** τοποθεσία **example/newproject/newproject** στη λίστα **INSTALLED_APPS**, προσθέτουμε τη γραμμή **'myapp'** (που είναι το όνομα της εφαρμογής σύμφωνα με το αρχείο **apps.py**) όπως φαίνεται στην εικόνα 26.

```
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'myapp'
41 ]
```

Εικόνα 26: Προσθήκη της εφαρμογής myapp στην λίστα INSTALLED_APPS

Έχοντας δημιουργήσει την διαδικτυακή εφαρμογή, το επόμενο βήμα είναι η υλοποίηση των λειτουργιών που εκτελεί, δηλαδή ο κώδικας για **το models.py, views.py, urls.py** όπως και αυτός που σχετίζεται με την διαχείριση των δεδομένων στο **admin.py**.

3.11.3 Δημιουργία Models

Με τον παρακάτω κώδικα δημιουργείται το model **MyMode**l, το οποίο περιέχει ένα field **name**.

```
1  from __future__ import unicode_literals
2
3  from django.db import models
4
5  class MyModel(model.Model):
6      name=models.CharField(max_length=200)
7
8      def __unicode__(self):
9          return self.name
10
```

Εικόνα 27: Δημιουργία model της εφαρμογής myapp

Μετά τη δημιουργία ενός model, απαιτείται η καταγραφή των αλλαγών στη βάση. Για να γίνει αυτό από την τοποθεσία **example/newproject** σε command prompt πληκτρολογούμε την εντολή **python manage.py makemigrations**, ώστε να δηλωθούν οι αλλαγές και ύστερα **python manage.py migrate** ώστε να πραγματοποιηθούν.

```
C:\Users\Chris\Desktop\example\newproject>python manage.py makemigrations
No changes detected

C:\Users\Chris\Desktop\example\newproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp, sessions
Running migrations:
  Applying myapp.0001_initial... OK
```

Εικόνα 28: Συγχρονισμός των models με τη βάση δεδομένων

Μετά την εκτέλεση των εντολών, οι αλλαγές στο επίπεδο Model των εφαρμογών που έχουν συμπεριληφθεί στην λίστα **INSTALLED_APPS**, αποθηκεύονται σε αντίστοιχους φάκελους migrations κάθε εφαρμογής.

3.11.4 Δημιουργία Views

Το επόμενο βήμα είναι η δημιουργία μιας συνάρτησης view που να καλεί τον κατάλληλο κώδικα html. Για να γίνει αυτό, στο αρχείο **views.py** στην τοποθεσία **example/newproject/myapp** πληκτρολογείται ο κώδικας που φαίνεται στην παρακάτω εικόνα.

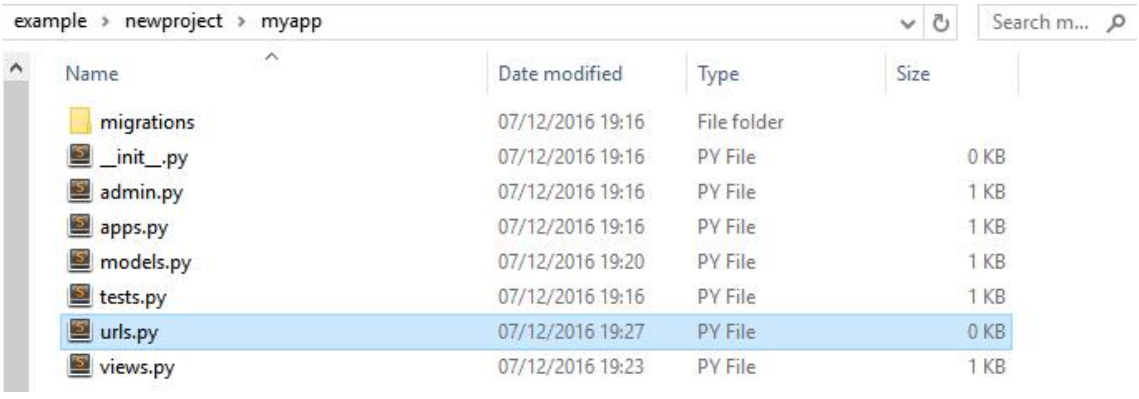
```
1  from django.shortcuts import render
2
3  from models import MyModel
4
5  def myview(request):
6      queryset=MyModel.objects.all()
7      context={'model_list':queryset}
8      return render(request,'mytemplate.html',context)
9
```

Εικόνα 29: Κώδικας του επιπέδου View της εφαρμογής myapp

Ο παραπάνω κώδικας ορίζει μια συνάρτηση **myview**(), η οποία δέχεται request, και επιστρέφει το αρχείο **mytemplate.html**. Μέσω του αρχείου αυτού θα προβληθεί ένα queryset, το οποίο ουσιαστικά είναι όλα τα αντικείμενα της κλάσης **MyMode**l που έχουν αποθηκευτεί στην βάση. Το **queryset** θα προβληθεί στο αντίστοιχο template μέσω της μεταβλητής **model_list**,όπως υποδηλώνει το dictionary context.

3.11.5 Δημιουργία URLconf

Προκειμένου να γίνεται κλήση της συνάρτησης **myview**() μετά την πληκτρολόγηση κάποιου URL από τον χρήστη, απαιτείται η δημιουργία **URLconf** που να διασφαλίζει την αντίστοχιση των URLs με τα κατάλληλα views. Ένας από τους τρόπους για να επιτευχθεί αυτό, είναι η δημιουργία αρχείου **urls.py** στην τοποθεσία **example/newproject/myapp**. Σε αντίθεση με το αρχείο **urls.py** που βρίσκεται στην τοποθεσία **example/newproject/newproject**, το συγκεκριμένο **urls.py** υλοποιεί τις αντιστοιχίσεις που αφορούν τα URLs που σχετίζονται μόνο με την υποεφαρμογή **myapp**.



Εικόνα 30: Δημιουργία αρχείου urls.py για την εφαρμογή myapp

Στο αρχείο **urls.py**, γράφουμε τον παρακάτω κώδικα:

```
1  from django.conf.urls import url
2  from .views import myview
3
4  urlpatterns=[
5  url(r'^',myview)
6  ]
```

Εικόνα 31: Κώδικας του URL conf της εφαρμογής myapp

Ο παραπάνω κώδικας περιέχει ένα urlpattern που αντιστοιχίζεται στην συνάρτηση **myview**(). Συγκεκριμένα, η κανονική έκφραση που αντιστοιχίζεται στην **myview()** είναι η έκφραση που δεν περιέχει κανέναν χαρακτήρα.

Προκειμένου το **URLconf** της εφαρμογής **myapp** να ληφθεί υπόψη, πρέπει να συμπεριληφθεί στο κεντρικό **URLconf**, δηλαδή στο αρχείο **urls.py** που βρίσκεται στην τοποθεσία **example/newproject/newproject**/. Ένας από τους τρόπους να γίνει αυτό είναι μέσω χρήσης της συνάρτησης **include()**. Στην παραπάνω εικόνα παρουσιάζονται τα urlpatterns.

```
16 from django.conf.urls import url,include
17 from django.contrib import admin
18
19
20 urlpatterns = [
21     url(r'^admin/', admin.site.urls),
22     url(r'^myappurls/',include("myapp.urls"))
23 ]
24
```

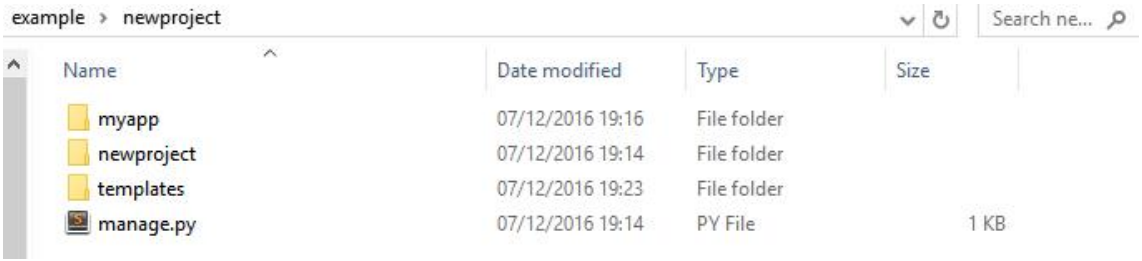
Εικόνα 32: Κώδικας του κεντρικού URLconf του newproject

Ο παραπάνω κώδικας αντιστοιχίζει ένα URL, το τελευταίο τμήμα του οποίου είναι το **myappurls**, με το αρχείο **urls.py** που βρίσκεται στην τοποθεσία **example/newproject/myapp**. Συγκεκριμένα, λόγω του κώδικα που περιέχει το **urls.py**, αν ένας χρήστης πληκτρολογήσει ένα URL της μορφής **my_domain_url/myappurls/,** τότε θα κληθεί η συνάρτηση **myview**(), η οποία θα προβάλει τα κατάλληλα δεδομένα (συγκεκριμένα όλα τα αντικείμενα της κλάσης **MyModel**) με τρόπο που υποδηλώνεται στο αρχείο **mytemplate.html**.

3.11.6 Δημιουργία Templates

Έχοντας περιγράψει τον τρόπο κλήσης ενός template μέσω ενός request, καθώς και τον τρόπο προβολής δεδομένων μέσω μιας συνάρτησης view, το τελευταίο βασικό βήμα για την υλοποίηση μιας διαδικτυακής εφαρμογής είναι η υλοποίηση των templates.

Για να γίνει αυτό, στις ρυθμίσεις πρέπει να συμπεριληφθεί η τοποθεσία στην οποία το Django θα αναζητά τα templates. Αρχικά, στην τοποθεσία **example/newproject/** δημιουργούμε έναν φάκελο templates, στο οποίο θα αποθηκεύονται όλα τα templates από δω και στο εξής.



Εικόνα 33: Δημιουργία φακέλου templates

Ύστερα, μεταβαίνουμε στο αρχείο **settings.py** στην τοποθεσία **example/newproject** και στην γραμμή **TEMPLATES**, στο **'DIRS' :** γράφουμε την γραμμή κώδικα **[os.path.join(BASE_DIR,'templates')]** όπως φαίνεται παρακάτω.

```
53 ROOT_URLCONF = 'newproject.urls'
54
55 TEMPLATES = [
56     {
57         'BACKEND': 'django.template.backends.django.DjangoTemplates',
58         'DIRS': [os.path.join(BASE_DIR,'templates')],
59         'APP_DIRS': True,
60         'OPTIONS': {
61             'context_processors': [
62                 'django.template.context_processors.debug',
63                 'django.template.context_processors.request',
64                 'django.contrib.auth.context_processors.auth',
65                 'django.contrib.messages.context_processors.messages',
66             ],
67         },
68     },
69 ]
```

Εικόνα 34: Κώδικας που δηλώνει την τοποθεσία των templates στο αρχείο settings.py

Στην παραπάνω εικόνα παρουσιάζεται ένας δυναμικός τρόπος δήλωσης της τοποθεσίας στην οποία βρίσκονται τα templates: δηλώνει στο Django ότι τα templates βρίσκονται στον φάκελο με τίτλο templates της βασικής τοποθεσίας της εφαρμογής, όποια και αν είναι αυτή.

Τελευταίο και πιο σημαντικό βήμα είναι η δημιουργία των html αρχείων, των templates αυτών

καθαυτών. Ένας τρόπος να γίνει αυτό, είναι η αξιοποίηση της δυνατότητας της κληρονομικότητας των templates που παρέχει το Django.

Στην τοποθεσία **example/newproject/templates** δημιουργούμε το αρχείο **base.html**, που θα αποτελέσει γονέα για τα υπόλοιπα templates της εφαρμογής.
Στο αρχείο **base.html** γράφουμε τον παρακάτω κώδικα:

```
1  <h1>My Template</h1>
2  {% block content %}
3  {% endblock %}
```

Εικόνα 35: Κώδικας του αρχείο base.html

Το **base.html** περιέχει την φράση My Template και μέσω της ετικέτας **{%block content%}** επιτρέπει σε templates – παιδιά του να εισάγουν τον δικό τους κώδικα. Το σημείο αυτό απαιτεί ιδιαίτερη προσοχή, γιατί αν ένα template – παιδί του **base.html**, έχει κώδικα εκτός των tags **{% block %}…{% endblock %}** ο κώδικας αυτός δεν θα ληφθεί υπόψη.

Ύστερα υλοποιούμε το αρχείο **mytemplate.html**, που όπως περιγράφηκε παραπάνω, χρησιμοποιείται από την συνάρτηση **myview()** για να προβληθούν τα κατάλληλα δεδομένα, όταν η διαδικτυακή εφαρμογή δεχθεί το URL **my_domain_url/myappsurls/**.

```
1  {% extends "base.html" %}
2  {% block content %}
3  {% for mymodel in model_list %}
4  {{mymodel.name}}
5  <br>
6  {% endfor %}
7  {% endblock %}
```

Εικόνα 36: Κώδικας του αρχείου mytemplate.html

Με τη γραμή **{%extends "base.html"%}** δηλώνεται ότι το **mytemplate.html** «κληρονομεί» τον κώδικα του **base.thml.** Ύστερα εντός των tags **{% block %}..{% endblock %},** συμπληρώνεται ο κώδικας που υλοποιεί τον τρόπο προβολής των δεδομένων. Όπως μπορεί να παρατηρήσει κανείς στην εικόνα, ο εν λόγω κώδικας εμφανίζει το όνομα (δηλαδή το field **name**) κάθε αντικειμένου mymodel που υπάρχει στην **model_list**,δηλαδή το queryset που έχει δημιουργηθεί μέσω της κλήσης της συνάρτησης **myview**().

3.11.7 Προβολή των Σελίδων της Εφαρμογής

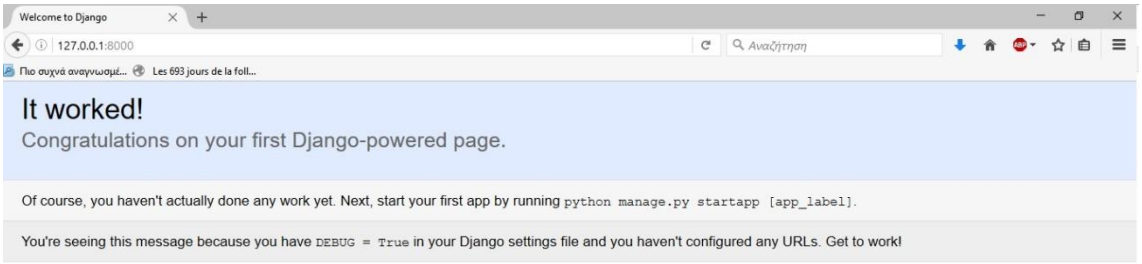
Προκειμένου να γίνει δυνατή η προβολή του κώδικα μιας εφαρμογής από έναν χρήστη, απαιτείται η μεταφορά του πηγαίου κώδικά της σε κάποιο **server**. Το Django, για τις ανάγκες ελέγχου κατά τη διάρκεια ανάπτυξης μιας εφαρμογής περιέχει ενσωματωμένο **development server**. Προκειμένου να λειτουργήσει, από την τοποθεσία **example/newproject** πληκτρολογούμε σε command prompt την εντολή **python manage.py runserver**.

```
C:\Users\Chris\Desktop\example\newproject>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
December 07, 2016 - 19:47:18
Django version 1.10.1, using settings 'newproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Εικόνα 37: Εκκίνηση του Django Development Server

Η προεπιλεγμένη IP του server είναι η IP του localhost (**127.0.0.1**) και η επικοινωνία πραγματοποιείται μέσω της πόρτας **8000**.

Πληκτρολογώντας σε browser **http:/127.0.0.1:8000**, πραγματοποιείται επιτυχής πρόσβαση στον server όπως φαίνεται στην εικόνα 38.



Εικόνα 38: Προεπιλεγμένη κεντρική σελίδα του Django

Η σελίδα αυτή είναι η σελίδα που επιστρέφει το Django ως απάντηση του αιτήματος για το URL 127.0.0.1:8000 και μπορεί να αλλάξει με κατάλληλες αλλαγές στο **URLconf**.

Έχει διαπιστωθεί ότι ο server λειτουργεί. Το επόμενο βήμα, είναι να διαπιστωθεί ότι η εφαρμογή που δημιουργήθηκε στις προηγούμενες παραγράφους λειτουργεί επίσης.
Προκειμένου να συμβεί αυτό, δημιουργούμε τρία αντικείμενα **MyModel** τα οποία αποθηκεύονται στη βάση δεδομένων με χρήση του command prompt και της εντολής **python manage.py shell**.

```
C:\Users\Chris\Desktop\example\newproject>python manage.py shell
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from myapp.models import MyModel
>>> MyModel.objects.create(name="model 1")
<MyModel: model 1>
>>> MyModel.objects.create(name="model 2")
<MyModel: model 2>
>>> MyModel.objects.create(name="model 3")
<MyModel: model 3>
>>>
```

Εικόνα 39: Δημιουργία αντικειμένων στη βάση δεδομένων με χρήση της κονσόλας

Ύστερα επανεκκινώντας τον server, εφόσον το κύριο URL (domain url) είναι το **127.0.0.1:8000**, πληκτρολογώντας το **127.0.0.1:8000/myappurls/**, παρατηρείται το παρακάτω αποτέλεσμα.

Εικόνα 40: Επιτυχής προβολή της σελίδας της εφαρμογής myapp

Παρατηρούμε την επιτυχή λειτουργία της εφαρμογής. Η πληκτρολόγηση του URL είχε ως αποτέλεσμα την επιστροφή του **mytemplate.html** το οποίο πρόβαλλε τα ονόματα όλων των αντικειμένων **MyModel** που είχαν καταχωρηθεί στη βάση δεδομένων. Παρατηρείται επίσης, ότι στη σελίδα εμφανίζεται και η επικεφαλίδα «**My Template»**, η οποία έχει κληρονομηθεί από το αρχείο **base.html**.

3.11.8 Σελίδα admin

Μία ακόμη χρήσιμη δυνατότητα, όπως περιγράφηκε και σε προηγούμενες παράγραφους, είναι η ενσωματωμένη εφαρμογή **admin** που παρέχει το Django. Η σελίδα admin που παρέχει η εφαρμογή δίνει εξαρχής την επιλογή διαχείρισης των χρηστών μιας διαδικτυακής εφαρμογής, και έχει τη δυνατότητα επέκτασης ώστε να επιτρέπει την διαχείριση όλων των δεδομένων της

βάσης δεδομένων.

Στο συγκεκριμένο παράδειγμα, για να αποκτηθεί η δυνατότητα διαχείρισης των αντικειμένων της κλάσης **MyMode**l, στο αρχείο **admin.py** που βρίσκεται στην τοποθεσία **example/newproject/myapp** , δημιουργούμε την κλάση **MyModelAdmin**, υποκλάση της κλάσης **ModelAdmin**. Η κλάση αυτή θα χρησιμοποιηθεί για τη διαχείριση των αντικειμένων της κλάσης **MyModel** από την σελίδα **admin.** Ύστερα με την γραμμή κώδικα **admin.site.register(MyModel,MyModelAdmin)** κάνουμε register το **MyModel** μαζί με την **MyModelAdmin** στην σελίδα admin. Η κλάση **MyModelAdmin** έχει ένα **attribute list_displa**y, το οποίο δηλώνει στο Django τα attributes του αντικειμένου της κλάσης **MyModel** που θα εμφανίζονται στη σελίδα. Στη συγκεκριμένη περίπτωση, αυτό είναι το name, δηλαδή το μοναδικό field της κλάσης **MyModel**.

```
1  from django.contrib import admin
2  from models import MyModel
3
4
5  class MyModelAdmin(admin.ModelAdmin):
6      list_display=["name"]
7
8
9  admin.site.register(MyModel,MyModelAdmin)
```

Εικόνα 41: Κώδικας εγγραφής της κλάσης MyModel στην σελίδα Django admin

Προκειμένου να αποκτηθεί πρόσβαση στην σελίδα admin, πρέπει να δημιουργηθεί χρήστης με δικαιώματα διαχειριστή. Για να γίνει αυτό, σε command prompt στην τοποθεσία **example/newproject** πληκτρολογούμε την εντολή **python manage.py createsuperuser**.

```
C:\Users\Chris\Desktop\example\newproject>python manage.py createsuperuser
Username (leave blank to use 'chris'): admin
Email address: admin@newproject.gr
Password:
Password (again):
Superuser created successfully.
```

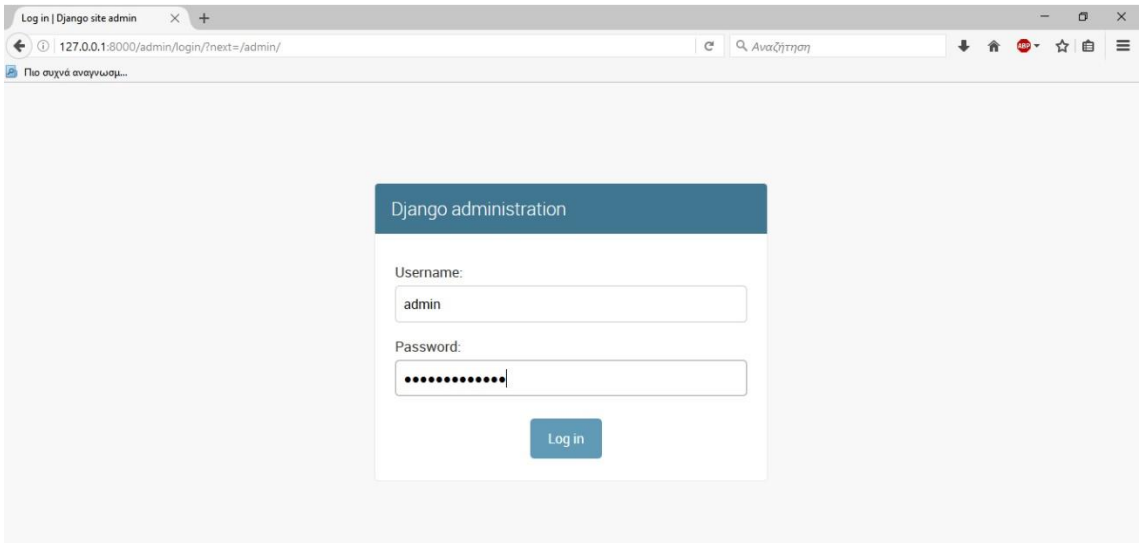
Εικόνα 42: Δημιουργία διαχειριστή (superuser) της διαδικτυακής εφαρμογής myapp

Όπως φαίνεται στην παραπάνω εικόνα, αφού εκτελεστεί η εντολή **createsuper**, ζητώνται στοιχεία για τον χρήστη που θα δημιουργηθεί. Για τις ανάγκες του παραδείγματος, δημιουργείται χρήστης με όνομα χρήστη **admin** και κωδικό **adminpassword**. Αξίζει να σημειωθεί ότι το Django χρησιμοποιεί πολιτική κωδικού για τη δημιουργία διαχειριστή που επιβάλλει ο κωδικός

να μην είναι μικρότερος από 6 χαρακτήρες, να μην είναι ίδιος με το όνομα χρήστη κ.α.

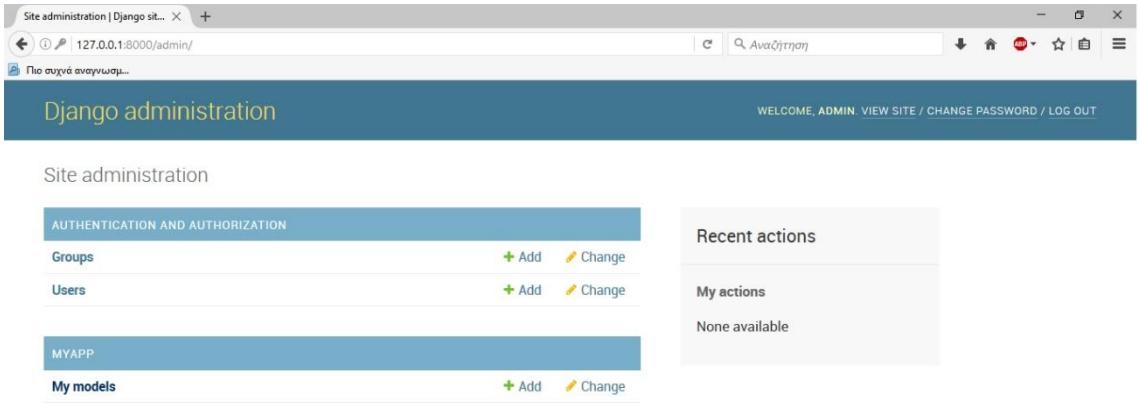
Το προεπιλεγμένο URL της σελίδας admin που παρέχει το Django έχει την μορφή **my_domain_url/admin/**, αν και υπάρχει η δυνατότητα αλλαγής του από το κεντρικό URL conf της εφαρμογής (του urls.py που βρίσκεται στην τοποθεσία **example/newproject/newproject**).

Πληκτρολογώντας **127.0.0.1:8000/admin**, προβάλλεται η παρακάτω οθόνη που απαιτεί τα στοιχεία σύνδεσης.



Εικόνα 43: Οθόνη σύνδεσης διαχειριστή

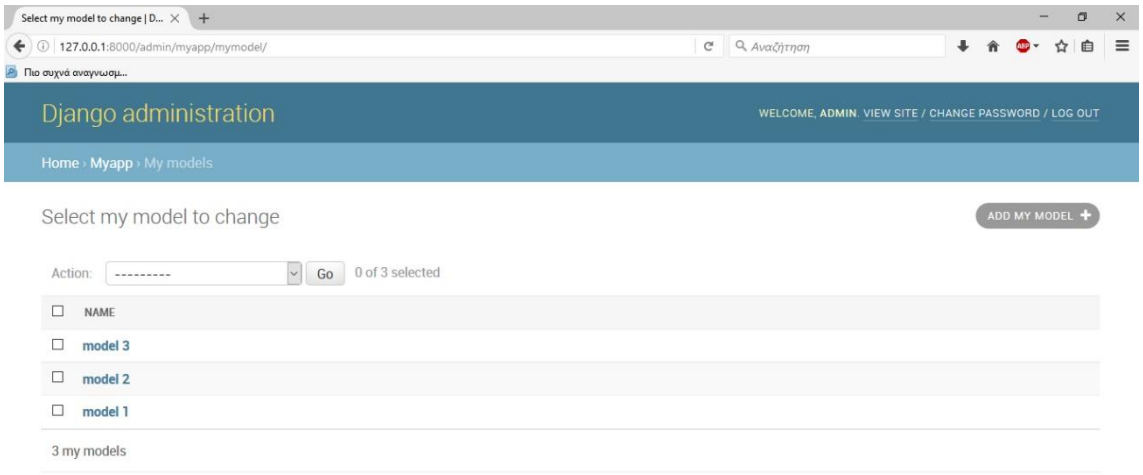
Εισάγοντας τα στοιχεία του χρήστη που δημιουργήθηκε, η πρόσβασή στην σελίδα admin είναι επιτυχής.



Εικόνα 44: Κεντρική σελίδα admin της διαδικτυακής εφαρμογής myapp

Όπως φαίνεται στην παραπάνω εικόνα, η σελίδα admin δίνει δυνατότητα παρέμβασης στους χρήστες που έχουν δημιουργηθεί, όπως και στα models της εφαρμογής **myapp**.

Επιλέγοντας το “My models”, παρατηρούμε ότι εμφανίζονται όλα τα αντικείμενα της κλάσης **MyModel** που έχουν αποθηκευτεί στην βάση δεδομένων. Το Django παρέχει τη δυνατότητα προσθήκης νέου αντικειμένου στην βάση (με την επιλογή «ADD MY MODE» πάνω δεξιά) ή αλλαγής των στοιχείων ενός αντικειμένου κάνοντας κλικ στο όνομά του στη λίστα.



Εικόνα 45: Προβολή όλων των αντικειμένων που έχουν καταχωρηθεί στην βάση στη σελίδα admin του Django

Κεφάλαιο 4: Η Ασφάλεια στο Προγραμματιστικό Πλαίσιο Django

Μετά την παρουσίαση των πιο κρίσιμων ευπαθειών και κινδύνων ασφάλειας, αλλά και την εισαγωγή στη φιλοσοφία και στον τρόπο υλοποίησης διαδικτυακών εφαρμογών με χρήση του Django (που πραγματοποιήθηκε στα Κεφάλαια 2 & 3 αντίστοιχα) επόμενο βασικό βήμα είναι η παρουσίαση των εργαλείων ασφάλειας που διαθέτει το Django για την αντιμετώπιση των κινδύνων και των ευπαθειών.

4.1 Αυθεντικοποιήση και Εξουσιοδότηση Χρηστών

Μία διαδικτυακή εφαρμογή πρέπει να διασφαλίζει την εμπιστευτικότητα, τη διαθεσιμότητα των ακεραιότητα των δεδομένων και των υπηρεσιών που παρέχει στους χρήστες της. Η υλοποίηση κατάλληλων μηχανισμών που να διασφαλίζουν τον περιορισμό πρόσβασης σε εμπιστευτικά δεδομένα, μέσω διαπιστευτηρίων ή ρόλων χρηστών αλλά και μηχανισμών που να καταγράφουν τη δραστηριότητα είναι θεμελιώδες ζήτημα για μία ασφαλή διαδικτυακή εφαρμογή.

Το προγραμματιστικό πλαίσιο Django παρέχει ένα πλήρες σύστημα αυθεντικοποίησης και εξουσιοδότησης χρηστών. Το σύστημα αυθεντικοποίησης του Django διαθέτει:

α) τη δυνατότητα δημιουργίας νέων χρηστών
β) τη δυνατότητα διαχείρισης των δικαιωμάτων (permissions) των χρηστών
γ) τη δυνατότητα δημιουργίας ρόλων και ομάδων χρηστών, στις οποίες δίνονται συγκεκριμένα δικαιώματα
δ) ρυθμιζόμενο σύστημα κρυπτογράφησης κωδικών
ε) έτοιμες φόρμες εγγραφής και εισόδου χρηστών
στ) μηχανισμούς ελέγχου ισχύος κωδικού
ζ) αντιμετώπιση λανθασμένων προσπαθειών εισόδου
η) δυνατότητα αυθεντικοποίησης χρηστών μέσω αρχής αυθεντικοποίησης.

Το σύστημα αυθεντικοποίησης του django υλοποιείται από την εφαρμογή '**django.contrib.auth'**. Προκειμένου αυτή η εφαρμογή να ενεργοποιηθεί σε μια διαδικτυακή εφαρμογή χρειάζεται να προσταθεί στη λίστα **INSTALLED_APPS** στο αρχείο **settings.py**.

4.1.1 Η Κλάση User

Οι χρήστες, στην εφαρμογή **django.contrib.auth** αναπαρίστανται από αντικείμενα της model κλάσης **User**. Με τη δημιουργία τους ή την αλλαγή κάποιου στοιχείου τους, το django πραγματοποιεί τις αντίστοιχες μετατροπές σε πίνακα στην βάση δεδομένων, όπως συμβαίνει με όλα τα υπόλοιπα models.

Η κλάση **User** διαθέτει attributes όπως **username, password, email, first_name, last_name**. Επίσης, η κλάση **User** διαθέτει κάποια attributes που είναι boolean μεταβλητές, τα οποία καθορίζουν τα δικαιώματα ενός αντικειμένου της κλάσης. Μερικές από αυτές τις μεταβλητές είναι οι παρακάτω:

α) is_staff: καθορίζει εάν ένας χρήστης έχει δικαίωμα πρόσβασης στο admin site.
β) is_active: καθορίζει εάν ο λογαριασμός ενός χρήστη είναι ενεργός. Αντί για διαγραφή

χρηστών, ο διαχειριστής μιας μπορεί απλά να θέτει False την τιμή αυτού του attribute.
γ) is_superuser: καθορίζει εάν ένας χρήστης έχει δικαιώματα διαχειριστή.
δ) is_authenticated: καθορίζει εάν ένας χρήστης έχει πραγματοποιήσει είσοδο στο λογαριασμό του κατά την τρέχουσα σύνοδο.

4.1.2 Δημιουργία Χρηστών

Υπάρχουν τρεις βασικοί τρόποι δημιουργίας χρηστών. Ο ένας είναι μέσω του **admin site** του Django. Ο δεύτερος είναι μέσω χρήσης φόρμας εγγραφής χρήστη. Ο τρίτος είναι μέσω του command prompt με χρήση της εντολής **create()**. Σε κάθε δημιουργία λογαριασμού χρήστη μέσω της εντολής **create()** , μέσω του admin site ή μέσω χρήσης φόρμας εγγραφή χρήστη στη διαδικτυακή εφαρμογή, το Django θέτει αυτόματα την τιμή **False** στις μεταβλητές εκείνες που ορίζουν τα δικαιώματα του χρήστη (εκτός και αν ο διαχειριστής ορίσει κάτι διαφορετικά). Αυτό σημαίνει πως οι λογαριασμοί που δημιουργούνται έτσι είναι λογαριασμοί απλών χρηστών χωρίς αυξημένα δικαιώματα.

Παράδειγμα:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john', 'lennon@thebeatles.com',
'johnpassword')

# At this point, user is a User object that has already been saved
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = 'Lennon'
>>> user.save()
```

Στο παραπάνω παράδειγμα, φαίνεται πώς μπορεί να δημιουργηθεί ένας χρήστης με χρήση του command prompt.

4.1.3 Δημιουργία superuser

Ένας χρήστης που έχει ρόλο superuser (ρόλο διαχειριστή μιας διαδικτυακής εφαρμογής), δηλαδή ένας χρήστης που έχει αυξημένα δικαιώματα, μπορεί να δημιουργηθεί πληκτρολογώντας την εντολή **manage.py createsuperuser** σε **command prompt**.

Παράδειγμα:

```
$ python manage.py createsuperuser --username=joe --
email=joe@example.com
```

Μετά την εκτέλεση της εντολής, το Django θα ζητήσει να πληκτρολογηθεί κωδικός για τον συγκεκριμένο χρήστη. Αν ο κωδικός πληροί κριτήρια που θέτει το Django (και περιγράφονται σε επόμενη παράγραφο), γίνεται αποδεκτός και ο διαχειριστής δημιουργείται επιτυχώς.

4.1.4 Αλλαγή Κωδικών

Η αλλαγή του κωδικού ενός χρήστη μπορεί να πραγματοποιηθεί με πολλούς τρόπους.
Το Django δεν αποθηκεύει τους κωδικούς των χρηστών σε μορφή απλού κειμένου, αλλά αποθηκεύει τα hash των κωδικών που εισάγονται. Για αυτό τον λόγο,σε περίπτωση που ένας διαχειριστής επιθυμεί να αλλάξει τον κωδικό ενός χρήστη, πρέπει να το κάνει με χρήση της συνάρτησης της μορφής **set_password('newpassword')** όπου **'newpassword'** ο νέος κωδικός σε μορφή string.

Παράδειγμα:

```
>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

4.1.5 Αυθεντικοποίηση Χρηστών

Για την αυθεντικοποίηση ενός χρήστη, η εφαρμογή **django.contrib.auth** παρέχει τη συνάρτηση **authenticate().** Η εν λόγω συνάρτηση χρησιμοποιεί ένα ζεύγος από **username** και **password** ως keyword arguments της και τα συγκρίνει με τα δεδομένα που βρίσκονται στο επιλεγμένο Authentication Backend (που ορίζεται από το αρχείο **settings.py**).Αν αυτός ο συνδυασμός υπάρχει, η **authenticate()** επιστρέφει το αντικείμενο της κλάσης **User** στο οποίο αυτά αντιστοιχούν. Σε περίπτωση που ο έλεγχος αποτύχει, τότε το Authentication Backend επιστρέφει την εξαίρεση **PermissionDenied**,και η μέθοδος **authenticate**() την τιμή **None**.

Παράδειγμα:

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # A backend authenticated the credentials
else:
    # No backend authenticated the credentials
```

4.1.6 Δικαιώματα και Εξουσιοδότηση

Όπως αναφέρθηκε και προηγουμένως,η εφαρμογή **django.contrib.auth** περιέχει ένα απλό σύστημα δικαιωμάτων. Αυτό το σύστημα παρέχει τη δυνατότητα καθορισμού των δικαιωμάτων συγκεκριμένων χρηστών ή συγκεκριμένων ομάδων χρηστών.Το σύστημα αυτό το χρησιμοποιεί το **admin site** του Django ως εξής:
α) η πρόσβαση στην φόρμα προσθήκης ενός αντικειμένου ("**ADD**") παρέχεται μόνο στους χρήστες που έχουν το δικαίωμα "**add**" για την model κλάση του αντικειμένου.
β) η πρόσβαση στην φόρμα αλλαγής ενός αντικειμένου ("**CHANGE**") παρέχεται μόνο στους χρήστες που έχουν το δικαίωμα "**change**"για την αντίστοιχη model κλάση του αντικειμένου.
γ) το δικαίωμα διαγραφής ενός αντικειμένου δίνεται στους χρήστες που έχουν το δικαίωμα "**delete**" για την model κλάση του αντικειμένου.

Τα δικαιώματα δεν καθορίζονται μόνο βάσει της κλάσης του αντικειμένου , αλλά και για συγκεκριμένα instances του αντικειμένου. Με χρήση των μεθόδων **has_add_permission()**, **has_change_permission()**, **has_delete_permission()** τις οποίες δέχεται η κλάση

ModelAdmin καθίσταται δυνατός ο καθορισμός δικαιωμάτων για διαφορετικά instances αντικειμένων της ίδιας κλάσης.

Τα αντικείμενα της κλάσης User έχουν δύο fields many – to – many: το **groups** και το **user_permissions**. Η πρόσβαση και ο καθορισμός αυτών πραγματοποιείται όπως σε οποιοδήποτε άλλο model στο Django και αποτελεί εναλλακτικό τρόπο του καθορισμού των δικαιωμάτων του χρήστη.

Παράδειγμα:

```
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

Κατά την έναρξη ενός project, το Django ορίζει αυτόματα τα δικαιώματα όλων των αντικειμένων της κλάσης User που είναι αποθηκευμένα στη βάση. Επίσης, κάθε φορά που η εντολή **python manage.py migrate** εκτελείται, το Django ορίζει δικαιώματα για κάθε νέο model που αποθηκεύεται στην βάση και πραγματοποιεί τις αλλαγές για τυχόν δικαιώματα που πραγματοποιήθηκαν για τα models που έχουν ήδη αποθηκευτεί.

Το Django παρέχει την ιδιότητα σε έναν προγραμματιστή να δημιουργήσει τα δικά του δικαιώματα (permissions) με χρήση του attribute **permissions** της κλάσης **Meta** μιας κλάσης Model.

Παράδειγμα:

```
class Task(models.Model):
    ...
    class Meta:
        permissions = (
            ("view_task", "Can see available tasks"),
            ("change_task_status", "Can change the status of
tasks"),
            ("close_task", "Can remove a task by setting its status
as closed"),
        )
```

Στο παραπάνω παράδειγμα τρεις νέες permissions δημιουργούνται για όλα τα αντικείμενα της κλάσης **Task**. Προκειμένου να ελεγχθεί εάν ένας χρήστης έχει κάποιο δικαίωμα χρειάζεται να πληκτρολογηθεί η εντολή **user.has_perm('app_name.'perm_name')**. Στο συγκεκριμένο παράδειγμα για να ελέγξουμε εάν ένας χρήστης έχει το δικαίωμα **change_task_status**, αρκεί να πληκτρολογήσουμε την εντολή **user.has_perm('app.change_task_status)**.

4.1.7 Αυθεντικοποίηση με Βάση τα requests

Το Django, με χρήση συνόδων και του **SessionMiddleware** «εφαρμόζει» το σύστημα αυθεντικοποίησης στα αντικείμενα της κλάσης **request** (δηλ. τα αντικείμενα που αναπαριστούν

τα εισερχόμενα requests).

Δημιουργείται έτσι, ένα attribute **user** των εκάστοτε αντικειμένων της κλάσης request, το οποίο αντιστοιχεί στον χρήστη που πραγματοποιεί το request. Αν ο χρήσης που πραγματοποιεί το request,δεν έχει πραγματοποιήσει είσοδο,τότε στο **request**.user εκχωρείται ένα instance της κλάσης **AnonymousUser**. Σε διαφορετική περίπτωση εκχωρείται ένα instance της κλάσης **User**.

Ο διαχωρισμός ανάμεσα σε χρήστες που έχουν πραγματοποιήσει είσοδο στον λογαριασμό τους και σε χρήστες που δεν το έχουν κάνει, γίνεται με την μεταβλητή **is_authenticated**, η οποία παίρνει τις τιμές **True** και **False** αντίστοιχα.

4.1.7.1 Είσοδος Χρήστη

Η είσοδος ενός χρήστη στη διαδικτυακή εφαρμογή πραγματοποιείται μέσω της συνάρτησης **django.contrib.auth.login().**Η συνάρτηση **login()** παίρνει ως είσοδο ένα αντικείμενο της κλάσης **HttpRequest** και ένα αντικεμενο της κλάσης **User**,και αποθηκεύει το ID του User στην τρέχουσα σύνοδο, με χρήση του **SessionMiddleware**. Αξίζει, να σημειωθεί ότι οποιοαδήποτε δεδομένα σχετικά με τη σύνοδο έχουν αποθηκευτεί στην ανώνυμη σύνοδο (δηλαδή όσο ένας χρήστης δεν έχει πραγματοποιήσει είσοδο στη διαδικτυακή εφαρμογή) , διατηρούνται στην ίδια σύνοδο και μετά την είσοδο του χρήστη. Το παρακάτω παράδειγμα δείχνει τον τρόπο χρήσης των συναρτήσεων **authenticate()** και **login()**σε μια συνάρτηση view.

Παράδειγμα:

```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

4.1.7.2 Έξοδος Χρήστη

Η έξοδος ενός χρήστη από μια διαδικτυακή εφαρμογή που έχει πραγματοποιηθεί με χρήση της **login(),** πραγματοποιείται μέσω της **django.contrib.auth.views.logout()**. Η συνάρτηση **logout()** δέχεται ως είσοδο ένα αντικείμενο της κλάσης **HttpRequest** και δεν επιστρέφει καμία τιμή.

Παράδειγμα:

```
from django.contrib.auth import logout
```

```
def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

Το παραπάνω παράδειγμα δείχνει τον τρόπο με τον οποίο μπορεί να χρησιμοποιηθεί η συνάρτηση **logout()** εντός μιας συνάρτησης view.

Με την κλήση της συνάρτησης **logout()** όλα τα δεδομένα συνόδου που σχετίζονται με το request διαγράφονται, μαζί με όλα τα υπάρχοντα δεδομένα. Αυτό συμβαίνει προκειμένου να αποτραπεί η τυχόν πρόσβαση στα δεδομένα της συνόδου του χρήστη, από τρίτο χρήστη που θα χρησιμοποιήσει τον ίδιο browser αμέσως μετά.

4.1.7.3 Έλεχος Πρόσβασης Χρηστών

Το Django παρέχει ποικίλους τρόπους να οριστεί ο βαθμός πρόσβασης των χρηστών ανάλογα με το αν είναι συνδεδεμένοι, με το ρόλο τους (δηλαδή το Group στο οποίο ανήκουν) και με τα δικαιώματα (permissions) που τους αντιστοιχούν.

Ο πιο απλός τρόπος περιορισμού της πρόσβασης ενός χρήστη είναι με χρήση του attribute **is_authenticated**,που έχει περιγραφεί αναλυτικά προηγουμένως.

Παράδειγμα:

```
from django.conf import settings
from django.shortcuts import redirect

def my_view(request):
    if not request.user.is_authenticated:
        return    redirect('%s?next=%s'    %    (settings.LOGIN_URL,
request.path))
    # ...
```

Στο παραπάνω παράδειγμα, η συνάρτηση **my_view()** δέχεται ένα request ως είσοδο, και ελέγχει την τιμή του **attribute is_authenticated()** που πραγματοποίησε το request. Έαν η τιμή αυτή είναι **False**, δηλαδή τα στοιχεία του χρήστη δεν έχουν αυθεντικοποιηθεί, τότε ο χρήστης ανακατευθύνεται σε άλλη σελίδα (η οποία έχει οριστεί στο αρχείο **settings.py**).

Εκτός από το συγκεκριμένο attribute, το Django αξιοποιεί και διάφορους decorators για να εμποδίσει την εκτέλεση views από χρήστες που δεν πληρούν τα αντίστοιχα κριτήρια.

4.1.7.4 Ο decorator @login_required

Ο συγκεκριμένος decorator χρησιμοποιείται πριν από ένα view και ελέγχει έαν ο χρήστης που πραγματοποίησε το request είναι συνδεδεμένος. Εάν είναι, η συνάρτηση view εκτελείται κανονικά. Εάν δεν είναι, ανακατευθύνεται σε ένα άλλο URL. Το URL αυτό ορίζεται από την τιμή της μεταβλητής **LOGIN_URL** στο αρχείο **settings.py**, ή μπορεί να εισαχθεί στον decorator ως keyword argument με όνομα '**login_url'**.

Παράδειγμα:

Στο παρακάτω παράδειγμα, εάν ο χρήστης δεν είναι συνδεδεμένος ανακατευθύνεται στο URL

'**accounts/login'**.

```
from django.contrib.auth.decorators import login_required

@login_required(login_url='/accounts/login/')
def my_view(request):
    ...
```

Σε περίπτωση που ο έλεγχος ολοκληρωθεί επιτυχώς, ο χρήστης ανακατευθύνεται σε URL το οποίο είναι αποθηκευμένο σε μεταβλητή με το όνομα '**next'**. Αν ένας προγραμματιστής επιθυμεί να δώσει διαφορετικό όνομα στην μεταβλητή αυτή, στη συνάρτηση **login_required()** πρέπει να εισάγει το προαιρετικό keyword argument '**redirect_field_name'**.

Παράδειγμα:

```
from django.contrib.auth.decorators import login_required

@login_required(redirect_field_name='my_redirect_field')
def my_view(request):
    ...
```

4.1.7.5 Ο decorator @user_passes_test

Το Django παρέχει την δυνατότητα περιορισμού πρόσβασης των χρηστών βάσει κριτηρίου το οποίο θα οριστεί από τον προγραμματιστή. Ο έλεγχος πρόσβασης μπορεί να υλοποιηθεί απευθείας με δύο τρόπους:

1) με χρήση χρήστη κατάλληλων μεθόδων της Python στο **request.user.**

Παράδειγμα:

Στο παράδειγμα που ακολουθεί, η συνάρτηση view εκτελείται μόνο εάν το e-mail του χρήση λήγει σε '@example.com'.

```
from django.shortcuts import redirect

def my_view(request):
    if not request.user.email.endswith('@example.com'):
        return redirect('/login/?next=%s' % request.path)
    # ...
```

2) με χρήση του decorator **@user_passes_text()**, ο οποίος δέχεται ως είσοδο μια συνάρτηση που πραγματοποιεί τον έλεγχο, και σε περίπτωση που ο χρήστης που πραγματοποιεί το request δεν ικανοποιεί τα κριτήρια του ελέγχου, τότε η **user_passes_text()** τον ανακατευθύνει σε ένα URL που ορίζεται και αυτό ως keyword argument στη συνάρτηση ή μέσω της μεταβλητής **LOGIN_URL** στο αρχείο **settings.py**.

Παράδειγμα:

```
from django.contrib.auth.decorators import user_passes_test
```

```
def email_check(user):
    return user.email.endswith('@example.com')

@user_passes_test(email_check, login_url='/login/'
)
def my_view(request):
    ...
```

Στο παραπάνω παράδειγμα, ορίζεται η συνάρτηση **email_check**(),η οποία ελέγχει εάν το email του χρήστη που πραγματοποιεί το request λήγει σε **'@example.com**'.Σε κάθε request,ο decorator **@user_passes_test** πραγματοποιεί τον έλεγχο που υλοποιείται στη συνάρτηση **email_check()** και επιστρέφει τις τιμές **True** ή **False** αντίστοιχα. Στην πρώτη περίπτωση, ο κώδικας της **my_view()** εκτελείται, ενώ στη δεύτερη, ο χρήστης ανακατευθύνται στο URL **/login/**.

4.1.7.6 Ο decorator @permission_required

Ο decorator **@permission_required** συντάσσεται με παρόμοιο τρόπο με τους δύο προηγούμενους και ελέγχει εάν ο χρήστης που πραγματοποίησε το request έχει την permission που έχει εισαχθεί στον decorator. Επιστρέφει την τιμή **True** ή **False** αντίστοιχα και στη δεύτερη περίπτωση ανακατευθύνει το χρήστη στο URL που έχει οριστεί. Αξίζει να σημειωθεί ότι η **@permission_required** δέχεται ως keyword argument το '**raise_exception'**.Σε περίπτωση που αυτό λάβει ως τιμή το **True**,τότε η συνάρτηση view, αντί να ανακατευθύνει τον χρήση στο URL που έχει οριστεί, επιστρέφει ένα σφάλμα '**403 Forbidden'**).

Παράδειγμα:

```
from django.contrib.auth.decorators import permission_required

@permission_required('polls.can_vote', login_url='/loginpage/')
def my_view(request):
    ...

@permission_required('polls.can_view',raise_exception=True)
def my_view_2(request):
    ...
```

Στο παραπάνω παράδειγμα,στη περίπτωση που ένας χρήστης δεν έχει το δικαίωμα **'can_vote'**,ανακατευθύνεται στο URL **'/loginpage/'**,ενώ αν δεν έχει το δικαίωμα '**can_view'**,βλέπει μία σελίδα που αναγράφει το σφάλμα '**403'**.

4.1.8 Views του django.contrib.auth

Το Django, μέσω της εφαρμογής **django.contrib.auth** παρέχει έτοιμες συναρτήσεις view οι οποίες υλοποιούν διαδικασίες όπως **login,logout**,αλλαγή κωδικού κ.α. Αυτές οι συναρτήσεις αξιοποιούν έτοιμες φόρμες, οι οποίες θα περιγραφούν αναλυτικότερα σε επόμενη παράγραφο του κεφαλαίου.

Υπάρχουν πολλοί τρόποι να χρησιμοποιηθούν οι έτοιμες συναρτήσεις views σε μία

διαδικτυακή εφαρμογή.

Ο πιο απλός είναι να συμπεριληφθεί το σύνολο των URLS που αντιστοιχούν στις έτοιμες views σε **URLconf** που χρησιμοποιείται στην εφαρμογή.

Παράδειγμα:

```
urlpatterns = [
    url('^', include('django.contrib.auth.urls')),
]
```

Με τον συγκεκριμένο κώδικα, στη διαδικτυακή εφαρμογή θα συμπεριληφθούν τα παρακάτω urlpatterns,τα οποία αντιστοιχούν σε κατάλληλα views :

```
^login/$ [name='login']
^logout/$ [name='logout']
^password_change/$ [name='password_change']
^password_change/done/$ [name='password_change_done']
^password_reset/$ [name='password_reset']
^password_reset/done/$ [name='password_reset_done']
^reset/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-
9A-Za-z]{1,20})/$ [name='password_reset_confirm']
^reset/done/$ [name='password_reset_complete']
```

Σε περίπτωση που ένας προγραμματιστής επιθυμεί να συμπεριλάβει μόνο ένα ή μερικά έτοιμα views, αρκεί να εργαστεί με τρόπο παρόμοιο με αυτόν που περιγράφεται στο παρακάτω παράδειγμα.

Παράδειγμα:

```
from django.contrib.auth import views as auth_views

urlpatterns = [
    url('^change-password/$', auth_views.password_change),
]
```

Αξίζει να σημειωθεί ότι το Django δεν παρέχει υλοποιημένα templates που αντιστοιχούν στα views και η υλοποίησή τους πρέπει να πραγματοποιηθεί από προγραμματιστή. Τα views που παρέχει το Django θα αναζητήσουν templates της μορφής '**view_name.html'** στην τοποθεσία **/registration**. Συνεπώς, για κάθε view που συμπεριλαμβάνεται στην εφαρμογή, ο δημιουργός της πρέπει να υλοποιήσει ένα template που θα έχει όνομα το όνομα του αντίστοιχου view εντός ενός φακέλου με το όνομα **registration**. Αυτό μπορεί να αλλάξει με χρήση του προαιρετικού keyword argument '**template_name'** (στο οποίο μπορεί να καθοριστεί το επιθυμητό template που θα επιστρέψει το αντίστοιχο view).

Παράδειγμα:

```
urlpatterns = [
    url(
        '^change-password/$',
        auth_views.password_change,
        {'template_name': 'change-password.html'}
    ),
```

```
]
```

Στο παραπάνω παράδειγμα, η **password_change()**καλεί το template **change-password.html**.Το Django παρέχει πληθώρα προαιρετικών αντίστοιχων keyword arguments τα οποία μπορεί να αλλάξουν τον τρόπο συμπεριφοράς των έτοιμων views που διαθέτει. Στη συνέχεια αναφέρονται οι έτοιμες views που παρέχει το Django. Λόγω της ομοιότητας συμπεριφοράς που παρουσιάζουν και των παρεμφερών παραμέτρων που δέχονται ως είσοδο, περιγράφεται αναλυτικά μόνο η πρώτη:

α) django.contrib.auth.views.login(request, template_name=`registration/login.html`, redirect_field_name='next', authentication_form=AuthenticationForm, current_app=None, extra_context=None, redirect_authenticated_user=False): υλοποιεί είσοδο ενός μη συνδεδεμένου χρήστη.

Η view **django.contrib.auth.views.login** δέχεται προαιρετικά τις εξής παραμέτρους:

i) template_name: ορίζει το template που καθορίζει την παρουσίαση της φόρμας εισόδου στο χρήστη.

ii)redirect_field_name: το όνομα ενός πεδίου GET το οποίο περιέχει το URL που θα ανακατευθυνθεί ο χρήστης μετά από επιτυχή σύνδεση.

iii)authentication_form: μία κλάση **Form** η οποία χρησιμοποιείται για την αυθεντικοποίηση του χρήστη.

iv)current_app: καθορίζει την εφαρμογή η οποία περιέχει τη συγκεκριμένη view.

v)redirect_autenticated_user: Μία boolean μεταβλητή η οποία καθορίζει εάν οι επιτυχώς συνδεδεμένοι χρήστες θα ανακατευθυνθούν όταν προσπαθήσουν να μπουν στη σελίδα login, με τον ίδιο τρόπο που ανακατευθύνθηκαν όταν πραγματοποίησαν επιτυχή είσοδο στη διαδικτυακή εφαρμογή.

Η view **login()** λειτουργεί ως εξής: αν κληθεί μέσω της HTTP μεθόδου GET, επιστρέφει μία φόρμα για σύνδεση χρήστη. Εάν κληθεί μέσω της POST -η οποία αποστέλει τα διαπιστευτήρια ενός χρήστη - επιχειρεί να πραγματοποίησει τη σύνδεση. Εάν η προσπάθεια είναι επιτυχής, η view ανακατευθύνει τον χρήστη στο URL που έχει οριστεί στο **'redirect_field_name'** ή στην μεταβλητή **LOGIN_REDIRECT_URL** (η τιμή της οποίας έχει οριστεί στο αρχείο settings.py). Σε περίπτωση που δεν είναι, η φόρμα σύνδεσης επιστρέφεται ξανά, μαζί με ένα μήνυμα που περιγράφει τους λόγους που η προσπάθεια σύνδεσης απέτυχε (πχ λάθος κωδικός ή λάθος όνομα χρήστη κ.ο.κ). Κατά την επιτυχή σύνδεση, η view **login()** μεταφέρει τιμές στις εξής template μεταβλητές: **form, next, site, site_name**.

β)django.contrib.auth.views.logout(request,next_page=None, template_name='registration/logged_out.html',redirect_field_name='next', current_app=None, extra_context=None): υλοποιεί έξοδο ενός συνδεδεμένου χρήστη.

γ)django.contrib.auth.views.logout_then_logi(request,login_url=None,current_app=None, extra_context=None): υλοποιεί έξοδο ενός συνδεδεμένου χρήστη και ύστερα τον ανακατευθύνει στο URL που αντιστοιχεί στο view σύνδεσης χρήστη.

δ)django.contrib.auth.views.password_change(request,template_name='registration/password_change_form.html',post_change_redirect=None,password_change_form=PasswordChangeForm, current_app=None, extra_context=None): υλοποιεί την αλλαγή κωδικού χρήστη.

ε)django.contrib.auth.views.password_change_done(request,template_name='registration/password_change_done.html',current_app=None,extra_context=None): καθορίζει τη σελίδα που θα προβληθεί στον χρήστη μετά την επιτυχή αλλαγή κωδικού.

στ)django.contrib.auth.views.password_reset(request,template_name='registration/password_reset_form.html', email_template_name='registration/password_reset_email.html', subject_template_name='registration/password_reset_subject.txt',password_reset_form=PasswordResetForm,token_generator=default_token_generator,post_reset_redirect=None,from_email=None,current_app=None,extra_context=None,html_email_template_name=None, extra_email_context=None): επιτρέπει σε ένα χρήστη να αλλάξει τον κωδικό του, δημιουργώντας ένα μοναδικό σύνδεσμο ο οποίος θα σταλεί στον χρήστη με e – mail.

ζ)django.contrib.auth.views.password_reset_done(request,template_name='registration/password_reset_done.html', current_app=None, extra_context=None): καθορίζει τη σελίδα που θα προβληθεί στο χρήστη αφού επιλέξει το σύνδεσμο που του έχει σταλεί σε mail.

η)django.contrib.auth.views.password_reset_confirm(request,uidb64=None, token=None,template_name='registration/password_reset_confirm.html',token_generator=default_token_generator,set_password_form=SetPasswordForm,post_reset_redirect=None, current_app=None, extra_context=None):
προβάλλει στο χρήση μία φόρμα για εισαγωγή νέου κωδικού.

θ)django.contrib.auth.views.password_reset_complete(request,template_name='registration/password_reset_complete.html', current_app=None, extra_context=None):προβάλλει στο χρήστη σελίδα που τον ενημερώνει ότι ο κωδικός του έχιε αλλάξει επιτυχώς.

1.9 Forms του django.contrib.auth

Το Django παρέχει υλοποιημένες φόρμες για όλες τις λειτουργίες που υλοποιούν οι έτοιμες views. Τις φόρμες αυτές αξιοποιούν οι αντίστοιχες έτοιμες views, ενώ μπορούν να αξιοποιηθούν και από διαφορετικά views,εάν ένας προγραμματιστής επιθυμεί να δημιουργήσει τα δικά του. Οι φόρμες αυτές βασίζονται στο model **User** που διαθέτει το **django.contrib.auth**. Σε περίπτωση που κάποιος υλοποιήσει διαφορετικό model για την αποθήκευση και διαχείριση χρηστών, ίσως να χρειαστεί να δημιουργήσει και τις δικές του φόρμες.

Οι κλάσεις **Form** που παρέχει το Django είναι οι εξής:
α)class AdminPasswordChangeForm: φόρμα για αλλαγή κωδικού χρήστη από διαχειριστή.

β)class AuthenticationForm: φόρμα για σύνδεση χρήστη.

γ)class PasswordChangeForm: φόρμα για αλλαγή κωδικού.

δ)class PasswordResetForm: φόρμα για δημιουργία μοναδικού link προκειμένου να γίνει reset ο κωδικός ενός χρήστη.

ε)class SetPasswordForm: φόρμα που επιτρέπει σε χρήστη να αλλάξει τον κωδικό του χωρίς να εισάγει τον παλιό.

στ)class UseChangeForm: φόρμα που χρησιμοποιείται στο admin interface για αλλαγή δεδομένων και δικαιωμάτων ενός χρήση από διαχειριστή.

η)class UserCreationForm: φόρμα που χρησιμοποιείται για εγγραφή χρήστη.

4.1.10 Αυθεντικοποίηση και Εξουσιοδότηση Χρηστών στα templates

Κατά τη διάρκεια της διαδικασίας του rendering, δηλαδή τη στιγμή που μέσω ενός view επιστρέφεται ένα αντικείμενο **RequestContext** ή το shortcut **render(),** οι πληροφορίες που αφορούν στο χρήστη που πραγματοποίησε το request για το view, μπορούν να ανακτηθούν και μέσω κατάλληλων template μεταβλητών στο αντίστοιχο template που καλείται μέσα από το **RequestContext** ή το **render()**.

Παράδειγμα:

```
{% if user.is_authenticated %}
    <p>Welcome, {{ user.username }}. Thanks for logging in.</p>
{% else %}
    <p>Welcome, new user. Please log in.</p>
{% endif %}
```

Στο παραπάνω παράδειγμα, εάν ένας χρήστης έχει πραγματοποιήσει σύνδεση,θα εμφανιστεί στην οθόνη το username του.

Η χρήση template μεταβλητών με τέτοιο τρόπο δίνει την δυνατότητα πραγματοποίησης ελέγχου πρόσβασης εντός των template αρχείων, όπως φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα:

```
{% if perms.foo %}
    <p>You have permission to do something in the foo app.</p>
    {% if perms.foo.can_vote %}
        <p>You can vote!</p>
    {% endif %}
    {% if perms.foo.can_drive %}
        <p>You can drive!</p>
    {% endif %}
{% else %}
    <p>You don't have permission to do anything in the foo app.</p>
{% endif %}
```

4.1.11 Διαχείριση Χρηστών μέσω του admin site του Django

Ένας χρήστης που έχει δικαιώματα διαχειριστή (superuser) μπορεί να πραγματοποίησει πρόσβαση στο default admin site του Django.Σε αυτό, μπορεί να διαχειριστεί τους καταγεγραμμένους χρήστες μέσω του interface, δηλαδή να προσθέσει, να διαγράψει ή να αλλάξει τα δικαιώματα των χρηστών που έχουν αποθηκευτεί στη βάση δεδομένων. Ένας διαχειριστής δεν μπορεί να δει τον κωδικό ενός χρήστη, όμως έχει τη δυνατότητα να τον αλλάξει, μέσω φόρμας στην οποία παραπέμπεται αν κάνει κλικ στην επιλογή **'Change**

Password'.Η δυνατότητα πρόσβασης ενός χρήστη στο admin site καθώς και των επιλογών που διαθέτει όσον αφορά στη διαχείριση χρηστών καθορίζεται, όπως έχει αναφερθεί, από τα δικαιώματα (permissions)του χρήστη αυτού [57].

4.2 Διαχείριση Κωδικών

Η ασφαλής διατήρηση των κωδικών σε μία διαδικτυακή εφαρμογή είναι θεμελιώδες ζήτημα για τη διασφάλιση της εμπιστευτικότητας των προσωπικών δεδομένων αλλά και για τη διαθεσιμότητα των υπηρεσιών εφαρμογής: η προστασία των κωδικών από κάθε πιθανότητα υποκλοπής τους εξασφαλίζει ότι ένας κακόβουλος χρήστης δεν θα αποκτήσει ανεπιθύμητη πρόσβαση και δικαιώματα στη διαδικτυακή εφαρμογή.

Το Django παρέχει ένα ευέλικτο και ασφαλές σύνολο εργαλείων που αναλαμβάνει την κρυπτογράφηση και την αποθήκευση των κωδικών, το οποίο θα περιγραφεί αναλυτικά στις παραγράφους που ακολουθούν.

4.2.1 Αποθήκευση Κωδικών

Η αποθήκευση κωδικών στο Django δεν γίνεται μέσω απλής κρυπτογράφησής τους. Για να αυξήσει την πολυπλοκότητα και να προστατεύσει τους κωδικούς από προσπάθειες cracking, το Django χρησιμοποιεί μία μεθολογία κρυπτογράφησης και αποθήκευσης που με είσοδο τον κωδικό που παρέχει ένας χρήστης κατά την εγγραφή του (ή κατά την αλλαγή κωδικού), παράγει ένα αρκετά πολύπλοκο αποτέλεσμα.

Ένας κωδικός αποθηκεύεται στο Django, στο attribute password του αντικειμένου User που εισήγαγε τον κωδικό ως ένα string της μορφής:

```
<algorithm>$<iterations>$<salt>$<hash>
```

Κάθε **<>** είναι ένα τμήμα του string. Τα τμήματα του string χωρίζονται μεταξύ τους με τον χαρακτήρα **'$'**. Το τμήμα **<algorithm>** είναι το όνομα του επαναληπτικού (ή μη επαναληπτικού) αλγόριθμου που χρησιμοποιείται για την κρυπτογράφηση, το τμήμα **<iterations>** είναι ο αριθμός των επαναλήψεων του αλγορίθμου, **<salt>** είναι ένα τυχαίο string και **<hash>** είναι το hash του password που θα προκύψει από τη διαδικασία [58].

Ως προεπιλογή, το Django χρησιμοποιεί τον αλγόριθμο **PBKDF2** ο οποίος εκτελεί πολλές φορές τη hashing συνάρτηση **SHA256**. Αυτός ο συνδυασμός είναι αρκετά ασφαλής, μιας και προκειμένου να βρεθεί κάποιος κωδικός με επιθέσεις όπως brute force ή dictionary attack, απαιτείται μεγάλος υπολογιστικός χρόνος και πολλοί πόροι.

Ο αλγόριθμος **PBKDF2 (Password Base Key Derivation Function 2)** προτείνεται από τον οργανισμό **NIST (National Institute of Standards & Delivery)**. Η λειτουργία του περιγράφεται παρακάτω [59],[60].

Το τελικό αποτέλεσμα του PBKDF2 είναι ένα κλειδί (DK) για το οποίο ισχύει:

```
DK = PBKDF2(PRF, Password, Salt,c, dkLen)
```

όπου

PRF είναι η ψευδοτυχαία συνάρτηση δύο παραμέτρων με μήκος εξόδου **hLen**
Password είναι ο κωδικός που έχει εισάγει ο χρήστης
Salt είναι μία τυχαία ακολουθία bits
c είναι ο αριθμός των επαναλήψεων
dkLen είναι το επιθυμητό μήκος του derived key

Το DK (derived key) υπολογίζεται από τον τύπο:

```
DK = T1 || T2 || ... || Tdklen/hlen, όπου Ti = F(Password, Salt, c, i) για κάθε i.
```

Στο παραπάνω, η συνάρτηση **F** είναι η πράξη **XOR** ανάμεσα στα αποτελέσματα της εφαρμογής των επαναλήψεων της PRF συνάρτησης, Η πρώτη συνάρτηση PRF χρησιμοποιεί τον κωδικό που θα δώσει ο χρήστης ως κλειδί και το salt προσαυξημένο από τον δείκτη i (encoded ως big - endian 32-bit integer).

Πιο συγκεκριμένα,

```
F(Password, Salt, c, i) = U1 ^ U2 ^ ... ^ Uc
```

,όπου:

```
U1 = PRF(Password, Salt || INT_32_BE(i))
U2 = PRF(Password, U1)
...
Uc = PRF(Password, Uc-1)
```

Ένας προγραμματιστής έχει τη δυνατότητα να επιλέξει διαφορετικό αλγόριθμο με διαφορετική hash συνάρτηση προκειμένου να κρυπτογραφήσει τους κωδικούς που εισάγουν οι χρήστες, όπως και να επιλέξει συγκεκριμένες ρυθμίσεις που αλλάζουν τη συμπεριφορά των αλγορίθμων αυτών καθαυτών. Η επιλογή του αλγορίθμου γίνεται με τις κατάλληλες αλλαγές στη λίστα **PASSWORD_HASHERS** που βρίσκεται στο αρχείο ρυθμίσεων **settings.py**. Η πρώτη καταχώρηση στη λίστα **PASSWORD_HASHERS**, χρησιμοποιείται για να κρυπτογραφήσει τους κωδικούς που παρέχουν οι χρήστες, ενώ οι επόμενες καταχωρήσεις της λίστας χρησιμοποιούνται για τον έλεγχο των ήδη καταχωρημένων κωδικών.

Με την έναρξη ενός project η λίστα **PASSWORD_HASHERS** στο αρχείο **settings.py** έχει ως εξής:

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
]
```

Όπως φαίνεται παραπάνω, ένας προγραμματιστής πορεί να επιλέξει να χρησιμοποιήσει τον αλγόριθμο **Argon2** ή τον αλγόριθμο **BCrypt** αντί του προεπιλεγμένου **PBKDF2** για την κρυπτογράφηση των κωδικών.

4.2.2 Χρήση αλγόριθμου Argon2

Ο αλγόριθμος **Argon2** ήταν ο νικητής του Διαγωνισμού Password Hasing 2015 [61]. Προκειμένου να χρησιμοποιηθεί, απαιτούνται τα εξής βήματα:

1) Εγκατάσταση της βιβλιοθήκης **argon2-cffi**, το οποίο μπορεί να γίνει με πληκτρολόγηση της εντολής **pip install argon2-cffi** σε command prompt.

2) Κατάλληλη αλλαγή της λίστας **PASSWORD_HASHERS** προκειμένου η καταχώρηση **django.contrib.auth.hashers.Argon2PasswordHasher** να μετακινηθεί στην πρώτη θέση.

Μετά τις αλλαγές, η λίστα PASSWORD_HASHERS θα έχει την εξής μορφή:

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
]
```

4.2.3 Χρήση Αλγόριθμου BCrypt

Προκειμένου να γίνει χρήση του αλγορίθμου **BCrypt** αντί του επιλεγμένου **PBDKF2** πρέπει να ακολουθηθούν τα εξής βήματα:

1) Εγκατάσταση της βιβλιοθήκης **bcrypt**,με πληκτρολόγηση της εντολής **pip install bcrypt** σε command prompt.
2) Αλλαγή της λίστας **PASSWORD_HASHERS** ώστε η καταχώρηση **django.contrib.auth.hashers.BCryptSHA256PasswordHasher** να βρεθεί στην πρώτη θέση.

Μετά τις αλλαγές, η λίστα **PASSWORD_HASHERS** θα έχει ως εξής:

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
]
```

Αξίζει να σημειθωεί ότι ο αλγόριθμος bcrypt περιορίζει τους όλους τους εισαγόμενους κωδικούς στους 72 χαρακτήρες. Αυτό σημαίνει ότι η hash τιμή που θα προκύψει από την εφαρμογή του bcrypt σε έναν κωδικό που περιέχει πάνω από 100 χαρακτήρες, θα ισούται με την hash τιμή που θα προκύψει σε ένων κωδικό με 72 χαρακτήρες. Αν και πρόκειται περί εξεζητημένου ζητήματος, μιας και συνήθως κανένας χρήστης δεν εισάγει κωδικό μεγαλύτερο των 72 χαρακτήρων, η χρήση του **django.contrib.auth.hashers.BCryptSHA256PasswordHasher** αντιμετωπίζει επιτυχώς το πρόβλημα αυτό, μιας και ο κωδικός που εισάγεται γίνεται hash με εφαρμογή της συνάρτησης SHA256.

4.2.4 Αναβάθμιση Πολυπλοκότητας Αλγορίθμων

Οι αλγόριθμοι hashing που χρησιμοποιούνται στο Django έχουν συγκεκριμένη πολυπλοκότητα, μιας και οι παράμετροι από την οποία αυτή εξαρτάται έχουν προεπιλεγμένες τιμές ως είσοδο. Ένας προγραμματιστής έχει τη δυνατότητα να να αυξήσει την πολυπλοκότητα των αλγορίθμων,κάνοντας ακόμη πιο δύσκολη την προσπάθεια cracking από κακόβουλους χρήστες. Όσον αφορά τους αλγορίθμους **PBDKF2** και **bcrypt**,αυτό μπορεί να γίνει μέσω της αλλαγής του attribute **iterations** (επαναλήψεις) της επιθυμητής κλάσης που αναπαριστά τον αλγόριθμο.

Παράδειγμα:

Το παρακάτω παράδειγμα περιγράφει τον τρόπο με τον οποίο οι επαναλήψεις για την εκτέλεση του αλγορίθμου PDKF2 κατά τη διάρκεια αποθήκευσης ενός κωδικού αυξάνονται.

Σε αρχείο **myapp/hashers.py** γράφεται ο εξής κώδικας:

```
from django.contrib.auth.hashers import PBKDF2PasswordHasher

class MyPBKDF2PasswordHasher(PBKDF2PasswordHasher):
    """
    A subclass of PBKDF2PasswordHasher that uses 100 times more
iterations.
    """
    iterations = PBKDF2PasswordHasher.iterations * 100
```

Στο αρχείο **settings.py** γράφεται ο εξής κώδικας:

```
PASSWORD_HASHERS = [
    'myproject.hashers.MyPBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
]
```

Όπως φαίνεται στο παράδειγμα,προκειμένου να αλλάξει ο αριθμός των φορών επανάληψης του αλγορίθμου, δημιουργείται μια υποκλάση της κλάσης **PBKDF2PasswordHasher** με αυξημένο αριθμό επανάληψης, η οποία ύστερα δηλώνεται ως αλγόριθμος που θα χρησιμοποιείται κατά την αποθήκευση των κωδικών που εισάγουν οι χρήστες.

Ο αλγόριθμος **Argon2** έχει τρεις παραμέτρους (attributes της κλάσης **django.contrib.auth.hashers.Argon2PasswordHasher**) οι οποίες μπορούν να αλλάξουν με τρόπο παρόμοιο με αυτόν που περιγράφηκε: το **time_cost** που είναι υπεύθυνο για τον χρόνο εκτέλεσης του αλγόριθμου, το **memory_cost** που καθορίζει το μέγεθος της μνήμης που θα χρησιμοποιηθεί για τον υπολογισμό του hash και το **parallelism** που ορίζει πόσοι επεξεργαστές θα χρησιμοποιηθούν παράλληλα για τον υπολογισμό του hash.

Ο ορισμός των παραπάνω παραμέτρων απαιτεί προσοχή γιατί καθορίζει την υπολογιστική ισχύ και άρα και τον χρόνο που θα χρειαστεί για την αποθήκευση ενός κωδικού κατά την εισαγωγή του από χρήστη. Ο ορισμός λάθος τιμών μπορεί να αποτελέσει εμπόδιο στην ομαλή λειτουργία μιας διαδικτυακής εφαρμογής.

4.2.5 Αυτόνομη Διαχείριση Κωδικών Χρήστη

Το **django.contrib.auth.hashers** διαθέτει, επίσης, ένα σύνολο συναρτήσεων οι οποίες μπορούν να χρησιμοποιηθουν για να δημιουργήσουν ή να ελέγξουν έναν κωδικό κατά τη διάρκεια της αυθεντικοποίησης χρήστη. Οι συναρτήσεις αυτές μπορούν να χρησιμοποιηθούν σε φόρμες και συναρτήσεις view.

Οι συναρτήσεις αυτές είναι οι εξής:

α)check_password(password,encoded): συγκρίνει κωδικό που θα εισαχθεί με το hash που βρίσκεται αποθηκευμένο στη βάση δεδομένων και επιστρέφει **True** σε περίπτωση ταύτισης ή **False** στην αντίθετη περίπτωση.

β)make_password(passwor,salt=None,hasher='default'): δημιουργεί hashed κωδικό με βάση την μορφή που περιγράφεται παραπάνω.

γ)is_password_usable(encoded_password): ελέγχει εάν ο κωδικός που εισάγεται μπορεί να αποτελεί hashed κωδικό (βάσει των hashers που χρησιμοποιούνται).

4.2.6 Πιστοποίηση Κωδικών

Στις περιπτώσεις διαδικτυακών εφαρμογών που ο κωδικός εισάγεται από τους χρήστες, χρειάζεται να διασφαλιστεί ότι ο κωδικός πληροί ορισμένα κριτήρια (πχ μέγεθος, τυχαιότητα) τέτοια ώστε να μην αποτελεί σημείο αδυναμίας της εφαρμογής (δηλαδή να μην καθιστά το cracking εύκολο). Το Django παρέχει ένα σύνολο από μηχανισμούς πιστοποίησης κωδικών (**password validators**). Ο προγραμματιστής μπορεί να επιλέξει να χρησιμοποιήσει όσους από αυτούς επιθυμεί, ή να υλοποιήσει κάποιο δικό του.

Κάθε password validator συνοδεύεται από ένα κείμενο που επεξηγεί στο χρήστη που πρόκειται να εισάγει τον κωδικό τα κριτήρια που πρέπει αυτός να πληροί.
Οι password validators που χρησιμοποιούνται καθορίζονται από τη λίστα **AUTH_PASWORD_VALIDATORS** στο αρχείο **settings.py**.

Οι έλεγχοι των password validators αξιοποιούνται στις φόρμες που ένας χρήστης, εισάγει, αλλάζει ή κάνει reset τον κωδικό του, καθώς και κατά την πληκτρολόγηση των εντολών **createsuperuser** και **changepassword** στο command prompt.

Το σύνολο των password validators που έχει το Django φαίνεται παρακάτω:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
tor',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {
            'min_length': 9,
        }
```

```
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

Οι παραπάνω password validators είναι κλάσεις που λειτουργούν ως εξής:

α)class UserAttributeSimilarityValidator (set_attributes=DEFAULT_USER_ATTRIBUTES, max_similarity=0.7): ελέγχει την ομοιότητα μεταξύ του κωδικού και ενός συνόλου attributes ενός χρήστη.

β)class MinimumLengthValidator(min_length=8): εξετάζει εάν ένας κωδικός είναι ίσος ή μεγαλύτερος με το ελάχιστο αποδεκτό πλήθος ψηφίων.

γ)class CommonPasswordValidator(password_list_path=DEFAULT_PASSWORD_LIST_PATH): συγκρίνει τον δοθέντα κωδικό με μία λίστα κωδικών και διασφαλίζει ότι δεν είναι ένας από αυτούς. Η προεπιλεγμένη λίστα είναι μία λίστα 1000 κωδικών που έχει δημιουργηθεί από τον Mark Brunnet.

δ)class NumericPasswordValidator: εξασφαλίζει ότι ο κωδικός δεν αποτελείται αποκλειστικά από ψηφία.

Όπως αναφέρθηκε, ένας προγραμματιστής έχει τη δυνατότητα να υλοποιήσει τον δικό του validator. Αυτό μπορεί να πραγματοποιηθεί με τη δημιουργία μιας κλάσης που να υλοποιεί τις εξής δύο μεθόδους:

1) validate(self, password, user=None): πιστοποιεί έναν κωδικό και επιστρέφει **None** εάν ο κωδικός περάσει επιτυχώς τον έλεγχο ή καλεί μία εξαίρεση της κλάσης **ValidationError** προβάλλοντας κατάλληλο μήνυμα σφάλματος στον χρήστη εάν αποτύχει.

2) get_help_text(): παρέχει κείμενο το οποίο ενημερώνει το χρήστη για τις απαιτήσεις που πρέπει να ικανοποιεί ο κωδικός που θα εισάγει.

Παράδειγμα:

Σε αρχείο validators.py:

```
from django.core.exceptions import ValidationError
from django.utils.translation import ugettext as _

class MinimumLengthValidator(object):
    def __init__(self, min_length=8):
        self.min_length = min_length

    def validate(self, password, user=None):
        if len(password) < self.min_length:
```

```
            raise ValidationError(
                _("This password must contain at least
%(min_length)d characters."),
                code='password_too_short',
                params={'min_length': self.min_length},
            )
    def get_help_text(self):
        return _(
            "Your password must contain at least %(min_length)d
characters."
            % {'min_length': self.min_length}
        )
```

Προφανώς, προκειμένου ο validator να χρησιμοποιηθεί κατά τη διάρκεια αποθήκευσης νέων κωδικών ο δημιουργός της διαδικτυακής εφαρμογής πρέπει να φροντίσει να αλλάξει τη λίστα **AUTH_PASSWORD_VALIDATORS** ως εξής:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':’myapp.validators.MinimumLengthValidator,
    },
…
]
```

Εκτός από την επιλογή δημιουργίας validator, όπως γίνεται κατανοητό και από το παραπάνω παράδειγμα, ένας προγραμματιστής έχει τη δυνατότητα χρήσης μεθόδων που παρέχει το **django.contrib.auth.password_validation** σε φόρμες, συναρτήσεις, κλάσεις κ.α. της επιλογής του.

Οι μέθοδοι αυτοί είναι οι εξής:

α)validate_password(password, user=None, password_validators=None): πιστοποιεί έναν κωδικό βάσει όλων των validators που έχουν συμπεριληφθεί.

β**)password_changed(password, user=None, password_validators=None):** ενημερώνει όλους τους password validators που χρησιμοποιούνται ότι ένας κωδικός έχει αλλάξει. Μπορεί να χρησιμοποιηθεί από validator που εμποδίζει την επανάχρηση κωδικού.

γ)password_validators_help_texts(password_validators=None): επιστρέφει μία λίστα με τα κείμενα όλων των validators που επεξηγούν στο χρήστη τις απαιτήσεις που πρέπει να ικανοποιεί ο κωδικός που θα εισαχθεί.

δ)password_validators_help_text_html(password_validators=None): ειστρέφει ένα HTML string με τη λίστα των κειμένων μορφοποιημένη με χρήση της html ετικέτας <ul>.

ε**)get_password_validators(validator_config):** επιστρέφει ένα σύνολο από validators, βάσει της παραμέτρου validator_config. Μπορεί να χρησιμοποιηθεί ώστε να καθορίσει συγκεκριμένους validators για τις ανάγκες μίας συνάρτησης μιας φόρμας ή μιας συνάρτησης view,διαφορετικούς από αυτούς που ορίζονται στη λίστα **AUTH_PASSWORD_VALIDATORS [58]**.

4.3 Προστασία Από Επιθέσεις

Εκτός από τη διάθεση ενός πλήρους συστήματος αυθεντικοποίησης και εξουσιοδότητς, το Django διαθέτει και πλήθος εργαλείων που διασφαλίζουν τη προστασία της διαδικτυακής εφαρμογής από επιθέσεις και κινδύνους ασφάλειας ελαχιστοποιώντας τις ευπάθειες που περιγράφονται στο Κεφάλαιο 2. Συγκεκριμένα το Django διαθέτει πλήθος **Middleware** που παρεμβάλλονται στον κύκλο των requests και responses και απευθύνονται στο πρόβλημα αυτό.

Την ίδια στιγμή, το Django διαθέτει εργαλεία κρυπτογράφησης και υπογραφής που διασφαλίζουν την πιστότητα των δεδομένων και του αποστολέα τους.

Σε αυτή την παράγραφο γίνεται αναλυτική περιγραφή όλων των παραπάνω.

4.3.1 Security Middleware

Ένα από τα βασικά middleware που χρησιμοποιεί το Django για την προστασία μιας διαδικτυακής εφαρμογής είναι το middleware **SecurityMiddleware**. Το security middleware παρεμβαίνει στην διαδικασία request/response προσθέτοντας παραμέτρους και ελέγχους που αυξάνουν την ασφάλεια. Αυτό επιτυγχάνεται κυρίως μέσω μεταβλητών που μπορούν να οριστούν κατάλληλα στο αρχείο **settings.py [51]**. Οι δυνατότητες του Security Middleware αξιοποιούνται στις παραγράφους που ακολουθούν.

4.3.2 Προστασία από Επιθέσεις Τύπου XSS (Cross – Site Scripting)

Τα templates του Django παρέχουν προστασία από την πλειοψηφία των επιθέσεων τύπου XSS. Αναλυτικότερα, η γλώσσα template του Django,υλοποιεί αυτόματα τη διαφυγή «επικίνδυνων» χαρακτήρων HTML, όσον αφορά στις μεταβλητές template. Το γεγονός αυτό προστατεύει τη διαδικτυακή εφαρμογή από τυχόν εισαγωγή κακόβουλου κώδικα ως τιμή μιας μεταβλητής με τον ξής τρόπο: ειδικοί HTML χαρακτήρες που χρησιμοποιούνται συνήθως για την εκτέλεση κακόβουλου κώδικα, δεν λαμβάνονται υπόψη ως τέτοιοι κατά τη διάρκεια της διαδικασίας παραγωγής του HTML κώδικα από ένα template (rendering) και αντί να εκτελεστούν, αντικαθίστανται με άλλους χαρακτήρες.

4.3.2.1 Αυτόματη διαφυγή Ειδικών Χαρακτήρων HTML

Συγκεκριμένα, η διαφυγή χαρακτήρων στην γλώσσα template του Django λειτουργεί ως εξής:

```
ο χαρακτήρας < στην έξοδο  μετατρέπεται σε &lt;
ο χαρακτήρας >  στην έξοδο μετατρέπεται σε &gt;
ο χαρακτήρας ' στην έξοδο μετατρέπεται σε  &#39;
ο χαρακτήρας « στην έξοδο μετατρέπεται σε &quot;
ο χαρακτήρας & στην έξοδο μετατρέπεται σε  &amp;
```

Όπως έχει ήδη αναφερθεί, η διαδικασία διαφυγής ειδικών χαρακτήρων στο Django είναι αυτόματη. Παρόλα αυτά, υπάρχει η δυνατότητα να απενεργοποιηθεί, σε περίπτωση που αυτό απαιτείται για τη λειτουργία κάποιας εφαρμογής. Αυτό επιτυγχάνεται με δύο τρόπους:
1) με χρήση της ετικέτας template **autoescape** και την παράμετρο **off**, όταν πρόκειται για ένα ολόκληρο block κώδικα.
2) με χρήση του template φίλτρου safe, όταν πρόκειται για μία μεμονωμένη template μεταβλητή.

Παράδειγμα:

```
{% autoescape off %}
    This will not be auto-escaped: {{ data }}.

This will not be escaped: {{ data|safe }}
```

Η απενεργοποίηση της λειτουργίας αυτόματης διαφυγής δεν προτείνεται. Ιδιαίτερη προσοχή χρειάζεται στα σημεία του κώδικα που υλοποιούν την εισαγωγή δεδομένων από το χρήστη, ή την προβολή δεδομένων από την βάση (τα οποία έχει εισάγει χρήστης).

4.3.2.2 Διαφυγή Ειδικών Χαρακτήρων Άλλων Γλωσσών

Αξίζει να σημειωθεί ότι η αυτόματη διαφυγή χαρακτήρων δεν παρέχει απόλυτη προστασία για γλώσσες εκτός της HTML, μιας και οι ειδικοί χαρακτήρες και οι εντολές ανά γλώσσα διαφέρουν. Για διαφορετικές γλώσσες, απαιτείται διαφορετική υλοποίηση διαφυγής.

Παράδειγμα:

Έστω η παρακάτω γραμμή κώδικα σε ένα αρχείο template:

```
<style class={{ var }}>...</style>
```

Η εισαγωγή της τιμής '**class 1 onmouseover=javascript.func()'**, στην μεταβλητή **var**,θα είχε ως αποτέλεσμα την εκτέλεση του Javascript κώδικα κατά τη διάρκεια της διαδικασίας rendering του template.

Ένα χρήσιμο φίλτρο template που αντιμετωπίζει το προαναφερθέν πρόβλημα είναι το **escapejs**.Με την χρήση του, ειδικοί χαρακτήρες Javascript αντικαθίστανται από άλλους [63].

Παράδειγμα:

Έστω η παρακάτω γραμμή κώδικα σε ένα template:

```
{{ value|escapejs }}
```

Αν η τιμή της μεταβλητής **value** είναι το string **"testing\r\njavascript \'string" escaping"**,τότε αυτό που θα εμφανιστεί στην οθόνη μετά το rendering του template θα είναι το **"testing\\u000D\\u000Ajavascript\\u0027string\\u0022 \\u003Cb\\u003Eescaping\\u003C/b\\u003E".**

4.3.2.3 Κεφαλίδα X – XSS – Protection

Διάφοροι περιηγητές έχουν τη δυνατότητα να εμποδίζουν την εμφάνιση περιεχομένου που θα μπορούσε να αποτελέσει επίθεση τύπου XSS. Η διαδικασία που ακολουθούν είναι η εξής: πρώτα αναζητούν Javascript κώδικα σε όλες τις POST ή GET παραμέτρους μιας σελίδας.

Ύστερα, εάν διαπιστωθεί ότι υπάρχει Javascript κώδικας και αυτός επανεμφανιστεί στο response του server, αντί για τη σελίδα στο χρήστη προβάλλεται ένα σφάλμα.

Η διαχείριση της συμπεριφοράς της συγκεκριμένης προστασίας γίνεται μέσω της HTTP κεφαλίδας **XSS – Protection**. Προκειμένου το φίλτρο XSS να ενεργοποιηθεί σε έναν browser και αυτός να οδηγηθεί στο να εμποδίζει όλα τα responses που θα μπορούσαν να υποθάλψουν επιθέσεις τύπου XSS, η συγκεκριμένη κεφαλίδα πρέπει να είναι ως εξής: **X-XSS-Protection: 1; mode=block** .

Το **SecurityMiddleware** πραγματοποιεί τις κατάλληλες αλλαγές σε όλα τα responses, εάν η μεταβλητή **SECURE_BROWSER_XSS_FILTER** στο αρχείο ρυθμίσεων **settings.py** λάβει την τιμή **True [51].**

4.3.3 Προστασία από επιθέσεις Cross – Site Request Forgery (CSRF)

Το Django έχει ενσωματωμένη προστασία από τα περισσότερα είδη επιθέσεων CSRF. Το μόνο που απαιτείται από πλευράς του προγραμματιστή είναι η ενεργοποίηση και η χρήση της σε όλα τα σημεία που είναι απαραίτητο.

Η προστασία από CSRF λειτουργεί ως εξής: κάθε request που χρησιμοποιεί την μέθοδο POST ελέγχεται κρυφά πριν εκτελεστεί. Το γεγονός αυτό διασφαλίζει ότι ένας κακόβουλος χρήστης δεν μπορεί να αναπαράξει μία φόρμα POST και να οδηγήσει έναν διαπιστευμένο χρήστη να υποβάλει στοιχεία μέσω αυτής άθελά του, μιας και θα έπρεπε να γνωρίζει το μυστικό κλειδί που διαφέρει για κάθε χρήστη (και πιστοποιείται με χρήση cookie).

Σε μια εφαρμογή που γίνεται χρήση πρωτοκόλλου HTTPS,το **CsrfViewMiddleware** ελέγχει ότι τα URLS στα οποία κατευθύνονται τα POST requests, ανήκουν στο ίδιο subdomain και χρησιμοποιούν το ίδιο port με το URL στο οποίο βρίσκεται η φόρμα.

Η υλοποίηση της προστασίας από επιθέσεις τύπου CSRF στο Django, βασίζεται στα εξής βήματα:

1. Ένα CSRF cookie που βασίζεται σε μία μυστική τυχαία τιμή, στην οποία δεν έχουν πρόσβαση άλλα sites. Αυτό το cookie καθορίζεται από το **CsrfViewMiddleware**. Αποστέλεται μαζί με κάθε απάντηση (response) η οποία έχει πραγματοποιήσει την κλήση της **django.middleware.csrf.get_token()** (μιας συνάρτησης που χρησιμοποιείται ώστε να ανακτήσει το **CSRF token** και καλείται αυτόματα εφόσον έχει ενεργοποιηθεί το **CsrfViewMiddleware**), αν δεν έχει ήδη καθοριστεί στο αίτημα (request).

Για λόγους περαιτέρω προστασίας, προκειμένου να καθοριστεί το CSRF token, στην μυστική τιμή παρεμβάλλονται με τυχαίο τρόπο τμήματα μίας ακόμη τυχαίας τιμής (salt), αφού όλο το salt προστεθεί στην μυστική τιμή . Επίσης, η μυστική τιμή αλλάζει κάθε φορά που ένας χρήστης συνδέεται στην εφαρμογή.

2. Ένα κρυφό πεδό φόρμας (form field) με το όνομα **'csrfmiddlewaretoken'** δημιουργείται εντός κάθε φόρμας POST η οποία αποστέλλει δεδομένα. Η τιμή του πεδίου είναι ο συνδιασμός μυστικής τιμής και salt. Το salt αναδημιουργείται με κάθε κλήση της συνάρτησης **get_token()**, ώστε η τιμή του πεδίου **csrfmiddlewaretoken** να αλλάζει με κάθε νέο HTTP response. Το παραπάνω υλοποιείται με τη χρήση της template ετικέτας **{% csrf_token %}**.

3. Για όλα τα εισερχόμενα requests που δεν χρησιμοποιούν τις HTTP μεθόδους **GET, HEAD, OPTIONS, TRACE**, το **CsrfViewMiddleware** ελέγχει την ύπαρξη **CSRF cookie** και **CSRF token**. Επίσης, συγκρίνει την μυστική τιμή του **CSRF cookie** με την μυστική τιμή που έχει το πεδίο **csrfmiddlewaretoken** (και όχι με ολόκληρο το **CSRF token**). Αυτό δίνει τη δυνατότητα δημιουργίας διαφορετικών token, χωρίς να δημιουργείται κάποια επιπλοκή στον έλεγχο. Εάν διαπιστωθεί κάποια αναντιστοιχία στις τιμές που ελέγχονται ή η ανυπαρξία κάποιου από τα **CSRF token** και **CSRF cookie**, τότε αποστέλλεται ένα '**403 Forbidden'** response στον χρήστη που πραγματοποίησε το request.

4. Για τα HTTPS requests, πραγματοποιείται ένας επιπλέον έλεγχος (έλεγχος της HTTP κεφαλίδας **Referer**) από το **CsrfViewMiddleware**. Αυτός ο έλεγχος διασφαλίζει ότι ένα subdomain, ακόμη και αν μπορεί να δημιουργήσει ή να αλλάξει cookies για το domain μιας εφαρμογής, δεν μπορεί να αναγκάσει έναν χρήστη να αποστείλει δεδομένα στη διαδικτυακή εφαρμογή, μιας το εισερχόμενο αίτημα δεν προέρχεται από το ακριβές domain της διαδικτυακής εφαρμογής (αλλά από ένα subdomain της εφαρμογής αυτής).

Ο έλεγχος αυτός πραγματοποιείται με βάση την κεφαλίδα **Referer** που περιέχουν τα requests (και δεν πραγματοποιείται όταν η επικοινωνία γίνεαι μέσω HTTP γιατί οι συγκεκριμένες κεφαλίδες δεν είναι αρκετά ασφαλείς μέσω HTTP επικοινωνίας). Σε περίπτωση που στο αρχείο ρυθμίσεων έχει οριστεί κάποια τιμή για **το CSRF_COOKIE_DOMAIN**, τότε η κεφαλίδα **Referer** συγκρίνεται με την τιμή αυτή. Σε διαφορετική περίπτωση, η κεφαλίδα Referer συγκρίνεται με την κεφαλίδα **Host.** Το Django παρέχει επίσης τη δυνατότητα να συμπεριλαμβάνονται ως έμπιστα στον παραπάνω έλεγχο domains πέρα από το domain της εφαρμογής και το domain για το οποίο έχει δημιουργηθεί το cookie, αρκεί αυτά να αναφερθούν στη ρύθμιση **CSRF_TRUSTED_ORIGINS** στο αρχείο **settings.py**.

Σε κάθε περίπτωση, αν οι συγκρινόμενες τιμές δεν ταυτιστούν, τότε ο server επιστρέφει ένα '**403 Forbidden'** response στον χρήστη που πραγματοποίησε το HTTPS request και η αποστολή δεδομένων αποτυγχάνει.

Αξίζει να σημειωθεί, ότι λόγω του τρόπου λειτουργίας της, η προστασία από CSRF δεν προστατεύει από επιθέσεις τύπου man – in – the – middle, ενώ παίρνει ως δεδομένο ότι το site δεν έχει ευπάθειες **Cross – Site Scripting** και η κεφαλίδα Host έχει πιστοποιηθεί. Αυτό αναδεικνύει την ανάγκη αξιοποίησης των υπόλοιπων εργαλείων και μεθόδων ασφάλειας που προσφέρει το Django, όπως η χρήση πρωτοκόλλου HTTPS, η προστασία από επιθέσεις τύπου XSS κ.α.

Προκειμένου να χρησιμοποιηθεί η προστασία από CSRF πρέπει να ακολουθηθούν τα εξής βήματα:

1. Το **middleware CsrfViewMiddleware** ενεργοποιείται αυτόματα στη ρύθμιση **MIDDLEWARE** στο αρχείο ρυθμίσεων της διαδικτυακής εφαρμογής. Σε περίπτωση που αυτή η ρύθμιση παρακαμφθεί, το middleware **'django.middleware.csrf.CsrfViewMiddleware'** πρέπει να χρησιμοποιείται πριν από κάθε middleware το οποίο θεωρεί ότι οι επιθέσεις τύπου CSRF έχουν αντιμετωπιστεί. Σε περίπτωση που το middleware CSRF έχει απενεργοποιηθεί από το αρχείο ρυθμίσεων (**settings.py**), προκειμένου να εφαρμοστεί η προστασία CSRF σε ένα συγκεκριμένο view πρέπει να γίνει χρήση του decorator **@csrf_protect**.

Ο συγκεκριμένος decorator είναι απαραίτητο να χρησιμοποιηθεί και στο view το οποίο καλεί τη φόρμα στην οποία γίνεται εισαγωγή δεδομένων, αλλά και στο view το οποίο θα δεχθεί τα δεδομένα αυτά.

Παράδειγμα:

```
from django.views.decorators.csrf import csrf_protect

from django.shortcuts import render
@csrf_protect
def my_view(request):
    c = {}
    # ...
    return render(request, "a_template.html", c)
```

2. Εφόσον το **CsrfViewMiddlewar**e είναι ενεργοποιημένο,σε κάθε φόρμα που χρησιμοποιεί την μέθοδο POST για να αποστείλει τις πληροφορίες σε URL που ανήκει στο domain της διαδικτυακής εφαρμογής, πρέπει να προστεθεί η ετικέτα template **{% csrf_token %}.**

Παράδειγμα:

```
<form action = "mydomain/url" method = "post">{% csrf_token %}
```

Είναι σημαντικό η χρήση της ετικέτας να μη γίνεται εντός φορμών που αποστέλλουν δεδομένα σε URL εκτός του domain, μιας και κάτι τέτοιο θα οδηγούσε στη διαρροή του CSRF token, καθιστώντας το site ευάλωτο.

3. Στις συναρτήσεις view που καλούν τα templates που χρησιμοποιούν την **ετικέτα {% csrf_token %}**, η διαδικασία rendering πρέπει να γίνει με κλήση της κλάσης **RequestContext,** ώστε η ετικέτα **{% csrf_token %}** να μεταφραστεί σωστά. Αυτό δεν ισχύει, σε περίπτωση που γίνεται χρήση έτοιμων generic views ή του shortcut **render(),** μιας και τα παραπάνω και χρησιμοποιούν ούτως ή άλλως τη **RequestContext.**

4.3.3.1 Απόρριψη requests

Αν ένα εισερχόμενο HTTP request αποτύχει τους ελέγχους που υλοποιούνται από το **CsrfMiddleware**, τότε ένα response με κωδικό '**403 Forbidden'** αποστέλλεται στο χρήστη που πραγματοποίησε το request. Οι αποτυχίες ελέγχου καταγράφονται από το Django. Η επιστροφή ενός **response 403** μπορεί να συμβεί είτε σε περίπτωση προσπάθειας επίθεσης CSRF, είτε σε περίπτωση που ο προγραμματιστής δεν έχει προσθέσει την ετικέτα template **{% csrf_token %}** σε φόρμα που απαιτείται. Το Django προβάλλει στον χρήστη μία σελίδα που του γνωστοποιεί τον κωδικό του σφάλματος. Παρόλα αυτά, ένας προγραμματιστής έχει τη δυνατότητα να ορίσει το δικό του view που θα προβάλλει κάποιο διαφορετικό αρχείο template, μέσω της ρύθμισης **CSRF_FAILURE_VIEW** στο αρχείο **settings.py**.

4.3.3.2 Χρήση Προστασίας από Επιθέσεις CSRF στην Template Engine Jinja2

Όπως έχει περιγραφεί στο προηγούμενο κεφάλαιο, το Django παρέχει δύο template engines, την προεπιλεγμένη (default template engine), και την **Jinja2**. Σε περίπτωση που έχει επιλεγεί η δεύτερη, τότε η ετικέτα **{{ csrf_input }}** (που αντιστοιχεί στην ετικέτα **{% csrf_token %}** της προεπιλεγμένης template engine) προστίθεται σε κάθε φόρμα τύπου POST.

Παράδειγμα:

```
<form action="" method="post">{{ csrf_input }}</form>
```

4.3.3.3 Ρυθμίσεις CSRF

Ένας προγραμματιστής έχει τη δυνατότητα μέσω του αρχείου **settings.py** να προσθέσει ή να τροποιήσει ρυθμίσεις οι οποίες καθορίζουν τη συμπεριφορά της προστασίας από επιθέσεις τύπου CSRF, όπως οι παρακάτω:

α) CSRF_COOKIE_AGE: καθορίζει τον χρόνο για τον οποίο θα διατηρηθεί το cookie (σε δευτερόλεπτα).

β)CSRF_COOKIE_DOMAIN: καθορίζει το domain για το οποίο δημιουργείται και στο οποίο μπορεί να χρησιμοποιηθεί το cookie.

γ) CSRF_COOKIE_HTTPONLY: μία boolean μεταβλητή, που σε περίπτωση που έχει την τιμή **True**, εμποδίζει την πρόσβαση στο cookie με χρήση Javascript**.** Χρησιμοποιείται για να εμποδίσει κακόβουλους χρήστες να παρακάμψουν την προστασία από επιθέσεις CSRF μέσω χρήσης Javascript.

δ) CSRF_COOKIE_NAME: Το όνομα του cookie, το οποίο μπορεί να είναι οποιοδήποτε επιλέξει ο εκάστοτε προγραμματιστής.

ε) CSRF_COOKIE_PATH: Το path του cookie, που μπορεί να είναι είτε ο φάκελος στον οποίο έχει εγκατασταθεί το Django, ή ένας φάκελος που τον περιέχει.

δ) CSRF_COOKIE_SECURE: μία boolean μεταβλητή που σε περίπτωση που έχει την τιμή **True**, εξαναγκάζει τον browser να αποστείλει τα cookies μόνο μέσω HTTPS (και όχι HTTP) σύνδεσης.

στ) CSRF_FAILURE_VIEW: καθορίζει το view που θα κληθεί για να επιστρέψει το σφάλμα **403** όταν ο έλεγχος ενός request από το **CsrfViewMiddleware** αποτύχει**.**

ζ) CSRF_HEADER_NAME: καθορίζει το όνομα της κεφαλίδας του request που χρησιμοποιείται για τον έλεγχο CSRF.

η) CSRF_TRUSTED_ORIGINS: καθορίζει τους hosts που θα χρησιμοποιηθούν στον έλεγο της κεφαλίδας **Referer** πέρα από το cookie domain και το domain της εφαρμογής [62].

4.3.4 Προστασία από Επιθέσεις SQL Injection

Το προγραμματιστικό πλαίσιο Django παρέχει αυτόματα προστασία από επιθέσεις τύπου SQL Injection. Η μόνη προϋπόθεση προκειμένου αυτή η προστασία να εφαρμοστεί, είναι η αξιοποίηση του Django ORM και η χρήση των querysets και όχι raw SQL κάθε φορά που απαιτείται να εκτελεστεί ένα query στη βάση δεδομένων. Μέσω της χρήσης των querysets, το Django υλοποιεί στο παρασκήνιο αυτόματη διαφυγή ειδικών SQL χαρακτήρων στην SQL εντολή που προκύπτει από κάθε query, πριν αυτή καταλήξει στη βάση δεδομένων. Έτσι,οι ειδικοί SQL χαρακτήρες που θα μπορούσαν να χρησιμοποιηθούν για την έγχυση ανεπιθύμητου SQL κώδικα δεν μεταφράζονται ως τέτοιοι, με αποτέλεσμα οι επιθέσεις τύπου SQL Injection να καθίστανται αδύνατες. Σε περίπτωση χρήσης raw SQL (μέσω του **Manager.raw()**) ή απευθείας εκτέλεσης SQL, χρειάζεται ειδική μέριμνα από τον ίδιο τον προγραμματιστή (όπως χρήση prepared statements) προκειμένου το site να μην είναι ευάλωτο σε τέτοιου είδους επιθέσεις [63].

4.3.5 Προστασία από Επιθέσεις Clickjacking

Οι σύγχρονοι περιηγητές έχουν τη δυνατότητα αναγνώρισης μιας HTTP κεφαλίδας με το όνομα **X-Frame-Options**.Η συγκεκριμένη κεφαλίδα καθορίζει αν τμήμα κώδικα μπορεί να φορτωθεί και να προβληθεί στα πλαίσια ενός HTML frame ή iframe. Κατά τη διάρκεια επιστροφής ενός HTTP response, αν η HTTP κεφαλίδα **X – Frame – Options** έχει την τιμή **'SAMEORIGIN'**, τότε ο browser θα προβάλλει τον κώδικα αν το request έχει πραγματοποιηθεί από το ίδιο site (στο οποίο έχει πραγματοποιηθεί το request). Αν η **κεφαλίδα X – Frame – Options** έχει την τιμή **'DENY'** τότε ο browser δεν θα επιτρέψει στον κώδικα να φορτωθεί, ανεξαρτήτως του site στο οποίο έχει πραγματοποιηθεί το request.

Το Django παρέχει σε έναν προγραμματιστή δύο βασικούς τρόπους που του επιτρέπουν να καθορίσει την τιμή αυτής της κεφαλίδας:

1) Με χρήση του middleware **XFrameOptionsMiddleware** το οποίο καθορίζει την τιμή της κεφαλίδας για όλα τα responses.

2) Με χρήση decorators οι οποίοι μπορούν να ορίσουν την τιμή της κεφαλίδας για responses που θα προκύψουν από συγκεκριμένες συναρτήσεις view, παρακάμπτοντας το middleware [64].

4.3.5.1 Ορισμός Κεφαλίδας X – Frame – Options για Όλα τα responses

Προκειμένου να οριστεί η κεφαλίδα X – Frame – Options για όλα τα responses, στο αρχείο των ρυθμίσεων της διαδικτυακής εφαρμογής (**settings.py**) πρέπει να προστεθεί η γραμμή **django.middleware.clickjacking.XFrameOptionsMiddleware** στη λίστα **MIDDLEWARE**.

Παράδειγμα:

```
MIDDLEWARE = [

    ...
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ...
```

```
]
```

Με αυτό τον τρόπο το middleware ενεργοποιείται. Κατά την έναρξη της δημιουργίας ενός site, το Django ενεργοποιεί το συγκεκριμένο middleware αυτόματα. Η τιμή της **κεφαλίδας X – Frame – Options** καθορίζεται από την μεταβλητή **X_FRAME_OPTIONS**. Αυτόματα λαμβάνει την τιμή **'SAMEORIGIN'**, αλλά ένας προγραμματιστής μπορεί να αλλάξει την τιμή της σε **'DENY'**, προσθέτοντας την παρακάτω γραμμή κώδικα στο αρχείο **settings.py**:

```
X_FRAME_OPTIONS = 'DENY'
```

Σε περίπτωση που για οποιοδήποτε λόγο ένας προγραμματιστής επιθυμεί να δοθεί η δυνατότητα στον browser να προβάλλει κώδικα από sites εκτός του domain της εφαρμογής όταν λαμβάνει συγκεριμένα requests, το μόνο που απαιτείται είναι να προσθέσει τον decorator **@x_frame_options_exempt** πάνω από τη view που διαχειρίζεται τα συγκεκριμένα requests, όπως φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα:

```
from django.http import HttpResponse
from           django.views.decorators.clickjacking           import
xframe_options_exempt

@xframe_options_exempt
def ok_to_load_in_a_frame(request):
    return HttpResponse("This page is safe to load in a frame on any
site.")
```

4.3.5.2 Ορισμός Κεφαλίδας X – Frame – Options για μια Συνάρτηση view

Το Django παρέχει decorators που μπορούν να χρησιμοποιηθούν για να καθορίσουν την τιμή της κεφαλίδας **X – Frame – Options** (σε '**DENY'** ή '**SAMEORIGIN'**) για requests τα οποία διαχειρίζονται συγκεκριμένες συναρτήσεις view. Το παρακάτω παράδειγμα δείχνει πως γίνεται αυτό.

Παράδειγμα:

```
from django.http import HttpResponse
from django.views.decorators.clickjacking import xframe_options_deny
from           django.views.decorators.clickjacking           import
xframe_options_sameorigin

@xframe_options_deny
def view_one(request):
    return HttpResponse("I won't display in any frame!")

@xframe_options_sameorigin
def view_two(request):
    return HttpResponse("Display in a frame if it's from the same
```

```
origin as me.")
```

Αυτό που απαιτεί ιδιαίτερη προσοχή είναι ότι όπως αναφέρθηκε και νωρίτερα, η κεφαλίδα **X – Frame – Options** υποστηρίζεται μόνο από σύγχρονους browsers. Συγκεκριμένα, οι εκδόσεις των browsers που υποστηρίζουν τη συγκεκριμένη κεφαλίδα είναι οι: **Internet Explorer 8+, Firefox 3.6.9+, Opera 10.5+, Safari 4+, Chrome 4.1+,Edge,Servo,Android Browser, Chrome for Android, Edge Mobile, Firefox for Android, Internet Explorer Mobile, Safari Mobile [51],[63].**

Για παλαιότερες εκδόσεις browser,χρειάζεται να υλοποιηθούν διαφορετικές τεχνικές προστασίας από επιθέσεις Clickjacking από τον ίδιο τον προγραμματιστή.

4.3.6 Προστασία από HTTP Header Attacks

Το Django κάνει χρήση της HTTP κεφαλίδας **Host** που υποδεικνύει τον host στον οποίο απευθύνεται ένα request σε πολλές περιπτώσεις. Προκειμένου να αποτραπεί η δυνατότητα εισαγωγής ψεύτικης τιμής στην κεφαλίδα Host με σκοπό την εκτέλεση επίθεσης, το Django παρέχει τον εξής τρόπο προστασίας: όταν γίνεται χρήση της μεθόδου **Django.http.HttpRequest.get_host()**, το Django διαπιστώνει εάν η κεφαλίδα Host είναι κάποια από αυτές που περιέχονται στην μεταβλητή **ALLOWED_HOSTS** (που ορίζεται στο αρχείο **settings.py**). Η μεταβλητή αυτή είναι μία λίστα από strings που είναι τα ονόματα των hosts που μπορούν να στείλουν request. Εάν ο host δεν ταυτοποιηθεί με κάποιον από τη λίστα, τότε η μέθοδος **get_host()** επιστρέφει την ένα instance του αντικειμένου εξαίρεσης **SuspiciousOperation** (και συγκεκριμένα της υποκλάσης του **SuspiciousOperation, DisallowedHost**).

Σε περίπτωση που γίνεται χρήση proxy server, η μεταβλητή **USE_X_FORWARED_HOST** χρειάζεται να λάβει την τιμή **True** ώστε να λαμβάνεται υπόψη η κεφαλίδα **X-Forwarded-Host** (την οποία ορίζει κατάλληλα ο proxy από το εισερχόμενο request) και όχι η κεφαλίδα **Host** για τον έλεγχο.

Αξίζει να σημειωθεί ότι αυτός ο τρόπος προστασίας εφαρμόζεται μόνο στη περίπτωση που η τιμή της κεφαλίδας Host ανακτάται μέσω της μεθόδου **get_host()**. Σε περίπτωση που ανακτηθεί με άλλο τρόπο, όπως με χρήση του αντικειμένου **request.META** (αντικείμενο που περιέχει όλες τις πληροφορίες που σχετίζονται με ένα HTTP request) τότε η προστασία αυτή παρακάμπτεται [63].

Παράδειγμα:

```
ALLOWED_HOSTS=['www.example1.com','.example2.com']
```

Στο παραπάνω παράδειγμα, το Django συγκρίνει την τιμή της κεφαλίδας Host των εισερχόμενων requests με το domain **www.example1.com** ή το **www.example2.com** καθώς και όλα τα subdomains του **example2.com** (πχ. **mail.example2.com**). –

4.4 Πρωτόκολλο HTTPS

Η χρήση πρωτοκόλλου HTTPS για την επικοινωνία με ένα site είναι απαραίτητη προκειμένου να ελαχιστοποιηθεί η πιθανότητα υποκλοπής στοιχείων αυθεντικοποίησης των χρηστών ή άλλων εμπιστευτικών πληροφοριών από κακόβουλους χρήστες.

Η χρήση του HTTPS γίνεται μέσω της ενεργοποίησής του στον server που χρησιμοποιεί η εφαρμογή (και ο τρόπος ενεργοποίησης διαφέρει ανάλογα με τον server). Το Django παρέχει ρυθμίσεις οι οποίες προσφέρουν περαιτέρω προστασία κατά τη διάρκεια χρήσης HTTPS.Οι ρυθμίσεις αυτές μπορούν να αξιοποιηθούν από έναν προγραμματιστή με πρόσθεσή τους στο αρχείο ρυθμίσεων **settings.py [63]**.

4.4.1 Χρήση SECURE_PROXY_SSL_HEADER

Η μεταβλητή αυτή είναι χρήσιμη σε περίπτωση που η διαδικτυακή εφαρμογή κάνει χρήση proxy server, προκειμένου να διασφαλιστεί η σωστή εκτέλεση της μεθόδου **is_secure()** που χρησιμοποιεί το Django προκειμένου να διαπιστώσει εάν ένα εισερχόμενο request είναι ασφαλές.

Η μέθοδος **is_secure()** λειτουργεί ως εξής: κάθε φορά που πραγματοποιείται ένα request, ελέγχεται εάν αυτό χρησιμοποιεί HTTPS, έαν δηλαδή το URL από το οποίο προέρχεται αρχίζει με το πρόθεμα **"https://"**. Σε περίπτωση που αυτό ισχύει, η μέθοδος **is_secure()** επιστρέφει την τιμή **True**, ενώ σε διαφορετική περίπτωση επιστρέφει την τιμή **False**. Η χρήση ενός proxy server, θα μπορούσε να οδογήσει την μέθοδο **is_secure()** να επιστρέφει συνεχώς **False** , μιας και σε κάθε request η αρχική επικοινωνία μεταξύ του proxy server και του Django θα γινόταν μέσω HTTP.

Προκειμένου να μην αντιμετωπιστεί το παραπάνω πρόβλημα, ένας προγραμματιστής χρειάζεται να ορίσει μία HTTP header την οποία θα ελέγχει το Django (και διαφέρει ανάλογα με τον proxy) προκειμένου να διαπιστώσει εάν το request έφτασε στον proxy server μέσω HTTPS σύνδεσης.
Η μεταβλητή **SECURE_PROXY_SSL_HEADER** ουσιαστικά ορίζει την κεφαλίδα την οποία πρέπει να ελέγξει το Django προκειμένου να διαπιστώσει εάν τα request στον proxy server χρησιμοιούν HTTPS [63],[65].

Παράδειγμα:

```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

Στο παραπάνω παράδειγμα, το Django θα ελέγχει την κεφαλίδα **X-Forwarded-Proto** σε κάθε request που έρχεται από τον proxy server και κάθε φορά που η τιμή της κεφαλίδας είναι '**https'**, θα θεωρεί ότι η επικοινωνία γίνεται μέσω HTTPS άρα το request είναι ασφαλές.
Αξίζει να σημειωθεί, ότι προκειμένου να γίνει η χρήση της μεταβλητής **SECURE_PROXY_SSL_HEADER** από έναν προγραμματιστή, πρέπει να ισχύουν τα εξής:
1) Ο proxy server που χρησιμοποιεί η διαδικτυακή εφαρμογή πρέπει να ελέγχεται από τον προγραμμα τιστή.

2) ο proxy server πρέπει να αφαιρεί την κεφαλίδα X-Forwarded-Proto ααπό τα εισερχόμενα requests.

3) ο proxy server πρέπει να είναι ρυθμισμένος κατάλληλα ώστε η τιμή **'https'** στην **κεφαλίδα X – Forwarded – Proto** να χρησιμοποιείται μόνο σε περίπτωση που τα εισερχόμενα requests χρησιμοποιούν HTTPS.

4.4.2 Χρήση SECURE_SSL_REDIRECT

Η τοποθέτηση της τιμής **True** σε αυτή την μεταβλητή ανακατευθύνει όλα τα HTTP requests σε HTTPS σύνδεση [63].

4.4.3 Ενεργοποίηση των Ασφαλών cookies

Η αρχική προσπάθεια ενός browser να συνδεθεί μέσω HTTP μπορεί να οδηγήσει σε διαρροή των ήδη υπάρχοντων cookies. Για αυτό το λόγο, το Django παρέχει τις μεταβλητές **SESSION_COOKIE_SECURE** και **CSRF_COOKIE_SECURE**. Η τοποθέτηση της τιμής **True** και στις δύο στο αρχείο των ρυθμίσεω **settings.py**, απαγορεύει στον browser να στείλει τα cookies μέσω HTTP. Αυτό σημαίνει ότι οι σύνοδοι δεν θα λειτουργούν ομαλά μέσω HTTP σύνδεσης και η προστασία από επιθέσεις CSRF θα αποτρέψει την αποστολή δεδομένων μέσω φορμών POST μέσω HTTP [63].

4.4.4 Χρήση HTTP Strict Transport Security (HSTS)

Η **HSTS** είναι μία HTTP κεφαλίδα η οποία ενημερώνει τον browser ότι οποιαδήποτε μελλοντική σύνδεση σε ένα συγκεκριμένο site πρέπει να πραγματοποιηθεί μέσω HTTPS. Σε συνδυασμό με τη διασφάλιση ανακατεύθυνσης όλων των HTTP αιτημάτων σε HTTPS, η κεφαλίδα αυτή διασφαλίζει τη χρήση SSL σε κάθε μορφής σύνδεση και επικοινωνία από και προς μια διαδικτυακή εφαρμογή.

Μέσω της μεταβλητής **SECURE_HSTS_SECONDS**, το Django ενεργοποιεί την κεφαλίδα για όλα τα HTTP responses για χρονικό διάστημα που ισούται με την τιμή της (σε δευτερόλεπτα).

Παράδειγμα:

```
SECURE_HSTS_SECONDS = 3600
```

Στο παραπάνω παράδειγμα η επικοινωνία πρέπει να γίνεται υποχρεωτικά μέσω HTTPS για μία ώρα.

Μέσω της μεταβλητής **SECURE_HSTS_INCLUDE_SUBDOMAINS**, το Django δίνει το προγραμματιστή την δυνατότητα να προσθέσει στην κεφαλίδα **HSTS** όλα τα subdomains του domain της διαδικτυακής εφαρμογής. Για να γίνει αυτό, αρκεί να δοθεί η τιμή **True** στη

συγκεκριμένη μεταβλητή [63].

Παράδειγμα:

```
SECURE_HSTS_SECONDS =  True
```

Στο παραπάνω παράδειγμα η σύνδεση σε οποιοδήποτε subdomain της εφαρμογής μέσω HTTP εμποδίζεται για όσο χρόνο ορίζει η μεταβλητή **SECURE_HSTS_SECONDS.**

4.5 Ασφαλής Διαχείριση Συνόδων

Ένα βασικό ζήτημα για τη λειτουργία αλλά και την ασφάλεια μιας διαδικτυακής εφαρμογής στο οποίο απευθύνεται το Django είναι η διαχείριση των συνόδων. Οι σύνοδοι ταυτοποιούν και καταγράφουν τις δραστηριότητες των χρηστών. Η γνώση του τρόπου λειτουργίας, αποθήκευσης και συμπεριφοράς των συνόδων για έναν προγραμματιστή είναι σημαντική προκειμένου να αποφευχθούν ευπάθειες ασφάλειας.

Το Django παρέχει ποικίλους τρόπους διαχείρισης και αποθήκευσης των συνόδων.

4.5.1 Ενεργοποίηση Συνόδων

Η λειτουργία που σχετίζεται με τις συνόδους (**sessions**) στο Django υλοποιείται μέσω του middleware **SessionMiddleware**. Προκειμένου να ενεργοποιηθεί το συγκκειρμένο middleware, στο αρχείο ρυθμίσεων **settings.py** στην λίστα **MIDDLEWARE** χρειάζεται να προστεθεί η γραμμή **'django.contrib.sessions.middleware.SessionMiddleware'**.Αυτό γίνεται αυτόματα με την έναρξη ενός νέου Django project.

Σε περίπτωση που ένας προγραμματιστής δεν επιθυμεί να χρησιμοποιήσει συνόδους, μπορεί να αφαιρέσει την γραμμή που αναφέρεται προηγουμένως από τη λίστα **MIDDLEWARE**,όπως και τη γραμμή από τη λίστα **INSTALLED_APPS**.

Το Django δίνει την δυνατότητα στον προγραμματιστεί να επιλέξει το μέρος στο οποίο θα αποθηκεύονται τα στοιχεία της συνόδου μέσω της μεταβλητής **SESSION_ENGINE**. Ο τρόπος με τον οποίο συμβαίνει αυτό περιγράφεται παρακάτω.

4.5.1.1 Datatabase – Backed Sessions

Εφόσον η εφαρμογή sessions είναι ενεργοποιημένη (δηλαδή το string **'django.contrib.sessions'** υπάρχει στη λίστα **INSTALED_APPS**, το Django θα αποθηκεύει τα στοιχεία των συνόδων στη βάση δεδομένων με χρήστη του model **django.contrib.sessions.models.Session**. Το Django έχει ως προεπιλογή την αποθήκευση των συνόδων στη βάση δεδομένων. Με την έναρξη ενός νέου project,μόλις πληκτρολογηθεί η εντολή **python manage.py migrate**,το Django δημιουργεί αυτόματα τον πίνακα της βάσης δεδομένων που αντιστοιχεί στο model **Session.**

4.5.1.2 Cached Sessions

Ένας προγραμματιστής έχει τη δυνατότητα να χρησιμοποιήσει την προσωρινή μνήμη cache (αφού αυτή οριστεί με χρήση της λίστας **CACHES** στο αρχείο **settings.py**) προκειμένου να αποθηκεύσει τα στοιχεία που σχετίζονται με τις συνόδους με δύο τρόπους:

1) Με χρήση της γραμμής '**django.contrib.sessions.backends.cache'** ως τιμή της μεταβλητής **SESSION_ENGINE** στο αρχείο **settings.py**. Αυτό θα οδηγήσει στην απλή αποθήκευση των συνόδων στο cache που έχει οριστεί, με αποτέλεσμα τα στοιχεία των συνόδων να διαγράφονται όταν το cache γεμίσει ή όταν ο server στον οποίο φιλοξενείται **επανεκκινηθεί**.

2) Με χρήση της **'django.contrib.sessions.bakends.cached_db'** ως τιμή της μεταβλητής **SESSION_ENGINE** στο αρχείο **settings.py**. Αυτό θα οδηγήσει στην αποθήκευση των στοιχείων των συνόδων στο cache αλλά και στη βάση δεδομένων. Με αυτό τον τρόπο, το Django θα ανατρέχει στη βάση να αναζητήσει τα στοιχεία μιας συνόδου εφόσον αυτά δεν βρίσκονται στο cache, δίνοντας έτσι τη δυνατότητα μόνιμης αποθήκευσης των συνόδων. Η χρήση αυτού του τρόπου αποθήκευσης προϋποθέτει την ύπαρξη της γραμμής **'django.contrib.sessions'** στη λίστα **INSTALLED_APPS** στο αρχείο **settings.py [66]**.

4.5.1.3 File – Based Sessions

Προκειμένου οι σύνοδοι να αποθηκεύονται σε μορφή αρχείων, αρκεί η μεταβλητή **SESSION_ENGINE** να λάβει την τιμή '**django.contrib.sessions.backends.file'**. Η επιλογή του φακέλου στον οποίο θα γίνεται η αποθήκευση των συνόδων γίνεται μέσω εισαγωγής του string του path του φακέλου στη μεταβλητή **SESSION_FILE_PATH** στο αρχείο **settings.py**. Η προεπιλεγμένη τιμή της μεταβλητής είναι η διαδρομή που προκύπτει ως αποτέλεσμα της μεθόδου **tempfile.gettempdir()** (συνήθως είναι ο φάκελος **/tmp**).

4.5.1.4 Cookie – Based Sessions

Προκειμένου τα δεδομένα των συνόδων να αποθηκεύονται στα cookies, η μεταβλητή **SESSION_ENGINE** πρέπει να λάβει την τιμή **'django.contrib.sessions.backends.signed_cookies'**.

Τα δεδομένα θα αποθηκεύονται υπογεγραμμένα με χρήση των εργαλείων του Django για παραγωγή κρυπτογραφημένης υπογραφής και αξιοποίηση της μεταβλητής **SECRET_KEY**.
Σε περίπτωση χρήσης **cookie – based sessions**, απαιτείται προσοχή στα παρακάτω:

1) Σε περίπτωση διαρροής του κλειδιού, ένας κακόβουλος χρήστης μπορεί να δημιουργήσει ψευδή στοιχεία συνόδου και να οδηγήσει το site στο να θεωρήσει ότι είναι αυθεντικά. Επίσης συγκεκριμένο serializer για το serialization της συνόδου, το **Pickleserializer,** μαζί με χρήση των **cookie – based sessions** δημιουργεί ευπάθεια με εκμετάλλευση της οποίας μπορεί να εκτελεστεί ανεπιθύμητος κώδικας.

2) Το γεγονός ότι τα δεδομένα συνόδου υπογράφονται, δε σημαίνει ότι τα δεδομένα κρυπτογραφούνται. Από αυτό προκύπτει ότι τα δεδομένα συνόδου μπορούν να διαβαστούν από τον client. Ένας ειδικός κώδικας, ο **MAC (Message Authentication Code)** χρησιμοποιείται προκειμένου να μην είναι δυνατή η τροποποίηση των δεδομένων των συνόδων από τους

clients: σε περίπτωση που τα δεδομένα αλλοιωθούν από πλευράς του χρήστη, τότε τα δεδομένα δεν πιστοποιούνται (με αποτέλεσμα να «χαθεί» η τρέχουσα σύνοδος).

Το ίδιο συμβαίνει, όμως, και όταν ένας client δεν μπορεί να αποθηκεύσει το **session cookie** λόγω μεγέθους, κάτι που είναι πιθανό να συμβεί (παρόλη τη συμπίεση που πραγματοποιεί το Django) όταν τα δεδομένα της συνόδου αποθηκεύονται σε cookies. Συνεπώς, η χρήση **cookie – based sessions**, μπορεί να οδηγήσει σε προβλήματα διατήρησης της συνόδου.

3) Παρόλο που ο κώδικας MAC διασφαλίζει την αυθεντικότητα των δεδομένων συνόδου και την ακεραιότητά τους, δεν υπάρχει τρόπος σύγκρισης των δεδομένων της συνόδου που έχει ένας client με τα δεδομένα της συνόδου που έχει ο server. Λόγω του ότι οι αλλαγές των δεδομένων δεν καταγράφονται στον server, αλλά στον client, η σύνοδος δεν λήγει όταν ένας χρήστης αποσυνδεθεί από την εφαρμογή.

Έτσι, αν ένας κακόβουλος χρήστης καταφέρει να υποκλέψει το cookie ενός διαπιστευμένου χρήστη, μπορεί να καταφέρει να συνδεθεί στην εφαρμογή (ή να να θεωρηθεί συνδεδεμένος απλώς με χρήση του cookie), ακόμη και αν ο χρήστης στον οποίο ανήκει η σύνοδος είναι αποσυνδεδεμένος. Τα cookies διαγράφονται μόνο εάν ξεπεραστεί το χρονικό διάστημα το οποίο έχει οριστεί στην μεταβλητή **SESSION_COOKIE_AGE** στο αρχείο **settings.py**.

4.5.2 Χρήση Συνόδων σε Views

Εφόσον το middleware **SessionMiddleware** έχει ενεργοποιηθεί, κάθε αντικείμενο της κλάσης **HttpRequest** αποκτά ένα attribute με όνομα **session**, το οποίο είναι ένα αντικείμενο που συμπεριφέρεται σαν Python dictionary. Το αντικείμενο αυτό μπορεί να ανακτηθεί ή να αλλάξει οποιαδήποτε στιγμή σε μια συνάρτηση view γράφοντας **request.session**.

Το αντικείμενο **request.session** (που βασίζεται στην κλάση **backends.base.SessionBase**) επιδέχεται μία σειρά μεθόδων, όπως γνωστές μεθόδους των Python dictionaries **__getitem__, __setitem__,keys(),items()** αλλά και άλλες μεθόδους που εξυπηρετούν τη διαχείριση συνόδων όπως τις **set_expiry(), get_expiry_date(),clear_expired()**.

4.5.3 Χρήση Συνόδων Εκτός View

Το Django παρέχει την δυνατότητα διαχείρισης των συνόδων εκτός των view συναρτήσεων (πχ από command prompt). Για τον σκοπό αυτό γίνεται αξιοποίηση αντικειμένων της κλάσης **django.contrib.sessions.backends.db.SessionStore**, που διαθέτει μεθόδους όπως **create(), delete()** κ.ο.κ. Το παρακάτω παράδειγμα δείχνει πώς μπορεί να συμβεί αυτό.

Παράδειγμα:

```
>>> from django.contrib.sessions.backends.db import SessionStore
>>> s = SessionStore()
>>> # stored as seconds since epoch since datetimes are not
serializable in JSON.
>>> s['last_login'] = 1376587691
>>> s.create()
```

```
>>> s.session_key
'2b1189a188b44ad18c35e113ac6ceead'
>>> s = SessionStore(session_key='2b1189a188b44ad18c35e113ac6ceead')
>>> s['last_login']
1376587691
```

4.5.4 Session Serialization

Το Django, ως προεπιλογή, κάνει χρήση JSON προκειμένου να κάνει serialize τα δεδομένα των συνόδων. Με χρήση της μεταβλητής **SESSION_SERIALIZER** ένας προγραμματιστής μπορεί να επιλέξει κάποιον serializer από αυτούς που προσφέρει το Django (**JSOnSerializer, PickleSerializer**) ή να υλοποιήσει τον δικό του.

4.5.5 Αποθήκευση Συνόδων

Το Django αποθηκεύει αυτόματα τα δεδομένα μιας συνόδου (με τον τρόπο που έχει καθοριστεί από το **SESSION_ENGINE**) κάθε φορά που υπάρχει κάποια αλλαγή σε αυτά, δηλαδή όταν ένα key της συνόδου λαμβάνει διαφορετική τιμή. Ένας προγραμματιστής έχει τη δυνατότητα να αλλάξει αυτή τη συμπεριφορά, με χρήση της μεταβλητής **SESSION_SAVE_EVERY_REQUEST**. Τοποθετώντας την τιμή **True** σε αυτή, η σύνοδος αποθηκεύεται με την πραγματοποίηση κάθε νέου request.

4.5.6 Λήξη Συνόδων

Ένας προγραμματιστής έχει τη δυνατότητα να επιλέγει πότε θα λήγει μία σύνοδος. Μπορεί να επιλέξει εάν οι σύνοδοι θα είναι μόνιμοι ή θα διαρκούν όσο παραμένοι ανοικτός ο browser του χρήστη.

Προκειμένου να συμβεί το τελευταίο, η μεταβλητή **SESSION_EXPIRE_AT_BROWSER_CLOSE** χρειάζεται να λάβει την τιμή **True** στο αρχείο **settings.py**. Εάν η μεταβλητή **SESSION_EXPIRE_AT_BROWSER_CLOSE** έχει την τιμή **False**, τότε μία σύνοδος λήγει μετά από όσα δευτερόλεπτα αναγράφονται στην μεταβλητή **SESSION_COOKIE_AGE**. Αξίζει να σημειωθεί, ότι ορισμένοι browsers (όπως ο Google Chrome) παρέχουν ρυθμίσεις που επιτρέπουν στους χρήστες να διατηρήσουν τη σύνοδό τους ενεργή ακόμη και σε περίπτωση που κλείσουν τον browser. Σε αυτή την περίπτωση η ρύθμιση **SESSION_EXPIRE_AT_BROWSER_CLOSE** θα παρακαμθεί, και, επομένως, ο ορισμός κατάλληλης τιμής στην μεταβλητή **SESSION_COOKIE_AGE** έχει εξαιρετική σημασία.

Παράδειγμα:

Στο αρχείο settings.py υπάρχει ο παρακάτω κώδικας:

```
SESSION_EXPIRE_AT_BROWSER_CLOSE = False
SESSION_COOKIE_AGE = 1209600
```

Στο παραπάνω παράδειγμα, οι σύνοδοι στη διαδικτυακή εφαρμογή θα λήγουν κάθε δύο εβδομάδες.

4.5.7 Διαγραφή Δεδομένων Συνόδων

Η διατήρηση των δεδομένων συνόδων στον server διαφέρει ανάλογα με τον τρόπο αποθήκευσής τους. Σε περίπτωση που τα δεδομένα δεν αποθηκεύονται μόνο στο cache ή στα cookies (τα οποία αποθηκεύονται στον browser του χρήστη), τα στοιχεία θα διατηρούται αποθηκευμένα ακόμη και αν αυτές λήξουν. Το Django δεν προσφέρει κάποιον αυτοματοποιημένο τρόπο προκειμένου τα δεδομένα αυτά να διαγράφονται.

Ένας διαχειριστής μιας διαδικτυακής εφαρμογής, προκειμένου να ελαχιστοποιήσει την πιθανότητα της ανάκτησης των δεδομένων από κακόβουλους χρήστες, πρέπει να φροντίσει για τη διαγραφή τους σε συχνή βάση εκτελώντας την εντολή **clearsessions** (η οποία διαγράφει τα δεδομένα όλων των συνόδων ανεξάρτητα από την τοποθεσία που αποθηκεύονται) από το command prompt.

4.5.8 Ρυθμίσεις Συνόδων

Προκειμένου να καθοριστεί ο τρόπος διαχείρισης των συνόδων, η χρήση της εφαρμογής **django.contrib.sessions** επιτρέπει στον δημουργό μιας διαδικτυακής εφαρμογής τη χρήση ποικίλων μεταβλητών που σχετίζονται με αυτές. Οι πιο βασικές από αυτές έχουν ήδη αναφερθεί παραπάνω.

Οι υπόλοιπες είναι οι εξής :

α)SESSION_CACHE_ALIAS: σε περίπτωση αποθήκευσης των συνόδων στο cache και ορισμού παραπάνω από ένα caches στη ρύθμιση **CACHES**, αυτή η μεταβλητή ορίζει ποιό cache θα χρησιμοποιηθεί για την αποθήκευση των δεδομένων συνόδου.

β)SESSION_COOKIE_DOMAIN: καθορίζει το domain που χρησιμοποιούν τα session cookies.

γ)SESSION_COOKIE_HTTPONLY: μία boolean μεταβλητή, που σε περίπτωση που έχει την τιμή **True**, τα **session cookies** αποκτούν έναν δείκτη **HTTPonly**. Αυτός ο δείκτης καθιστά αδύνατη την ανάκτηση ή χρήση των cookies από κώδικα Javascript. Έτσι επιθέσεις τύπου XSS δεν μπορούν να χρησιμοποιηθούν προκειμένου να υποκλέψουν τη σύνοδο ενός χρήστη.

δ)SESSION_COOKIE_NAME: καθορίζει το όνομα των session cookies. Η προεπιλεγμένη τιμή της μεταβλητής είναι '**sessionid'**.

ε**)SESSION_COOKIE_PATH:** καθορίζει το path που θα χρησιμοποιούν τα cookies. Η τιμή της μεταβλητής χρειάζεται να ταυτίζεται με την τοποθεσία του φακέλου στον οποίο έχει εγκατασταθεί το Django, ή με κάποια τοποθεσία που περιέχει τον φάκελο αυτό.

στ)SESSION_COOKIE_SECURE: μία boolean μεταβλητή, που όπως αναφέρθηκε σε προηγούμενη παράγραφο, σε περίπτωση που είναι **True**, επιτρέπει την αποστολή δεδομένων

των session cookies μόνο μέσω HTTPS. Είναι σαφές ότι η τιμή **True** πρέπει να εισαχθεί μόνο σε περίπτωση που γίνεται χρήση HTTPS σύνδεσης.

Αξίζει να σημειωθεί ότι τα subdomains ενός domain μιας διαδικτυακής εφαρμογής έχουν την δυνατότητα να ορίσουν cookies για κάποιον χρήστη, τα οποία να ισχύουν σε ολόκληρο το domain. Το γεγονός αυτό, μπορεί να δημιουργήσει ευπάθεια που καθιστά δυνατές επιθέσεις τύπου Session Fixation, εάν το domain δεχθεί ως έγκυρα cookies τα οποία προέκυψαν από subdomain που ελέγχεται από κακόβουλο χρήστη. Έστω, για παράδειγμα, ότι ένας κακόβουλος χρήστης συνδέεται στο **a.domain.com**. Αν έχει καταφέρει να αποκτήσει τον έλεγχο του **b.domain.com**, τότε μπορεί να το χρησιμοποιήσει ώστε να στείλει το session key στο θύμα. Όταν το θύμα επισκεφθεί το **a.domain.com**, τότε θα συνδεθεί με τα στοιχεία του κακόβουλου χρήστη, και οποιοδήποτε προσωπικό δεδομένο καταχωρήσει θα είναι προσβάσιμο από αυτόν.

Ο ορισμός των domain τα οποία μπορούν να χρησιμοποιήσουν τα cookies στην μεταβλητή **SESSION_COOKIE_DOMAIN** πρέπει να γίνει με ιδιαίτερη προσοχή [66].

4.6 Η Κρυπτογραφία στο Django

4.6.1 SECRET_KEY

Κατά τη διάρκεια έναρξης ενός project,το Django δημιουργεί ένα τυχαίο κλειδί, το οποίο είναι μοναδικό για κάθε εφαρμογή και αποθηκεύεται στην τιμή **SECRET_KEY** στο αρχείο **settings.py**. Η τιμή του **SECRET_KEY** χρησιμοποιείται για την κρυπτογραφημένη υπογραφή δεδομένων,για τις συνόδους σε περίπτωση που χρησιμοποιείται οποιοδήποτε backend εκτός του **django.contrib.sessions.backed.cache**, καθώς και για τα **password_reset() tokens**.

Το **SECRET_KEY** είναι ουσιαστικά το κλειδί το οποίο πιστοποιεί την ταυτότητα ενός site στους χρήστες του. Η προστασία του είναι ζωτικής σημασίας για την ασφάλεια μιας διαδικτυακής εφαρμογής, μιας και η απόκτησή του από οποιοδήποτε χρήστη, μπορεί να οδηγήσει σε ανεπιθύμητη αναβάθμιση των δικαιωμάτων του, πρόσβαση σε εμπιστευτικά δεδομένα και δυνατότητα εκτέλεσης επιθέσεων [65].

4.6.2 Κρυπτογραφημένη Υπογραφή

Κατά τη διάρκεια της λειτουργίας μιας διαδικτυακής εφαρμογής, μπορεί να γίνει χρήση μη – διαπιστευμένων μέσων για διοχέυτεση δεδομένων και πληροφοριών. Η προσθήκη κρυπτογραφημένης υπογραφής στα δεδομένα είναι αυτή που διασφαλίζει ότι τα δεδομένα αυτά δεν έχουν υποστεί κάποια ανεπιθύμητη επεξεργασία (αλλοίωση από κακόβουλους χρήστες κ.α.), μιας και η οποιαδήποτε αλλαγή στις τιμές τους θα επιφέρει και αλλαγή της υπογραφής. Το Django παρέχει APIs που υπογράφουν και αναγνωρίζουν υπογεγραμμένα δεδομένα (cookies κ.α.).

Τα εργαλεία αυτά μπορεί να χρησιμοποιηθούν σε μια σειρά από λειτουργίες του Django, όπως τη δημιουργία URL που χρησιμοποιούνται για την αλλαγή κωδικού σε χρήστες που τον

έχουν ξεχάσει, διασφάλιση μη – αλλοίωσης των δεδομένων σε κρυφά πεδία φορμών κ.α.

Οι μέθοδοι που διαθέτει το Django για την υπογραφή δεδομένων βρίσκονται στο **python module django.core.signing.**

4.6.2.1 Υπογραφή μιας Τιμής

Προκειμένου ένας προγραμματιστής να υπογράψει μια τιμή μεταβλητής πρέπει να δημιουργήσει ένα instance της κλάσης **Signer** (που βρίσκεται στο module **django.core.signing**). Ο τρόπος για να πραγματοποιηθεί αυτό φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα:

```
>>> from django.core.signing import Signer
>>> signer = Signer()
>>> value = signer.sign('My string')
```

Η υπογραφή θα προστεθεί στο τέλος του string, ακολουθώντας τον χαρακτήρα **':'**, όπως φαίνεται παρακάτω:

```
>>> value
'My string:GdMGD6HNQ_qdgxYP8yBZAdAIV1w'
```

Για να διαβαστεί η αρχική τιμή μιας υπογεγραμμένης τιμής μεταβλητής, χρησιμοποιείται η μέθοδος **unsign()**.

Παράδειγμα:

```
>>> original = signer.unsign(value)
>>> original
'My string'
```

Σε περίπτωση που εντοπιστεί ανεπιθύμητη αλλοίωση των δεδομένων η μέθοδος **unsign()** αντί για την αρχική τιμή μεταβλητής επιστρέφει μία εξαίρεση της κλάσης **BadSignature**.

Παράδειγμα:

```
>>> from django.core import signing
>>> value += 'm'
>>> try:
...     original = signer.unsign(value)
... except signing.BadSignature:
...     print("Tampering detected!")
```

Η κλάση **Signer** χρησιμοποιεί το **SECRET_KEY** που έχει οριστεί στο αρχείο **settings.py** προκειμένου να εκτελέσει την υπογραφή δεδομένων, αλλά ένας προγραμματιστής έχει τη δυνατότητα να χρησιμοποιήσει κλειδί της επιλογής του χρησιμοποιώντας το ως παράμετρο όταν δημιουργεί το instance της κλάσσης **Signer**.

Παράδειγμα:

```
>>> signer = Signer('my-other-secret')
```

```
>>> value = signer.sign('My string')
>>> value
'My string:EkfQJafvGyiofrdGnuthdxImIJw'
```

4.6.2.2 Χρήση salt

Επίσης, η κλάση **Signer** δέχεται την προαιρετική παράμετρο salt. Με τη χρήση της, στην υπογραφή μιας τιμής προστίθεται κάθε φορά ένα salt (ακολουθία από χαρακτήρες που ορίζεται από τον προγραμματιστή).

Παράδειγμα:

```
>>> signer = Signer()
>>> signer.sign('My string')
'My string:GdMGD6HNQ_qdgxYP8yBZAdAIV1w'
>>> signer = Signer(salt='extra')
>>> signer.sign('My string')
'My string:Ee7vGi-ING6n02gkcJ-QLHg6vFw'
>>> signer.unsign('My string:Ee7vGi-ING6n02gkcJ-QLHg6vFw')
'My string'
```

Ένας προγραμματιστής έχει τη δυνατότητα να αυξάνει την δυσκολία αποκάλυψης της υπογραφής, προσθέτοντας κάθε φορά στην υπογραφή την ώρα και ημερομηνία (timestamp). Προκειμένου να συμβεί αυτό, πρέπει να κάνει χρήση μιας συγκεκριμένης υποκλάσης της κλάσης **Signer**, την **TimestampSigner**, όπως φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα:

```
>>> from datetime import timedelta
>>> from django.core.signing import TimestampSigner
>>> signer = TimestampSigner()
>>> value = signer.sign('hello')
>>> value
'hello:1NMg5H:oPVuCqlJWmChm1rA2lyTUtelC-c'
>>> signer.unsign(value)
'hello'
>>> signer.unsign(value, max_age=10)
...
SignatureExpired: Signature age 15.5289158821 > 10 seconds
>>> signer.unsign(value, max_age=20)
'hello'
>>> signer.unsign(value, max_age=timedelta(seconds=20))
'hello'
```

4.6.2.3 Υπογραφή Δομών Δεδομένων

Το module **django.core.signing** εκτός από την υπογραφή απλών μεταβλητών, διαθέτει και συναρτήσεις οι οποίες μπορούν να υπογράψουν ή να ανακτήσουν τιμές ολόκληρων Python dictionaries, λιστών Python κ.α.

Οι δύο αυτές μέθοδοι είναι οι εξής:

α) dumps(obj, key=None, salt='django.core.signing', compress=False):επιστρέφει ένα ασφαλές υπογεγραμμένο (με χρήση SHA1) και συμπιεσμένο (base64) JSON string. Η υπογραφή υλοποιείται με χρήση της κλάσης **TimestampSigner**.

Παράδειγμα:

```
>> from django.core import signing
>>> value = signing.dumps({"foo": "bar"})
>>> value
'eyJmb28iOiJiYXIifQ:1NMg1b:zGcDE4-TCkaeGzLeW9UQwZesciI'
>>> signing.loads(value)
{'foo': 'bar'}
```

β) loads(string, key=None, salt='django.core.signing', max_age=None): ανακτά τις αρχικές τιμές που έχουν υπογραφεί με χρήση της συνάρτησης **dumps()**, ενώ επιστρέφει εξαίρεση της κλάσης **BadSignature** άμα διαπιστωθεί αλλοίωση των δεδομένων [67].

Παράδειγμα*:*

```
>> from django.core import signing
>>> value = signing.dumps({"foo": "bar"})
>>> value
'eyJmb28iOiJiYXIifQ:1NMg1b:zGcDE4-TCkaeGzLeW9UQwZesciI'
>>> signing.loads(value)
{'foo': 'bar'}
```

4.7 Έλεγχος Περιεχομένου που Παρέχεται από Χρήστες

Μεταξύ των σημείων τα οποία απαιτούν ιδιαίτερη προσοχή κατά τη διάρκεια σχεδιασμού μιας διαδικτυακής εφαρμογής, ιδιαίτερη θέση έχουν τα σημεία στα οποία εισάγονται δεδομένα από τους χρήστες της, μιας και υπάρχει κίνδυνος εκμετάλλευσής τους με στόχο την εισαγωγή ανεπιθύμητου κώδικα.

Στις προηγούμενες παραγράφους έχει περιγραφεί ο τρόπος που το Django διασφαλίζει την προστασία από επιθέσεις τύπου SQL Injection και Cross – Site Scripting. Ένα ακόμη σημείο το οποίο μπορεί να αποτελέσει ευπάθεια είναι η δυνατότητα ανεβάσματος αρχείων.

4.7.1 ImageField

Το Django για την αποθήκευση εικόνων, εκτός από το **FileField** μπορεί να χρησιμοποιήσει field **ImageField**. Το συγκεκριμένο field (που για τη δυνατότητα χρήσης του απαιτείται η εγκατάσταση της βιβλιοθήκης **pillow**), κάθε φορά που δέχεται ένα αρχείο το ελέγχει για να διαπιστώσει εάν είναι όντως εικόνα, πριν αυτό αποθηκευτεί στη βάση δεδομένων. Αυτός ο

έλεγχος από μόνος του δεν είναι αρκετός, μιας και ένας κακόβουλος χρήστης θα μπορούσε να μεταμφιέσει κακόβουλο HTML κώδικα σε αρχείο εικόνας [63].

4.7.2 Κεφαλίδα X – Content – Type – Options

Μερικοί browsers προσπαθούν να μαντέψουν τον τύπο του περιεχομένου που πρόκειται να ανακτήσουν για να προβάλλουν στο χρήστη. Ως αποτέλεσμα, αγνοούν την κεφαλίδα **Content – Type** που περιέχουν τα αντίστοιχα HTTP requests. Αυτό μπορεί να αποτελέσει κίνδυνο ασφάλειας, μιας και η «λάθος εκτίμηση» ενός browser μπορεί να οδηγήσει σητν εκτέλεση κακόβουλου HTML ή Javascript κώδικα.

Για να αποτραπεί η διαδικασία αγνόησης της κεφαλίδας **Content – Type**, χρειάζεται η κεφαλίδα **X – Content – Type – Options** να καθοριστεί ως: **X – Content – Type – Options: nosniff**. Αυτό υλοποιείται για όλα τα responses αυτόματα από το **SecurityMiddleware**, θέτοντας την τιμή **True** στην μεταβλητή **SECURE_CONTENT_TYPE_NOSNIFF** στο αρχείο **settings.py [51]**.

Κεφάλαιο 5: Ασφαλής Σχεδιασμός Διαδικτυακής Εφαρμογής με Χρήση Προγραμματιστικού Πλαισίου Django

Έχοντας παρουσιάσει τον τρόπο λειτουργίας αλλά και τα εργαλεία ασφάλειας που διαθέτει το προγραμματιστικό πλαίσιο Django, σε αυτό το κεφάλαιο παρουσιάζεται ο ασφαλής σχεδιασμός μιας διαδικτυακής εφαρμογής με χρήση του συγκεκριμένου framework. Ειδικότερα, παρουσιάζεται η λειτουργία της διαδικτυακής εφαρμογής **myphoto** που δημιουργήθηκε για τις ανάγκες της εργασίας, τα εργαλεία ασφάλειας που αξιοποιήθηκαν για την προστασία της, καθώς και ο τρόπος με τον οποίο αυτό έγινε.

5.1 Δημιουργία Εργαστηριακού Περιβάλλοντος Ανάπτυξης

5.1.1 Εγκατάσταση Python και Django

Για τη δημιουργία του back – end μέρους του **myphoto,** χρησιμοποιήθηκε η έκδοση του **Django 1.10.5** (τελευταία αξιόπιστη έκδοση την περίοδο εκπόνησης της εργασίας) και η έκδοση της **Python 2.7.12**. Ο τρόπος εγκατάστασής τους περιγράφηκε αναλυτικά στην παράγραφο 3.11.1.

5.1.2 Εγκατάσταση Βιβλιοθήκης Pillow

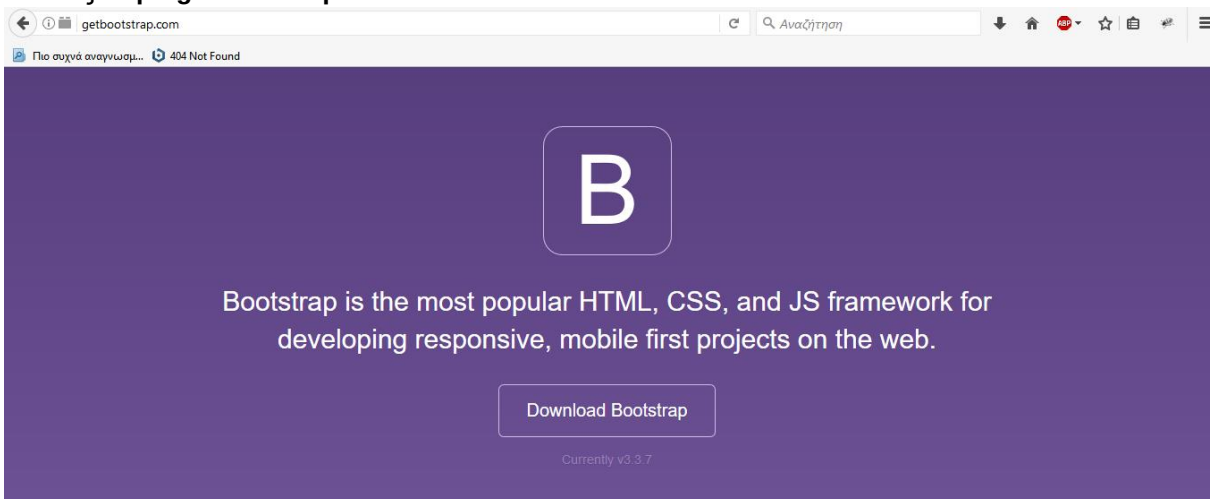
Η εφαρμογή myphoto κάνει χρήση της Python βιβλιοθήκης **pillow**, μιας βιβλιοθήκης που διευκολύνει τη διαχείριση αρχείων εικόνων. Η εγκατάστασή της έγινε μέσω πληκτρολόγησης της εντολής **pip install pillow** σε command prompt.

Εικόνα 46: Εγκατάσταση βιβλιοθήκης Pillow

5.1.3 Εγκατάσταση Bootstrap

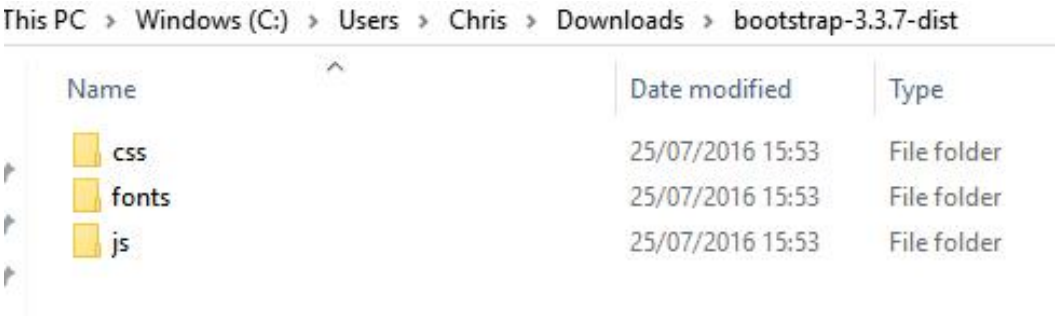
Για τις ανάγκες του front – end τμήματος της διαδικτυακής εφαρμογής, δηλαδή της μορφοποίησης της εμφάνισης του site έγινε χρήση του framework **Bootstrap 3.3**. Το Bootstrap είναι ένα front - end framework,που παρέχοντας έτοιμα τμήματα κώδικα CSS και Javascript, επιταχύνει τη διαδικασία σχεδιασμού ιστοσελίδων. Η εγκατάστασή του επιτυγχάνεται μέσω της

σελίδας **http://getbootstrap.com/**.



Εικόνα 47: Πρόσβαση στη σελίδα του Bootstrap

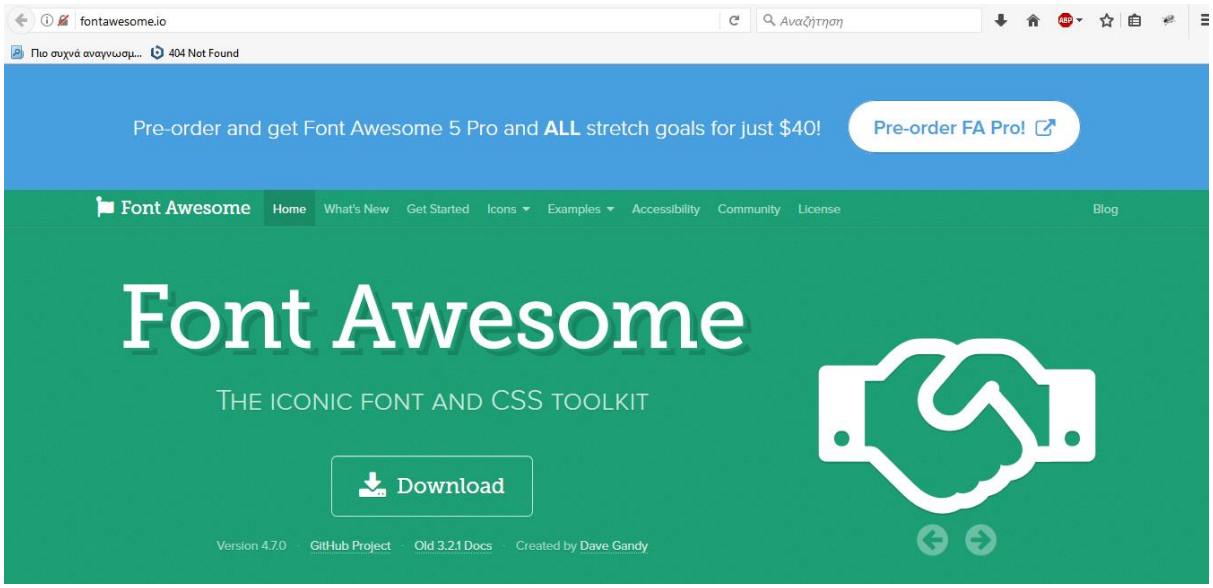
Μετά τη λήψη του Boostrap παρατηρούμε ότι στον φάκελο **bootstrap-3.3.7-dist** περιέχονται οι φάκελοι που φαίνονται στην εικόνα 48, που με τη σειρά τους περιέχουν τα αρχεία που υποδηλώνει και ο τίτλος του κάθε φάκλου. Για τις ανάγκες της εμφάνισης του site θα χρησιμοποιηθούν μόνο μερικά από αυτά.



Εικόνα 48: Αρχεία του Bootstrap

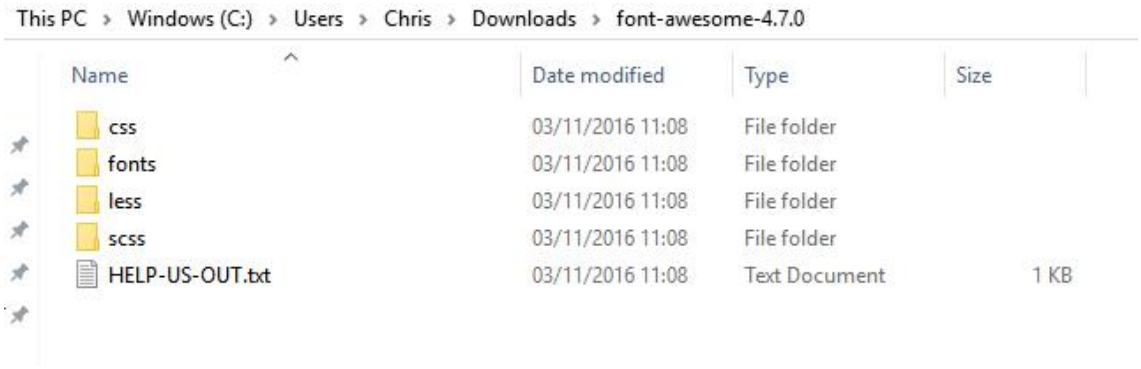
5.1.4 Εγκατάσταση Font – Awesome

Εκτός από το Boostrap, για τις ανάγκες της εμφάνισης της διαδικτυακής εφαρμογής myphoto έγινε χρήση του **Font – Awesome 4.7**. Το Font – Awesome αποτελεί ένα σύνολο γραφικών εικονιδίων που αναπαριστώνται με τη βοήθεια χρήσης CSS [68].
Η εγκατάστασή του Font – Awesome πραγματοποιείται μέσω της πρόσβασης στην ιστοσελίδα **http://fontawesome.io/**.



Εικόνα 49: Πρόσβαση στη σελίδα λήψηστου Font – Awesome

Μετά τη λήψη του **Font – Awesome** παρατηρούμε ότι έχει δημιουργηθεί ένας **φάκελος font-awesome-4.7.0** ο οποίος περιέχει φάκελους που περιέχουν τα αναγκαία αρχεία. Όπως και στην περίπτωση του **Bootstrap**, έτσι και εδώ, θα γίνει χρήση συγκεκριμένων εξ' αυτών.



Εικόνα 50: Αρχεία του Font – Awesome

5.1.5 Εγκατάσταση Sublime

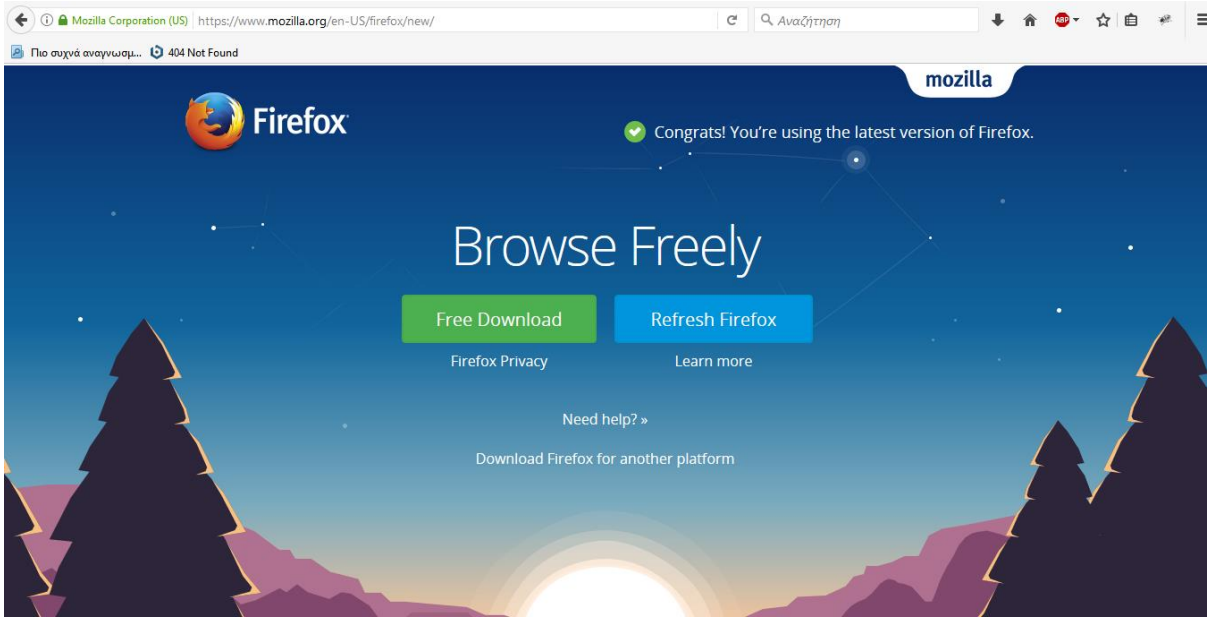
Η συγγραφή του κώδικα έγινε με χρήση του text editor **Sublime 3** . Για την εγκατάστασή του πραγματοποιήθηκε η πρόσβαση στη σελίδα **www.sublimetext.com** και η επιλογή **Download**

For Windows.

Εικόνα 51: Πρόσβαση στη σελίδα λήψης του Sublime

5.1.6 Εγκατάσταση Mozilla Firefox και Development Server

Για την προβολή και το debugging της διαδικτυακής εφαρμογής κατά τη διάρκεια της ανάπτυξης χρησιμοποιήθηκε ο browser Mozilla Firefox 51.0.1. Η εγκατάστασή του πραγματοποιήθηκε μέσω πρόσβασης στη σελίδα **https://www.mozilla.org/enUS/firefox/new/.**



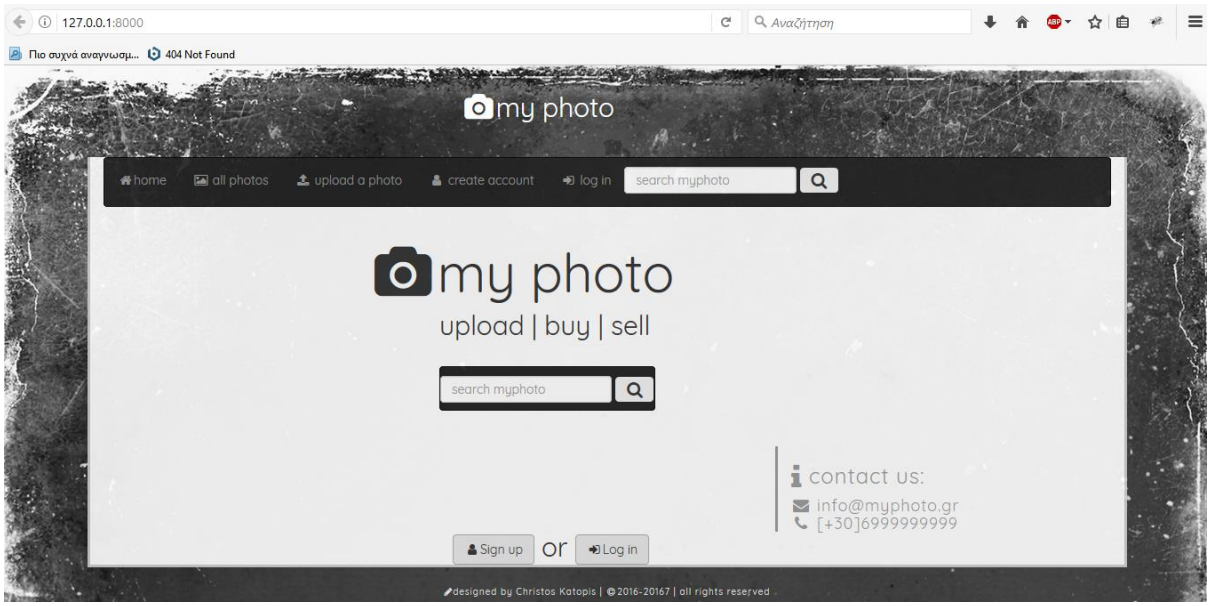
Εικόνα 52: Πρόσβαση στη σελίδα λήψης του Mozilla Firefox

Επίσης, χρησιμοποιήθηκε ο development server που έχει ενσωματωμένο το προγραμματιστικό πλαίσιο Django, η λειτουργία του οποίου έχει περιγραφεί στο Κεφάλαιο 3. Η ανάπτυξη της διαδικτυακής εφαρμογής έγινε στο λειτουργικό σύστημα Windows 10.

5.2 Περιγραφή Λειτουργίας Διαδικτυακής Εφαρμογής

Για την καλύτερη κατανόηση του σχεδιασμού της, κρίνεται σκόπιμο να γίνει μια συνοπτικής περιγραφή της λειτουργίας της διαδικτυακής εφαρμογής myphoto.

Το **myphoto** αποτελεί μία online gallery φωτογραφιών (και γενικότερα εικόνων) στην οποία οι χρήστες έχουν τη δυνατότητα να διαθέσουν φωτογραφίες τους προς πώληση ή να αγοράσουν φωτογραφίες άλλων χρηστών. Πληκτρολογώντας σε command prompt στην τοποθεσία **thesisproject/myphoto** (όπου έχει ήδη δημιουργηθεί το project,με βάση τα βήματα που περιγράφονται στην παράγραφο 3.11, καθώς και ο αναγκαίος κώδικας) την εντολή **python manage.py runserver,** και ύστερα στον browser την διεύθυνση **127.0.0.1:8000** παρατηρούμε την αρχική σελίδα της διαδικτυακής εφαρμογής:



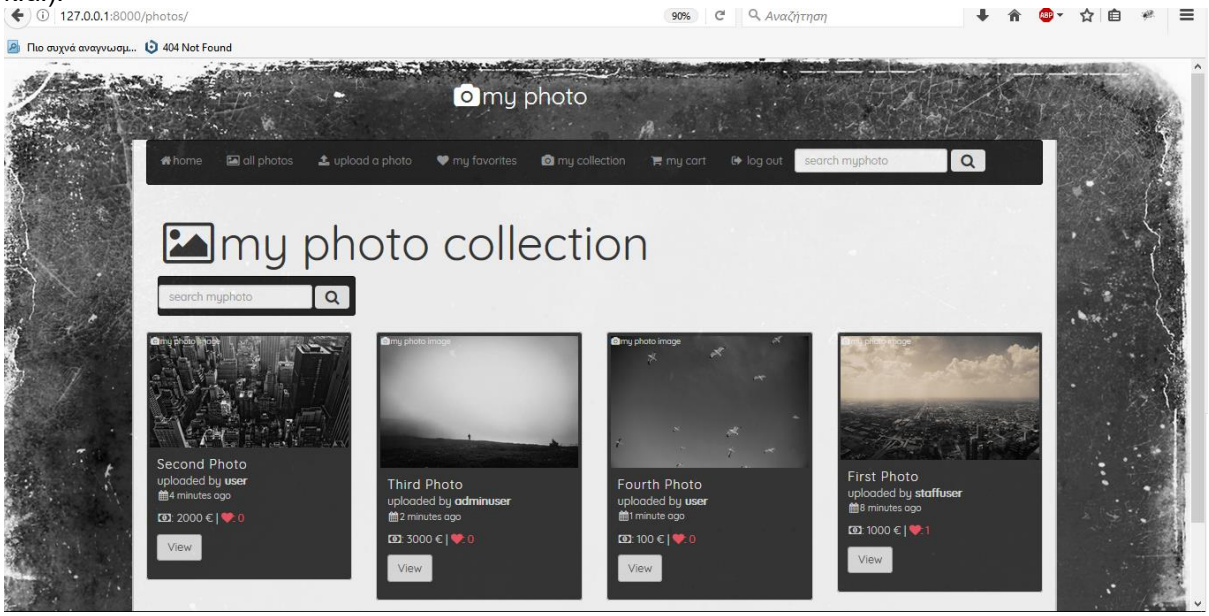
Εικόνα 53: Η αρχική σελίδα της διαδικτυακής εφαρμογής myphoto

Στις παραγράφους που ακολουθούν περιγράφονται αναλυτικότερα οι υπηρεσίες που παρέχει στους χρήστες το myphoto.

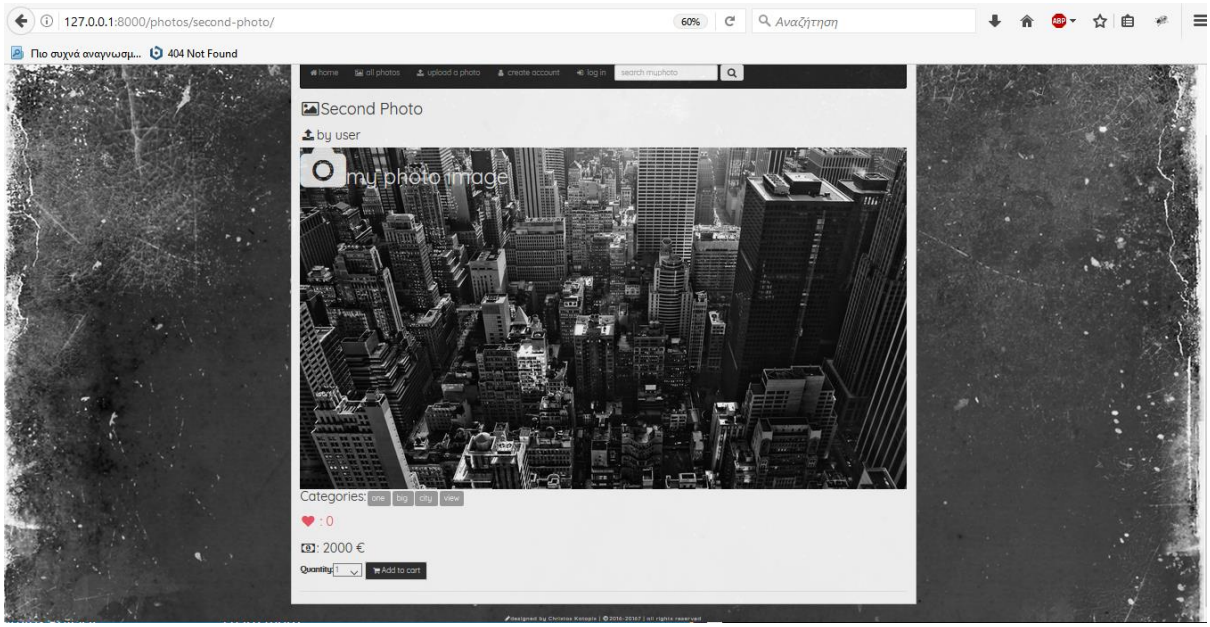
5.2.1 Προβολή – Αναζήτηση Φωτογραφιών

Παρέχεται η δυνατότητα σε όλους τους χρήστες (αυθεντικοποιημένους ή όχι) να δουν το σύνολο των φωτογραφιών που έχουν ανέβει στο myphoto, όπως και να αναζητήσουν φωτογραφίες με βάση τον χρήστη, τον τίτλο ή τις κατηγορίες στις οποίες ανήκουν. Επίσης ένας χρήστης μπορεί

να προβάλλει μια φωτογραφία σε μεγάλο μέγεθος, μαζί με σχετικές λεπτομέρειες (σχόλια, τιμή κ.α.).



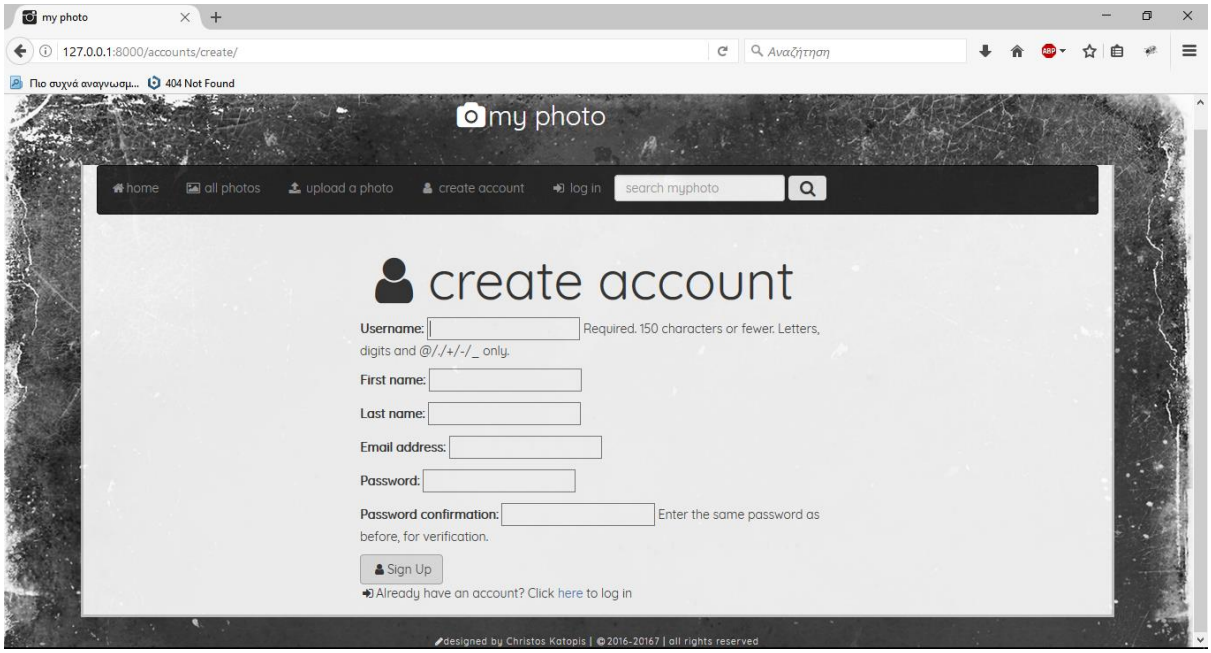
Εικόνα 54: Προβολή όλων των φωτογραφιών του myphoto



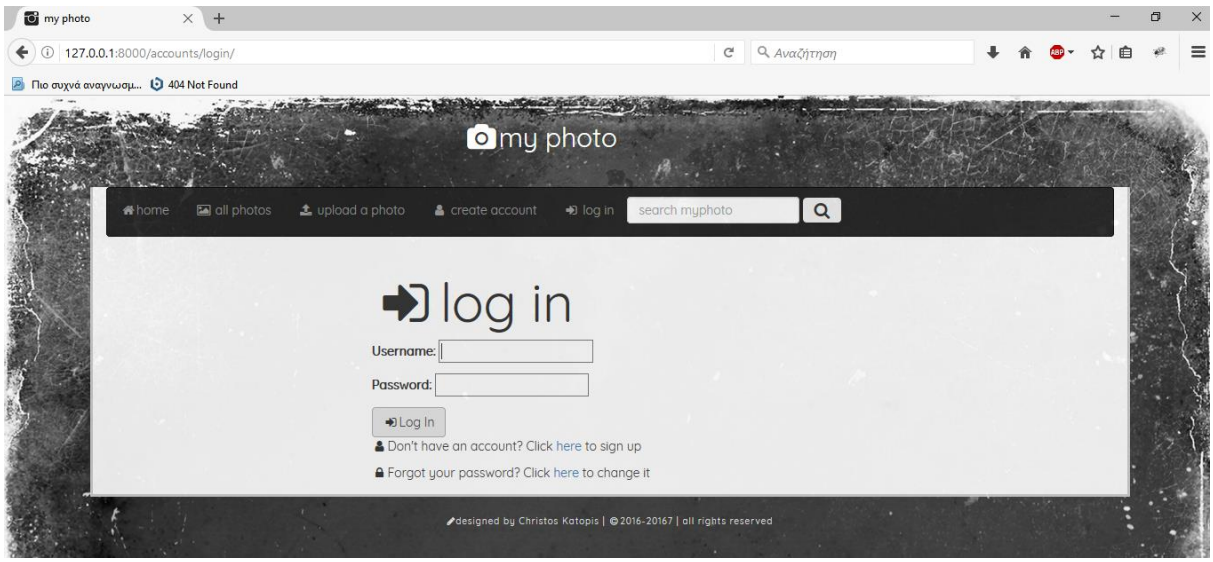
Εικόνα 55: Προβολή συγκεκριμένης φωτογραφίας του myphoto

5.2.2 Εγγραφή – Σύνδεση Χρήστη

Ένας μη αυθεντικοποιημένος χρήστης έχει τη δυνατότητα δημιουργίας λογαριασμού, όπως φαίνεται στην εικόνα 56, ώστε να έχει πρόσβαση σε περισσότερες υπηρεσίες που προσφέρει το myphoto. Επίσης, ένας χρήστης ο οποίος έχει ήδη δημιουργήσει λογαριασμό, πρέπει να συνδεθεί σε αυτόν προκειμένου να έχει πρόσβαση στις υπηρεσίες αυτές όπως φαίνεται στην εικόνα 57.



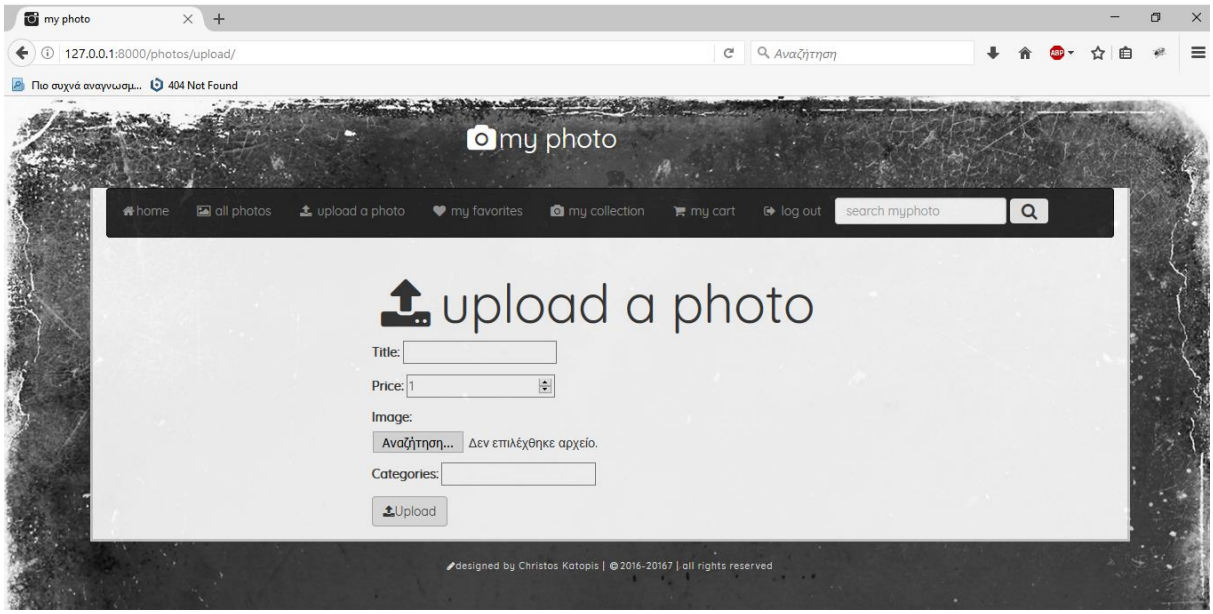
Εικόνα 56: Οθόνη εγγραφής νέου χρήστη στο myphoto



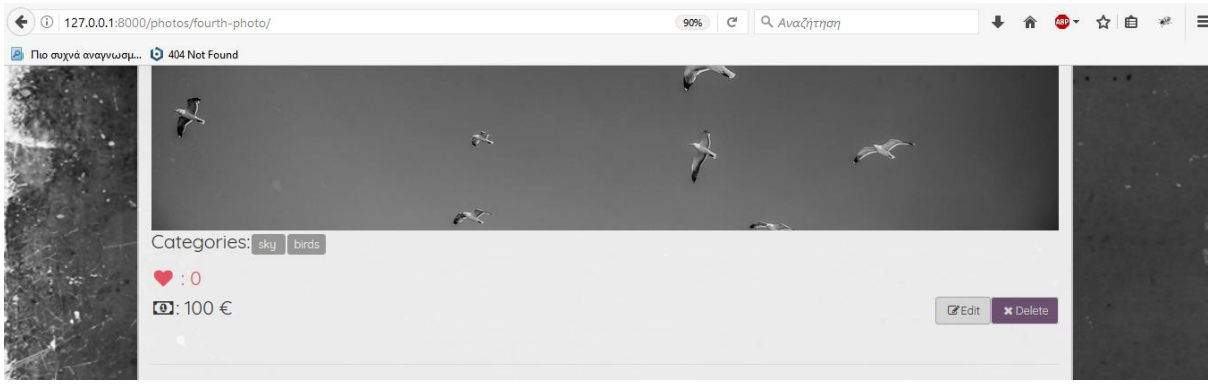
Εικόνα 57: Οθόνη σύνδεσης εγεγγραμένου χρήστη στο myphoto

5.2.3 Μεταφόρτωση – Διαγραφή – Επεξεργασία Φωτογραφίας

Οι αυθεντικοποιημένοι χρήστες του myphoto έχουν τη δυνατότητα να μεταφορτώσουν μία νέα φωτογραφία παρέχοντας σχετικές πληροφορίες με αυτή (τιμή, τίτλος, κατηγορίες) όπως φαίνεται στην εικόνα 58. Ο χρήστης ο οποίος ο οποίος έχει ανεβάσει τη φωτογραφία, έχει τη δυνατότητα να την επεξεργαστεί ή να τη διαγράψει, όπως φαίνεται στην εικόνα 59.



Εικόνα 58: Μεταφόρτωση φωτογραφίας στο myphoto

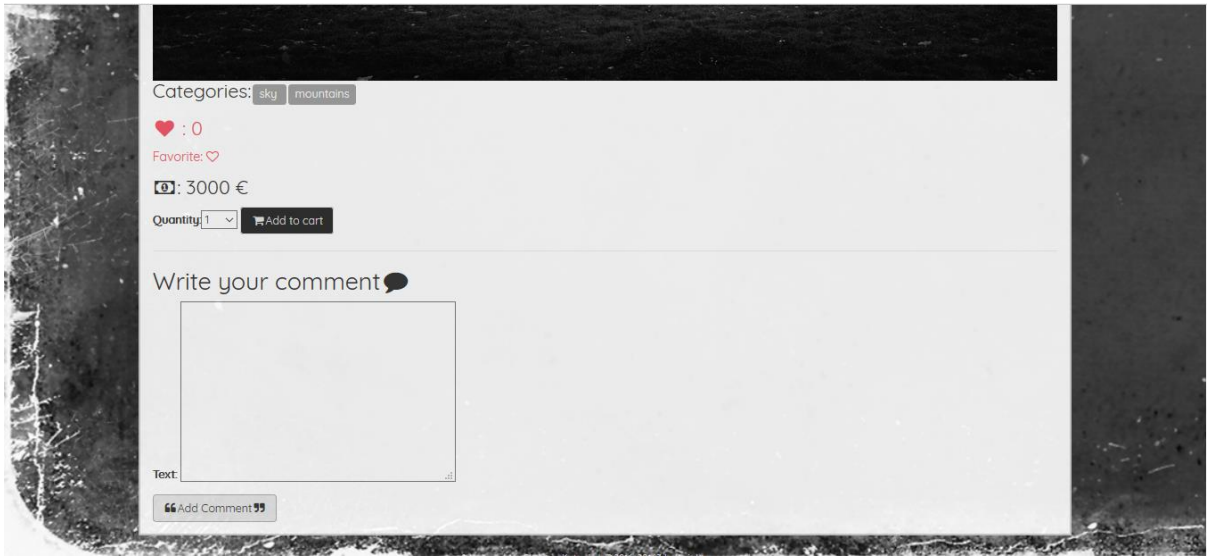


Εικόνα 59: Επεξεργασία και διαγραφή φωτογραφίας στο myphoto

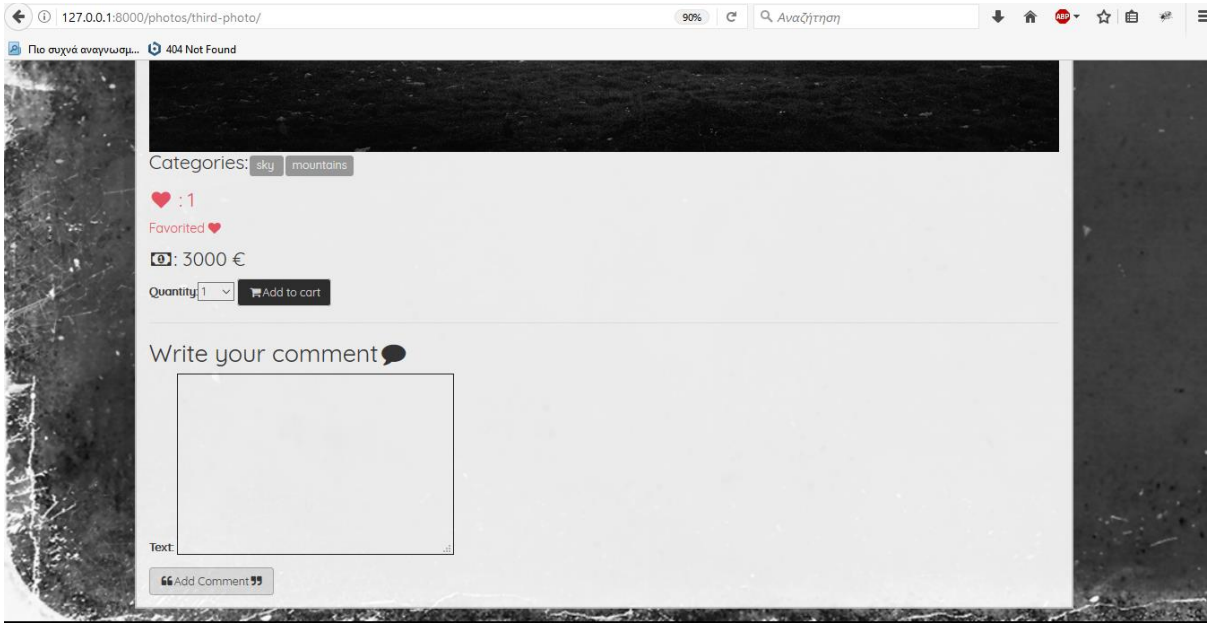
5.2.4 Προσθήκη Φωτογραφίας στα Αγαπημένα

Το **myphoto** παρέχει τη δυνατότητα σε αυθεντικοποιημένους χρήστες να προσθέσουν μία φωτογραφία άλλου χρήστη στα αγαπημένα τους (favorite), όπως φαίνεται στις εικόνες 60 και

61.



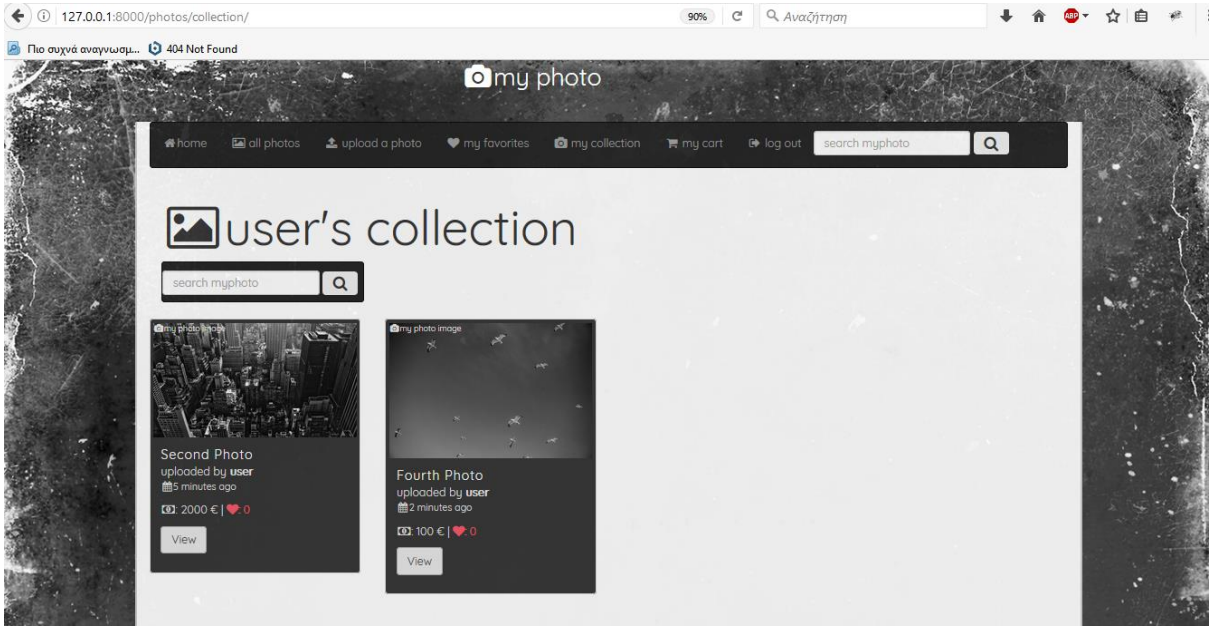
Εικόναα 60: Προσθήκη φωτογραφίας στα αγαπημένα



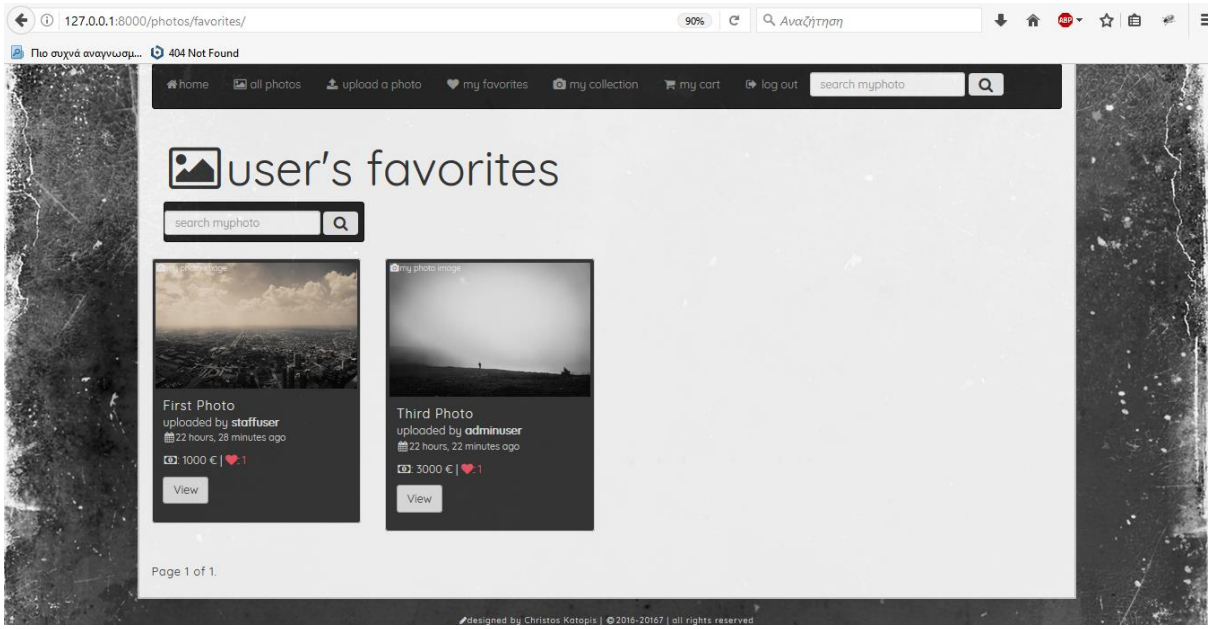
Εικόνα 61: Επιτυχής προσθήκη φωτογραφίας στα αγαπημένα

5.2.5 Προβολή Συλλογής – Προβολή Αγαπημένων

Ένας αυθεντικοποιημένος χρήστης, έχει τη δυνατότητα να προβάλλει τη συλλογή του, δηλαδή τις φωτογραφίες που έχει ο ίδιος προσθέσει στη διαδικτυακή εφαρμογή, ή τα αγαπημένα του, δηλαδή τις φωτογραφίες άλλων χρηστών που έχει προσθέσει στα αγαπημένα του.



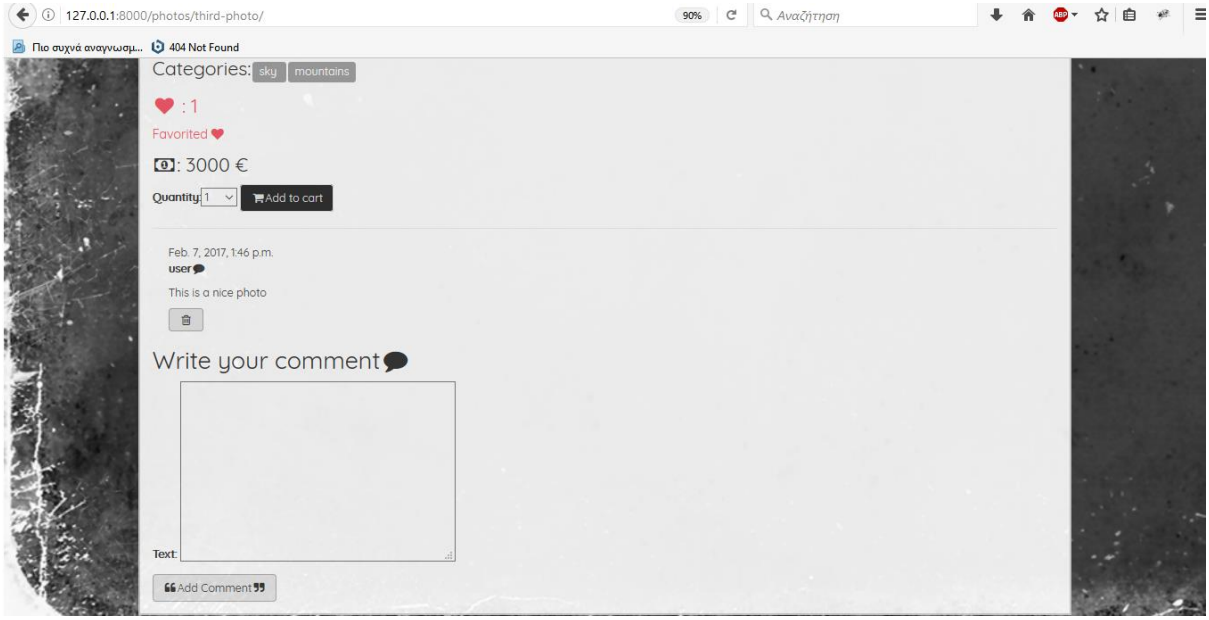
Εικόνα 62: Συλλογή φωτογραφίων που έχει μεταφορτώσει ο χρήστης user



Εικόνα 63: Συλλογή φωτογραφιών που έχει προσθέσει ο χρήστης user στα αγαπημένα του

5.2.6 Σχολιασμός Φωτογραφιών

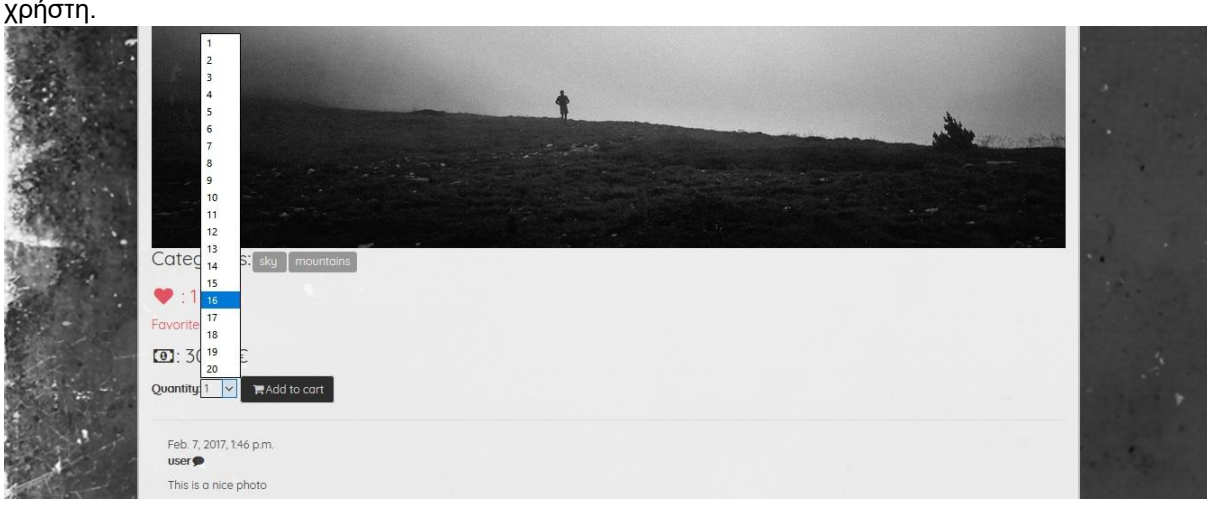
Παρέχεται η δυνατότητα σε αυθεντικοποιημένους χρήστες να σχολιάσουν οποιαδήποτε φωτογραφία. Επίσης, παρέχεται η δυνατότητα διαγραφής των σχολίων που έχουν πραγματοποιήσει οι ίδιοι.



Εικόνα 64: Προσθήκη σχολίου σε φωτογραφία

5.2.7 Αγορά Φωτογραφίας

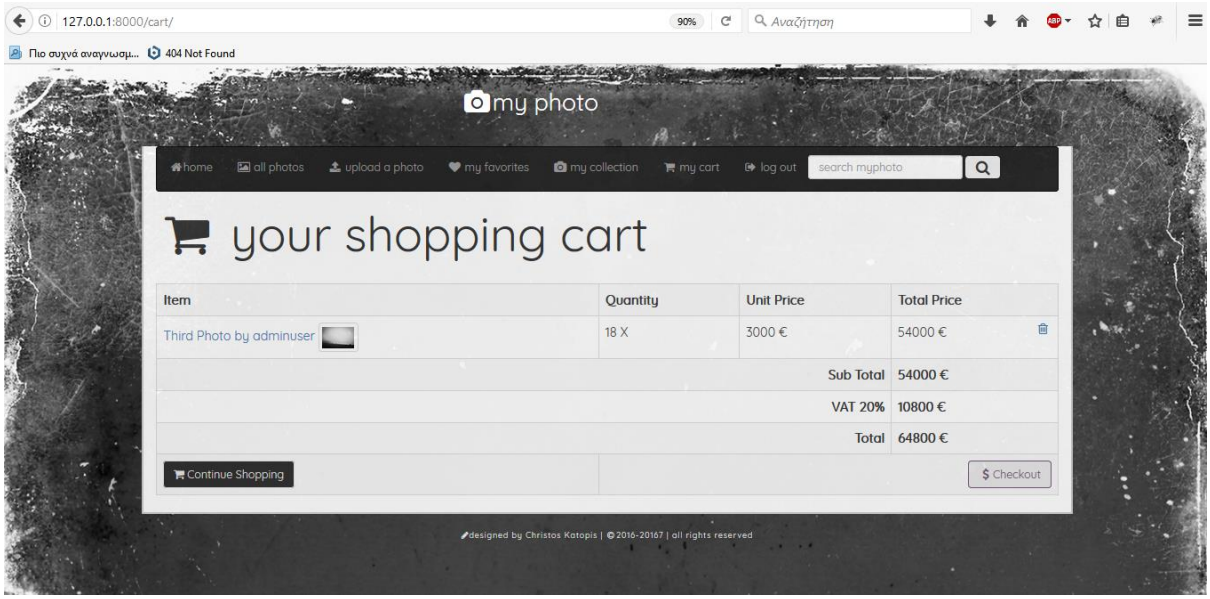
Ένας χρήστης (αυθεντικοποιημένος ή μη) του myphoto έχει τη δυνατότητα αγοράς μιας φωτογραφίας, επιλέγοντας την επιθυμητή ποσότητά της και πατώντας το κουμπί **Add to cart**. Η ολοκλήρωση της παραγγελίας μίας φωτογραφίας προϋποθέτει την αυθεντικοποίηση του χρήστη.



Εικόνα 65: Επιλογή ποσότητας μιας φωτογραφίας για προσθήκη στο καλάθι

5.2.8 Προβολή – Διαχείριση Καλαθιού

Ένας χρήστης (αυθεντικοποιημένος ή μη) έχει τη δυνατότητα να προβάλλει και να διαχειριστεί το καλάθι αγορών, διαγράφοντας κάποια φωτογραφία ή προσθέτοντας κάποια άλλη φωτογραφία στο καλάθι του.



Εικόνα 66: Προβολή και διαχείριση καλαθιού

5.2.9 Διαχείριση Δεδομένων από Προσωπικό (staff) και Διαχειριστές (superusers)

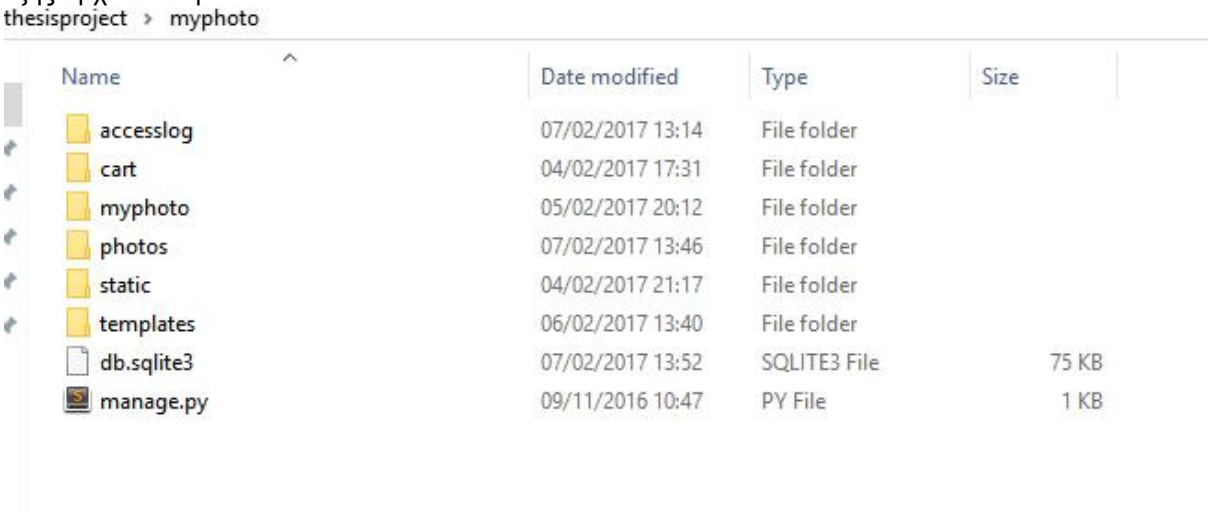
Το myphoto επιτρέπει σε χρήστες με αυξημένα δικαιώματα (staff και superusers), τη διαχείριση διάφορων δεδομένων που αφορούν σε χρήστες, φωτογραφίες, σχόλια, παραγγελίες φωτογραφιών κ.α. Η περιγραφή των λειτουργιών αυτών θα γίνει λεπτομερώς σε επόμενη παράγραφο του παρόντος κεφαλαίου.

5.3 Πηγαίος Κώδικας της Διαδικτυακής Εφαρμογής myphoto

Για την καλύτερη κατανόηση του τρόπου λειτουργίας της διαδικτυακής εφαρμογής, πριν τη λεπτομερή περιγραφή του σχεδιασμού της ασφάλειας του **myphoto**, είναι απαραίτητο να πραγματοποιηθεί σύντομη παρουσίαση του πηγαίου κώδικά του.

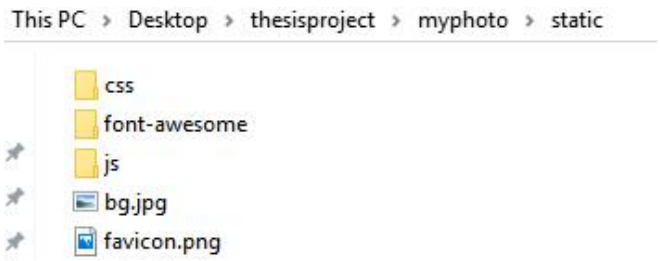
Ο κώδικας που σχετίζεται με το myphoto δημιουργήθηκε και αποθηκεύτηκε σε φάκελο με τίτλο **thesisproject**. Μεταβαίνοντας στο directory **thesisproject/myphoto** παρατηρούνται τα

εξής αρχεία και φάκελοι:



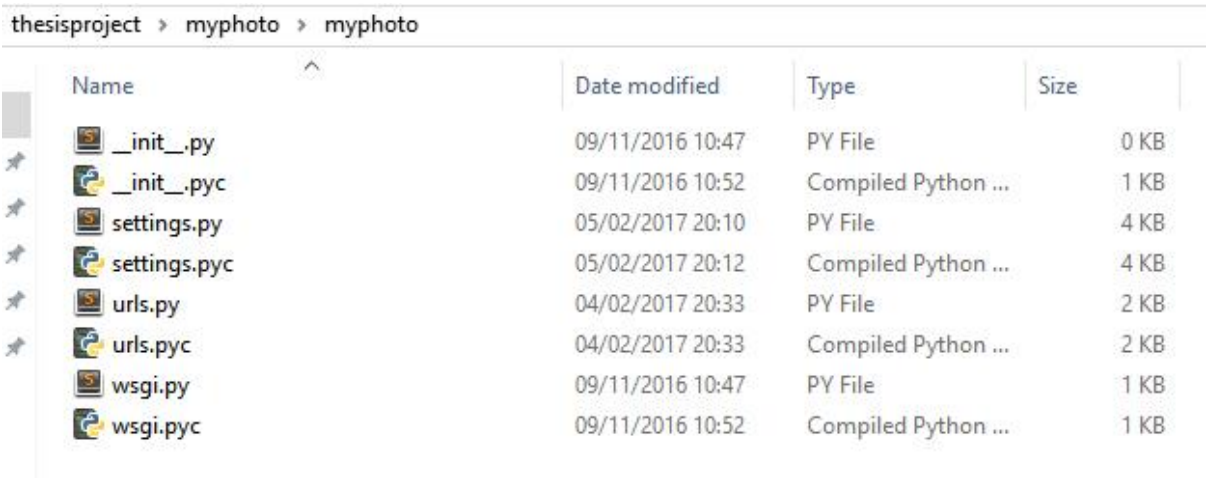
Εικόνα 67: Φάκελοι και αρχεία του myphoto

Στον φάκελο **static** περιέχονται τα αρχεία που σχετίζονται με την εμφάνιση της διαδικτυακής εφαρμογής, όπως τα τμήματα του κώδικα του **Boostrap** και του **Font – Awesome**, και τα αρχεία .css.



Εικόνα 68: Ο φάκελος static

Ο φάκελος **myphoto** περιέχει αρχεία που καθορίζουν το σύνολο της διαδικτυακής εφαρμογής,όπως το αρχείο των ρυθμίσεων **settings.py** και το αρχείο του συνολικού **URLconf urls.py**.

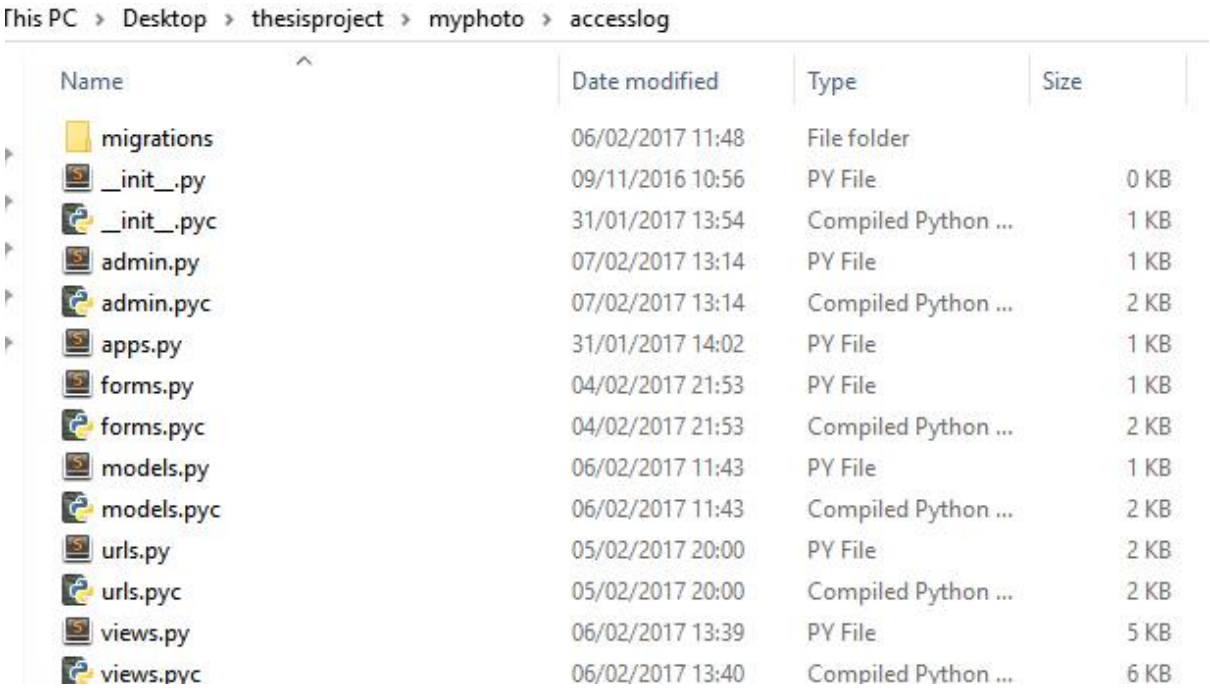


Εικόνα 69: Ο φάκελος myphoto

Στους υπόλοιπους τρεις φακέλους (**accesslog, photos, cart**) περιέχεται ο κώδικας των τριών υποεφαρμογών στις οποίες βασίζεται η λειτουργία της διαδικτυακής εφαρμογής myphoto. Κάθε υποεφαρμογή υλοποιεί τμήμα των συνολικών υπηρεσιών που παρέχει η διαδικτυακή εφαρμογή myphoto. Η λειτουργία αυτών των υποεφαρμογών θα εξεταστεί αναλυτικότερα στις επόμενες παραγράφους.

5.3.1 Η Εφαρμογή accesslog

Η εφαρμογή accesslog σχετίζεται με την αυθεντικοποίηση και την εξουσιοδότηση των χρηστών, με τη διαχείριση δεδομένων των χρηστών κ.α. Μεταβαίνοντας στον φάκελο accesslog, παρατηρούνται τα εξής αρχεία:



Εικόνα 70: Ο φάκελος accesslog

Όπως και κάθε εφαρμογή του προγραμματιστικού πλαισίου Django, ο κώδικας της υποεφαρμογής accesslog αποτελείται από models, views, URLs που βρίσκονται σε αντίστοιχα αρχεία .py. Στις επόμενες παραγράφους θα σκιαγραφηθεί το περιεχόμενο τους.

5.3.1.1 Accesslog Models

```python
from django.conf import settings
from django.db import models
from django.utils import timezone


class AccessAttempt(models.Model):
 user_ip=models.CharField(max_length=100)
 login_attempts=models.PositiveIntegerField(default=1)
 attempt_time= models.DateTimeField(default=timezone.now)
 blocked=models.BooleanField(default=False)


 class Meta:
     ordering=['-attempt_time']

 def __unicode__(self):
        return self.user_ip+" attempt"
```

Εικόνα 71: Το αρχείο models.py της εφαρμογής accesslog

Το model της accesslog είναι η κλάση **AccessAttempt**. Η κλάση αυτή χρησιμοποιείται για την καταγραφή των προσπαθειών σύνδεσης των χρηστών. Τα fields της είναι:

α) το user_ip, στο οποίο αποθηκεύεται η IP του χρήστη.
β) το login_attempts που καταγράφει τον αριθμό των προσπαθειών σύνδεσης ενός χρήστη.
γ) το attempt_time που καταγράφει την ώρα της τελευταίας προσπάθειας σύνδεσης.
δ) το blocked, ένα field της κλάσης **BooleanField** που καθορίζει εάν ο χρήστης με τη συγκεκριμένη IP έχει δικαίωμα σύνδεσης στο myphoto ή όχι.

5.3.1.2 Accesslog Forms

```python
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User


class UserForm(UserCreationForm):
 class Meta:
     model=User
     fields= {
          "username",
          "email",
          "first_name",
          "last_name"
     }

 def __init__(self, *args, **kwargs):
     super(UserForm,self).__init__(*args, **kwargs)
     self.fields['email'].required = True
```

Εικόνα 72: Το αρχείο forms.py της εφαρμογής accesslog

Η φόρμα **UserForm** του accesslog κληρονομεί τις ιδιότητές της από την κλάση **UserCreationForm** που παρέχει το **django.contrib.auth**, και χρησιμοποιείται για τη δημιουργία λογαριασμού χρήστη. Κληρονομεί όλες τις ιδιότητες της **UserCreationForm,** με τη διαφορά ότι τα πεδία που καλείται να συμπληρώσει ο χρήστης κατά τη δημιουργία λογαριασμού είναι, εκτός από το **username** και το **password**, το όνομα και το επίθετό του (**first_name, last_name**) καθώς και το **email** του.

Ταυτόχρονα, η κλάση **UserForm** διασφαλίζει ότι ένας χρήστης δεν μπορεί να υποβάλει φόρμα της συγκεκριμένης κλάσης αν δεν συμπληρώσει το πεδίο με το e – mail του.

5.3.1.3 Accelog Views

Οι συναρτήσεις view της εφαρμογής accesslog (που περιέχονται στο αρχείο **views.py**) είναι η **is_logged_in(),** η **check_ip()** , η **is_locked_out(),** η **create_account() και η activation_confirm().** Στη συγκεκριμένη παράγραφο θα πραγματοποιηθεί αναλυτική περιγραφή τους.

```
1  from django.conf import settings
2  from django.utils import timezone
3  from django.contrib.auth.views import login
4  from django.shortcuts import render
5  from django.http import HttpResponseRedirect,QueryDict
6  from django.contrib.auth.models import User
7  from django.http import HttpResponse,HttpResponseRedirect,Http404
8  from django.shortcuts import render
9  from django.core.mail import send_mail
10 from django.contrib.auth.tokens import default_token_generator
11 from django.contrib.sites.shortcuts import get_current_site
12 from django.utils.encoding import force_text
13 from django.utils.http import is_safe_url, urlsafe_base64_decode,urlsafe_base64_encode
14 from django.utils.six.moves.urllib.parse import urlparse, urlunparse
15 from django.template import loader
16 from django.utils.encoding import force_bytes
17 from .models import AccessAttempt
18 from .forms import UserForm
19
20 def is_logged_in(function):
21  def wrap(request):
22    user=request.user
23    if user.is_authenticated()==True:
24      return HttpResponseRedirect('/')
25    else:
26      return function(request)
27  wrap.__doc__=function.__doc__
28  wrap.__name__=function.__name__
29  return wrap
30
```

Εικόνα 73: Η συνάρτηση view is_logged_in

α) is_logged_in(request):

Η συνάρτηση view **is_logged_in()** χρησιμοποιείται ως decorator πριν από την εκτέλεση μιας συνάρτησης. Ελέγχει εάν ο χρήστης που έχει πραγματοποιήσει το request έχει συνδεθεί στο myphoto. Εάν έχει, τότε τον κάνει redirect στην κεντρική σελίδα. Εάν όχι, τότε αφήνει τη συνάρτηση να εκτελεστεί κανονικά.

```
def check_ip(request):
 x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
 if x_forwarded_for:
   ip = x_forwarded_for.split(',')[0]
 else:
   ip = request.META.get('REMOTE_ADDR')
 return ip


@is_logged_in
def is_locked_out(request):
   if request.method=="POST":
     ip = check_ip(request)
     userlocked,created=AccessAttempt.objects.get_or_create(user_ip=ip,user=request.user)
     if created:
       return login(request)
     else:
       blocked=userlocked.blocked
       if blocked:
         context={"blocked":blocked,"time":""}
         return render(request,"registration/login_failed.html",context)
       else:
         if userlocked.login_attempts > 3:
           now=timezone.now()
           time_passed= int ( (now - userlocked.attempt_time).total_seconds() )
           if time_passed < settings.LOCKOUT_TIME:
             time= settings.LOCKOUT_TIME - time_passed
             context={"blocked":blocked,"time":time}
             return render(request,"registration/login_failed.html",context)
           else:
             userlocked.delete()
             return login(request)
         else:
           userlocked.login_attempts=userlocked.login_attempts + 1
           userlocked.attempt_time=timezone.now()
           userlocked.save()
           return login(request)
   else:
     return login(request)
```

Εικόνα 74: Οι συναρτήσεις view check_ip, is_locked_out

β) check_ip(request):
Η συνάρτηση check_ip δέχεται ως είσοδο το request που πραγματοποιήθηκε και επιστρέφει την IP του χρήστη που το πραγματοποίησε. Αρχικά, ελέγχει την HTTP κεφαλίδα **HTTP_X_FORWARDED_FOR**, ώστε να διαπιστωθεί εάν ο χρήστης χρησιμοποιεί proxy server. Σε περίπτωση που χρησιμοποιείται, τότε η **check_ip()** επιστρέφει το πρώτο τμήμα της κεφαλίδας (δηλαδή την IP του χρήστη). Σε περίπτωση που δεν χρησιμοποιείται, η **check_ip()** ελέγχει την κεφαλίδα **REMOTE_ADDR** στην οποία αναγράφεται η IP του χρήστη.

γ) is_locked_out(request):
Η συνάρτηση **is_locked_out()** υλοποιεί την είσοδο του χρήστη στη διαδικτυακή εφαρμογή myphoto. Αρχικά, δέχεται ως decorator τη συνάρτηση **is_logged_in()**. Με αυτόν τον τρόπο διασφαλίζεται ότι ένας χρήστης που έχει ήδη συνδεθεί στη σελίδα, δεν μπορεί να ξανασυνδεθεί χωρίς πρώτα να πραγματοποιήσει έξοδο. Ύστερα ελέγχεται έαν το request πραγματοποιήθηκε με την HTTP μέθοδο POST, δηλαδή εάν ο χρήστης έχει αποστείλει στοιχεία εισόδου.

Σε περίπτωση που το request δεν έχει πραγματοποιηθεί με την μέθοδο POST,τότε η συνάρτηση **is_locked_out()** επιστρέφει τη συνάρτηση **django.contrib.auth.views.login().** Στη συνέχεια, καλείται η συνάρτηση **check_ip()** προκειμένου να διαπιστωθεί η IP του χρήστη και ελέγχεται εάν ο χρήστης με τη συγκεκριμένη IP έχει ήδη προσπαθήσει να συνδεθεί στο myphoto με τη γραμμή **userlocked,created=AccessAttempt.objects.get_or_create(user_ip=ip).**

Η εκτέλεση της συγκεκριμένης γραμμής έχει ως αποτέλεσμα την επιστροφή ή της δημιουργία εκ νέου ενός instance της κλάσης **AccessAttempt** στην μεταβλητή **userlocked** και της αντίστοιχης εκχώρησης της τιμής **False** ή **True** στην boolean μεταβλητή **created**. Σε περίπτωση που το συγκεριμένο instance δημιουργείται πρώτη φορά (δηλαδή **created = True**) η συνάρτηση **is_locked_out()** επιστρέφει τη συνάρτηση **login()** προκειμένου ο χρήστης να συμπληρώσει τα διαπιστευτήριά του και να προσπαθήσει να εισέλθει.

Σε διαφορετική περίπτωση ελέγχεται εάν ο χρήστης με τη συγκεκριμένη IP έχει δικαίωμα εισόδου. Εάν δεν έχει (**blocked = True**), τότε η συνάρτηση **is_locked_out()** επιστρέφει αρχείο template που εμφανίζει κατάλληλο μήνυμα. Σε περίπτωση που έχει, η συνάρτηση **is_locked_out()** ελέγχει τις προσπάθειες εισόδου που έχει πραγματοποιήσει ο χρήστης. Εάν ο χρήστης έχει πραγματοποιήσει παραπάνω από 4 αποτυχημένες προσπάθειες σύνδεσης, η **is_locked_out()** επιστρέφει template με μήνυμα που εξηγεί ότι πρέπει να περιμένει.
Σε κάθε άλλη περίπτωση, η **is_locked_out()** αυξάνει το χαρακτηριστικό **login_attempts** του αντικειμένου **userlocked** που αντιστοιχεί στον χρήστη με τη συγκεκριμένη IP κατά 1 και θέτει στο χαρακτηριστικό **attempt_time** την ώρα που πραγματοποιήθηκε η τελευταία προσπάθεια. Το instance του **AccessAttempt** που αντιστοιχεί στον χρήστη με τη συγκεκριμένη IP διαγράφεται μετά από επιτυχή είσοδο στο myphoto.

```
72  @is_logged_in
73  def create_account(request):
74   form=UserForm(request.POST or None)
75   if form.is_valid():
76      user = form.save(commit=False)
77      user.is_active=False
78      user.save()
79      token_generator=default_token_generator
80      domain=get_current_site(request).domain
81      if request.is_secure():
82        protocol="https"
83      else:
84        protocol="http"
85      context = {
86         'domain': domain,
87         'uid': urlsafe_base64_encode(force_bytes(user.pk)),
88         'token': token_generator.make_token(user),
89         'protocol':protocol,
90           }
91      subject="my photo account activation"
92      email=loader.render_to_string("registration/activation_email.html",context)
93      send_mail(subject,email,"myphoto.team@gmail.com",[user.email])
94      return HttpResponseRedirect("/accounts/activation")
95   return render(request,"create_account.html",{"form":form})
96
```

Εικόνα 75: Η συνάρτηση view create_account

γ) create_account(request):
Η view συνάρτηση **create_account()** υλοποιεί τη δημιουργία λογαριασμού ενός χρήστη με χρήση της φόρμας **UserForm**. Η εκτέλεσή της πραγματοποιείται μόνο σε περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος (αυτό διασφαλίζεται με χρήση του decorator **is_logged_in()**). Σε περίπτωση που ένας χρήστης έχει αποστείλει τα στοιχεία του μέσω της φόρμας και η φόρμα έχει πιστοποιηθεί, τότε ένα νέο instance του model **django.contrib.auth.User** δημιουργείται και η συνάρτηση **create_account()** αποστέλλει, με χρήση της συνάρτησης **django.core.mail.send_mail**(), e – mail με σύνδεσμο ενεργοποίησης του λογαριασμού στην διεύθυνση e – mail που αντιστοιχεί στο χρήστη που συμπλήρωσε τα στοιχεία του.

Ο σύνδεσμος αυτός παράγεται κρυπτογραφόντας το πρωτεύον κλειδί του αντικειμένου της κλάσης User που αναπαριστά το χρήστη και χρησιμοποιώντας token που παράγεται από το **default_token_generator** που παρέχει το **django.utils**.

```
98   def activation_confirm(request,uidb64=None, token=None):
99     token_generator=default_token_generator
100    assert uidb64 is not None and token is not None
101    try:
102     uid = force_text(urlsafe_base64_decode(uidb64))
103     user = User.objects.get(pk=uid)
104    except (TypeError, ValueError, OverflowError, UserModel.DoesNotExist):
105     user = None
106    if user is not None and token_generator.check_token(user, token):
107     validlink = True
108     date = timezone.now()
109     days_passed = (date - user.date_joined).days
110     if (days_passed < settings.ACCOUNT_ACTIVATION_DAYS):
111       context= {
112           'validlink':validlink,
113           'error_message':''
114       }
115       user.is_active=True
116       user.save()
117     else:
118       validlink=False
119       context = {
120         'validlink': validlink,
121         'error_message':'Your account has expired. Please create a new one.'
122        }
123    else:
124     validlink = False
125     context = {
126       'validlink': validlink,
127       'error_message':'The link is not valid. Please try again or create a new account.'
128      }
129    return render(request,"registration/activation_confirm.html",context)
130
131
132
```

Εικόνα 76: Η συνάρτηση view activation_confirm

δ) activation_confirm(request,uid=None,token=None):
Η συνάρτηση **activation_confirm()** καλείται σε περίπτωση που ένας χρήστης έχει καταχωρήσει τα στοιχεία του για δημιουργία λογαριασμού στο myphoto και έχει ακολουθήσει το σύνδεσμο ενεργοποίησης του λογαργιασμού του που έχει σταλεί μέσω mail μετά την εκτέλεση της **create_account()**. Η συνάρτηση **activation_confirm()** ελέγχει το πρωτεύον κλειδί του αντικειμένου **User** που αναπαριστά το χρήστη που πραγματοποίησε το request, και το token που έχει χρησιμοποιηθεί για την παραγωγή του URL ενεργοποίησης. Σε περίπτωση που ο σύνδεσμος είναι έγκυρος, το attribute **is_active** του αντίστοιχου αντικειμένου της κλάσης User λαμβάνει την τιμή **True** και ο αντίστοιχος χρήστης αποκτά τη δυνατότητα να πραγματοποιήσει είσοδο στο myphoto.

5.3.1.4 Accesslog Urls

```
1  from django.conf.urls import url
2  from django.contrib.auth import views as auth_views
3  from .views import create_account,activation_confirm,is_locked_out
4  from django.views.generic import TemplateView
5  from django.contrib.auth.decorators import login_required
6
7
8
9  urlpatterns=[
10  url(r'^create/',create_account,name="create_account"),
11  url(r'^activation/$',TemplateView.as_view(template_name="registration/activation.html")),
12  url(r'^activation/(?P<uidb64>[0-9A-za-z]+)-(?P<token>.+)/',activation_confirm,name="activation"),
13  url(r'^loggedout/',TemplateView.as_view(template_name="registration/loggedout.html")),
14  url(r'^login/',is_locked_out,name="login"),
15  url(r'^logout/',login_required(auth_views.logout,redirect_field_name=None),name="logout"),
16  url(r'^password/reset/$',auth_views.password_reset,{"post_reset_redirect":"/accounts/password/reset/done"},name="password_reset"),
17  url(r'^password/reset/done/$',auth_views.password_reset_done),
18  url(r'^password/reset/(?P<uidb64>[0-9A-za-z]+)-(?P<token>.+)/$',auth_views.password_reset_confirm,{"post_reset_redirect":"/account
19  url(r'^password/reset/complete/$',auth_views.password_reset_complete)
20  ]
```

Εικόνα 77: Το αρχείο urls.py της εφαρμογής accesslog

Τα urls του accesslog σχετίζονται με τη δημιουργία του account, την ενεργοποίησή του, την είσοδο του χρήστη, την έξοδό του, καθώς και την αλλαγή κωδικού. Συγκεκριμένα:

α) το urlpattern /create/ αντιστοιχεί στη view **create_account()**.
β) τα patterns που ξεκινούν με /activation/ σχετίζονται με την ενεργοποίηση λογαριασμού μέσω link που παρέχεται με e - mail (και ένα εξ'αυτών αντιστοιχεί στη view **activation_confirm())**.
γ) το pattern /login/ καλεί τη view **is_locked_out()**.
δ) το pattern /logout/ αντιστοιχεί στην συνάρτηση **django.contrib.auth.views.logout()**.
ε) τα patterns που ξεκινούν με το password/reset/ σχετίζονται με την αλλαγή κωδικού του χρήστη και αντιστοιχούν σε views που έχει ενσωματωμένα το ίδιο το django, όπως τη συνάρτηση **django.contrib.auth.views.password_reset()**.

5.3.1.5 Accesslog Admin

```python
from django.contrib import admin
from django.contrib.auth.models import User,Group

from .models import AccessAttempt


class AccessAdmin (admin.ModelAdmin):
  list_display = ["user_ip","attempt_time","login_attempts","blocked"]
  search_fields=["user_ip","attempt_time"]
  list_editable=["blocked"]
  list_filter=["blocked"]
class Meta:
    model = AccessAttempt


class UserAdmin(admin.ModelAdmin):
 list_display=["username","email","first_name","last_name","is_staff","is_superuser","is_active","last_login"]
 search_fields=["username"]
 list_filter=["is_staff","is_superuser","is_active"]
 list_editable=["is_active"]

 class Meta:
    model=User



admin.site.unregister(Group)
admin.site.unregister(User)
admin.site.register(User,UserAdmin)
admin.site.register(AccessAttempt,AccessAdmin)
```

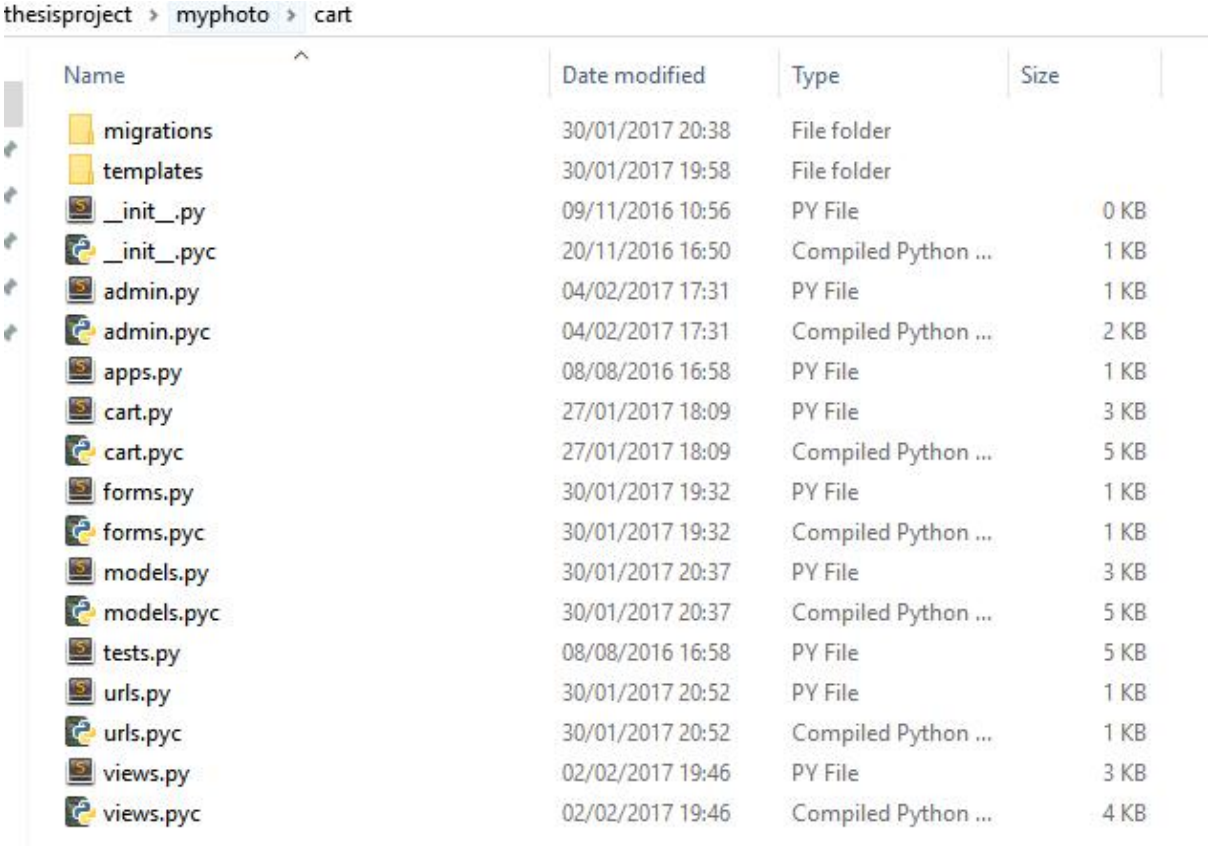
Εικόνα 78: Το αρχείο admin.py της εφαρμογής accesslog

Στο αρχείο admin.py της υποεφαρμογής accesslog δημιουργούνται δύο υποκλάσεις της κλάσης **ModelAdmin**:
α) Η υποκλάση AccessAdmin που σχετίζεται με τη διαχείριση των αντικειμένων **AccessAttempt** στο **django admin interface**.
β) Η υποκλάση UserAdmin που σχετίζεται με τη διαχείριση των αντικειμένων της κλάσης **User** στο django admin interface.

5.3.2 Η Εφαρμογή Cart

Η εφαρμογή **cart** υλοποιεί όλες τις λειτουργίες που σχετίζονται με την αγορά φωτογραφιών και την διαχείριση του προσωπικού καλαθιού του χρήστη.

Μεταβαίνοντας στον φάκελο **cart** παρατηρούμε τα εξής αρχεία:



Εικόνα 79: Ο φάκελος cart

Στις επόμενες παραγράφους ακολουθεί η αναλυτικότερη περιγραφή τους.

5.3.2.1 Το Αρχείο cart.py

Θεμελιώδης κώδικας για τη λειτουργία της υποεφαρμογής cart, είναι ο κώδικας που βρίσκεται στο αρχείο **cart.py**. Ο κώδικας αυτός ορίζει μία κλάση Cart που αναπαριστά το καλάθι των αγορών. Στην κλάση ορίζονται επίσης οι μέθοδοι **add()**, **save(), remove(), get_total_price(), get_vat(), get_sub_total(), _len_(),_iter_(),clear()**.

Τα αντικείμενα της κλάσης **Cart** αποθηκεύονται στις συνόδους των χρηστών και σταματούν να υπάρχουν όταν η σύνοδος λήξει. Κατά την αρχικοποίηση ενός αντικειμένου της κλάσης **Cart**, ελέγχεται εάν αυτό υπάρχει στη σύνοδο. Εάν δεν υπάρχει, δημιουργείται με την μορφή ενός κενού Python dictionary.

```python
from decimal import Decimal
from django.conf import settings
from photos.models import Photo

class Cart(object):

    def __init__(self, request):
        """
        Initialize the cart.
        """
        self.session = request.session
        cart = self.session.get(settings.CART_SESSION_ID)
        if not cart:
            # save an empty cart in the session
            cart = self.session[settings.CART_SESSION_ID] = {}
        self.cart = cart

    def add(self, product, quantity=1, update_quantity=False):
        """
        Add a product to the cart or update its quantity.
        """
        product_id = str(product.id)
        if product_id not in self.cart:
            self.cart[product_id] = {'quantity': 0,
                                     'price': str(product.price)}
        if update_quantity:
            self.cart[product_id]['quantity'] = quantity
        else:
            self.cart[product_id]['quantity'] += quantity
        self.save()

    def save(self):
        # update the session cart
        self.session[settings.CART_SESSION_ID] = self.cart
        # mark the session as "modified" to make sure it is saved
        self.session.modified = True
```

Εικόνα 80: Οι μέθοδοι add(), save()

α) add(self, product, quantity, update_quantity):
Η μέθοδος **add()** κατά την εκτέλεσή της ελέγχει εάν το προϊόν που έχει καταχωρηθεί ως είσοδός της υπάρχει στο καλάθι. Εάν δεν υπάρχει, δημιουργεί αντίστοιχη καταχώρηση για το προϊόν και στη συνέχεια προσθέτει όσα τεμάχια από αυτό αντιστοιχούν την τιμή της μεταβλητής **quantity**. Εάν υπάρχει, ανανεώνει την ποσότητα της ήδη υπάρχουσας καταχώρησης.

β) save(self): Η μέθοδος **save()** αποθηκεύει το καλάθι στην σύνοδο και υποδηλώνει ότι η σύνοδος έχει αλλάξει ώστε να αποθηκευτεί.

```
37
38      def remove(self, product):
39          """
40          Remove a product from the cart.
41          """
42          product_id = str(product.id)
43          if product_id in self.cart:
44              del self.cart[product_id]
45              self.save()
46
47      def __len__(self):
48          """
49          Count all items in the cart.
50          """
51          return sum(item['quantity'] for item in self.cart.values())
52
53      def get_total_price(self):
54          return sum(Decimal(item['price']) * item['quantity'] for item in self.cart.values()) + 20*(sum(Decimal(item['price']) * item['
55
56      def get_vat(self):
57          return 20*(sum(Decimal(item['price']) * item['quantity'] for item in self.cart.values()))/100
58
59      def get_sub_total(self):
60          return sum(Decimal(item['price']) * item['quantity'] for item in self.cart.values());
61
62      def __iter__(self):
63          """
64          Iterate over the items in the cart and get the products
65          from the database.
66          """
67          product_ids = self.cart.keys()
68          # get the product objects and add them to the cart
69          products = Photo.objects.filter(id__in=product_ids)
70          for product in products:
71              self.cart[str(product.id)]['product'] = product
72
73          for item in self.cart.values():
74              item['price'] = Decimal(item['price'])
75              item['total_price'] = item['price'] * item['quantity']
76              yield item
77
78      def clear(self):
79          # remove cart from session
80          del self.session[settings.CART_SESSION_ID]
81          self.session.modified = True
```

Εικόνα 81: Οι μέθοδοι _len_, get_total_price,get_vat,get_sub_total,_iter_,clear

γ) remove(self, product):
η μέθοδος **remove()** διαγράφει την καταχώρηση του προϊόντος **product** από το αντικείμενο της κλάσης **Cart**.

δ) _len_(self):
Η μέθοδος **_len_()** υπολογίζει το άθροισμα των ποσοτήτων όλων των προϊόντων που έχουν καταχωρηθεί στο αντικείμενο της κλάσης **Cart**.

ε) get_sub_total(self):
Η μέθοδος **get_sub_total()** υπολογίζει το συνολικό κόστος των προϊόντων που είναι αποθηκευμένα στο καλάθι.

στ) get_vat(self):
Η μέθοδος **get_vat()** υπολογίζει το επιπλέον χρηματικό ποσό που πρέπει να πληρωθεί (ποσοστό 20% - έξοδα αποστολής, φόροι και ποσοστό παρακράτησης του myphoto).

ζ) get_total_price(self):
Η μέθοδος **get_total_price()** υπολογίζει το άθροισμα του αποτελέσματος των παραπάνω δύο μεθόδων.

η) _iter_(self):

Η μέθοδος _**iter(self)** δημιουργεί keys του dicionary **cart** που αντιστοιχούν στις φωτογραφίες που βρίσκονται αποθηκευμένες στη βάση δεδομένων (και αναπαριστώνται από το model **Photo**).

ι) clear(self):
Η μέθοδος **clear()** διαγράφει την καταχώρηση **Cart** από την τρέχουσα σύνοδο και αποθηκεύει τη σύνοδο.

5.3.2.2 Cart Models

```python
from django.db import models
from django.conf import settings


class Order(models.Model):
 amount=models.PositiveIntegerField()
 date_created=models.DateTimeField(auto_now=False,auto_now_add=True)
 user= models.ForeignKey(settings.AUTH_USER_MODEL,default=1)
 is_completed=models.BooleanField(default=False)


```

Εικόνα 82: Το αρχείο models.py της εφαρμογής cart

Το model της υποεφαρμογής cart είναι η κλάση **Order**. Η κλάση **Order** χρησιμοποιείται για την καταγραφή των παραγγελιών στη βάση δεδομένων, αφού αυτές πραγματοποιηθούν.Περιέχει τα attributes:
α) amount, που ορίζει το συνολικό κόστος της αγοράς.
β) date_created στο οποίο εκχωρείται η ημερομηνία της παραγγελίας.
γ) user που είναι **ForeignKey** στο αντικείμενο της κλάσης User που αναπαριστά τον χρήστη που πραγματοποίησε την παραγγελία.
γ) is_completed, το οποίο είναι ένα **BooleanField** που ορίζει εάν η παραγγελία ενός χρήστη έχει αποσταλλεί ή όχι.

5.3.2.3 Cart Forms

```python
from django import forms

PRODUCT_QUANTITY_CHOICES = [(i, str(i)) for i in range(1, 21)]

class CartAddProductForm(forms.Form):
    quantity = forms.TypedChoiceField(
                                choices=PRODUCT_QUANTITY_CHOICES,
                                coerce=int)
    update = forms.BooleanField(required=False,
                                initial=False,
                                widget=forms.HiddenInput)


```

Εικόνα 83: Το αρχείο forms.py της εφαρμογής car

Το αρχείο **forms.py** της υποεφαρμογής cart περιέχει μία υποκλάση της κλάσης **Form** με όνομα **CartAddProductForm**. Αυτή η φόρμα χρησιμοποιείται προκειμένου να πραγματοποιηθεί η προσθήκη μιας φωτογραφίας στο καλάθι του χρήστη.

5.3.2.4 Cart Views

Οι συναρτήσεις views της υποεφαρμογής cart είναι οι **cart_add(), cart_remove(), cart_detail()** και **checkout()**.

```
7   from .forms import CartAddProductForm
8   from .models import Order
9   from django.contrib.auth.decorators import login_required
10  from django.contrib.auth.hashers import check_password
11  from django.core.mail import send_mail
12  
13  @require_POST
14  def cart_add(request, id, slug=None):
15      cart = Cart(request)
16      product = get_object_or_404(Photo, id=id)
17      product_id=product.id
18      form = CartAddProductForm(request.POST)
19      if form.is_valid():
20          cd = form.cleaned_data
21          cart.add(product=product,
22                   quantity=cd['quantity'],
23                   update_quantity=cd['update'])
24      return redirect('cart:cart_detail')
```

Εικόνα 84: Η συνάρτηση view cart_add

α) cart_add(requqst,id,slug=None):
Η view συνάρτηση **cart_add()** δέχεται ως είσοδο το request που πραγματοποιεί ο χρήστης καθώς και την μεταβλητή **id**. Ανακτά ή δημιουργεί ένα αντικείμενο της κλάσης **Cart** που αντιστοιχεί στην σύνοδο που σχετίζεται με το request, και, με χρήση της μεθόδου **add()** που έχει οριστεί στο αρχείο cart.py και μιας φόρμας της κλάσης **CartAddProductForm** προσθέτει την φωτογραφία με το αντίστοιχο **id** στο κατάλληλο αντικείμενο της κλάσης **Cart**. Η ποσότητα καθορίζεται από την ποσότητα που έχει εισάγει στη φόρμα της κλάσης **CartAddProductForm** ο χρήστης. Με χρήση του decorator **@require_POST** διασφαλίζεται ότι η συγκεκριμένη συνάρτηση θα εκτελεστεί μόνο σε περίπτωση που το request έχει πραγματοποιηθεί με τη μέθοδο POST.

```
def cart_remove(request, id,slug=None):
    cart = Cart(request)
    product = get_object_or_404(Photo, id=id)
    cart.remove(product)
    return redirect('cart:cart_detail')

def cart_detail(request):
 cart = Cart(request)
 return render(request,'cart_detail.html',{"cart":cart})

@login_required(redirect_field_name=None)
def checkout(request):
 cart=Cart(request)
 user=request.user
 amount=cart.get_total_price()
 if amount > 0:
     form=AuthenticationForm()
     context1={"form":form}
     if request.POST:
         if request.POST.get('checkuser')==request.user.username:
             if check_password(request.POST.get('checkpass'),request.user.password):
                 neworder=Order(user=request.user,amount=amount)
                 neworder.save()
                 cart.clear()
                 context2={"order_message":"Your order has been submitted!Thank you for using myphoto."}
                 send_mail('my photo order','Your order: %d euros. Thank you for using myphoto \n -the myphoto team'%amount,'myphot
                 return render(request,"order.html",context2)

         else:
             context2={"order_message":"Your credentials are invalid!Please try again."}
             return render(request,"order.html",context2)
     else:
         return render(request,'checkout.html',context1)
 else:
     context2={"order_message":"Your cart is empty! Please add something before checking out."}
     return render(request,"order.html",context2)
```

Εικόνα 85: Οι view συναρτήσεις cart_remove,cart_detail,checkout

β) cart_remove(request,id,slug=None):
Η συνάρτηση view **cart_remove()** με χρήση της μεθόδου **remove()** που ορίζεται στο αρχείο **cart.py,** υλοποιεί την αφαίρεση της φωτογραφίας που αντιστοιχεί στο **id** που δέχθηκε ως είσοδο από το αντικείμενο της κλάσης **Cart** που αντιστοιχεί στην τρέχουσα σύνοδο.

γ) cart_detail(request):
Η view **cart_detail()** δέχεται ως είσοδο ένα request και επιστρέφει ένα αρχείο template που παρουσιάζει το αντικείμενο της κλάσης **Cart** που αντιστοιχεί στην τρέχουσα σύνοδο.

δ) checkout(request):
Η συνάρτηση **checkout()** υλοποιεί την ολοκλήρωση της παραγγελίας του χρήστη, ως εξής: αρχικά ανακτά το αντικείμενο της κλάσης **Cart** που σχετίζεται με την σύνοδο που ανττιστοιχεί στο request που πραγματοποιήθηκε. Ύστερα ελέγχει εάν το συνολικό ποσό του καλαθιού είναι μεγαλύτερο από μηδέν, δηλαδή εάν το καλάθι είναι άδειο. Σε περίπτωση που είναι επιστρέφει ένα αρχείο template που εμφανίζει αντίστοιχο μήνυμα στον χρήστη. Σε περίπτωση που δεν είναι, με βάση τις πληροφορίες που έχουν εισαχθεί σε φόρμα της κλάσης **AuthenticationForm**, η συνάρτηση ελέγχει εάν το **username** και το **password** που θα εισαχθούν ταυτίζονται με αυτά του χρήστη που έχει πραγματοποιήσει το request. Εάν ταυτίζονται, με χρήση της συνάρτησης **django.core.mail.send_mail()** ο χρήστης ενημερώνεται για την παραγγελία του στη διεύθυνση e – mail που έχει δώσει κατά την εγγραφή του και με τη τη χρήση της μεθόδου **clear()** που έχει οριστεί στο **cart.py**, το καλάθι “αδειάζει”.

Ταυτόχρονα ένα αντικείμενο του model **Order** με το **username**, την ημερομηνία της παραγγελλίας και το ποσό της παραγγελίας δημιουργείται και αποθηκεύεται στη βάση

δεδομένων. Σε περίπτωση που το ζεύγος των διαπιστευτηρίων είναι λανθασμένο, η συνάρτηση **checkout()** επιστρέφει αρχείο template το οποίο εμφανίζει σχετικό μήνυμα.

5.3.2.5 Cart Urls

```
from django.conf.urls import url
from .views import cart_add,cart_remove,cart_detail,checkout
urlpatterns = [
    url(r'^$',cart_detail, name='cart_detail'),
    url(r'^add/photos/(?P<id>\d+)/$',
        cart_add,
        name='cart_add'),
    url(r'^remove(?P<id>\d+)/$',
        cart_remove,
        name='cart_remove'),
    url(r'^checkout/$',checkout,name='checkout')

]
```

Εικόνα 86: Το αρχείο urls.py της εφαρμογής cart

Το **URLconf** της υποεφαρμογής cart αποτελείται από τα εξής patterns:

α) το pattern / , το οποίο καλεί την view **cart_detail().**
β) το pattern /add/photo/?P<id>\d+), το οποίο καλεί την view **cart_add().**
γ) το pattern /remove/photo/?P<id>\d+), το οποίο καλεί την view **cart_remove().**
δ) το pattern /checkout, που καλεί τη view **checkout().**

5.3.2.6 Cart Admin

```

from django.contrib import admin
from .models import Order

class OrderAdmin (admin.ModelAdmin):
  list_display = ["user","date_created","amount","is_completed"]
  list_display_links=["user"]
  search_fields=["user","amount"]
  list_editable=["is_completed"]

  def has_add_permission(self, request, obj=None):
     return False

  class Meta:
    model = Order

admin.site.register(Order,OrderAdmin)
```

Εικόνα 87: Το αρχείο admin.py της εφαρμογής admin

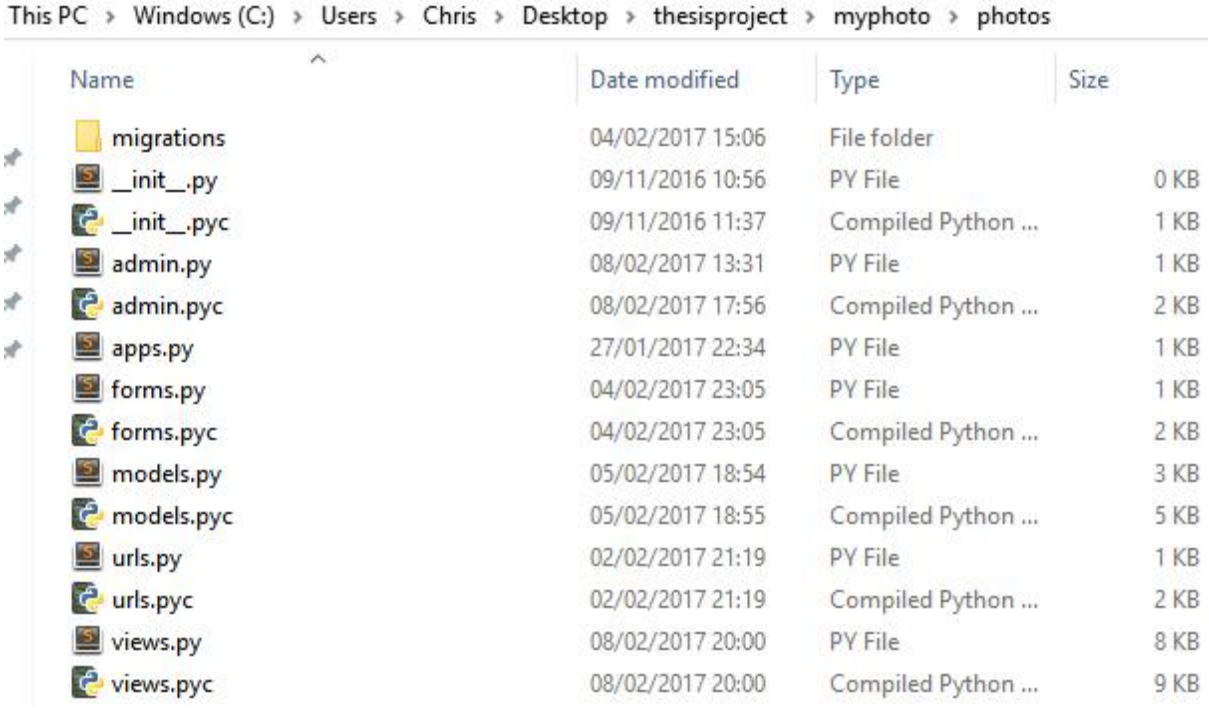
Το αρχείο admin.py περιέχει την υποκλάση της κλάσης **ModelAdmin**, **OrderAdmin** βάσει της

οποίας καθίσταται εφικτή η διαχείριση των αντικειμένων της κλάσης **Order** στο **admin interface**.

5.3.3 Η Εφαρμογή Photos

Η υποεφαρμογή **photos** υλοποιεί τις λειτουργίες που αφορούν στη διαχείριση των φωτογραφιών και των σχολίων από τους χρήστες.

Μεταβαίνοντας στον φάκελο photos, παρατηρούμε τα εξής αρχεία:



Εικόνα 88: Ο φάκελος myphoto

Στις επόμενες παραγράφους θα περιγραφεί αναλυτικότερα ο κώδικας που σχετίζεται με την εφαρμογή photos.

5.3.3.1 Photos Models

Στο αρχείο models.py ορίζονται οι συναρτήσεις **upload_location(), create_slug(), pre_save_post_receiver()** και τα models **Photo, Favorite** και **Comment**.

```python
from __future__ import unicode_literals
from django.conf import settings
from django.db import models
from django.core.urlresolvers import reverse
from django.db.models.signals import pre_save
from django.utils.text import slugify
from django.utils import timezone
from django.core.validators import MinValueValidator



def upload_location(instance,filename):
    user=instance.user
    return "%s/%s/%s" %(user.username,instance.slug,filename)

    def __unicode__(self):
        return self.name

class Photo(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,default=1)
    title = models.CharField(max_length=120)
    slug = models.SlugField(unique=True)
    categories=models.CharField(max_length=300)
    image=models.ImageField(upload_to=upload_location,null=True,blank=True,width_field="width_field",height_field="height_field")
    height_field=models.IntegerField(default=0)
    width_field=models.IntegerField(default=0)
    price = models.PositiveIntegerField(default=1,validators=[MinValueValidator(1)])
    likes=models.PositiveIntegerField(default=0)
    updated = models.DateTimeField(auto_now=True,auto_now_add=False)
    timestamp = models.DateTimeField(auto_now=False,auto_now_add=True)

    def total_likes(self):
        return self.likes.count()

    def __unicode__(self):
        return self.title

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse("photos:detail", kwargs={"slug":self.slug})

class Meta:
    ordering=['-likes']

class Favorite(models.Model):
    user=models.ForeignKey(settings.AUTH_USER_MODEL,default=1)
    photo=models.ForeignKey(Photo)
    created=models.DateTimeField(auto_now_add=True)

class Comment(models.Model):
 photo=models.ForeignKey('Photo',related_name='comments')
 author = models.ForeignKey(settings.AUTH_USER_MODEL,default=1)
 text = models.TextField()
 created_date = models.DateTimeField(default=timezone.now)


 def __unicode__(self):
        return self.text
```

Εικόνα 89: Η συνάρτηση upload_location και τα models Photo, Favorite, Comment

α) upload_location(instance,filename):
Η συνάρτηση **upload_location()** καθορίζει την τοποθεσία αποθήκευσης των αρχείων των φωτογραφιών που ανεβαίνουν στο myphoto. Το μονοπάτι ενός αποθηκευμένου αρχείου έχει την μορφή **username/slug/filename.img** όπου **username** το username του χρήστη που έχει ανεβάσει τη φωτογραφία, slug το slug της αντίστοιχης φωτογραφίας και **filename** το όνομα του αρχείου της φωτογραφίας.

β) class Photo:
η κλάση **Photo** είναι το model που χρησιμοποιείται για την αναπαράσταση των φωτογραφιών που ανεβαίνουν στο myphoto στη βάση δεδομένων.
Διαθέτει τα attributes:
i) user, ένα field της κλάσης **ForeignKey** στο model **django.contrib.auth.models.User** που χρησιμοποιείται για να καθορίσει τον χρήστη που ανέβασε τη φωτογραφία.

ii) title, ένα field της κλάσης **CharField** στο οποίο αποθηκεύεται το όνομα της φωτογραφίας.
iii) slug, ένα field της κλάσης **SlugField** στο οποίο αποθηκεύεται μοναδικό slug που αντιστοιχεί σε μοναδικό αντικείμενο της κλάσης **Photo** (με τρόπο που περιγράφεται παρακάτω).
iv) categories, ένα field της κλάσης **CharField** που χρησιμοποιείται για να αποθηκευτούν strings που καθορίζουν τις κατηγορίες στις οποίες ανήκει μια φωτογραφία
v)price, ένα field της κλάσης **PositiveIntegerField** που χρησιμοποιείται για να αναπαραστήσει την τιμή μιας φωτογραφίας. Η χρήση του **django.core.validators.MinValueValidator** διασφαλίζει ότι το attribute **price** κατά την αποθήκευση ενός αντικειμένου του model **Photo** θα είναι μεγαλύτερο ή ίσο του 1.
vi) updated, ένα field που καταγράφει την ημερομηνία και ώρα που μια φωτογραφία έχει υποστεί επεξεργασία.
vii) timestamp, ένα field που καταγράφει την ημερομηνία και ώρα που μία φωτογραφία ανέβηκε στο myphoto.
viii) image, ένα field της κλάσης **ImageField** που χρησιμοποιείται για να αποθηκεύσει στην βάση δεδομένων το αρχείο εικόνας που αντιστοιχεί σε αντικείμενο της κλάσης **Photo**.
ix) likes, ένα field που χρησιμοποιείται για να αποθηκεύσει τον αριθμό των αντικειμένων του model **Favorite** που συσχετίζονται με κάθε αντικείμενο του model **Photo**.
x) height_field, ένα field της κλάσης **IntegerField** που χρησιμοποιείται για την αποθήκευση του ύψους του αρχείου εικόνας που σχετίζεται με αντικείμενο του model Photo.
xi) width_field, ένα field της κλάσης **IntegerField** που χρησιμοποιείται για την αποθήκευση του πλάτους του αρχείου εικόνας που σχετίζεται με αντικείμενο του model Photo.

Το model **Photo** δέχεται επίσης την μέθοδο **get_absolute_url()** η οποία επιστρέφει το URL στο οποίο αντιστοιχεί η κάθε φωτογραφία. Αυτό γίνεται με χρήση του shortcut **reverse(). Επίσης, το** model **Photo** δέχεται την μέθοδο **total_likes()** η οποία επιστρέφει τον αριθμό των likes που έχει ένα αντικείμενο της κλάσης Photo.

γ) class Favorite:
Το model **Favorite** περιέχει τα attributes:
i) user, της κλάσης **ForeignKey** στο **model django.contrib.auth.models.User**.
ii) photo, της κλάσης **ForeignKey** στο model **Photo**. **iii) created** καταγράφει την ημερομηνία δημιουργίας του κάθε αντικειμένου του συγκεκριμένου model.
Τα αντικείμενα του model **Favorite** χρησιμοποιούνται για την καταγραφή τυχόν συσχέτισης των αντικειμένων model **User** και **Photo**, και δημιουργούνται ή διαγράφονται κατά την κλήση της view συνάρτησης **favorite()** (η οποία θα περιγραφεί παρακάτω).

δ) class Comment: Το model **Comment** έχει τα attributes:
i) author και
ii) photo, της κλάσης **ForeignKey** που αξιοποιούνται για την καταγραφή συσχέτισης ανάμεσα στα models **django.contrib.auth.User** και **Photo** αντίστοιχα.
iii) text, της κλάσης **TextField** που καταγράφει το περιεχόμενο των σχολίων.
iv) created_date,της κλάσης **DateTimeField** αποθηκεύει την ημερομηνία και ώρα δημιουργίας ενός σχολίου.
Το model **Comment** χρησιμοποιείται για την αναπαράσταση και αποθήκευση των σχολίων που αφήνουν οι χρήστες κάτω από μία φωτογραφία.

```
63  def create_slug(instance,new_slug=None):
64      slug = slugify(instance.title)
65      if new_slug is not None:
66          slug=new_slug
67      qs= Photo.objects.all().filter(slug=slug).order_by("-id")
68      exists=qs.exists()
69      if exists:
70          new_slug="%s-%s" %(slug,qs.first().id)
71          return create_slug(instance,new_slug=new_slug)
72      return slug
73
74
75  def pre_save_post_receiver(sender,instance,*args,**kwargs):
76      if not instance.slug:
77          instance.slug=create_slug(instance)
78
79
80
81
82
83  pre_save.connect(pre_save_post_receiver,sender=Photo)
```

Εικόνα 90: Οι συναρτήσεις create_slug, pre_save_post_receiver

ε) create slug(instance, new_slug=None):
Η συνάρτηση **create_slug(),** με χρήση της συνάρτησης **slugify()** που παρέχει το Django, δημιουργεί ένα μοναδικό slug για κάθε φωτογραφία που αποθηκεύεται. Το slug είναι ένα string της μορφής **photo-title** όπου “Photo Title” το attribute **title** του αντικειμένου του model **Photo** . Κατά την κλήση της, η **create_slug()** ελέγχει εάν υπάρχει άλλο ίδιο slug, δηλαδή αν ένα αντικείμενο του model **Photo** έχει ίδιο τίτλο. Σε αυτή την περίπτωση, το slug παίρνει την μορφή **photo-title-id** όπου “id” το id του συγκεκριμένου αντικειμένου του model **Photo**. Τα slugs χρησιμοποιούνται στην κατασκευή μοναδικού URL για κάθε φωτογραφία.

στ)pre_save_post_receiver(sender,instance,*args,kwargs):**
Η συνάρτηση **pre_save_post_receiver()** εκτελείται στην αρχή της κλήσης της μεθόδου **save()** για κάθε αντικείμενο του model Photo και διασφαλίζει τη δημιουργία slug για αυτό το αντικείμενο καλώντας την συνάρτηση **create_slug()**.

5.3.3.2 Photos Forms

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm

from .models import Photo,Comment
from django.contrib.auth.models import User

class PhotoForm(forms.ModelForm):
    class Meta:
        model=Photo
        fields= ["title","price","image","categories"]


class CommentForm(forms.ModelForm):
    class Meta:
     model=Comment
     fields=["text"]

```

Εικόνα 91: Το αρχείο forms.py της εφαρμογής photos

Στο αρχείο **forms.py** της υποεφαρμογής **photos** ορίζονται οι υποκλάσεις της κλάσης **ModelForm PhotoForm** και **CommentForm**. Η κλάση **PhotoForm** χρησιμοποιείται για τις φόρμες στις οποίες οι χρήστες υποβάλλουν στοιχεία για μια φωτογραφία, ενώ η **CommentForm** για την υποβολή σχολίων.

5.3.3.3 Photos Views

Μεταβαίνοντας στο αρχείο **views.py**, παρατηρούμε ότι τα views που υλοποιούνται στην υποεφαρμογή **photos** είναι τα εξής: **is_uploader(), home(), photo_upload(), photo_delete(), photo_edit(), photo_list(), photo_detail(), my_collection(), my_favorites(), favorite(), add_comment(), delete_comment().**

```
from django.contrib.auth.forms import UserCreationForm
from django.http import HttpResponse,HttpResponseRedirect,Http404
from django.shortcuts import render, get_object_or_404, redirect
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.db.models import Q
from django.core.exceptions import PermissionDenied
from django.core.mail import send_mail
from django.utils import timezone
from cart.forms import CartAddProductForm
from cart.models import Cart,Item
from django.contrib.auth.decorators import login_required

from accesslog.models import AccessAttempt
from accesslog.views import check_ip
from .models import Photo,Favorite,Comment
from .forms import PhotoForm,CommentForm


def is_uploader(function):
  def wrap(request,slug):
        user=request.user
        if user.is_authenticated()==False:
         return HttpResponseRedirect('/accounts/login')
        else:
         photo=get_object_or_404(Photo,slug=slug)
         if ( user==photo.user or user.is_staff ):
           return function(request,slug)
         else:
            raise PermissionDenied
  wrap.__doc__=function.__doc__
  wrap.__name__=function.__name__
  return wrap


def home(request):
 if request.user.is_authenticated:
   ip=check_ip(request)
   userlocked=AccessAttempt.objects.all().filter(user_ip=ip)
   if userlocked:
     userlocked.delete()
 context={
   "title":"my photo",
  }
 return render(request,"home.html",context)



@login_required(redirect_field_name=None)
def photo_upload(request):
 form = PhotoForm(request.POST or None,request.FILES or None)
 if form.is_valid():
   instance=form.save(commit=False)
   instance.user=request.user
   instance.save()
   return HttpResponseRedirect(instance.get_absolute_url())

 context = {
   "form":form
 }
 return  render(request,"photo_form.html",context)
```

Εικόνα 92: Οι view συναρτήσεις is_uploader, home, photo_upload

α) is_uploader(function):
Η **is_uploader()** είναι μια συνάρτηση που χρησιμοποιείται ως decorator σε συναρτήσεις που δέχονται ως είσοδο requests και slugs. Αρχικά, η **is_uploader()** με χρήση της μεθόδου **is_authenticated()** ελέγχει εάν ο χρήστης που πραγματοποίησε το request έχει συνδεθεί στο myphoto. Σε περίπτωση που δεν έχει, τον ανακατευθύνει στο URL που αντιστοιχεί στη σελίδα της σύνδεσης στο myphoto (login page). Σε περίπτωση που έχει, ανακτά το αντικείμενο του model **Photo** που αντιστοιχεί στο slug και ελέγχει εάν ο χρήστης είναι ο χρήστης που το έχει ανεβάσει ή εάν έχει αναβαθμισμένα δικαιώματα (με χρήση του boolean attribute **is_staff** του model **django.contrib.auth.models.User**). Σε περίπτωση που τουλάχιστον ένα από τα δύο ισχύει, η συνάρτηση **is_uploader()** επιστρέφει τη συνάρτηση την οποία έχει δεχθεί ως είσοδο. Στην αντίθετη περίπτωση επιστρέφει μία εξαίρεση **PermissionDenied**, υποδηλώνοντας ότι ο χρήστης που πραγματοποίησε το request δεν έχει δικαίωμα να εκτελέσει τη συνάρτηση.

β) home(request):
Η view **home()** χρησιμοποιείται για να επιστρέψει το template αρχείο που αντιστοιχεί στην κεντρική σελίδα του myphoto. Πριν από αυτό, με χρήση της συνάρτησης **check_ip()** και του model **AccessAttempt** που ορίζεται στην υποεφαρμογή accesslog, ελέγχει έαν υπάρχει καταχώρηση του model **AccessAttempt** που αντιστοιχεί στην IP του χρήστη που πραγματοποίησε το request. Εάν υπάρχει, η καταχώρηση αυτή διαγράφεται. Η λειτουργία αυτή υλοποιείται ώστε να διασφαλιστεί η διαγραφή των αποτυχημένων προσπαθειών σύνδεσης του χρήστη σε περίπτωση επιτυχούς σύνδεσης του (μιας και σε περίπτωση επιτυχούς σύνδεσης οι χρήστες ανακατευθύνονται στην κεντρική σελίδα του myphoto).

γ) photo_upload(request):
Η view **photo_upload()** δέχεται τον decorator **django.contrib.auth.decorators.login_required** ώστε η εκτέλεσή της να επιτρέπεται μόνο σε αυθεντικοποιημένους χρήστες.
Σε περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος, ανακατευθύνεται στο URL που αντιστοιχεί στη σελίδα σύνδεσης του myphoto. Με χρήση της **PhotoForm** η συγκεκριμένη view δημιουργεί φόρμα που χρησιμοποιείται για την υποβολή μιας φωτογραφίας στο myphoto. Σε περίπτωση που το request πραγματοποιηθεί με την μέθοδο POST, δηλαδή ο χρήστης έχει πραγματοποιήσει την υποβολή μιας φωτογραφίας, η view ελέγχει εάν τα στοιχεία που έχουν υποβοληθεί στα πεδία της φόρμας πιστοποιούνται με βάση τους validators που ορίζονται στο αρχείο **settings.py** (με χρήση της μεθόδου **is_valid()**). Αν πιστοποιούνται, ένα αντικείμενο του model **Photo** με attributes τα δεδομένα που έχουν εκχωρηθεί στα αντίστοιχα πεδία της φόρμας δημιουργείται. Το αντικείμενο αυτό αποθηκεύεται στη βάση δεδομένων.

```python
@is_uploader
def photo_edit(request,slug=None):
 user=request.user
 instance=get_object_or_404(Photo,slug=slug)
 form = PhotoForm(request.POST or None,request.FILES or None,instance=instance)
 if form.is_valid():
   instance=form.save(commit=False)
   instance.save()
   return HttpResponseRedirect(instance.get_absolute_url())
 else:
   context = {
     "title":instance.title,
     "instance":instance,
     "form":form
    }
   return render(request,"photo_form.html",context)

@is_uploader
def photo_delete(request,slug=None):
 instance=get_object_or_404(Photo,slug=slug)
 items=Item.objects.all().filter(object_id=instance.id)
 if items:
    items.delete()
 instance.delete()
 return redirect("photos:list")

def photo_list(request):
 queryset_list=Photo.objects.all().order_by("likes")
 query =request.GET.get('q')
 if query:
   queryset_list=queryset_list.filter(
       Q(title__icontains=query) |
       Q(user__username__icontains=query) |
       Q(categories__icontains=query)
      )
 paginator = Paginator(queryset_list,4)
 page = request.GET.get('page')
 try:
     queryset = paginator.page(page)
 except PageNotAnInteger:
     # If page is not an integer, deliver first page.
     queryset = paginator.page(1)
 except EmptyPage:
     # If page is out of range (e.g. 9999), deliver last page of results.
     queryset = paginator.page(paginator.num_pages)
 context= {
       "object_list":queryset,
       "title":"my photo collection",
       "failure":"There are not any results matching your search"
       }
 return render(request,"photo_list.html",context)
```

Εικόνα 93: Οι view συναρτήσεις photo_edit, photo_delete,photo_list

δ)photo_edit(request,slug=None):

Το **photo_edit()** δέχεται τον decorator **is_uploader()**. Στην εκτέλεσή του επιστρέφει μία φόρμα της κλάσης **PhotoForm** η οποία αντιστοιχεί στο αντικείμενο του model **Photo** που έχει το slug που η συνάρτηση view δέχθηκε ως είσοδο. Εάν κληθεί με την μέθοδο POST, η συγκεκριμένη view αποθηκεύει τα νέα δεδομένα που έχουν εισαχθεί την φόρμα ανανεώνοντας την καταχώρηση που αντιστοιχεί στο συγκεκριμένο αντικείμενο της κλάσης **Photo**.

ε)photo_delete(request,slug=None):

Σε περίπτωση που ο χρήστης επιτύχει τον έλεγχο που πραγματοποιεί ο decorator **is_uploader()**, η **photo_delete()** διαγράφει από τη βάση δεδομένων την καταχώρηση που αντιστοιχεί στο αντικείμενο του model **Photo** που έχει το συγκεκριμένο slug που έχει δεχθεί ως

είσοδο. Μετά την εκτέλεσή της, ανακατευθύνει το χρήστη στη σελίδα που απεικονίζονται όλες οι φωτογραφίες του myphoto.

στ)photo_list(request):
Η συνάρτηση view **photo_list()** επιστρέφει ένα αρχείο template (**photo_list.html**) το οποίο απεικονίζει στο χρήστη όλες τις φωτογραφίες που έχουν ανέβει στο myphoto. Με χρήση του **django.core.paginator**, οι καταχωρήσεις απεικονίζονται ανά τέσσερις σε διαφορετική σελίδα. Επίσης, εάν έχει πραγματοποιηθεί κάποια αναζήτηση στην μπάρα αναζήτησης, αξιοποιώντας instance της κλάσης **Q**, η view **photo_list()** επιστρέφει τα αντικείμενα του model **Photo** που περιέχουν το string που αναζητήθηκε σε κάποιο από τα attributes **title, user** ή **categories**.

```
def photo_detail(request,slug=None):
 user=request.user
 instance=get_object_or_404(Photo,slug=slug)
 if user.is_authenticated():
   liked=Favorite.objects.all().filter(user=request.user,photo_id=instance.id)
 else:
   liked=None;
 cart_product_form=CartAddProductForm()
 commentform=CommentForm()
 if instance.categories:
   categories=instance.categories
   categories_list=categories.split(',')
 context={
   "title":instance.title,
   "instance":instance,
   "liked":liked,
   "cart_product_form":cart_product_form,
   "commentform":commentform,
   "user":user,
   "categories":categories_list
   }
 return render(request,"photo_detail.html",context)


@login_required(redirect_field_name=None)
def my_collection(request):
 user=request.user
 queryset_list=Photo.objects.filter(user=user)
 query =request.GET.get('q')
 if query:
   queryset_list=queryset_list.filter(
       Q(title__icontains=query) |
       Q(description__icontains=query)
       )
 paginator = Paginator(queryset_list, 4)
 page = request.GET.get('page')
 try:
     queryset = paginator.page(page)
 except PageNotAnInteger:
     # If page is not an integer, deliver first page.
     queryset = paginator.page(1)
 except EmptyPage:
     # If page is out of range (e.g. 9999), deliver last page of results.
     queryset = paginator.page(paginator.num_pages)
 context = {
  "object_list":queryset,
  "title":user.username+"'s collection",
  "failure":"You have not uploaded anything yet!"
 }
 return render(request,"photo_list.html",context)
```

Εικόνα 94: Οι view συναρτήσεις photo_detail, my_collection

ζ) photo_detail(request,slug=None):
Η συνάρτηση view **photo_detail()** επιστρέφει ένα αρχείο template που απεικονίζει στον χρήστη την φωτογραφία που αντιστοιχεί στο slug που η **photo_detail()** έχει δεχθεί ως είσοδο. Σε περίπτωση που ο χρήστης είναι αυθεντικοποιημένος, η **photo_detail()** του παρέχει τη δυνατότητα να προσθέσει τη φωτογραφία στα αγαπημένα του.

η) my_collection(request): Η view **my_collection(),** σε περίπτωση που ο χρήστης είναι αυθεντικοποιημένος (που διασφαλίζεται με χρήση του decorator **login_required()**) επιστρέφει το template **photo_list.html** που απεικονίζει στο χρήστη τα αντικείμενα του model **Photo** που ο ίδιος έχει ανεβάσει. Με τρόπο παρόμοιο με την **photo_list()**, σε περίπτωση αναζήτησης στο αρχείο template απεικονίζονται τα κατάλληλα αποτελέσματα.

```
@login_required(redirect_field_name="None")
def favorite(request,slug):
 photo=get_object_or_404(Photo,slug=slug)
 user=request.user
 if request.user != photo.user:
   obj,created=Favorite.objects.get_or_create(user=request.user,photo_id=photo.id,photo=photo)
   if not created:
     photo.likes=photo.likes-1
     obj.delete()
   else:
     photo.likes=photo.likes+1
     obj.save()
   photo.save()
   return HttpResponseRedirect(photo.get_absolute_url())
 else:
    raise PermissionDenied

@login_required(redirect_field_name=None)
def add_comment(request,slug):
 instance = get_object_or_404(Photo,slug=slug)
 if request.method == "POST":
   commentform = CommentForm(request.POST)
   if commentform.is_valid():
     comment = commentform.save(commit=False)
     comment.photo = instance
     comment.author=request.user
     comment.save()
   else:
     commentform=CommentForm()
 return HttpResponseRedirect(instance.get_absolute_url())

@login_required(redirect_field_name=None)
def delete_comment(request,slug,pk):
 user=request.user
 instance=get_object_or_404(Photo,slug=slug)
 comment=get_object_or_404(Comment,photo=instance,pk=pk)
 if (comment.author==user or user.is_staff):
   comment.delete()
 return HttpResponseRedirect(instance.get_absolute_url())
```

Εικόνα 95: Οι view συναρτήσεις favorite, add_comment,delete_comment

θ) favorite(request,slug):
Η view **favorite()** εκτελείται σε περίπτωση που ο χρήστης είναι αυθεντικοποιημένος. Ελέγχει εάν υπάρχει αντικείμενο της κλάσης **Favorite** που να καταγράφει συσχέτιση του αντικειμένου της κλάσης **Photo** που αντιστοιχεί στο slug που η **favorite()** δέχεται ως είσοδο και του χρήστη που πραγματοποιεί το request. Σε περίπτωση που υπάρχει, η συγκεριμένη view το διαγράφει και μειώνει το attribute **likes** του αντίστοιχου αντικειμένου της κλάσης **Photo** κατά 1. Σε περίπτωση που δεν υπάρχει, η συνάρτηση **favorite()** δημιουργεί το αντικείμενο και αυξάνει το attribute **likes** κατά 1. Η συνάρτηση ύστερα ανακατευθύνει τον χρήστη στο URL που αντιστοιχεί στο template που απεικονίζει την κατάλληλη φωτογραφία.

ι) add_comment(request):
Η συνάρτηση **add_comment()**,κληθεί με την μέθοδο POST και ο χρήστης είναι αυθεντικοποιημένος, δημιουργεί ένα instance της κλάσης **CommentForm** το οποίο δέχεται τα δεδομένα που έχει εισάγει ο χρήστης και ύστερα τα αποθηκεύει σε ένα αντικείμενο του model **Comment**. Στη συνέχεια, η συνάρτηση **add_comment()** ανακατευθύνει τον χρήστη στο URL

που αντιστοιχεί στην κατάλληλη φωτογραφία.

κ) delete_comment(request,slug,pk): Η συνάρτηση **delete_comment()** δέχεται ως είσοδο το request που πραγματοποίησε ο χρήστης, το slug στο οποίο αντιστοιχεί ένα αντικείμενο της κλάσης **Photo** και ένα πρωτεύον κλειδί στο οποίο αντιστοιχεί ένα αντικείμενο της κλάσης **Comment** . Με τη χρηση της γραμμής **get_object_or_404()** ανακτά από τη βάση δεδομένων ένα αντικείμενο του model **Comment** που να αντιστοιχεί στα δεδομένα που έχει δεχθεί ως είσοδο ή επιστρέφει σφάλμα . Σε περίπτωση επιτυχούς εύρεσής του, η καταχώρηση που αντιστοιχεί στο αντικείμενο αυτό διαγράφεται.

```
208  @login_required(redirect_field_name=None)
209  def my_favorites(request):
210   user=request.user
211   favorites=user.favorite_set.all()
212   queryset_list=[favorite.photo for favorite in favorites]
213   query =request.GET.get('q')
214   if query:
215    queryset_list=queryset_list.filter(
216        Q(title__icontains=query) |
217        Q(description__icontains=query) |
218        Q(user__username__icontains=query)
219        )
220   paginator = Paginator(queryset_list, 4)
221   page = request.GET.get('page')
222   try:
223       queryset = paginator.page(page)
224   except PageNotAnInteger:
225       # If page is not an integer, deliver first page.
226       queryset = paginator.page(1)
227   except EmptyPage:
228       # If page is out of range (e.g. 9999), deliver last page of results.
229       queryset = paginator.page(paginator.num_pages)
230   context={
231       "object_list":queryset,
232       "title":user.username+"'s favorites",
233       "failure":"You have no favorites yet!"
234    }
235   return render(request,"photo_list.html",context)
```

Εικόνα 96: Η view συνάρτηση my_favorites

λ) my_favorites(request):
Η view συνάρτηση **my_favorites()** επιστρέφει στον χρήστη που πραγματοποίησε το request το template **photo_list.html** το οποίο αναπαριστά τα αντικείμενα της κλάσης **Photo** για τα οποία υπάρχει αντικείμενο του model **Favorite** που υποδηλώνει συσχέτιση. Παρέχει επίσης τη δυνατότητα αναζήτησης με τρόπο όμοιο με τη συνάρτηση view **photo_list()**.

5.3.3.4 Photos Urls

```
from django.conf.urls import url
from .views import (photo_detail,photo_upload,photo_list,my_collection,favorite,my_favorites,photo_edit,photo_delete,add_comment,de

urlpatterns = [
 url(r'^upload/',photo_upload,name="upload"),
 url(r'^collection/',my_collection,name="my_collection"),
 url(r'^favorites/',my_favorites,name="my_favorites"),
 url(r'^$',photo_list,name='list'),
 url(r'^(?P<slug>[\w-]+)/$',photo_detail,name="detail"),
 url(r'^(?P<slug>[\w-]+)/edit/$',photo_edit,name='edit'),
 url(r'^(?P<slug>[\w-]+)/delete/',photo_delete,name='delete'),
 url(r'^(?P<slug>[\w-]+)/favorite/$',favorite,name="favorite"),
 url(r'^(?P<slug>[\w-]+)/add_comment/$',add_comment,name="add_comment"),
 url(r'^(?P<slug>[\w-]+)/(?P<pk>\d+)/delete_comment/$',delete_comment,name="delete_comment")
]
```

Εικόνα 97: Το αρχείο urls.py της εφαρμογής photos

Όπως φαίνεται και στην εξής εικόνα το **URLconf** της υποεφαρμογής photos περιέχει τα εξής patterns:
α) το pattern upload/, το οποίο αντιστοιχεί στη view **photo_upload().**
β) το pattern collection/, το οποίο αντιστοιχεί στη view **my_collection().**
γ) το pattern /,το οποίο αντιστοιχεί στη view **photo_list().**
δ)το pattern (?P<slug>[\w-]+)/, το οποίο αντιστοιχεί στη view **photo_detail().**
ε)το pattern (?P<slug>[\w-]+)/edit,το οποίο αντισοιχεί στη view **photo_edit().**
στ)το pattern (?P<slug>[\w-]+)/delete,το οποίο αντιστοιχεί στη view **photo_delete().**
ζ)το pattern (?P<slug>[\w-]+)/favorite,το οποίο αντιστοιχεί στη **view favorite().**
η)το pattern r'^(?P<slug>[\w-]+)/add_comment/,το οποίο αντιστοιχεί στη view **add_comment().**
θ)το pattern r'^(?P<slug>[\w-]+)/(?P<pk>\d+)/delete_comment/,το οποίο αντιστοιχεί στη view **delete_comment().**

5.3.3.5 Photos Admin

```python
from django.contrib import admin
from django.contrib import messages

from .models import Photo,Comment


class PhotoModelAdmin (admin.ModelAdmin):
  list_display = ["title","timestamp","updated"]
  list_display_links=["updated"]
  list_filter=["timestamp"]
  search_fields=["title"]


  class Meta:
    model = Photo


class CommentAdmin(admin.ModelAdmin):
 list_display=["author","created_date","text","photo"]
 list_display_links=["author"]
 search_fields=["author"]

 class Meta:
    model=Comment


admin.site.register(Photo,PhotoModelAdmin)
admin.site.register(Comment,CommentAdmin)
```

Εικόνα 98: Το αρχείο admin.py της εφαρμογής photos

Στο αρχείο **admin.py** της υποεφαρμογής photos, ορίζονται οι κλάσεις **CommentAdmin** και **PhotoAdmin** που χρησιμοποιούνται για την διαχείριση των αντικειμένων των models **Comment** και **Photo** αντίστοιχα στο admin interface.

5.3.4 URLs της Διαδικτυακής Εφαρμογής myphoto

```python
from django.conf.urls import url,include
from django.contrib import admin
from photos import views
from django.conf import settings
from django.conf.urls.static import static


urlpatterns = [
    url(r'^myphoto_root/', admin.site.urls),
    url(r'^photos/', include("photos.urls",namespace="photos")),
    url(r'^$',views.home,name="home"),
    url(r'^cart/', include('cart.urls', namespace='cart')),
    url(r'^accounts/',include("accesslog.urls",namespace="auth"))
```

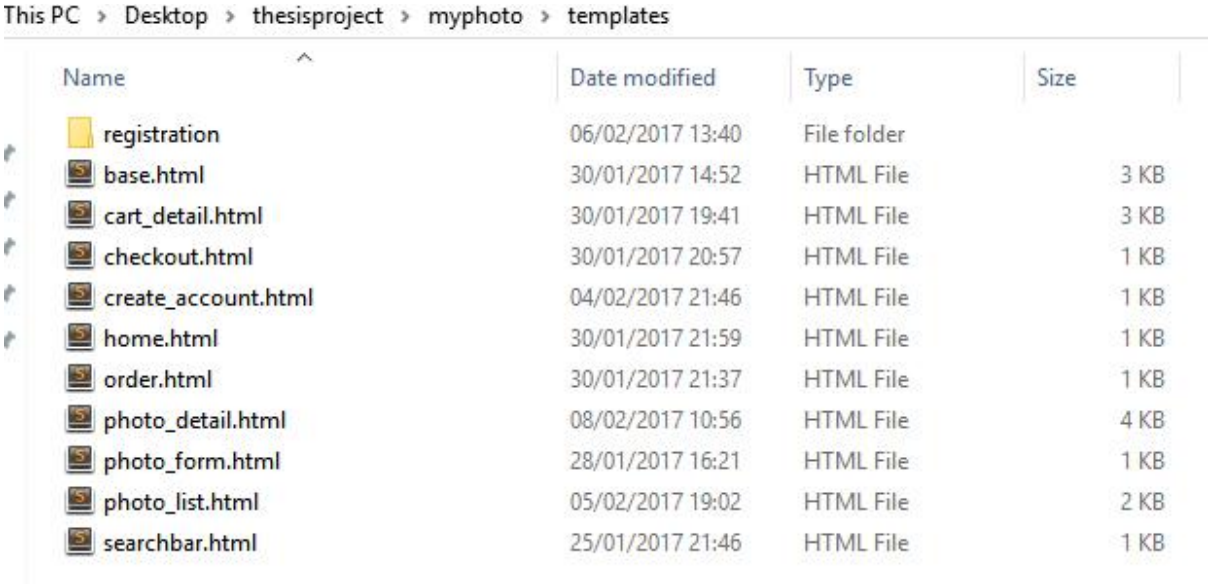
Εικόνα 99: Το αρχείο urls.py της εφαρμογής myphoto

Όπως φαίνεται στην εικόνα, το κεντρικό **URLconf** του myphoto περιλαμβάνει τα εξής urlpatterns:
α) το pattern /myphoto_root, που αντιστοιχεί στα **admin.site.urls** δηλαδή στα urlpatterns που περιλαμβάνει το **django.contrib.admin** και αντιστοιχούν στο admin interface του Django.

β) το pattern /photos που περιλαμβάνει το **URLconf** της υποεφαρμογής photos.
γ) το pattern / που αντιστοιχεί στην view συνάρτηση **home()**.
δ) το pattern /cart που περιλαμβάνει το **URLconf** της υποεφαρμογής cart.
e) το pattern /accounts που περιλαμβάνει το **URLconf** της υποεφαρμογής accesslog.

5.3.5 Templates της Διαδικτυακής Εφαρμογής myphoto

Τα templates της διαδικτυακής εφαρμογής περιέχονται στους φάκελους **templates** και **templates/registration**. Στον φάκελο **templates** περιέχονται τα αρχεία .html που καθορίζουν τον τρόπο απεικόνισης των δεδομένων που επιστρέφουν τα views των υποεφαρμογών photos και cart.



Εικόνα 100: Ο φάκελος templates της εφαρμογής myphoto

Το αρχείο **base.html** είναι το αρχείο template το οποίο κληρονομούν όλα τα υπόλοιπα με τρόπο

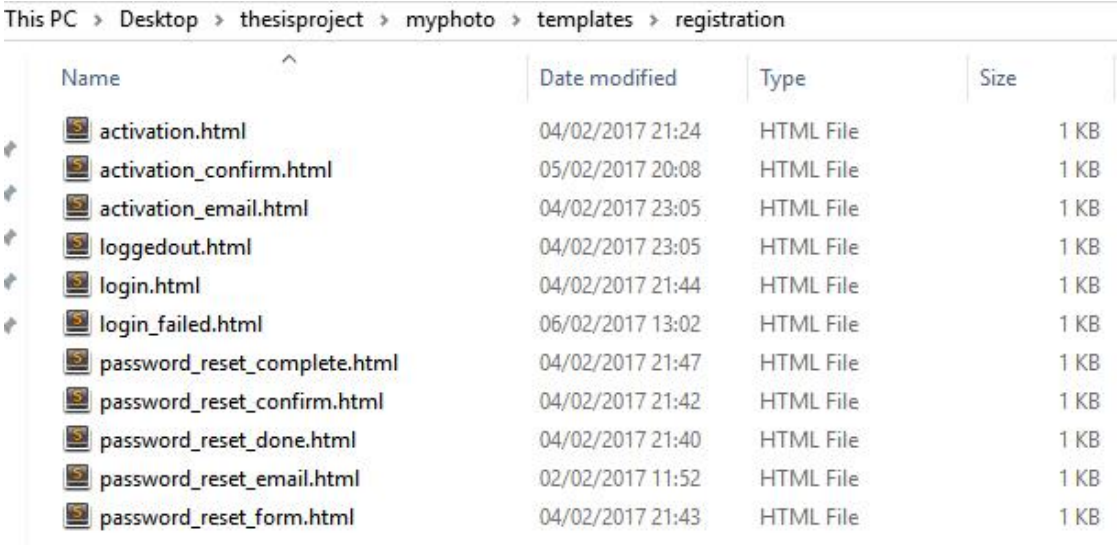
που περιγράφεται στην παράγραφο 3.7.3.

```
<html>
<!--Title-->
<head><title>{% block head_title %}my photo{% endblock head_title %}</title>

<!--Static Files-->
{% load static %}


<link rel="stylesheet" href='{% static "css/bootstrap.css" %}'>
<script src="{% static 'prevention.js' %}"></script>

<link rel="stylesheet" href='{% static "css/bootstrap-theme.min.css" %}'>


<!--Fonts-->
<link href="https://fonts.googleapis.com/css?family=Bungee+Hairline|Cuprum|Fjalla+One|Oswald|Poiret+One|Quicksand" rel="stylesheet">

<link rel="stylesheet" href='{%static "font-awesome/css/font-awesome.css" %}'>
<link rel="stylesheet" href='{%static "font-awesome/css/font-awesome.min.css" %}'>
<link rel="shortcut icon" type="image/png" href='{%static "/favicon.png" %}'>
<link rel='stylesheet' href='{% static "css/base.css" %}'/>

</head>


<body>
<!--Javascript-->
<script src="{% static 'js/prevention.js' %}"></script>


<h2 class="title"><i class="fa fa-camera fa-fw"></i>my photo</h2>
<div class='container'>
<nav class="navbar navbar-inverse">
 <ul class="nav navbar-nav">
   <li class="nav-item"> <a class="nav-link" href="{% url 'home' %}"><i class="fa fa-home fa-fw"></i>home</li></a>
      <li class="nav-item">
      <a class="nav-link" href="{% url 'photos:list' %}"><i class="fa fa-picture-o fa-fw"></i> all photos</a>
      </li>
      <li class="nav-item">
      <a class="nav-link" href="{%url 'photos:upload' %}"><i class="fa fa-upload fa-fw "></i> upload a photo</a>
      </li>
 {% if request.user.is_authenticated %}
      <li class="nav-item">
      <a class="nav-link" href="{% url 'photos:my_favorites' %}"><i class="fa fa-heart fa-fw"></i> my favorites</a>
      </li>
      <li class="nav-item">
      <a class="nav-link" href="{% url 'photos:my_collection' %}"><i class="fa fa-camera fa-fw"></i> my collection</a>
      </li>
      <li class="nav-item">
      <a class="nav-link" href="{% url 'cart:cart_detail' %}"><i class="fa fa-shopping-cart fa-fw"></i> my cart</a>
      </li>
      <li class="nav-item">
      <a class="nav-link" href="{%url 'auth:logout' %}"><i class="fa fa-sign-out fa-fw"></i> log out</a>
      </li>
      </ul>
 {% else %}
     <li class="nav-item">
     <a class="nav-link" href="{% url 'auth:create_account' %}"><i class="fa fa-user fa-fw"></i> create account</a>
     </li>
     <li class="nav-item">
     <a class="nav-link" href="{% url 'auth:login' %}"><i class="fa fa-sign-in fa-fw"></i> log in</a>
     </li>
 {% endif %}
      </ul>
      {% include "searchbar.html" %}
  </nav>

{% block content %}
```

Εικόνα 101: Το template αρχείο base.html

Στον φάκελο **templates/registration** περιέχονται τα αρχεία .html που καθορίζουν τον τρόπο

απεικόνισης των δεδομένων που επιστρέφονται από τα views της υποεφαρμογής accesslog.



Εικόνα 106: Ο φάκελος registration στην εφαρμογή myphoto

Όλα τα αρχεία .html του φάκελου **templates** διαμορφώνονται με κώδικα CSS, με αξιοποίηση του κώδικα του **Bootstrap** και του **Font - Awesome** που βρίσκονται στον φάκελο **static**.

5.4 Η Ασφάλεια στη Διαδικτυακή Εφαρμογή myphoto

Έχοντας περιγράψει τη λειτουργία της διαδικτυακής εφαρμογής που δημιουργήθηκε με χρήση Django, σε αυτή την παράγραφο περιγράφεται η ενσωμάτωση και η αξιοποίηση των εργαλείων που προσφέρει το Django και σχετίζονται με την ασφάλεια μιας διαδικτυακής εφαρμογής.

5.4.1 Αυθεντικοποίηση Χρηστών

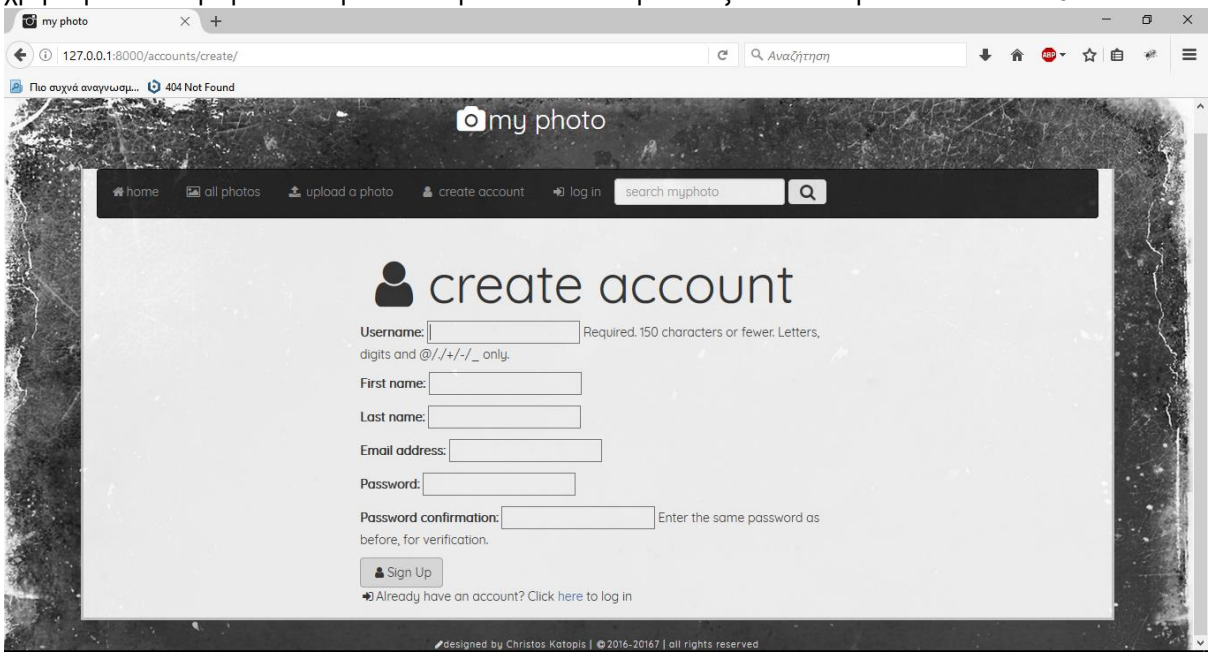
Για τις ανάγκες αυθεντικοποίησης και εξουσιοδότησης των χρηστών, η διαδικτυακή εφαρμογή myphoto αξιοποιεί την εφαρμογή **django.contrib.auth** και τις δυνατότητες ασφάλειας που αυτή προσφέρει. Το γεγονός αυτό σημαίνει ότι ο διαχειριστής του server, δηλαδή του υπολογιστικού συστήματος στο οποίο είναι αποθηκευμένος ο πηγαίος κώδικας του myphoto , έχει τη δυνατότητα να διαχειριστεί τους χρήστες του myphoto μέσω του command prompt με τρόπους που αναφέρονται στις παραγράφους 4.1 και 4.2. Στις παραγράφους που ακολουθούν θα γίνει περιγραφή των λειτουργιών που υλοποιούνται με απομακρυσμένη πρόσβαση, δηλαδή μέσα από το περιβάλλον διεπαφής χρήστη της διαδικτυακής εφαρμογής myphoto.

5.4.1.1 Δημιουργία Χρήστη

Για την αποθήκευση και την αναπαράσταση των χρηστών, χρησιμοποιείται το model **django.contrib.auth.models.User**, που περιγράφεται αναλυτικά στην παράγραφο 4.1.1. Η δημιουργία χρήστη, γίνεται με χρήση της φόρμας της κλάσης **UserForm**.

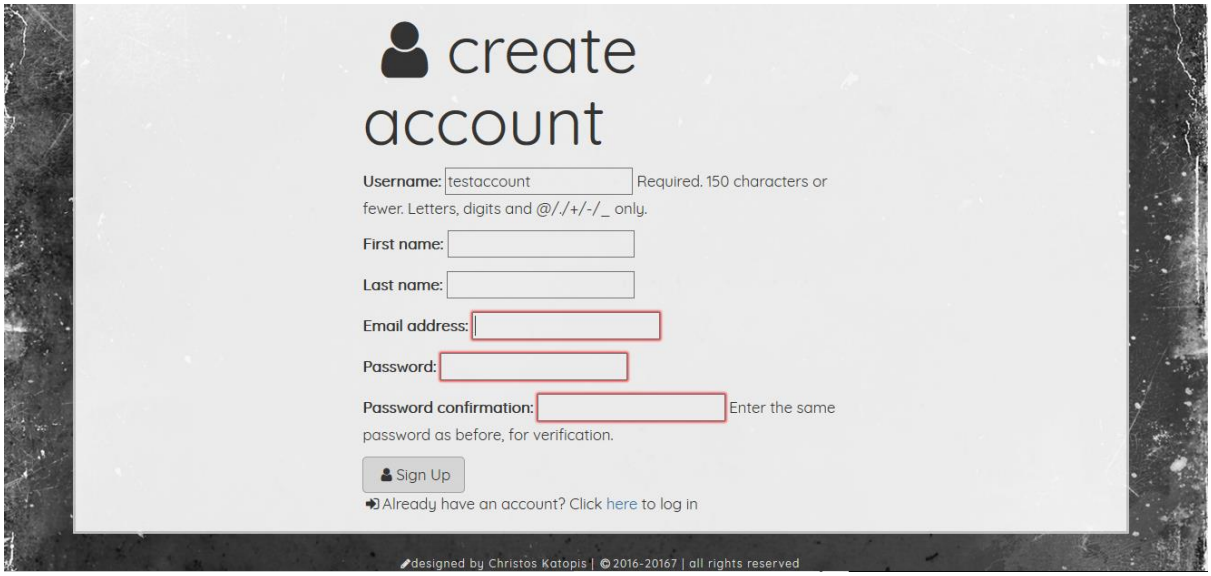
Το myphoto παρέχει δύο τρόπους δημιουργίας χρήστη. Ο πρώτος είναι μέσω του myphoto. Ο δεύτερος είναι μέσω του admin interface από χρήστη με κατάλληλα δικαιώματα. Σε αυτή την παράγραφο θα περιγραφεί ο πρώτος.

Προκειμένου ένας μη αυθεντικοποιημένος χρήστης να δημιουργήσει λογαριασμό, αρκεί στην κεντρική σελίδα να επιλέξει την επιλογή **create account** από το κεντρικό menu. Τότε, στον χρήστη επιστρέφεται η οθόνη που παρουσιάζεται στην εικόνα 107 .



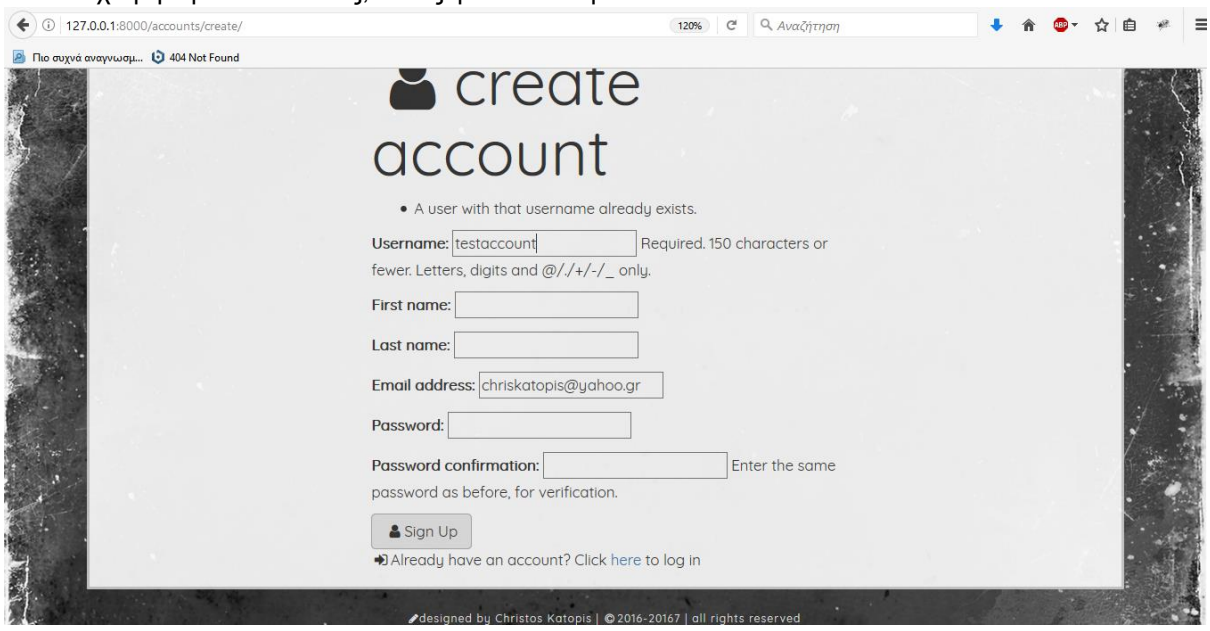
Εικόνα 107: Δημιουργία χρήστη στην διαδικτυακή εφαρμογή myphoto

Με την υποβολή των στοιχείων, η μέθοδος **is_valid()** (που περιγράφεται στην παράγραφο 3.8.2), που παρέχει το προγραμματιστικό πλαίσιο Django και χρησιμοποιεί η view συνάρτηση **create_account()**, ελέγχει εάν τα στοιχεία ικανοποιούν τους validators που χρησιμοποιεί το Django και η εφαρμογή myphoto. Οι validators πιστοποιούν ότι ο χρήστης θα εισάγει οπωσδήποτε **username** και **password**, ότι το **username** του χρήστη δεν υπάρχει στη βάση δεδομένων του myphoto, ότι ο κωδικός ικανοποιεί συγκεκριμένα κριτήρια, ότι οι δύο κωδικοί ταριάζουν, και, λόγω της κλάσης **UserForm**, ότι ο χρήστης θα εισάγει οπωσδήποτε έγκυρη διεύθυνση e – mail.



Εικόνα 108: Μήνυμα λάθους σε περίπτωση παράλειψης εισαγωγής κωδικού κατά τη δημιουργία χρήστη

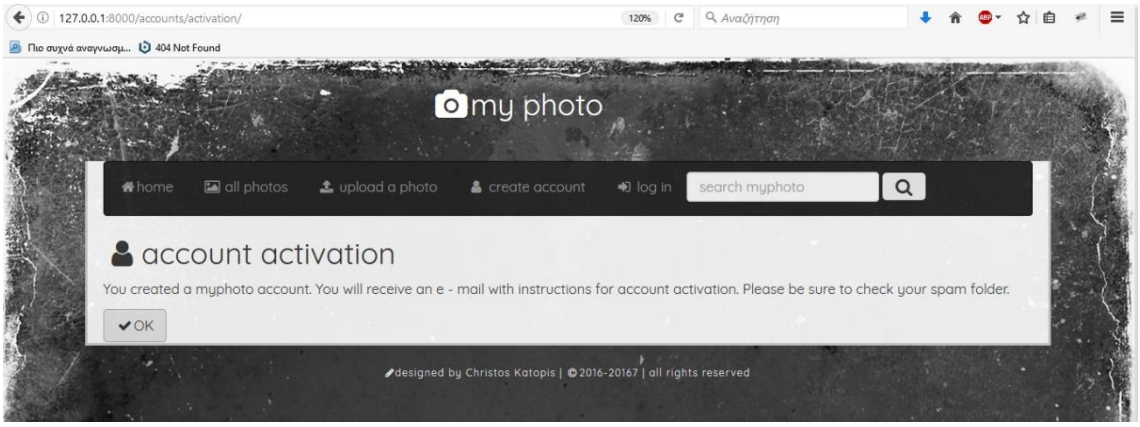
Σε περίπτωση που ένα από τα στοιχεία που εισήγαγε δεν ικανοποιεί τους ελέγχους των validators, ο χρήστης κατευθύνεται στη σελίδα δημιουργίας του λογαριασμού και λαμβάνει το αντίστοιχο μήνυμα του λάθους, όπως φαίνεται στην εικόνα.



Εικόνα 109: Μήνυμα σε περίπτωση εισαγωγής username το οποίο χρησιμοποιεί άλλος χρήστης

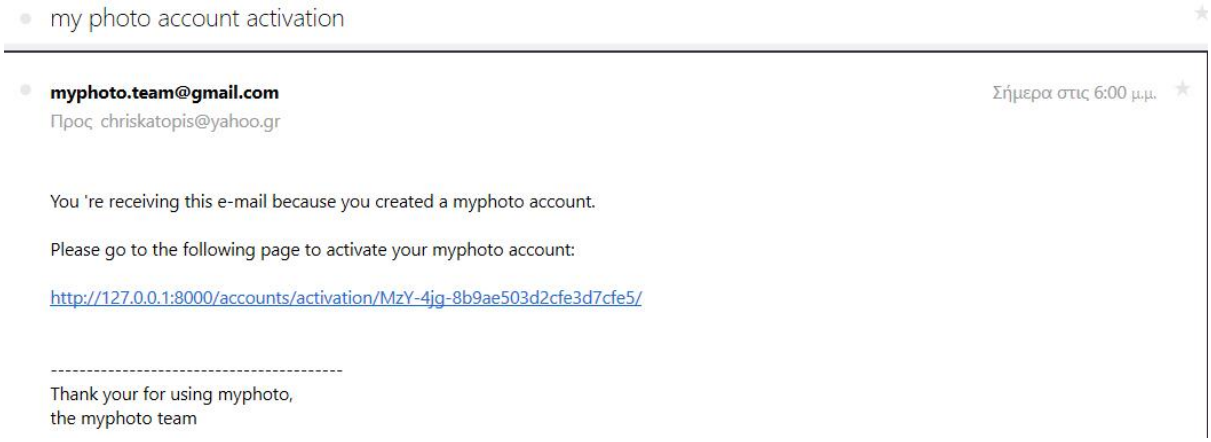
Σε περίπτωση που τους ικανοποιεί, ένα αντικείμενο της model κλάσης **User** με attributes τα στοιχεία που καταχωρήθηκαν δημιουργείται. Το attribute **is_active** λαμβάνει την τιμή **False**, με αποτέλεσμα ο χρήστης να μην μπορεί να συνδεθεί στο myphoto απευθείας.

Με την επιτυχή δημιουργία λογαριασμού, ο χρήστης λαμβάνει ένα μήνυμα που τον πληροφορεί σχετικά και τον ενημερώνει για τη διαδικασία ενεργοποίησης λογαριασμού.



Εικόνα 110: Μήνυμα επιτυχούς δημιουργίας λογαριασμού

Ταυτόχρονα, λαμβάνει ένα e – mail στη διεύθυνση που έχει εισάγει, με ένα URL που πρέπει να ακολουθήσει προκειμένου να έχει τη δυνατότητα σύνδεσης στο myphoto.



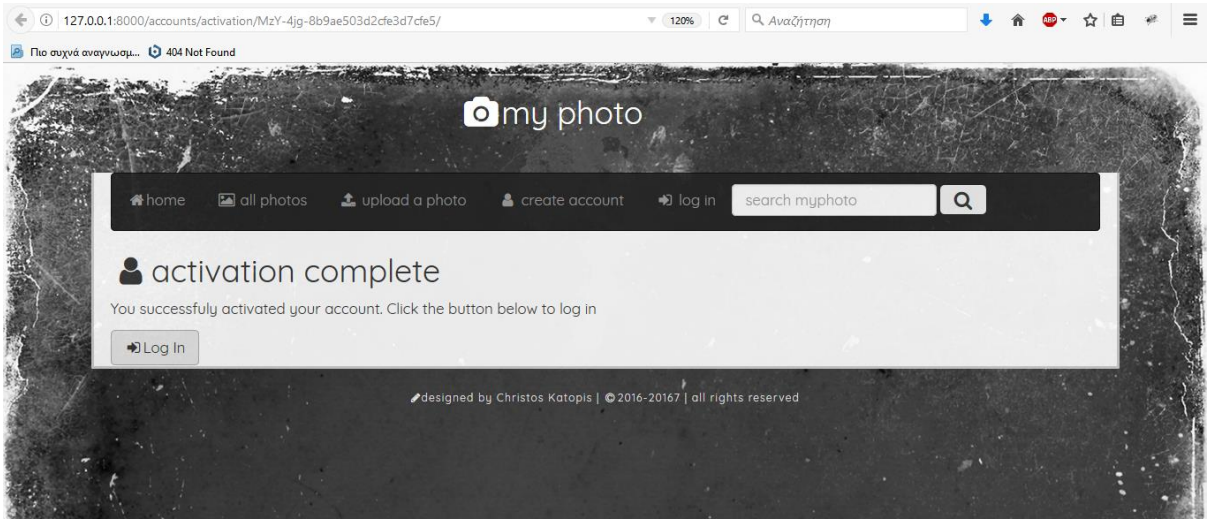
Εικόνα 111: E - mail ενεργοποίησης λογαριασμού χρήστη

Το URL αυτό δημιουργείται με βάση το κρυπτογραφημένο πρωτεύον κλειδί του αντικειμένου του model **User** που δημιουργείται κατά την κλήση της view **create_account()** (με χρήση της συνάρτησης **urlsafe_base64_encode()**) και ένα μοναδικό token που αντιστοιχεί στον χρήστη και δημιουργείται με χρήση του **default_token_generator** που παρέχει το Django. Το request για το συγκεκριμένο URL καλεί την view **activation_confirm()** που αποκρυπτογραφεί το πρωτεύον κλειδί που δέχεται ως είσοδο, ανακτά το αντικείμενο του model **User** που αντιστοιχεί σε αυτό και με χρήση της συνάρτησης **default_token_generator.check_token(user,token)** ελέγχει εάν το ζεύγος **token** και **user** είναι έγκυρο. Σε περίπτωση που είναι, η view συνάρτηση **activation_confirm()** ελέγχει πόσες ημέρες έχουν περάσει από την ημερομηνία δημιουργίας του αντικειμένου του model **User** (δηλαδή τις ημερές που έχουν περάσει από την ημερομηνία δημιουργίας του λογαριασμού και τις συγκρίνει με την μεταβλητή **ACCOUNT_ACTIVATION_DAYS** που βρίσκεται στο αρχείο **myphoto/settings.py**).

```
ACCOUNT_ACTIVATION_DAYS=7
```

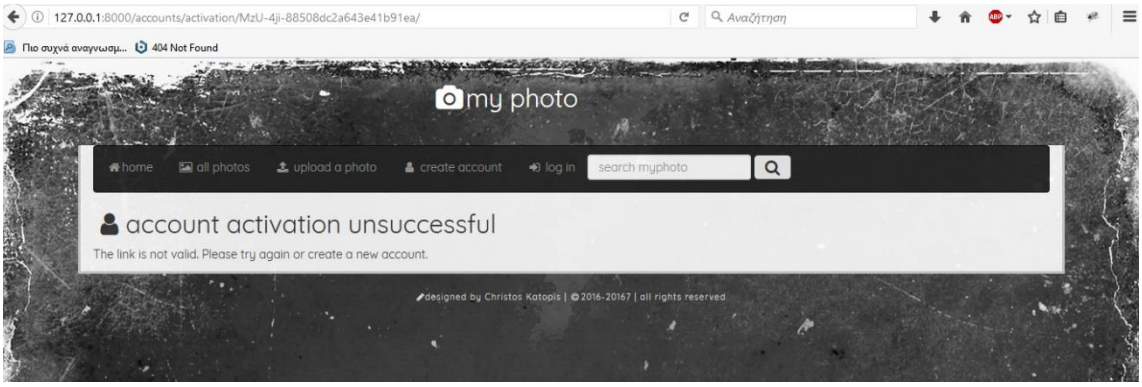
Εικόνα 112: Ορισμός της μεταβλητής ACCOUNT_ACTIVATION_DAYS

Εάν οι ημέρες που έχουν περάσει είναι λιγότερες από την τιμή του **ACCOUNT_ACTIVATION_DAYS**, το attribute **is_active** του κατάλληλου χρήστη λαμβάνει την τιμή **True** και στον χρήστη επιστρέφεται αρχείο template που εμφανίζει το μήνυμα που φαίνεται στην εικόνα 112.



Εικόνα 113: Επιτυχής ενεργοποίηση λογαριασμού χρήστη

Σε περίπτωση που κάτι από τα παραπάνω δεν ισχύει, η συνάρτηση **activation_confirm()** επιστρέφει αρχείο template που εμφανίζει κατάλληλο μήνυμα, ενώ η τιμή του attribute **is_active** του αντικειμένου **User που** αναπαριστά τον χρήστη παραμένει **False**.

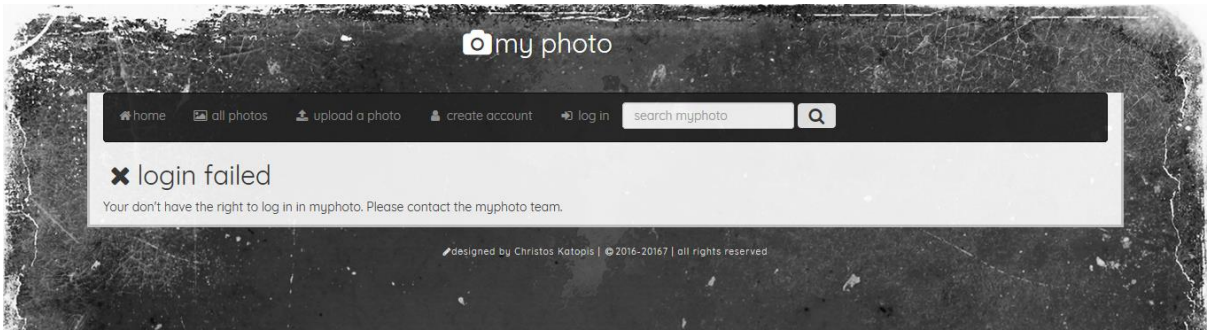


Εικόνα 114: Μήνυμα αποτυχίας ενεργοποίησης λογαριασμού χρήστη

5.4.1.2 Είσοδος Χρήστη

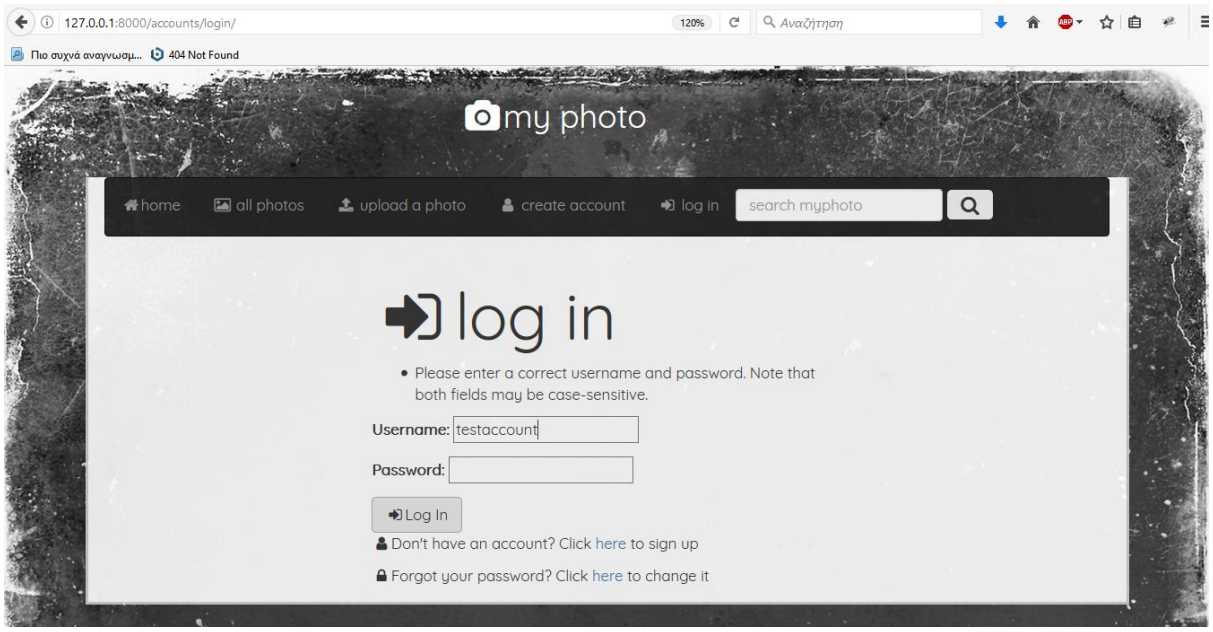
Η είσοδος ενός μη αυθεντικοποιημένου χρήστη επιτυγχάνεται με την επιλογή του **log in** από το κεντρικό menu. Υλοποιείται από τη view συνάρτηση **is_locked_out()** η οποία δέχεται τον decorator **is_logged_in()** ώστε να διασφαλιστεί ότι η κλήση της μπορεί να γίνει μόνο από χρήστη που δεν έχει πραγματοποιήσει είσοδο στο myphoto στην τρέχουσα σύνοδο.

Η συνάρτηση **is_locked_out(),** αρχικά, χρησιμοποιεί την **check_ip()** προκειμένου να διαπιστωθεί εάν η IP από την οποία ο χρήστης πραγματοποιεί το request έχει το δικαίωμα σύνδεσης ή όχι (ελέγχοντας το boolean attrubute του **blocked** του κατάλληλου αντικειμένου της model κλάσης **AccessAttempt**).



Εικόνα 115: Μήνυμα αποτυχίας εισόδου σε περίπτωση που η IP του χρήστη έχει μπλοκαριστεί

Εάν επιτρέπεται, η **is_locked_out()** ελέγχει τις αποτυχημένες προσπάθειες σύνδεσης που έχει πραγματοποιήσει ο χρήστης. Σε περίπτωση που δεν έχει ξεπεραστεί το ανώτατο όριο αποτυχημένων προσπαθειών (4) η **is_locked_out()** επιστρέφει την συνάρτηση **django.contrib.auth.views.login()**.

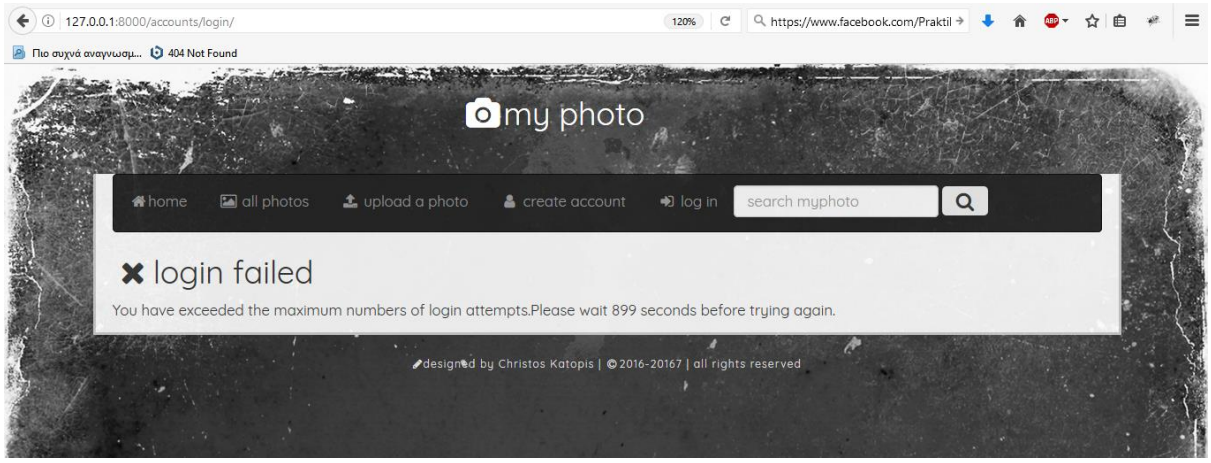


Εικόνα 116: Μήνυμα λάθους σε περίπτωση εισαγωγής λάθους διαπιστευτηρίων κατά την είσοδο χρήστη στο myphoto

Σε περίπτωση που οι αποτυχημένες προσπάθειες υπερβαίνουν τις τέσσερις, η **is_locked_out()** δεν επιτρέπει στον χρήστη να συνδεθεί για χρονικό διάστημα που καθορίζεται από την τιμή της μεταβλητής **LOCKOUT_TIME** (που ορίζεται στο αρχείο **settings.py**), επιστρέφοντάς του αρχείο template που εμφανίζει το κατάλληλο μήνυμα.

```
LOCKOUT_TIME=900
```

Εικόνα 117: Καθορισμός μεταβλητής LOCKOUT_TIME στο αρχείο settings.py



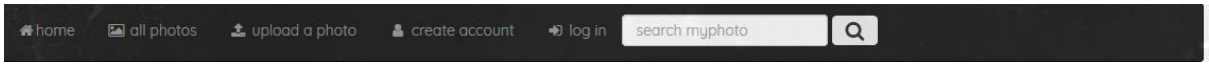
Εικόνα 118: Μήνυμα σε περίπτωση πολλών αποτυχημένων προσπαθειών εισόδου στο myphoto

Η επιτυχής είσοδος του χρήστη, δηλαδή η εισαγωγή έγκυρου ζεύγους διαπιστευτηρίων κατά τη διάρκεια εισόδου, έχει ως αποτέλεσμα την ανακατεύθυνση του χρήστη στην αρχική σελίδα και τη διαγραφή του αντικειμένου της model κλάσης **AccessAttempt** που καταγράφει τις αποτυχημένες προσπάθειες σύνδεσης.

5.4.2 Δικαιώματα και Εξουσιότηση

Οι λειτουργίες του myphoto μπορούν να χρησιμοποποιηθούν από αυθεντικοποιημένους και μη αυθεντικοποιημένους χρήστες.

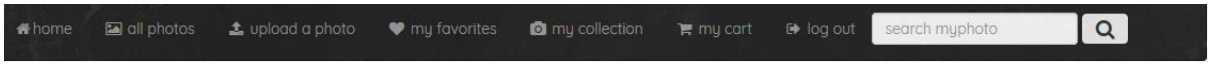
Ένας μη αυθεντικοποιημένος χρήστης, κατά την προβολή οποιαδήποτε σελίδας του myphoto, παρατηρεί το menu που φαίνεται στην εικόνα.



Εικόνα 119: Μενού myphoto για μη – αυθεντικοποιημένο χρήστη

Ένας μη αυθεντικοποιημένος χρήστης, μπορεί να δημιουργήσει λογαριασμό στο myphoto ή να συνδεθεί σε λογαριασμό που έχει ήδη δημιουργήσει και ενεργοποιήσει, να προβάλλει ή να αναζητήσει φωτογραφίες, να προσθέσει φωτογραφίες στο καλάθι του και γενικότερα να το διαχειριστεί.

Το menu που εμφανίζεται κατά την προβολή οποιασδήποτε σελίδας του myphoto σε έναν χρήστη που έχει πραγματοποιήσει είσοδο στο myphoto είναι αυτό που φαίνεται στην εικόνα.



Εικόνα 120: Μενού myphoto για αυθεντικοποιημένους χρήστες

Οι αυθεντικοποιημένοι χρήστες, διακρίνονται σε τρεις κατηγορίες:

α) τους απλούς χρήστες
β) τους χρήστες της κατηγορίας staff
γ) τους superusers που έχουν αναβαθμισμένα δικαιώματα.

Η διάκριση γίνεται μέσω της κατάλληλης τιμής στο boolean attribute **is_staff** και **is_superuser** που διαθέτει το model **django.contrib.auth.models.User**, το οποίο χρησιμοποιείται για την αναπαράσταση χρηστών.

Ένας απλός χρήστης (δηλαδή ένας χρήστης τέτοιος ώστε για τα attributes του αντίστοιχου αντικειμένου της model κλάσης **User** να ισχύει **is_staff= False** και **is_superuser=False**) έχει δικαίωμα να ανεβάσει φωτογραφίες, να κάνει favorite φωτογραφίες άλλων, να ολοκληρώσει παραγγελία που έχει πραγματοποιήσει, να επεξεργαστεί ή να διαγράψει φωτογραφίες του, να διαγράψει σχόλιά του,να προβάλλει τις φωτογραφίες που έχει ανεβάσει ή τις φωτογραφίες που έχει κάνει favorite.

Ένας χρήστης που ανήκει στην κατηγορία **staff** (δηλαδή ένας χρήστης τέτοιος ώστε για τα attributes του αντίστοιχου αντικειμένου της model κλάσης **User** να ισχύει **is_staff=True** και **is_superuser=False**) έχει τη δυνατότητα πραγματοποίησης όλων των παραπάνω λειτουργιών καθώς και δυνατότητα εισόδου στο **admin site** του Django -χωρίς όμως τη δυνατότητα προβολής οποιουδήποτε αντικειμένου model- και επεξεργασίας και διαγραφής φωτογραφιών και σχολίων οποιοδήποτε χρήστη μέσω του interface του myphoto.

Ένας χρήστης που ανήκει στην κατηγορία **superuser** (δηλαδή ένας χρήστης τέτοιος ώστε για τα attributes του αντίστοιχου αντικειμένου της model κλάσης User να ισχύει **is_staff=True** και **is_superuser=False**) έχει τη δυνατότητα αξιοποίησης όλων των δυνατοτήτων ενός χρήστη **staff**, καθώς και τη δυνατότητα διαχείρισης των χρηστών, των αντικειμένων της κλάσης **AccessAttempt** και των παραγγελιών μέσω του interface του admin site του myphoto.

5.4.3 Έλεγχος Πρόσβασης Χρηστών

Ο έλεγχος πρόσβασης των χρηστών στο myphoto υλοποιείται με τους τρόπους που περιγράφονται στην παράγραφο που ακολουθεί.

5.4.3.1 Έλεγχος Πρόσβασης σε Views

Στο myphoto, περιλαμβάνονται και δημιουργούνται κατάλληλα decorators ώστε η εκτέλεση των συναρτήσεων views να γίνεται μόνο από χρήστες που έχουν το αντίστοιχο δικαίωμα.

α) Χρήση decorator is_logged_in

Ο decorator **is_logged_in** είναι μια συνάρτηση που επιτρέπει την εκτέλεση μιας view συνάρτησης μόνο εάν ο χρήστης δεν έχει πραγματοποιήσει είσοδο στο myphoto. Έτσι, η view **create_account()** και **is_locked_out()** μπορούν να εκτελεστούν μόνο σε περίπτωση που ο χρήστης που πραγματοποιεί το request για το URL που αντιστοιχεί σε αυτές δεν έχει είναι αυθεντικοποιημένος.

```
@is_logged_in
def create_account(request):
 form=UserForm(request.POST or None)
 if form.is_valid():
    user = form.save(commit=False)
    user.is_active=False
    user.save()
    token_generator=default_token_generator
    domain=get_current_site(request).domain
    if request.is_secure():
      protocol="https"
    else:
      protocol="http"
    context = {
       'domain': domain,
       'uid': urlsafe_base64_encode(force_bytes(user.pk)),
       'token': token_generator.make_token(user),
       'protocol':protocol,
         }
    subject="my photo account activation"
    email=loader.render_to_string("registration/activation_email.html",context)
    send_mail(subject,email,"myphoto.team@gmail.com",[user.email])
    return HttpResponseRedirect("/accounts/activation")
 return render(request,"create_account.html",{"form":form})
```

Εικόνα 121: Ο decorator is_logged_in

β) Χρήστη decorator login_required

Το myphoto αξιοποιεί τον decorator **django.contrib.auth.decorators.login_required** για να περιορίσει την εκτέλεση view συναρτήσεων μόνο σε αυθεντικοποιημένους χρήστες. Με αυτό τον τρόπο, μόνο ένας αυθεντικοποιημένος χρήστης μπορεί να προσθέσει μια φωτογραφία στα αγαπημένα του, να ολοκληρώσει την παραγγελία του ή να μεταφορτώσει κάποια φωτογραφία. Εάν δεν είναι αυθεντικοποιημένος, ανακατευθύνεται στην login σελίδα του myphoto.

```
@login_required(redirect_field_name="None")
def favorite(request,slug):
 photo=get_object_or_404(Photo,slug=slug)
 user=request.user
 if request.user != photo.user:
   obj,created=Favorite.objects.get_or_create(user=request.user,photo_id=photo.id,photo=photo)
   if not created:
     photo.likes=photo.likes-1
     obj.delete()
   else:
     photo.likes=photo.likes+1
     obj.save()
   photo.save()
   return HttpResponseRedirect(photo.get_absolute_url())
 else:
    raise PermissionDenied
```

Εικόνα 122: Χρήση decorator login_required στη view συνάρτηση favorite

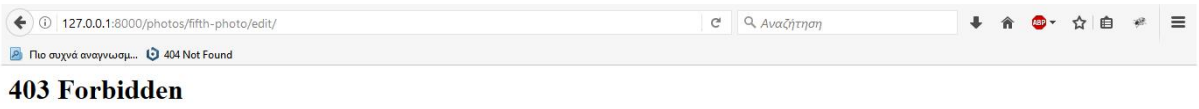
γ) Χρήση decorator is_uploader

Στον κώδικα του myphoto, δημιουργείται ο decorator **is_uploader()**. Ο συγκεκριμένος decorator χρησιμοποιείται στις view συναρτήσεις που υλοποιούν κάποια λειτουργία η οποία μπορεί να γίνει μόνο από τον χρήστη που έχει ανεβάσει μια φωτογραφία ή από χρήστη που ανήκει στην κατηγορία **staff** (για παράδειγμα επεξεργασία ή διαγραφή φωτογραφίας).

```
@is_uploader
def photo_edit(request,slug=None):
 user=request.user
 instance=get_object_or_404(Photo,slug=slug)
 form = PhotoForm(request.POST or None,request.FILES or None,instance=instance)
 if form.is_valid():
   instance=form.save(commit=False)
   instance.save()
   return HttpResponseRedirect(instance.get_absolute_url())
 else:
   context = {
     "title":instance.title,
     "instance":instance,
     "form":form
    }
   return render(request,"photo_form.html",context)

@is_uploader
def photo_delete(request,slug=None):
 instance=get_object_or_404(Photo,slug=slug)
 items=Item.objects.all().filter(object_id=instance.id)
 if items:
    items.delete()
 instance.delete()
 return redirect("photos:list")
```

Εικόνα 123: Ο decorator is_uploader

Σε περίπτωση που ένας χρήστης προσπαθήσει να εκτελέσει μία τέτοια λειτουργία, πραγματοποιώντας request για το URL που αντιστοιχεί στην κατάλληλη view συνάρτηση, ο decorator ελέγχει εάν ο χρήστης είναι αυθεντικοποιημένος. Εάν δεν είναι, τότε ανακατευθύνεται στη σελίδα login του myphoto. Εάν είναι, τότε ο decorator **is_uploader()** εξετάζει εάν έχει το δικαίωμα να καλέσει την αντίστοιχη view συνάρτηση. Εάν δεν το έχει, επιστρέφει μια εξαίρεση της κλάσης **PermissionDenied** (HTTP σφάλμα 403).



Εικόνα 124: Σελίδα που εμφανίζεται σε περίπτωση που ο χρήστης δεν έχει δικαίωμα εκτέλεσης κάποιας view συνάρτησης

5.4.3.2 Έλεγχος Πρόσβασης σε Templates

Ο περιορισμός των λειτουργιών στους χρήστες που έχουν τα αντίστοιχα δικαιώματα, επιτυγχάνεται με χρήση των κατάλληλων decorators με τον τρόπο που περιγράφεται παραπάνω. Η διατήρηση μιας συμπαγούς επιχειρησιακής λογικής, υποβάλλει και την απόκρυψη αυτών των δυνατοτήτων από το περιβάλλον διεπαφής του χρήστη. Έτσι, στο myphoto χρησιμοποιούνται template tags τέτοια ώστε να μην εμφανίζουν στο χρήστη τα html κουμπιά που οδηγούν σε views τα οποία δεν έχει το δικαίωμα να καλέσει (ώστε να αποφευχθεί η συχνή εμφάνιση εξαιρέσεων της κλάσης **PermissionDenied**).

α) Χρήση μεθόδου is_authenticated

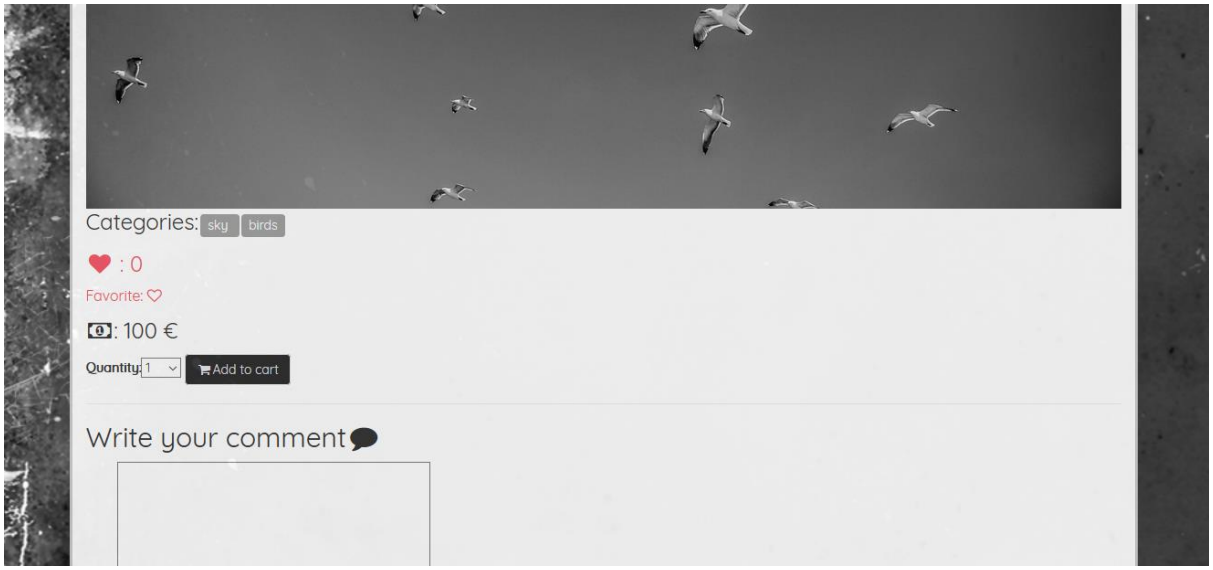
Με τη χρήση της μεθόδου **is_authenticated()** που επιστρέφει **False** εάν ο χρήστης που πραγματοποιεί ένα request δεν έχει πραγματοποιήσει είσοδο στο myphoto κατά την τρέχουσα σύνοδο και True εάν έχει στα αρχεία template, εμφανίζονται ή αποκρύπτονται δεδομένα.

```
 {% if user.is_authenticated %}
<h2>Write your comment<i class="fa fa-comment fa-fw"></i></h2>
<form action="{{instance.get_absolute_url}}add_comment/" method="POST">{{ commentform.as_p }}{% csrf_token %}<button type="submit"
class="btn btn-default"><i class="fa fa-quote-left fa-fw"></i>Add Comment<i class="fa fa-quote-right fa-fw"></i></button></form>{%
endif %}
```

Εικόνα 125: Χρήση μεθόδου is_authenticated εντός template αρχείου

Για παράδειγμα, κατά τη διάρκεια προβολής μιας φωτογραφίας σε έναν αυθεντικοποιημένο χρήστη εμφανίζεται η δυνατότητα να κάνει favorite και να προσθέσει κάποιο σχόλιο, ενώ σε έναν μη αυθεντικοποιημένο χρήστη όχι, όπως φαίνεται στις εικόνες 126 και 127.

Εικόνα 126: Απόκρυψη του κουμπιού Favorite για μη – αυθεντικοποιημένους χρήστες



Εικόνα 127: Εμφάνιση του κουμπιού Favorite για αυθεντικοποιημένους χρήστες

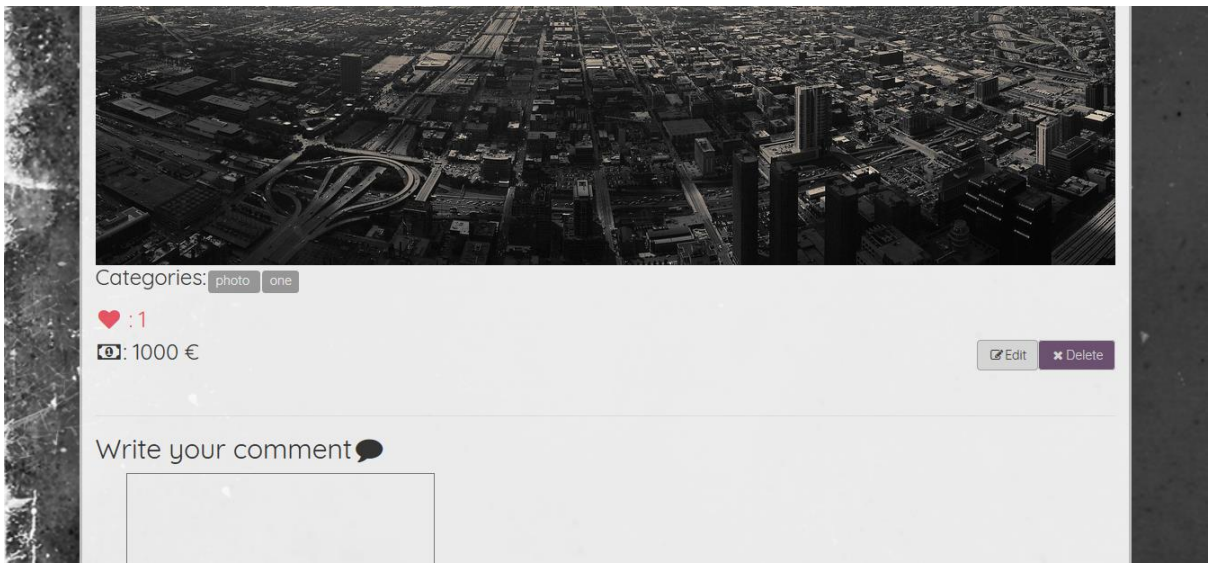
β) Χρήση attribute user

Με τη χρήση του attribute **user** που έχει το αντικείμενο που αναπαριστά τα requests και του attribute **user** που έχει το model **Photo** εντός των templates, σε έναν χρήστη που έχει ανεβάσει μια φωτογραφία (ή ανήκει στην κατηγορία **staff**) εμφανίζεται η δυνατότητα να επεξεργαστεί ή να διαγράψει μια φωτογραφία, ενώ σε έναν χρήστη που δεν ανήκει σε κάποια από τις δύο κατηγορίες όχι.

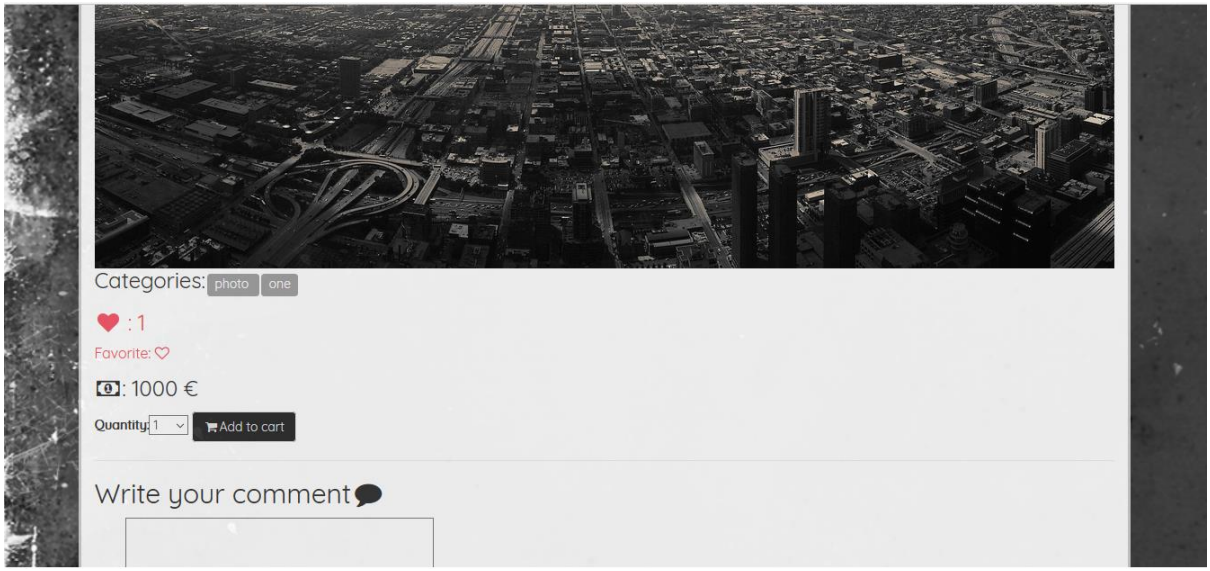
```
{% if instance.user == user or  user.is_staff %}
    <span class="pull-right"><form action="{{instance.get_absolute_url}}edit/" method="POST" style="display:inline">{% csrf_token
    %}<button type="submit" class="btn btn-default"><i class="fa fa-pencil-square-o fa-fw"></i>Edit</button></form><form action="
    {{instance.get_absolute_url}}delete/" method="GET" style="display:inline"><button type="submit" class="btn btn-success"><i
    class="fa fa-remove fa-fw"></i>Delete</button></form></span>
{% endif %}
 {% if instance.user != user %}
   <form action="{{instance.get_absolute_url}}favorite" method="GET"><button type="submit" class="likes post-likes" id="heart-icon"
    data-photo-id="{{ instance.id }}" >
```

Εικόνα 128: Χρήση attribute user εντός template αρχείου

Ταυτόχρονα, το attribute **user** χρησιμοποιείται ώστε να διασφαλίσει εάν ένας χρήστης έχει το δικαίωμα να προσθέσει στο καλάθι του μια φωτογραφία. Σε περίπτωση που ο χρήστης είναι αυτός που έχει ανεβάσει τη φωτογραφία, τότε δεν έχει δυνατότητα να την προσθέσει στο καλάθι του.



Εικόνα 128: Δυνατότητες αυθεντικοποιημένου χρήστη που έχει μεταφορτώσει τη συγκεκριμένη φωτογραφία

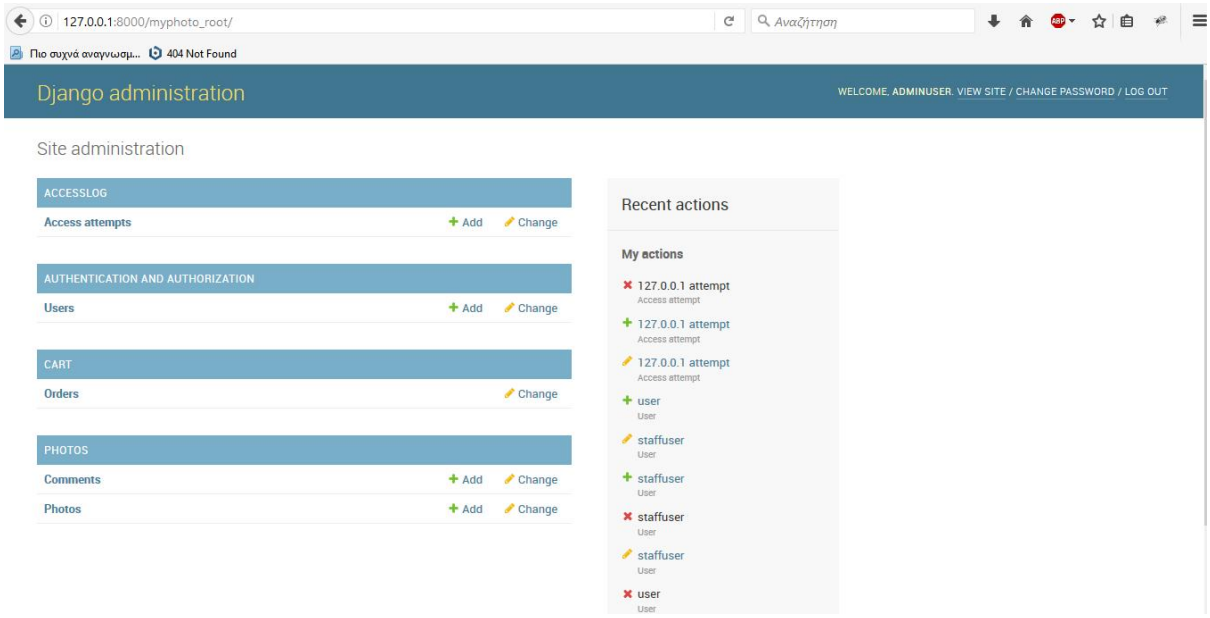


Εικόνα 129: Δυνατότητες αυθεντικοποιημένου χρήστη που δεν έχει μεταφορτώσει τη συγκεκριμένη φωτογραφία

5.4.4 Admin Site της Διαδικτυακής Εφαρμογής myphoto

Το myphoto αξιοποιεί την εφαρμογή **django.contrib.admin** για την υλοποίηση του admin site που αντιστοιχεί στη διαχείριση των δεδομένων του myphoto

Η είσοδος στο admin site γίνεται μέσω του URL **/myphoto_root** (και όχι του συνήθους pattern **/admin** για λόγους ασφαλείας) και μετά από εισαγωγή έγκυρου ζεύγους διαπιστευτηρίων στην φόρμα εισόδου χρήστη. Η κεντρική σελίδα του admin site φαίνεται στην εικόνα 130 .



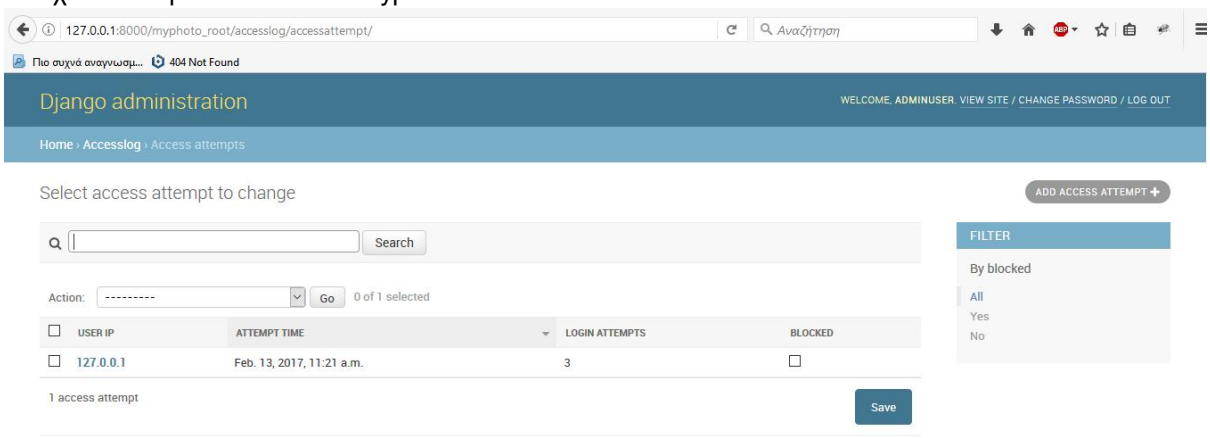
Εικόνα 130: Κεντρική σελίδα admin site του myphoto

Όπως φαίνεται, ένας χρήστης που ανήκει στην κατηγορία superusers, μπορεί να διαχειριστεί αντικείμενα των model κλάσεων **AccessAttempt, User, Photo, Comment, Order.**

5.4.4.1 Διαχείριση Αντικειμένων της Model Κλάσης AccessAttempt

Ένας διαχειριστής μπορεί να να διαγράψει, να προσθέσει ή να επεξεργαστεί τα αντικείμενα της model κλάσης AccessAttempt που είναι αποθηκευμένα στη βάση δεδομένων. Με ενημέρωση του attribute blocked ο διαχειριστής μπορεί να καθορίσει εάν **έ**νας χρήστης με συγκεκριμένη **IP**

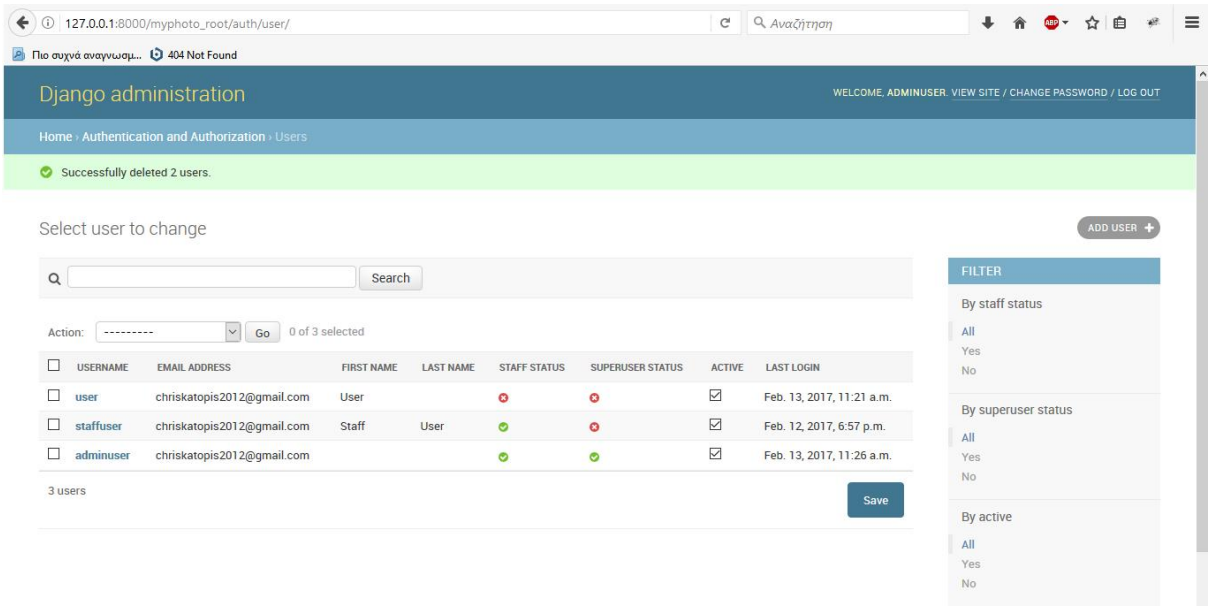
θα έχει δικαίωμα εισόδου στο myphoto.



Εικόνα 131: Αντικείμενα του model AccessAttempt

5.4.4.2 Διαχείριση Χρηστών

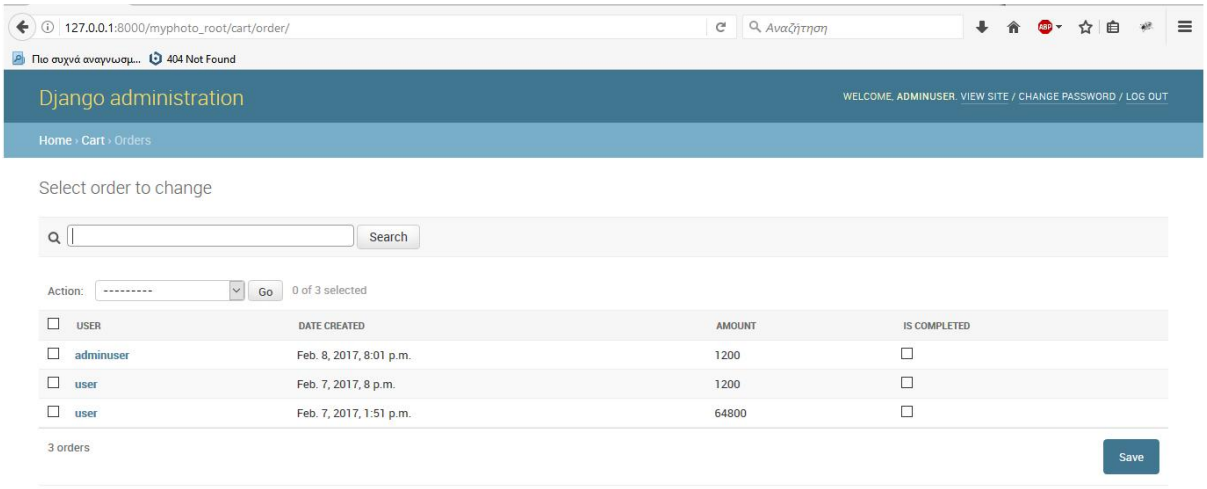
Ένας διαχειριστής μπορεί να δημιουργήσει, να διαγράψει ή να επεξεργαστεί τα αντικείμενα της model κλάσης User που έχουν αποθηκευτεί στη βάση δεδομένων.Ταυτόχρονα μπορεί να δει εάν ο λογαριασμός αυτών των χρηστών έχει ενεργοποιηθεί και πότε ήταν η ημερομηνία και ώρα που πραγματοποίησαν τελευταία φορά είσοδο στη διαδικτυακή εφαρμογή.



Εικόνα 132: Εγγεγραμμένοι χρήστες του myphoto

5.4.4.3 Διαχειρίση Παραγγελιών

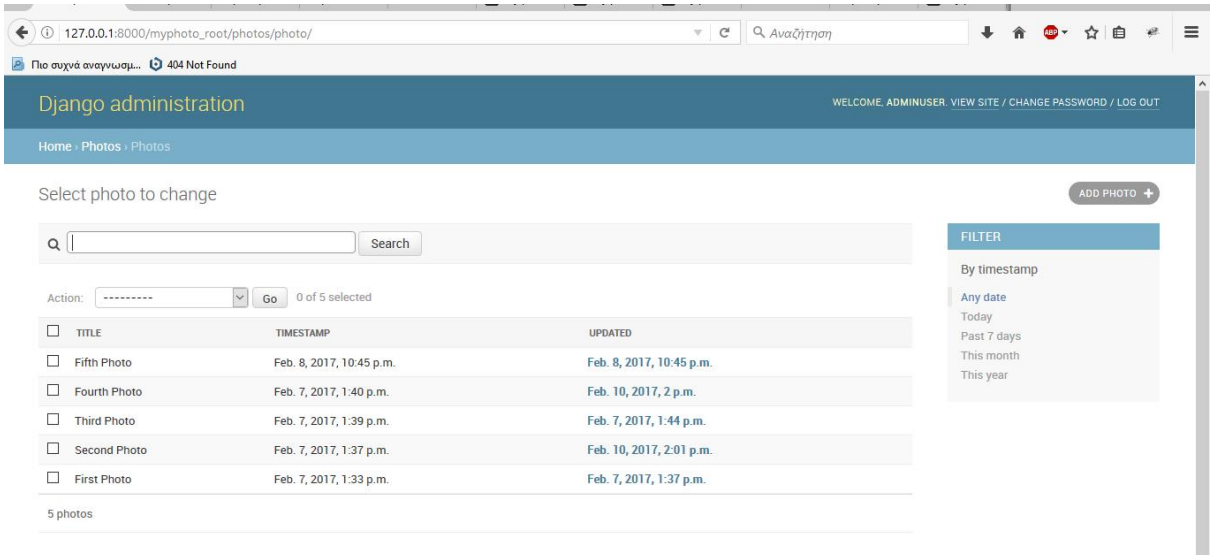
Ένας διαχειριστής μπορεί να διαγράψει ή να επεξεργαστεί τα αντικείμενα της model κλάσης **Order** που χρησιμοποιούνται για να καταγράφονται οι παραγγελίες που έχουν ολοκληρώσει οι χρήστες. Δεν έχει δικαίωμα προσθήκης νέου αντικειμένου (για λόγους ασφάλειας), ενώ με επεξεργασία του attribute **completed** ο διαχειριστής μπορεί να ενημερώσει τη βάση δεδομένων σε περίπτωση ολοκλήρωσης της αποστολής της παραγγελίας.



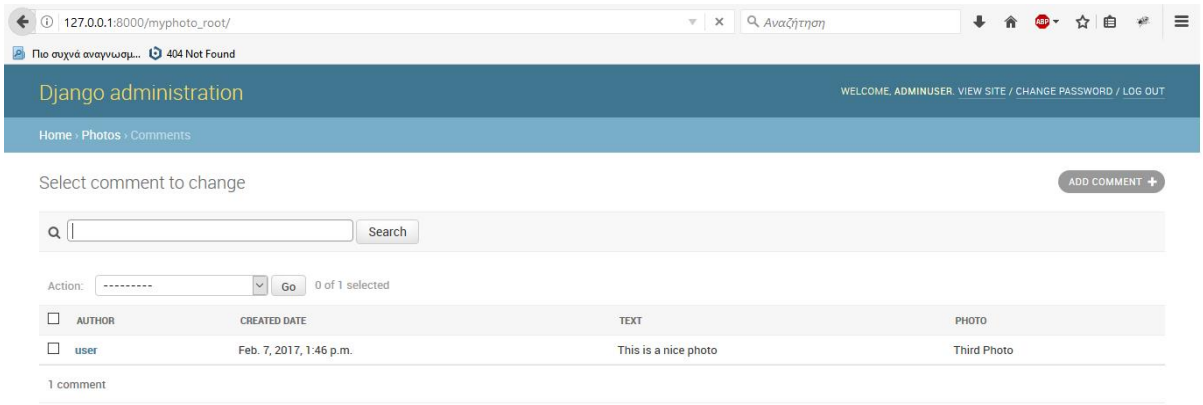
Εικόνα 133: Παραγγελίες του myphoto

5.4.4.4 Διαχείριση Φωτογραφιών & Σχολίων

Ένας διαχειριστής μπορεί να διαγράψει, να επεξεργαστεί και να προσθέσει νέα αντικείμενα των model κλάσεων **Photo** και **Comment**.



Εικόνα 134: Φωτογραφίες που έχουν μεταφορτωθεί στο myphoto



Εικόνα 135: Σχόλια που έχουν υποβληθεί σε φωτογραφίες που έχουν μεταφορτωθεί στο myphoto

5.4.5 Διαχείριση Κωδικών

5.4.5.1 Αποθήκευση Κωδικών

Για την κρυπτογράφηση κωδικών πριν την αποθήκευσή τους στη βάση, χρησιμοποιείται ο αλγόριθμος Argon2 (καλύτερος αλγόριθμος hashing για το 2015 όπως φαίνεται και στην παράγραφο 4.2.2) με χρήση της κλάσης **Argon2PasswordHasher**, όπως φαίνεται από την λίστα **PASSWORD_HASHERS** στο αρχείο **settings.py**

```
PASSWORD_HASHERS=['django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher'
]
```

Εικόνα 136: PASSWORD_HASHERS του myphoto

Προκειμένου να χρησιμοποιηθεί, απαιτήθηκε η εγκατάσταση της βιβλιοθήκης **argon2-cffi** που εγκαταστάθηκε με εκτέλεση της εντολής **pip install argon2-cffi** σε command prompt, όπως φαίνεται στην εικόνα.

```
C:\Users\Chris\Desktop\thesisproject\myphoto>pip install argon2-cffi
Collecting argon2-cffi
  Downloading argon2_cffi-16.3.0-cp27-cp27m-win32.whl
Collecting enum34 (from argon2-cffi)
  Downloading enum34-1.1.6-py2-none-any.whl
Collecting cffi>=1.0.0 (from argon2-cffi)
  Downloading cffi-1.9.1-cp27-cp27m-win32.whl (145kB)
    100% |################################| 153kB 799kB/s
Collecting six (from argon2-cffi)
  Using cached six-1.10.0-py2.py3-none-any.whl
Collecting pycparser (from cffi>=1.0.0->argon2-cffi)
  Downloading pycparser-2.17.tar.gz (231kB)
    100% |################################| 235kB 673kB/s
Building wheels for collected packages: pycparser
  Running setup.py bdist_wheel for pycparser ... done
  Stored in directory: C:\Users\Chris\AppData\Local\pip\Cache\wheels\a8\0b\41\dc
95621f9d3a0da7bc191b8a71f0e8182ffd3cc5f33ac55005
Successfully built pycparser
Installing collected packages: enum34, pycparser, cffi, six, argon2-cffi
Successfully installed argon2-cffi-16.3.0 cffi-1.9.1 enum34-1.1.6 pycparser-2.17
 six-1.10.0
```

Εικόνα 137: Εγκατάσταση βιβλιοθήκης argon2-cffi

5.4.5.2 Πιστοποίηση Κωδικών

Το myphoto για τον έλεγχο εγκυρότητες των κωδικών που εισάγονται από χρήστες χρησιμοποιεί τέσσερις validators, οι οποίοι ορίζονται στο αρχείο **settings.py**.

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

Εικόνα 138: Validators που χρησιμοποιεί η διαδικτυακή εφαρμογή myphoto

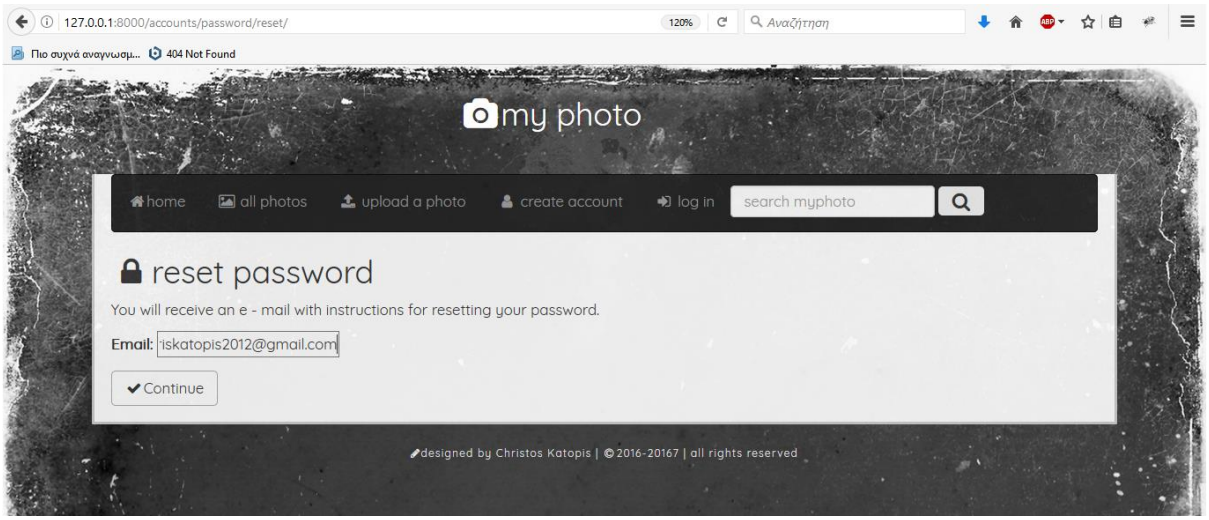
Συνεπώς, προκειμένου ένας κωδικός να θεωρηθεί έγκυρος και να αποθηκευτεί στην βάση δεδομένων κατά την εισαγωγή του, πρέπει να ικανοποιεί τους ελέγχους που πραγματοποιούνται από τους validators. Δηλαδή, να έχει πάνω από 8 χαρακτήρες, να μην αποτελείτα αποκλειστικά από αριθμούς, να μην ταυτίζεται με κάποιο άλλο attribute του αντικειμένου του model **User** που αντιστοιχεί στον χρήστη που τον εισάγει, και να μην ταυτίζεται με κάποια λέξη από τον κατάλογο των πιο κοινών κωδικών που περιέχει το Django.

5.4.5.3 Αλλαγή Κωδικού

Ένας χρήστης έχει την επιλογή να αλλάξει τον κωδικό του, κάνοντας κλικ στην επιλογή **«Forgot your Password? Click here to reset it»**. στη σελίδα εισόδου χρήστη.

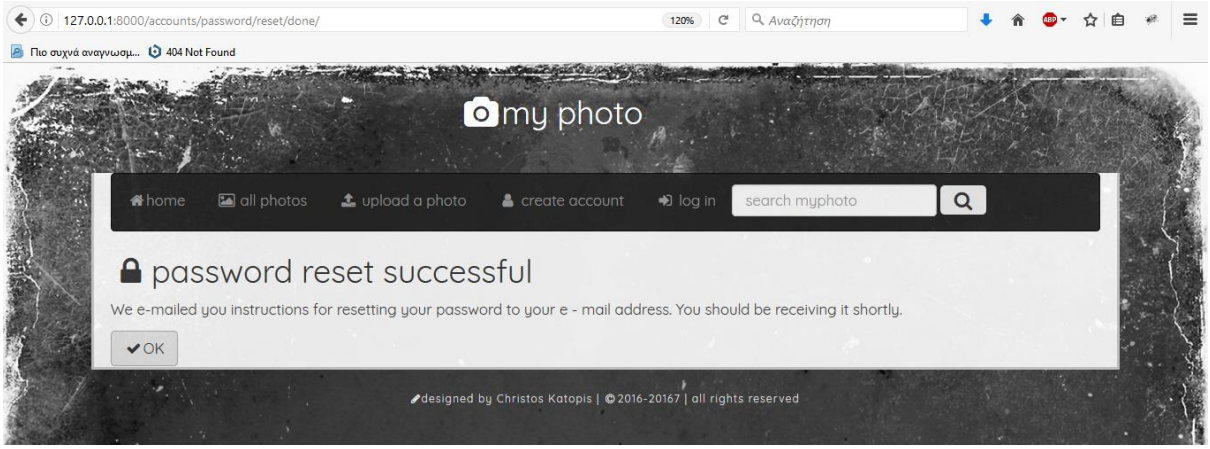
Το myphoto με χρήση του view **django.contrib.auth.views.password_reset**,

στέλνει mail στη διεύθυνση e – mail που θα εισάγει σε αντίστοιχη φόρμα όπως φαίνεται στην εικόνα 139.



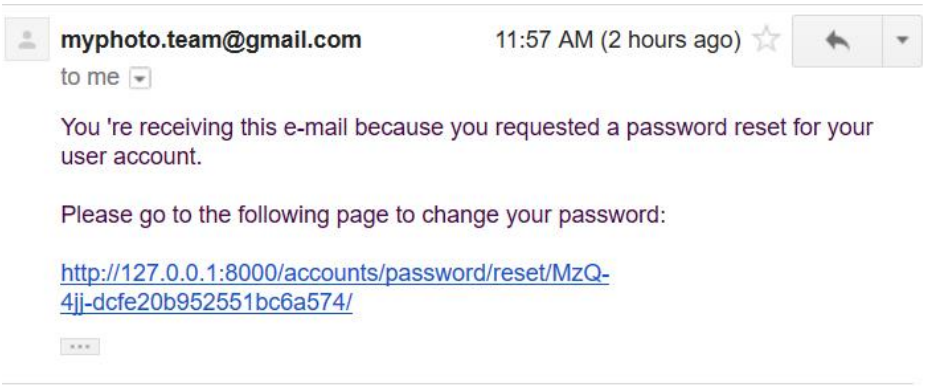
Εικόνα 139: Εισαγωγή διεύθυνσης e – mail στην οποία θα αποσταλλεί ο σύνδεσμος για αλλαγή κωδικού

Ύστερα, επιστρέφει αρχείο template με σχετικό μήνυμα στον χρήστη,



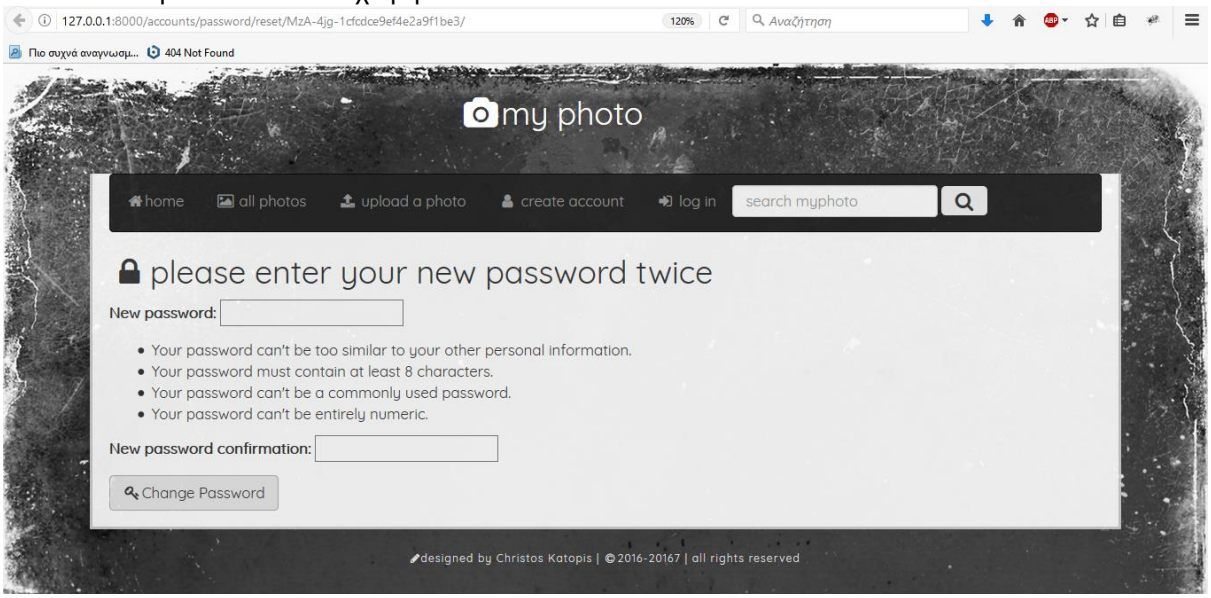
Εικόνα 140: Μήνυμα επιτυχημένης αποστολής συνδέσμου για αλλαγή κωδικού στη διεύθυνση e – mail που έχει εισαχθεί

Η συνάρτηση **django.contrib.views.password_reset** δημιουργεί ένα ζεύγος token και κρυπτογραφημένου πρωτεύοντος κλειδιού του αντικειμένου της model κλάσης **User** που αντιστοιχεί στο χρήστη, και αποστέλλει e – mail στο χρήστη με URL που έχει δημιουργηθεί βάση του token και του κρυπτογραφημένου πρωτεύοντος κλειδιού.



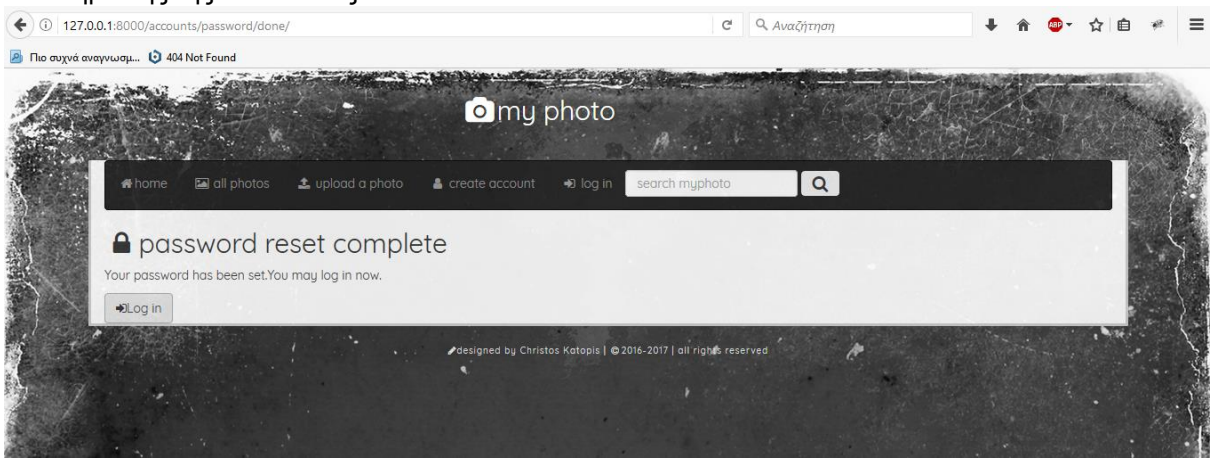
Εικόνα 141: E – mail που περιέχει το URL το οποίο πρέπει να ακολουθηθεί για την αλλαγή κωδικού

Ύστερα εάν ο χρήστης ακολουθήσει το URL, η **django.contrib.views.password_reset_confirm** ελέγχει εάν το ζεύγος κρυπτογραφημένου κλειδιού και token είναι έγκυρο, αφού πρώτα το πρωτεύον κλειδί, και εάν η διεύθυνση e – mail έχει καταχωρηθεί στη βάση δεδομένων.Σε περίπτωση που είναι, τότε ο χρήστης κατευθύνεται σε σελίδα στην οποία θα καταχωρήσει το νέο κωδικό.



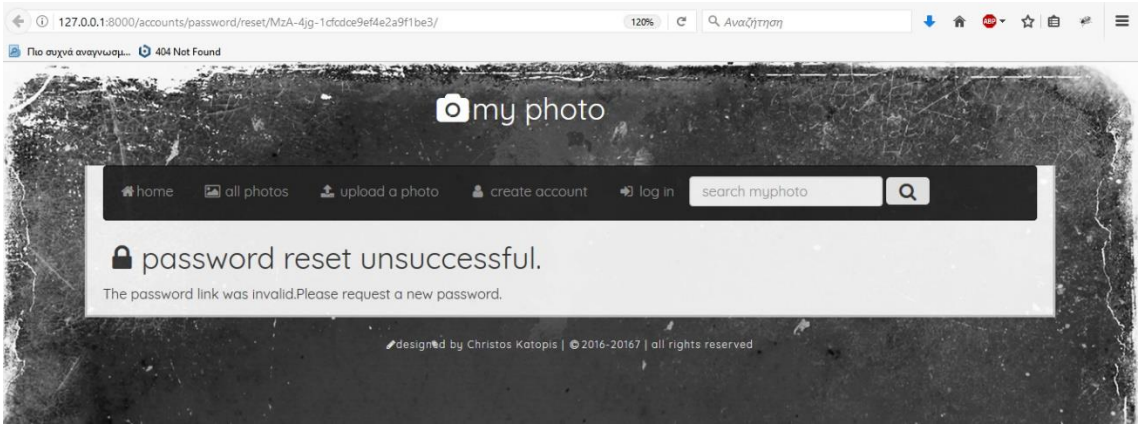
Εικόνα 142: Σελίδα καταχώρησης νέου κωδικού χρήστη του myphoto

Μετά την καταχώρησή του, επιστρέφεται αρχείο template που εμφανίζει μήνυμα ολοκλήρωσης της διαδικασίας.



Εικόνα 143: Μήνυμα ολοκλήρωσης της διαδικασίας αλλαγής κωδικού

Σε διαφορετική περίπτωση, δηλαδή σε περίπτωση που το ζεύγος token και πρωτεύον κλειδί δεν είναι έγκυρο ή που έχει ήδη γίνει request για το ίδιο URL ήδη μία φορά, στον χρήστη επιστρέφεται template αρχείο που αναφέρει το μήνυμα λάθους.



Εικόνα 144: Μήνυμα λάθους σε περίπτωση μη έγκυρου URL αλλαγής κωδικού

5.4.6 Προστασία από Επιθέσεις

Σημαντικό τμήμα της ασφάλειας μιας διαδικτυακής εφαρμογής είναι η αποφυγή βασικών ευπαθειών.

Η διαδικτυακή εφαρμογή myphoto αξιοποιεί πληθώρα εργαλείων που προσφέρει το προγραμματιστικό πλαίσιο Django για αποφυγή αυτών των ευπαθειών. Όπως φαίνεται στην εικόνα 145, Middlewares όπως το **SecurityMiddleware**, το **CsrfViewMiddleware** και το **XFrameOptionsMiddleware** που είναι υπεύθυνα για την προστασία από τις αντίστοιχες

επιθέσεις είναι ενεργοποιημένα, καθώς έχουν προστεθεί στη λίστα **MIDDLEWARE** του αρχείου **settings.py**.

```
47  MIDDLEWARE = [
48      'django.middleware.security.SecurityMiddleware',
49      'django.contrib.sessions.middleware.SessionMiddleware',
50      'django.middleware.common.CommonMiddleware',
51      'django.middleware.csrf.CsrfViewMiddleware',
52      'django.contrib.auth.middleware.AuthenticationMiddleware',
53      'django.contrib.messages.middleware.MessageMiddleware',
54      'django.middleware.clickjacking.XFrameOptionsMiddleware',
55  ]
56
```

Εικόνα 145: Η λίστα MIDDLEWARE στο αρχείο settings.py

5.4.6.1 Προστασία από Επιθέσεις Τύπου XSS

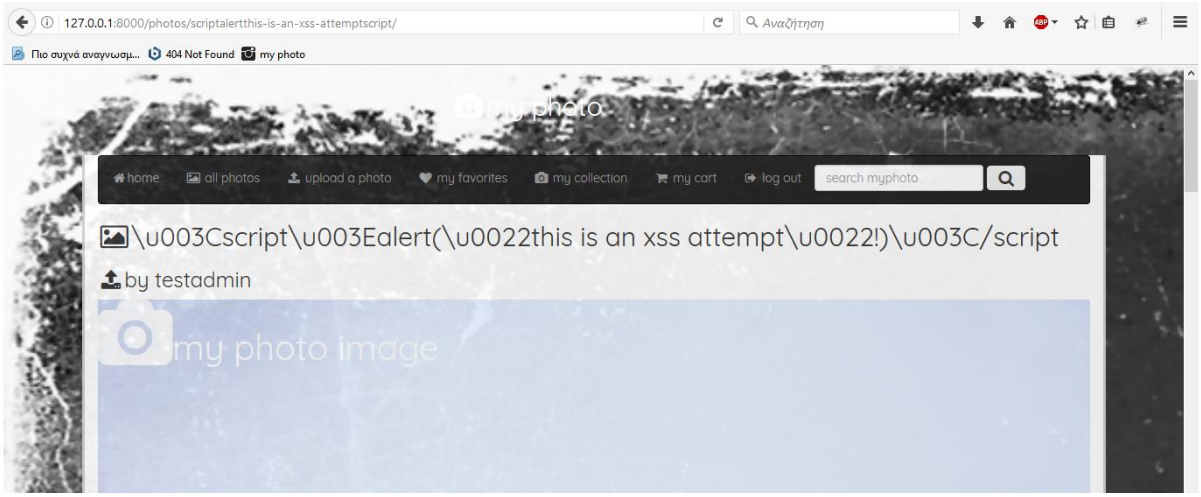
Το myphoto αξιοποιεί την αυτόματη διαφυγή ειδικών HTML χαρακτήρων που προσφέρει η default template engine του Django (την οποία και χρησιμοποιεί).

Επίσης, γίνεται χρήση του template filter **escapejs** σε όλα τα σημεία που ανακτώνται δεδομένα από τη βάση δεδομένων ώστε να διασφαλιστεί η διαφυγή ειδικών χαρακτήρων της Javascript.

```
{% extends "base.html" %}

{% block content %}
<h2><i class="fa fa-picture-o fa-fw"></i>{{ instance.title|escapejs}}</h2>
<h3><i class="fa fa-upload fa-fw"></i>by {{ instance.user|escapejs }}</h3>
<div class="watermark"><i class="fa fa-camera fa-2x"></i>my photo image</div><img src="{{instance.image.url}}" class='
img-responsive'>
```

Εικόνα 146: Χρήστη template φίλτρου escapejs

Όπως φαίνεται και στην παρακάτω εικόνα, οι ειδικοί χαρακτήρες της Javascript δεν μεταφράζονται ως τέτοιοι:



Εικόνα 146: Διαφυγή ειδικών χαρακτήρων Javascript

Εικόνα 147: Διαφυγή ειδικών χαρακτήρων Javascript

Επίσης με χρήση κατάλληλου κώδικα στο αρχείο **settings.py**, όπως φαίνεται στην εικόνα 148, διασφαλίζεται ο έλεγχος από τον browser του client για Javascript κώδικα πρωτού επιστραφεί ένα response στο χρήστη.

```
122  #XSS PROTECTION
123  SECURE_BROWSER_XSS_FILTER=True
```

Εικόνα 148: Καθορισμός μεταβλητής SECURE_BROWSER_XSS_FILTER στο αρχείο settings.py

5.4.6.2 Προστασία από Επιθέσεις CSRF

Όπως αναφέρεται και στην αρχή της συγκεκριμένης παραγράφου, στη λίστα **MIDDLEWARE** του αρχείου settings.py του myphoto συμπεριλαμβάνεται το **CsrfViewMiddleware**. Έτσι, με χρήση του tag **{% csrf_token %}** σε κάθε φόρμα που αποστέλλει δεδομένα με την HTTP μέθοδο POST, διασφαλίζεται η αποστολή και ο έλεγχος CSRF tokens που πιστοποιούν την ακεραιότητα και την προέλευση των δεδομένων ενάντια σε επιθέσεις Cross Site Request Forgery.

```
4   {% block content %}
5   <div class='col-sm-6 col-sm-offset-3'>
6   <h1><i class="fa fa-upload fa-fw"></i>upload a photo</h1>
7   <form method='POST' action='' enctype='multipart/form-data'>{% csrf_token %}
8   {{ form.as_p }}
9   <button type='submit' class='btn btn-default'><i class="fa fa-upload"></i>Upload</button>
10  </form>
11  </div>
12
13  {% endblock content %}
```

Εικόνα 149: Χρήση της template ετικέτας {% csrf_token %}

Επίσης, στο αρχείο **settings.py** καθορίζονται κατάλληλες ρυθμίσεις που περιορίζουν την ισχύ των CSRF cookies στα 3600 δευτερόλεπτα (δηλαδή μία ώρα), και αποτρέπουν την χρήση τους από κώδικα Javascript, γεγονός που μειώνει τον κίνδυνο διαρροής του σε κακόβουλους χρήστες.

```
125  #CSRF PROTECTION
126  CSRF_COOKIE_AGE=3600
127  CSRF_COOKIE_HTTPONLY=True
```

Εικόνα 150: Καθορισμός ρυθμίσεων CSRF στο αρχείο settings.py

5.4.6.3 Προστασία από Επιθέσεις Clickjacking

Όπως φαίνεται στην εικόνα 145, στο myphoto γίνεται χρήση του **XFrameOptionsMiddleware**. Ταυτόχρονα, με χρήση της γραμμής **X_FRAME_OPTIONS='DENY'** στο αρχείο **settings.py,** η HTTP κεφαλίδα **X_FRAME_OPTIONS** λαμβάνει την τιμή **'DENY'**. Έτσι, ένας browser χρήστη δεν αναπαράγει κώδικα από οποιαδήποτε πηγή εντός frame ή iframe σε σελίδα του myphoto ενισχύοντας την αποφυγή ευπαθειών Clickjacking.

```
130  #CLICKJACKING PROTECTION
131  X_FRAME_OPTIONS='DENY'
```

Εικόνα 152: Καθορισμός μεταβλητής X_FRAME_OPTIONS στο αρχείο settings.py

5.4.6.4 Προστασία από Επιθέσεις Τύπου SQL Injection

Το myphoto αξιοποιεί το Django ORM που εξασφαλίζει την προστασία από SQL Injection ελέγχοντας τα queries πριν αυτά αλληλεπιδράσουν με τη βάση δεδομένων. Το γεγονός ότι κάθε SQL query γράφεται με κατάλληλη μορφή (όπως φαίνεται στην εικόνα 153) χωρίς να χρησιμοποιείται raw SQL πουθενά στον κώδικα του myphoto διασφαλίζει την προστασία της διαδικτυακής εφαρμογής.

```
89   def photo_list(request):
90    queryset_list=Photo.objects.all().order_by("likes")
91    query =request.GET.get('q')
```

Εικόνα 153: Αξιοποίηση του Django ORM στον κώδικα του myphoto

5.4.6.5 Ασφαλής Διαχείριση Συνόδων

Το myphoto αξιοποιεί το **SessionMiddleware** και την default ρύθμιση του Django για αποθήκευση των συνόδων: τα δεδομένα που σχετίζονται με τις συνόδους αποθηκεύονται στη βάση δεδομένων.

Αυτό δίνει την δυνατότητα πιστοποίησης και ελέγχου των δεδομένων συνόδου από την πλευρά του server και δεν αφήνεται αποκλειστικά στην πλευρά του client.

Στο αρχείο **settings.py** υλοποιούνται οι εξής ρυθμίσεις:

```
134  #SESSION SETTINGS
135  SESSION_EXPIRE_AT_BROWSER_CLOSE=True
136  SESSION_COOKIE_AGE=172800
137  SESSION_COOKIE_HTTPONLY=True
```

Εικόνα 154: Καθορισμός μεταβλητών που σχετίζονται με τις συνόδους στο αρχείο settings.py

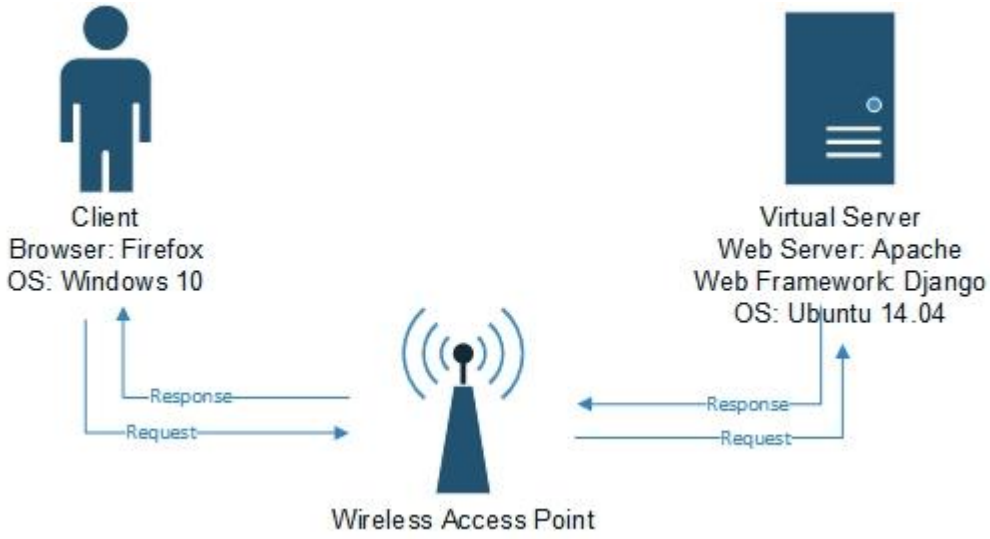
Με τη χρήση των παραμάνω ρυθμίσεων μια σύνοδος λήγει εάν ο χρήστης κλείσει τον browser του, ενώ έχει μέγιστη διάρκεια **172800** δευτερόλεπτα (2 μέρες).

Τα cookies των συνόδων δεν μπορούν να χρησιμοποιηθούν ή να ανακτηθούν με χρήση γλώσσας Javascript, κάτι που εμποδίζει τις επιθέσεις XSS που έχουν ως στόχο την αξιοποίηση των session cookies για υλοποίηση επιθέσεων.

5.5 Δημιουργία Εργαστηριακού Περιβάλλοντος Παραγωγής

Ένα σημαντικό βήμα για την ολοκλήρωση του ασφαλούς σχεδιασμού μιας διαδικτυακής εφαρμογής σε προγραμματιστικό πλαίσιο Django, είναι η διασφάλιση της λειτουργίας της μέσω πρωτοκόλλου HTTPS, δηλαδή η εγκατάσταση πιστοποιητικού SSL στον web server που την εξυπηρετεί. Αυτό απαιτεί την τοποθέτηση της εφαρμογής σε περιβάλλον παραγωγής και δεν μπορεί να καλυφθεί από τον development server που προσφέρει το προγραμματιστικό πλαίσιο Django. Έτσι, σε αυτή την παράγραφο, θα περιγραφεί η δημιουργία περιβάλλοντος παραγωγής για τη διαδικτυακή εφαρμογή myphoto.

Για τις ανάγκες της εργασίας, το περιβάλλον παραγωγής θα υλοποιηθεί με χρήση εικονικού λειτουργικού συστήματος. Αναλυτικότερα, θα εγκατασταθεί εικονικό λειτουργικό σύστημα **Ubuntu 14.04** με χρήση **Virtualbox**. Με χρήση του web server Apache και τις κατάλληλες ρυθμίσεις, το εικονικό σύστημα των **Ubuntu** θα προσομοιάζει το σύστημα στο οποίο είναι εγκατεστημένος ο server, ενώ το λειτουργικό σύστημα Windows 10 στον οποίο αναπτύχθηκε αρχικά η διαδικτυακή εφαρμογή myphoto,με χρήση του browser Firefox, θα προσομοιάζει τον **client**.Η λειτουργία του εργαστηριακού περιβάλλοντος περιγράφεται σχηματικά στην εικόνα 155.



Εικόνα 155: Σχήμα εργαστηριακού περιβάλλοντος

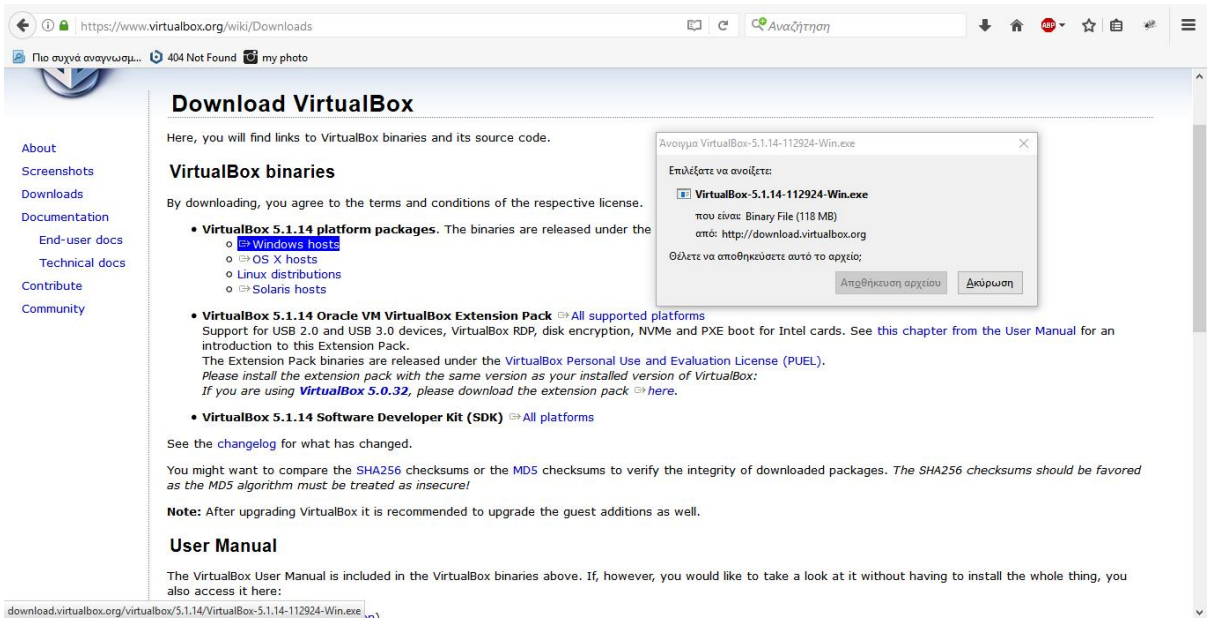
Στις επόμενες παραγράφους, επιχείρεται μία λεπτομερέστερη περιγραφή της διαδικσίας δημιουργίας περιβάλλοντος παραγωγής.

5.5.1 Εγκατάσταση Virtualbox

Το **Virtualbox** είναι ένα πρόγραμμα **virtualization** (εικονικοποίησης).

Η εικονικοποίηση είναι μια διαδικασία που καθιστά δυνατή τη λειτουργία ενός λειτουργικού συστήματος εντός ειδικού περιβάλλοντος, όπως ένα άλλο λειτουργικό σύστημα. Η εικονικοποίηση διαφέρει από άλλες αντίστοιχες διαδικασίες, μιας και ο κώδικας του εικονικού λειτουργικού συστήματος δεν εκτελείται και στο περιβάλλον που το φιλοξενεί. Αυτό δίνει τη δυνατότητα ταυτόχρονης λειτουργίας δύο ανεξάρτητων μεταξύ τους λειτουργικών συστημάτων, πράγμα που επιτρέπει την προσομοίωση λειτουργίας δύο διαφορετικών ηλεκτρονικών υπολογιστών. Το εικονικό λειτουργικό σύστημα καλείται **guest**, ενώ το λειτουργικό σύστημα που το φιλοξενεί καλείται **host** [69].

Πραγματοποιώντας πρόσβαστη στη σελίδα **www.virtualbox.org/wiki/downloads** και επιλέγοντας την έκδοση για λειτουργικό σύστημα Windows, πραγματοποιείται η λήψη του Virtualbox.

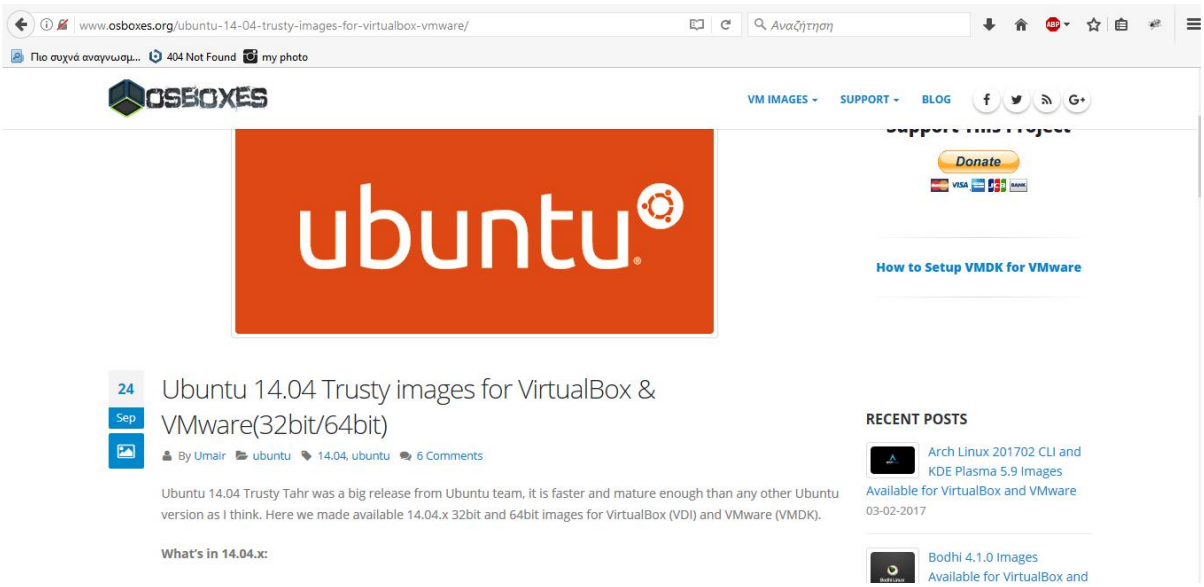


Εικόνα 156: Σελίδα λήψης του Virtualbox

5.5.2 Εγκατάσταση Ubuntu

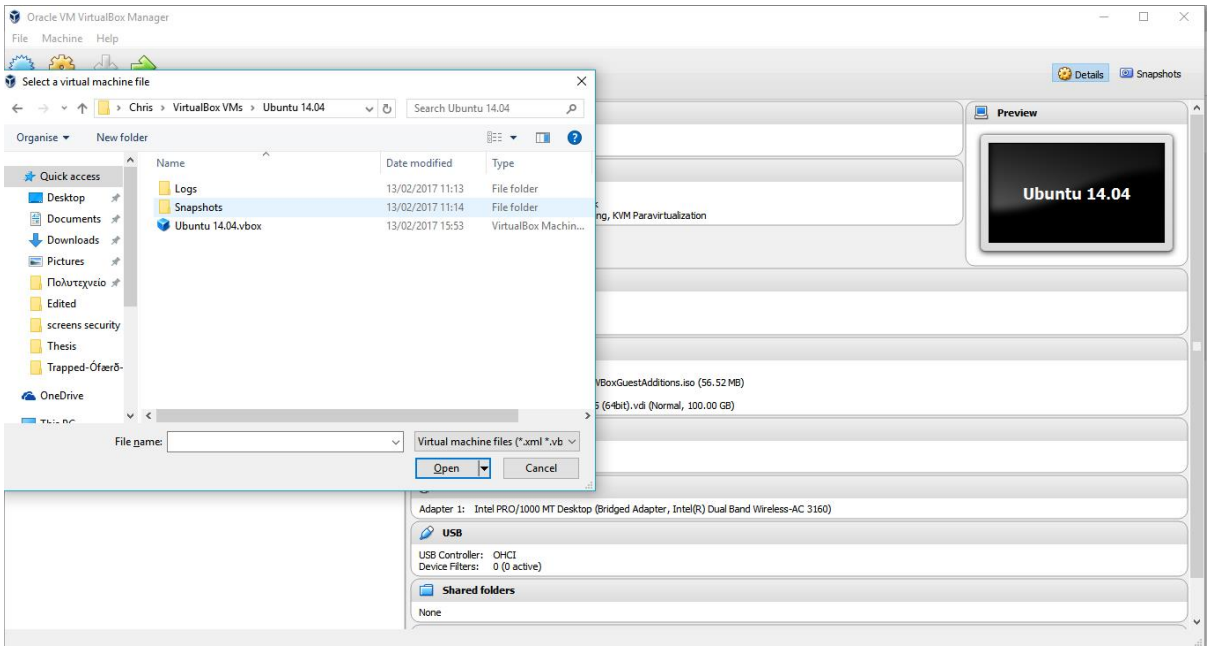
Έχοντας ολοκληρώσει τη λήψη και την εγκατάσταση του **Virtualbox**, χρειάζεται να πραγματοποιήσουμε εγκατάσταση του εικονικού λειτουργικού συστήματος. Όπως έχει περιγραφεί προηγουμένως αυτό θα είναι το **Ubuntu 14.04**. Συγκεκριμένα, θα γίνει χρήση εικόνας του **Ubuntu 14.04**, η οποία έχει δημιουργηθεί ειδικά για λειτουργία μέσω Virtualbox. Πραγματοποιώντας πρόσβαση στη σελίδα **www.osboxes.org/ubuntu-14.04/trusty-images-**

for-virtualbox-vmware/ και επιλέγοντας την αντίστοιχη επιλογή, γίνεται λήψη της εικόνας του **Ubuntu 14.04**.



Εικόνα 157: Σελίδα λήψης της εικόνας του Ubuntu 14.04

Το επόμενο βήμα είναι η δημιουργία του εικονικού λειτουργικού συστήματος Ubuntu 14.0.4. Αυτό πραγματοποιείται ως εξής: μετά την εκκίνηση του Virtualbox και την επιλογή “New”, επιλέγουμε το αρχείο **Ubuntu14.04.vbox**. Μετά την ολοκλήρωση της διαδικασίας που απαιτεί καθορισμό διάφορων ρυθμίσεων (για παράδειγμα πόση RAM θα διατεθεί στο εικονικό λειτουργικό σύστημα) , το εικονικό λειτουργικό σύστημα **Ubuntu 14.04** είναι έτοιμο προς χρήση.



Εικόνα 158: Δημιουργία εικονικού λειτουργικού συστήματος Ubuntu 14.04

5.5.3 Εγκατάσταση Apache

Μετά την εγκατάσταση του εικονικού λειτουργικού συστήματος Ubuntu 14.04, το επόμενο βήμα είναι η εγκατάσταση του **web server** που είναι απαραίτητος για την εξυπηρέτηση της διαδικτυακής εφαρμογής myphoto. Όπως αναφέρθηκε, ο web server αυτός θα είναι ο **apache2**. Πληκτρολογώντας την εντολή **apache2 -v** στο terminal των Ubuntu 14.04 που μόλις δημιουργήθηκαν, διαπιστώνεται ότι ο **apache2** είναι ήδη εγκατεστημένος στη συγκεκριμένη εικόνα των **Ubuntu 14.04** που χρησιμοποιείται.

```
thesisproject@osboxes:/etc/apache2/sites-enabled$ apache2 -v
Server version: Apache/2.4.7 (Ubuntu)
Server built:   Jul 15 2016 15:34:04
```

Εικόνα 159: Έκδοση του apache web server που έχει εγκατασταθεί στα Ubuntu 14.04

5.5.4 Ορισμός της IP του server

Για να κατασταθεί δυνατή η επικοινωνία μεταξύ των δύο λειτουργικών συστημάτων, guest και host, και συγκεκριμένα της πρόσβασης στον εικονικό server **Ubuntu 14.04** , θα χρησιμοποιηθεί

στατική IP. Έτσι η πρόσβαση στον server και άρα και στη διαδικτυακή εφαρμογή myphoto θα είναι δυνατή με την πληκτρολόγηση αυτής της IP σε browser οποιουδήποτε υπολογιστή ή λειτουργικού συστήματος έχει συνδεθεί στο ίδιο σημείο πρόσβασης Wi – Fi (Wi – Fi Access Point).

Η IP ορίζεται κατάλληλα στο αρχείο **/etc/network/interfaces**, όπως φαίνεται στην εικόνα 160.

```
  GNU nano 2.2.6         File: /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo eth0
iface lo inet loopback
iface eth0 inet static
          address 192.168.1.100
          netmask 255.255.255.0
          gateway 192.168.1.1
          dns-nameservers 8.8.8.8
```

Εικόνα 160: Ορισμός ethernet tnterface για το λειτουργικό σύστημα Ubuntu 14.04

Όπως φαίνεται παραπάνω, στον εικονικό server ορίζεται η IP **192.168.168.1.100,** με Subnet Mask κατάλληλη για το δίκτυο στο οποίο θα συνδεθεί ο server και το host λειτουργικό σύστημα στο οποίο θα λειτουργεί. Ως Gateway ορίζεται η IP του router (στο συγκεκριμένο παράδειγμα η **192.168.1.1**) και ως DNS server θα αξιοποιηθεί αυτός που αντιστοιχεί στην IP **8.8.8.8**, δηλαδή ο DNS server της google.

Εφόσον καθοριστεί η IP, τα αρχεία που περιέχουν τον πηγαίο κώδικα της διαδικτυακής εφαρμογής myphoto μεταφέρονται στο directory **/var/www**, στο οποίο είθισται να φιλοξενούνται τα αρχεία τα οποία εξυπηρετεί ο web server apache.

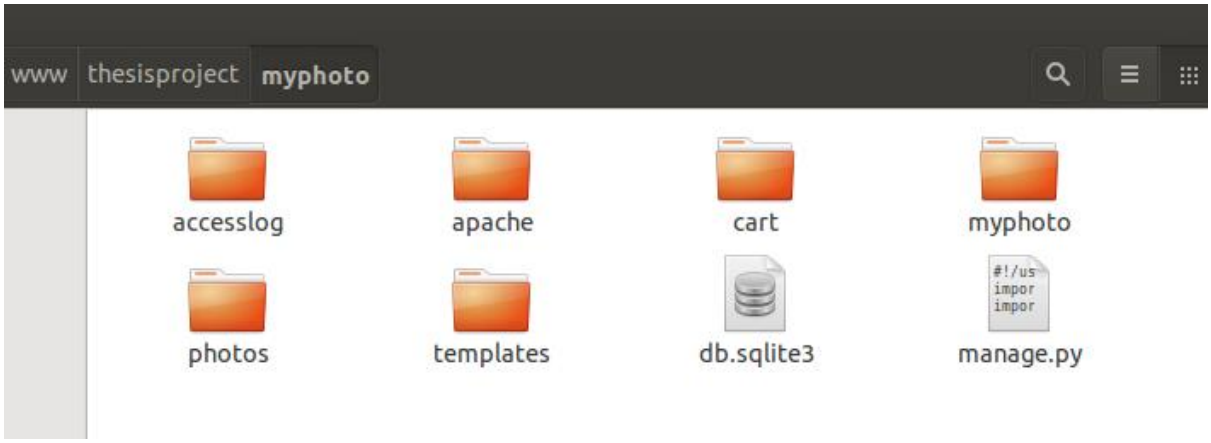
Στον φάκελο **/var/www/thesisproject** παρατηρούνται οι παρακάτω φάκελοι:

Εικόνα 161: Φάκελοι που σχετίζονται με την διαδικτυακή εφαρμογή myphoto στο εικονικό λειτουργικό σύστημα Ubuntu 14.04

Ο φάκελος **myphoto** περιέχει τον κώδικα που είναι απαραίτητος για τη λειτουργία του

myphoto (και περιγράφηκε αναλυτικά στην παράγραφο 5.3), ο φάκελος **static_cdn** θα περιέχει τα αρχεία css και τις εικόνες που καθορίζουν την εμφάνιση του myphoto, ενώ ο φάκελος **media_cdn** θα περιέχει τα αρχεία των φωτογραφιών, προκειμένου αυτά να προβληθούν στο myphoto.

Εντός του φακέλου myphoto, δημιουργείται ένας φάκελος με το όνομα **apache**. Σε αυτόν τον φάκελο θα δημιουργηθεί python module **wsgi.py**, το οποίο είναι απαραίτητο προκειμένου μία διαδικτυακή εφαρμογή Django να εξυπηρετηθεί με τη χρήση apache web server.



Εικόνα 162: Δημιουργία φάκελου apache στον οποίο θα δημιουργηθεί το αρχείο wsgi.py

5.5.5 Δημιουργία Web Server Gateway Interface

Όπως έχει περιγραφεί σε προηγούμενες παραγράφους, το web framework Django ουσιαστικά παρεμβάλλεται ανάμεσα στα HTTP requests προκειμένου να τα διαχειριστεί και να επιστρέψει κατάλληλα αποτελέσματα στους χρήστες που τα πραγματοποιούν. Παρόλα αυτά, σε περιβάλλον παραγωγής ένας web server ο οποίος θα λάβει τα requests αυτά σε πρώτο επίπεδο είναι απαραίτητος, μιας και το Django δεν υλοποιεί αυτή τη λειτουργία από μόνο του (αυτό γίνεται μόνο σε περιβάλλον ανάπτυξης με χρήση του Django development server)

Προκειμένου να κατασταθεί δυνατή η επικοινωνία μιας εφαρμογής που έχει δημιουργηθεί με χρήση Django και γενικά με χρήση Python και ενός web server, είναι απαραίτητη η χρήση μιας υλοποίησης του **WSGI interface (Web Server Gateway Interface)**.

Το **WSGI interface** είναι κώδικας ο οποίος επιτρέπει σε διαδικτυακές εφαρμογές που έχουν δημιουργηθεί σε γλώσσα Python να εξυπηρετηθούν με χρήση διάφορων web servers..

Το WSGI interface αποτελείται από δύο τμήματα: το τμήμα λειτουργίας που υλοποιείται από τον web server και το τμήμα λειτουργίας που υλοποιείται από τη διαδικτυακή εφαρμογή που εξυπηρετείται.

Η πλευρά του server ανακτά ένα αντικείμενο το οποίο παρέχεται από την πλευρά της εφαρμογής (και αναπαριστά την ίδια την εφαρμογή). Ο τρόπος με τον οποίο ανακτάται το αντικείμενο αυτό εξαρτάται από τον web server που χρησιμοποιείται. Κάποιοι servers απαιτούν

από τον προγραμματιστή να υλοποιήσει ένα τμήμα κώδικα το οποίο θα δημιουργεί ένα instance του server και θα παρέχει το αντικείμενο που ατντιστοιχεί στην εφαρμογή. Άλλοι servers χρησιμοποιούν αρχεία ρυθμίσεων ώστε να καθορίσουν την τοποθεσία από την οποία θα ανακτηθεί το αντικείμενο που αναπαριστά την εφαρμογή [70].

5.5.5.1 Εγκατάσταση mod_wsgi

Το πακέτο **mod_wsgi** είναι μια συγκεκριμένη υλοποίηση του τμήματος του server του **WSGI interface** το οποίο επιτρέπει σε έναν apache web server να εξυπηρετήσει μία οποιαδήποτε διαδικτυακή εφαρμογή Python η οποία υποστηρίζει τη χρήση WSGI interface [71]. Σύμφωνα με τους δημιουργούς του Django framework, η χρήση apache web server και mod_wsgi είναι ο προτεινόμενος τρόπος εξυπηρέτησης μιας διαδικτυακής εφαρμογής που έχει δημιουργηθεί με χρήση Django [72].

Προκειμένου να γίνει χρήση του mod_wsgi, απαιτείται η εγκατάστασή του. Λόγω της χρήσης **Python 2.7,** σε terminal στο λειτουργικό σύστημα Ubuntu 14.04 πληκτρολογούμε την εντολή **sudo apt-get install libapache2-mod-wsgi**:

```
thesisproject@osboxes:~$ sudo apt-get install  libapache2-mod-wsgi
```

Εικόνα 164: Εγκατάσταση mod_wsgi

5.5.5.2 Δημιουργία Αρχείου wsgi.py

Μετά την ολοκλήρωση της εγκατάστασης του mod_wsgi, προκειμένου να γίνει δυνατή η επικοινωνία της διαδικτυακής εφαρμογής myphoto με τον apache, απαιτείται να υλοποιηθεί το τμήμα του WSGI interface που σχετίζεται με την διαδικτυακή εφαρμογή. Για τον λόγο αυτό, στο python module **wsgi.py** που έχει δημιουργηθεί εντός του φακέλου **thesisproject/apache**,

υλοποιείται ο κώδικας που φαίνεται στην εικόνα 165 .

```
"""
WSGI config for myphoto project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/1.10/howto/deployment/wsgi/
"""

import os,sys
apache_configuration=os.path.dirname(__file__)
project=os.path.dirname(apache_configuration)
workspace=os.path.dirname(project)
sys.path.append(workspace)
sys.path.append(project)
sys.path.append('/var/www/')

from django.core.wsgi import get_wsgi_application

os.environ["DJANGO_SETTINGS_MODULE"] = "myphoto.settings"

application=get_wsgi_application()
```

Εικόνα 165: Το αρχείο wsgi.py του myphoto

Εφόσον ολοκληρωθεί η υλοποίηση του WSGI κώδικα που αφορά στη διαδικτυακή εφαρμογή και η εγκατάσταση του mod_wsgi,το επόμενο βήμα είναι να δηλωθεί στον apache η τοποθεσία στην οποία βρίσκεται ο κώδικας που σχετίζεται με το WSGI και αφορά στην εφαρμογή.

Αυτό επιτυγχάνεται με πρόσβαση στις ρυθμίσεις του apache (που βρίσκονται στο αρχείο **/etc/apache2/sites-available/000-default.conf** και προσθήκη των κατάλληλων γραμμών.

```
ServerAdmin webmaster@localhost
DocumentRoot /var/www/
Alias /static /var/www/thesisproject/static_cdn
Alias /media /var/www/thesisproject/media_cdn
<Directory /var/www/thesisproject/myphoto/apache>
     Require all granted
</Directory>

<Directory /var/www/thesisproject/myphoto/apache>
  <Files wsgi.py>
     Require all granted
  </Files>
</Directory>
```

Εικόνα 166: Ρυθμίσεις στο αρχείο 000-default.conf #1

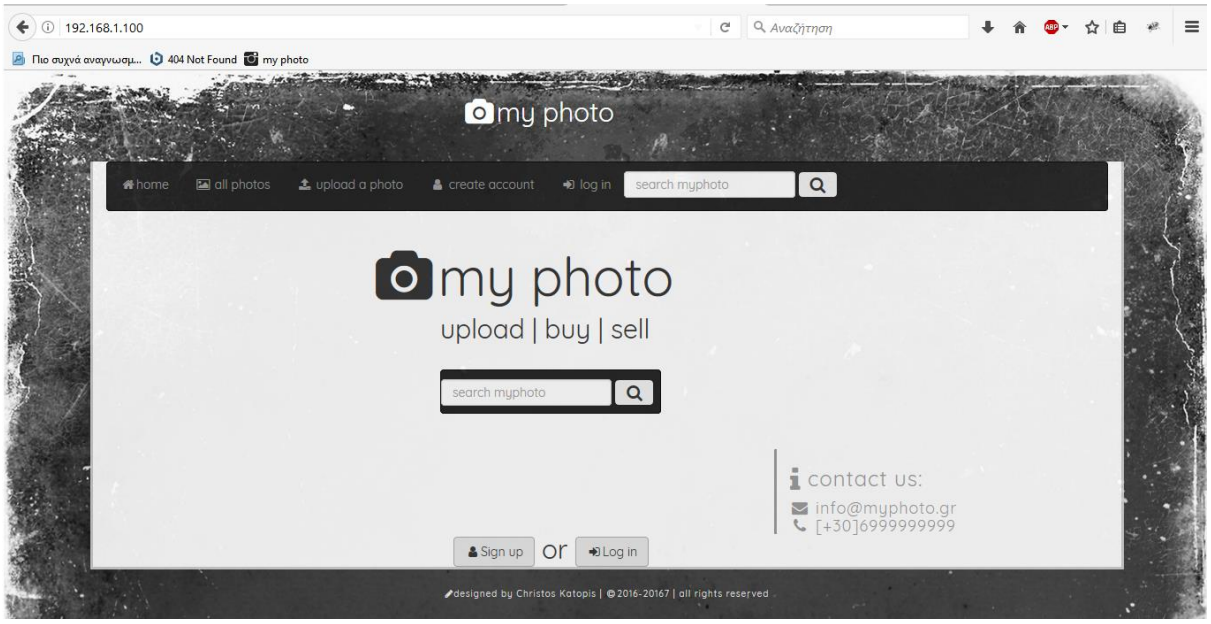
Με τις παραπάνω γραμμές, δηλώνονται ο φάκελος που πρόκειται να φιλοξενήσει τα αρχεία static και αυτός που σχετίζεται με πολυμέσα (φωτογραφίες). Επίσης, δίνονται δικαιώματα πρόσβασης στο αρχείο **wsgi.py** και στον φάκελο που αυτό περιέχεται.

```
WSGIScriptAlias / /var/www/thesisproject/myphoto/apache/wsgi.py
WSGIPythonPath /var/www/thesisproject/myphoto
```

Εικόνα 167: Ρυθμίσεις στο αρχείο 000-default.con #2

Στην παραπάνω εικόνα ορίζεται η τοποθεσία του κώδικα Python που σχετίζεται με τη διαδικτυακή εφαρμογή που πρόκειται να εξυπηρετήσει ο apache (**WSGIPythonPath**), όπως και η αρχική σελίδα στην οποία θα εκτελεστεί ο κώδικας που υπάρχει στο αρχείο **wsgi.py** (**WSGIScriptAlias**). Λόγω της χρήσης του χαρακτήρα **"/"** η πληκτρολόγηση της IP που αντιστοιχεί στον εικονικό server είναι αυτή που οδηγεί στην εκτέλεση του WSGI κώδικα και οδηγεί στην πρόσβαση στη διαδικτυακή εφαρμογή myphoto.

Μετά από όλα τα πραπάνω βήματα, πληκτρολογώντας την IP **192.168.1.100** στο host λειτουργικό σύστημα Windows 10 (που για τις ανάγκες της εργασίας επιτελεί τον ρόλο του client) παρατηρείται η επιτυχής πρόσβαση στη διαδικτυακή εφαρμογή myphoto.



Εικόνα 168: Πρόσβαση στη διαδικτυακή εφαρμογή myphoto

5.6 Πρωτόκολλο HTTPS

Το περιβάλλον παραγωγής της διαδικτυακής εφαρμογής myphoto έχει ολοκληρωθεί. Το τελευταίο βήμα για τον ασφαλή σχεδιασμό μιας διαδικτυακής εφαρμογής είναι η υλοποίηση δυνατότητας επικοινωνίας με την διαδικτυακή εφαρμογή μέσω HTTPS σύνδεσης.

5.6.1 Δημιουργία Πιστοποιητικού SSL

Για να συμβεί αυτό θα γίνει δημιουργηθεί πιστοποιητικό SSL με χρήση του openssl, ενός προγράμματος που αξιοποιεί αλγόριθμους κρυπτογράφησης για να δημιουργήσει τα απαραίτητα πιστοποιητικά. Με πληκτρολόγηση της εντολής **openssl version** σε terminal φαίνεται ότι το openssl είναι ήδη εγκατεστημένο στο εικονικό λειτουργικό σύστημα Ubuntu 14.04 που επιτελεί τον ρόλο του server.

```
thesisproject@osboxes:~$ openssl version
OpenSSL 1.0.1f 6 Jan 2014
```

Εικόνα 169: Έκδοση του openssl που είναι εγκατεστημένη στα Ubuntu 14.04

Προκειμένου να καταστεί δυνατή η επικοινωνία μέσω HTTPS με έναν οποιοδήποτε web server, απαιτείται η δημιουργία μοναδικού πιστοποιητικού που βεβαιώνει την ταυτότητά του. Προκειμένου πιστοποιητικό να θεωρείται αξιόπιστο, υπογράφεται από κάποια τρίτη Αρχή Πιστοποίησης (Certificate Authority) η οποία θεωρείται αξιόπιστη.

Για τις ανάγκες της εργασίας, θα δημιουργηθεί αυτοϋπογεγραμμένο πιστοποιητικό, ή αλλιώς, θα δημιουργηθεί μια Αρχή Πιστοποίησης στο ίδιο λειτουργικό σύστημα στο οποίο είναι εγκατεστημένος ο apache web server.

Για τις ανάγκες του πρωτοκόλλου SSL, δημιουργείται φάκελος **/ssl** εντός του directory **/etc/apache2**.

Προκειμένου να δημιουργηθεί το ιδιωτικό κλειδί που ανήκει στην Αρχή Πιστοποίησης, πληκτρολογούμε σε terminal την εντολή που φαίνεται στην εικόνα 170.

```
thesisproject@osboxes:/etc/apache2/ssl$ openssl genrsa -aes256 -out ca.key
Generating RSA private key, 4096 bit long modulus
...++
.....................................................................
.....................................................................
e is 65537 (0x10001)
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:
thesisproject@osboxes:/etc/apache2/ssl$
```

Εικόνα 170: Δημιουργία ιδιωτικού κλειδιού της Αρχής Πιστοποίησης

Ύστερα, δημιουργείται το πιστοποιητικό της αρχής πιστοποίησης, που περιέχει το δημόσιο κλειδί της Αρχής Πιστοποίησης, με χρήση του ιδιωτικού κλειδιού **ca.key**(που δημιουργήθηκε στο προηγούμενο βήμα), όπως φαίνεται στην εικόνα 171 . Με τη δημιουργία του, εισάγονται κάποιες βασικές πληροφορίες που αφορούν στο πιστοποιητικό.

```
thesisproject@osboxes:/etc/apache2/ssl$ openssl req -new -x509 -days 365 -key ca
.key -out ca.crt
Enter pass phrase for ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:GR
State or Province Name (full name) [Some-State]:Attica
Locality Name (eg, city) []:Athens
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Thesis Project CA
Organizational Unit Name (eg, section) []:TP CA
Common Name (e.g. server FQDN or YOUR name) []:Thesis Project CA
Email Address []:chriskatopis@yahoo.gr
thesisproject@osboxes:/etc/apache2/ssl$
```

Εικόνα 171: Δημιουργία πιστοποιητικού της Αρχής Πιστοποίησης

Το επόμενο βήμα είναι η δημιουργία ιδιωτικού κλειδιού που αντιστοιχεί στον server, που πραγματοποιείται με την εντολή που φαίνεται στην εικόνα 172.

```
thesisproject@osboxes:/etc/apache2/ssl$ openssl genrsa -aes256 -out server.key 4
096
Generating RSA private key, 4096 bit long modulus
.....
.....
.....++
.....
.............++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
thesisproject@osboxes:/etc/apache2/ssl$
```

Εικόνα 172: Δημιουργία ιδιωτικού κλειδιού του web server

Ύστερα, με χρήση του ιδιωτικού κλειδιού του server ,**server.key**, δημιουργείται ένα αρχείο **server.csr** το οποίο αντιστοιχεί στο αίτημα του server για έκδοση πιστοποιητικού.

```
thesisproject@osboxes:/etc/apache2/ssl$ openssl req -new -key server.key -out se
rver.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:GR
State or Province Name (full name) [Some-State]:Attica
Locality Name (eg, city) []:Athens
Organization Name (eg, company) [Internet Widgits Pty Ltd]:myphoto Inc.
Organizational Unit Name (eg, section) []:myphoto
Common Name (e.g. server FQDN or YOUR name) []:myphoto
Email Address []:info@myphoto.gr

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:thesis
An optional company name []:thesis
thesisproject@osboxes:/etc/apache2/ssl$
```

Εικόνα 173: Δημιουργία αιτήματος για πιστοποιητικό του server

Με χρήση του πιστοποιητικού της Αρχής Πιστοποίησης **ca.crt** και του ιδιωτικού κλειδιού **ca.key**, το αρχείο **server.csr** που αντιστοιχεί στο αίτημα του server υπογράφεται και έτσι δημιουργείται το αρχείο **server.crt** που αναπαριστά το πιστοποιητικό του server.

```
thesisproject@osboxes:/etc/apache2/ssl$ openssl x509 -req -days 365 -in server.c
sr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
Signature ok
subject=/C=GR/ST=Attica/L=Athens/O=myphoto Inc./OU=myphoto /CN=myphoto /emailAdd
ress=info@myphoto.gr
Getting CA Private Key
Enter pass phrase for ca.key:
thesisproject@osboxes:/etc/apache2/ssl$
```

Εικόνα 174: Δημιουργία πιστοποιητικού του server

Μετά την ολοκλήρωση της διαδικασίας έκδοσης πιστοποιητικού SSL για τον web server,ο φάκελος /**etc/apache2/ssl** που δημιουργήθηκε προηγουμένως περιέχει τα εξής αρχεία:

Εικόνα 175: Αρχεία ιδιωτικών κλειδιών και πιστοποιητικών της Αρχής Πιστοποίησης και του server

Προκειμένου να γίνει χρήση του πιστοποιητικού και του ιδιωτικού κλειδιού του server από τον apache, πρέπει η τοποθεσία τους να δηλωθεί στις ρυθμίσεις του apache. Για αυτό το λόγο, στο αρχείο /**etc/apache2/sites-available/default-ssl.conf** (το αρχείο ρυθμίσεων του apache σε περίπτωση που γίνεται χρήση πρωτοκόλλου SSL) προσθέτουμε τις γραμμές που φαίνονται στην εικόνα.

```
SSLCertificateFile      /etc/apache2/ssl/server.crt
SSLCertificateKeyFile /etc/apache2/ssl/server.key
```

Εικόνα 176: Ορισμός τοποθεσίας πιστοποιητικού και ιδιωτικού κλειδιού server στο αρχείο ρυθμίσεων του apache

Επίσης, προκειμένου να επικοινωνήσει ο apache web server με τη διαδικτυακή εφαρμογή myphoto μέσω HTTPS προσθέτονται οι κατάλληλες ρυθμίσεις που είχαν προστεθεί και στο αρχείο **etc/apache2/000-default.conf**, όπως φαίνεται στην εικόνα.

```
<IfModule mod_ssl.c>
        <VirtualHost *:443>
                ServerAdmin webmaster@localhost

                DocumentRoot /var/www/
        Alias /static /var/www/thesisproject/static_cdn
        Alias /media /var/www/thesisproject/media_cdn
        <Directory /var/www/thesisproject/myphoto/apache>
            Require all granted
        </Directory>

        <Directory /var/www/thesisproject/myphoto/apache>
          <Files wsgi.py>
             Require all granted
          </Files>
        </Directory>
```

Εικόνα 177: Ρυθμίσεις του default-ssl.conf #1

```
        </VirtualHost>
WSGIScriptAlias / /var/www/thesisproject/myphoto/apache/wsgi.py
WSGIPythonPath /var/www/thesisproject/myphoto
</IfModule>
```

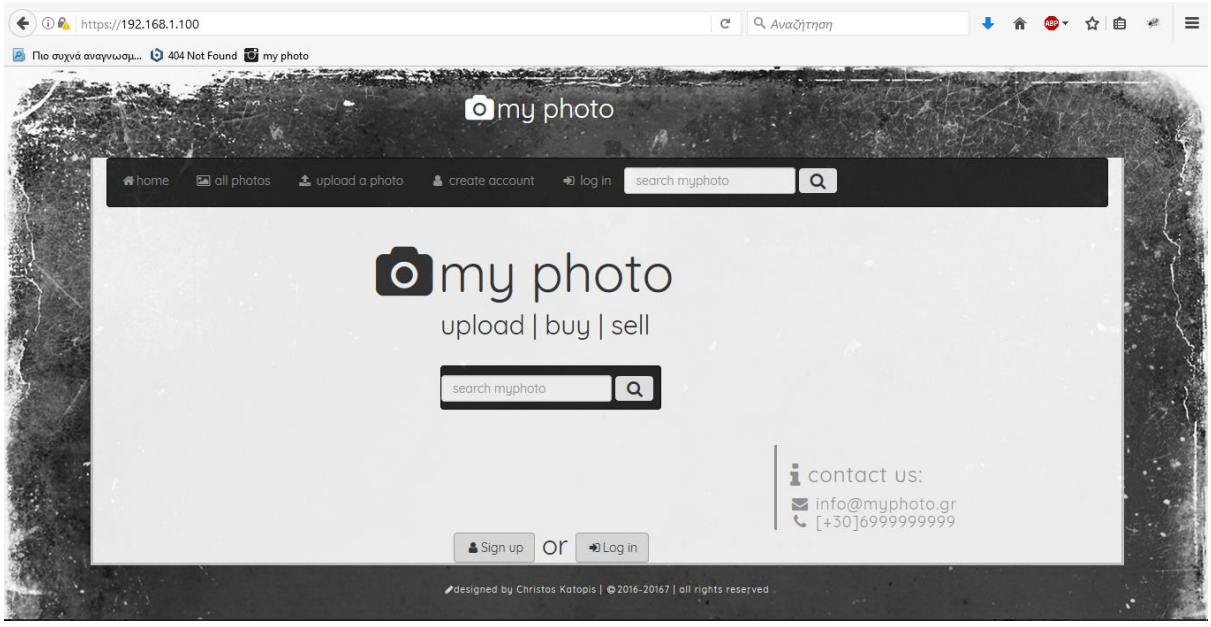
Εικόνα 178: Ρυθμίσεις του default-ssl.conf #2

Μετά την ολοκλήρωση των παραπάνω βημάτων, πραγματοποιείται επανεκκίνηση του web server. Όπως φαίνεται στην εικόνα , ο apache πλέον ζητάει τη φράση που δόθηκε κατά τη διάρκεια δημιουργίας του πιστοποιητικού του server, προκειμένου να αποκρυπτογραφήσει τα κλειδιά που χρησιμοποιούνται . Σε περίπτωση λάθους απάντησης, ο web server δεν εκκινείται.

```
thesisproject@osboxes:/etc/apache2/sites-available$ sudo service apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
Apache needs to decrypt your SSL Keys for 127.0.1.1:443 (RSA)
Please enter passphrase:                                              [ OK ]
thesisproject@osboxes:/etc/apache2/sites-available$
```

Εικόνα 179: Επανεκκίνηση του apache web server

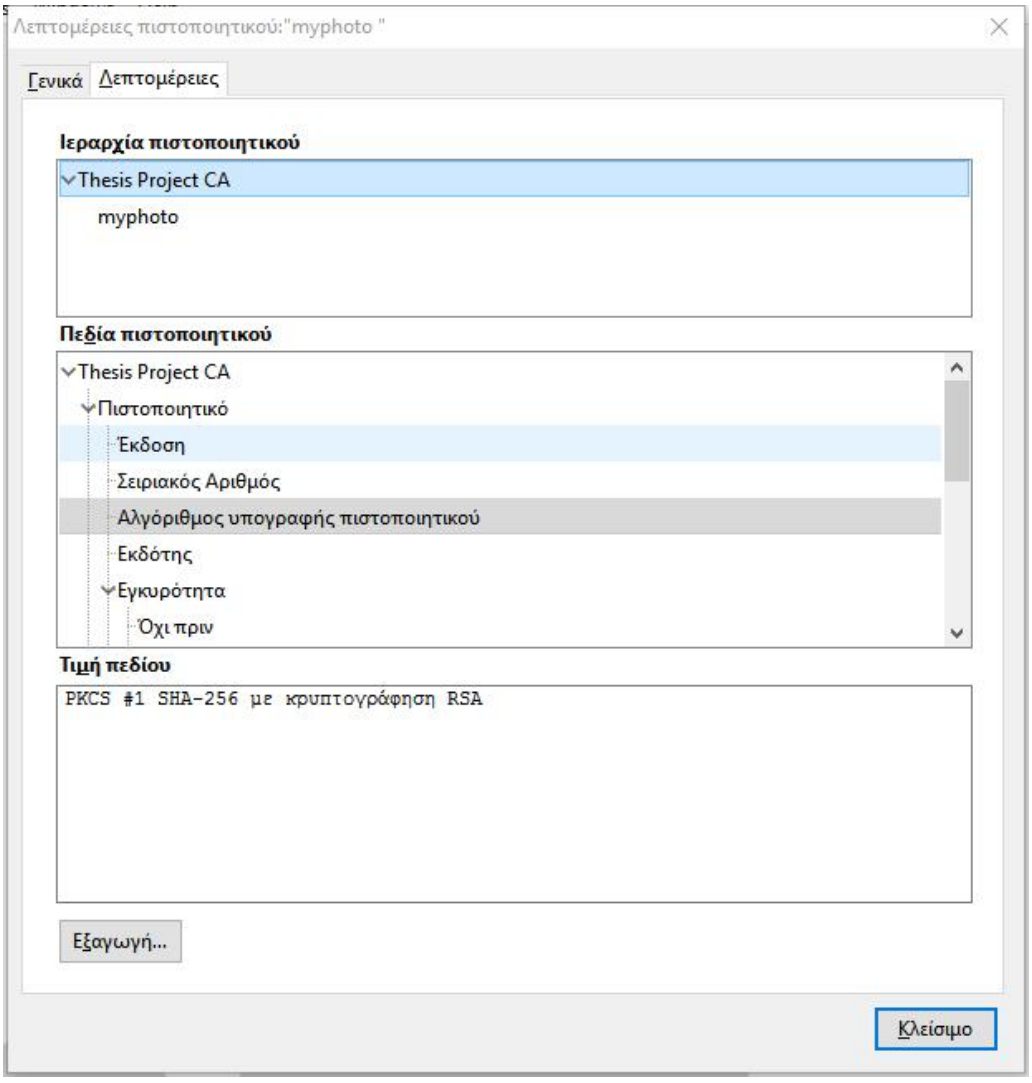
Έχοντας ολοκληρώσει τα παραπάνω βήματα, από το λειτουργικό σύστημα Windows 10 πληκτρολογείται η διεύθυνση **https://192.168.1.100**. Όπως φαίνεται στην εικόνα, η πρόσβαση μέσω HTTPS πλέον είναι δυνατή.



Εικόνα 180: Πρόσβαση στη διαδικτυακή εφαρμογή myphoto μέσω HTTPS

Σημείωση: *Είναι αρκετά πιθανή η εμφάνιση μηνύματος προειδοποίησης σε διάφορους περιηγητές κατά την πρώτη φορά πρόσβασής τους στη διαδικτυακή εφαρμογή myphoto . Αυτό συμβαίνει επειδή το πιστοποιητικό της Αρχής Πιστοποίησης που έχει δημιουργηθεί προηγουμένως δεν θεωρείται έγκυρο, μιας και δεν πρόκειται για μια «γνωστή» Αρχή Πιστοποίησης αλλά για μία που δημιουργήθηκε για τις ανάγκες της εργασίας. Παρόλα αυτά, το πιστοποιητικό του web server και ο τρόπος που υλοποιήθηκε η σύνδεση μέσω HTTPS είναι σωστά και η σύνδεση είναι ασφαλής.*

Κάνοντας κλικ στο κατάλληλο σημείο, παρατηρούνται οι πληροφορίες για το πιστοποιητικό του apache web server:



Εικόνα 181: Πιστοποιητικό του apache web server

5.6.2 Ρυθμίσεις HTTPS

```
139  #HTTPS SETTINGS
140  SECURE_SSL_REDIRECT=True
141  SESSION_COOKIE_SECURE=True
142  CSRF_COOKIE_SECURE=True
143  SECURE_HSTS_SECONDS=3600
144  SECURE_HSTS_INCLUDE_SUBDOMAINS=True
```

Εικόνα 182: Καθορισμός μεταβλητών που σχετίζονται με το πρωτόκολλο HTTPS στο αρχείο settings.py

Έχοντας δημιουργήσει το πιστοποιητικό SSL, με τις παραπάνω ρυθμίσεις στο αρχείο **settings.py** διαφαλίζεται η ανακατεύθυνση όλων των HTTP requests μέσω HTTPS, τα session και CSRF cookies αποστέλλονται μόνο μέσω HTTPS, ενώ γίνεται χρήση HTTPS Strict Transport Security. Συγκεκριμένα με τη μεταβλητή **SECURE_HTSTS_SECONDS=3600** διασφαλίζεται ότι οποιαδήποτε μελλοντική σύνδεση στο myphoto μπορεί να γίνει μόνο μέσω HTTPS ενώ με τη γραμμή **SECURE_HSTS_INCLUDE_SUBDOMAINS** η HTTPS Strict Transport Security διασφαλίζεται για όλα τα subdomains της εφαρμογής .

5.7 Προστασία από HTTP Header Attacks

```
ALLOWED_HOSTS = ['127.0.0.1','192.168.1.100']
```

Εικόνα 183: Καθορισμός της λίστας ALLOWED_HOSTS στο αρχείο settings.py

Με τον καθορισμό της λίστας **ALLOWED_HOSTS** στο αρχείο **/thesisproject/myphoto/myphoto/settings.py** διασφαλίζεται ότι η HTTP κεφαλίδα HOST στα HTTP requests μπορεί να έχει ως τιμή μόνο την IP του εικονικού server (192.168.1.100) ή την IP του localhost.

5.8 Έλεγχος Περιεχομένου που Παρέχεται από Χρήστες

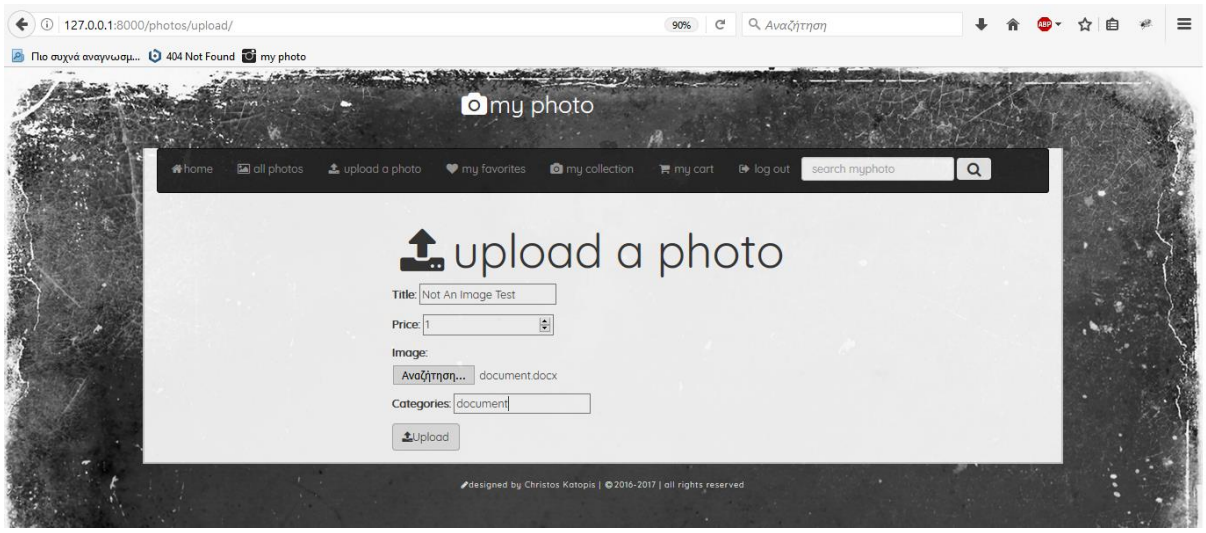
Εφόσον το myphoto βαρίζει τη λειτουργία του σε περιεχόμενο που παρέχεται από χρήστες (αρχεία εικόνας), η διασφάλιση αποφυγής ευπαθειών λόγω του γεγονότος αυτού είναι ζωτικής σημασίας.

5.8.1 Έλεγχος Περιεχομένου σε Επίπεδο Εφαρμογής

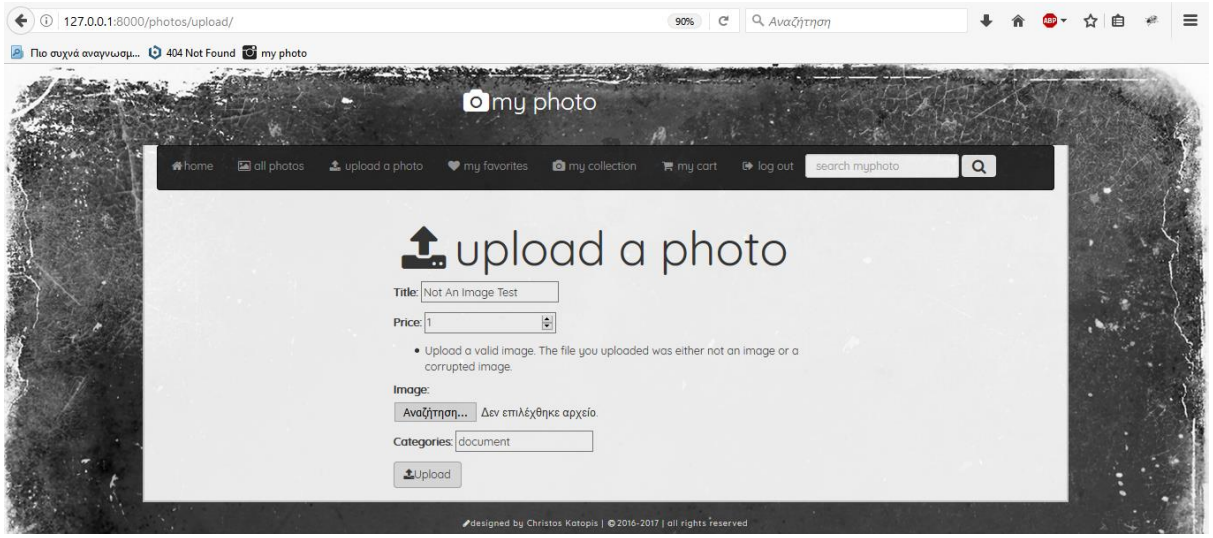
Για το λόγο αυτό, όπως έχει περιγραφεί παραπάνω, χρησιμοποιείται η βιβλιοθήκη pillow και το field **ImageField** για την αποθήκευση των αρχείων εικόνων.

Με τη χρήση τους, κάθε φορά που ένας χρήστης επιλέγει να ανεβάσει μια φωτογραφία, το myphoto ελέγχει ότι όντως πρόκειται περί αρχείου εικόνας.

Για παράδειγμα στην εικόνα 184 ένας χρήστης προσπαθεί να ανεβάσει ένα αρχείο κειμένου, και όπως φαίνεται στην εικόνα 185 το myphoto δεν το αποθηκεύει και εμφανίζει σχετικό μήνυμα λάθους.



Εικόνα 184: Προσπάθεια μεταφόρτωσης αρχείου κειμένου στο myphoto



Εικόνα 185: Μήνυμα λάθους μετά την ολοκλήρωση της προσπάθειας μεταφόρτωσης του αρχείου κειμένου

Παρεμφερές μήνυμα θα εμφανιστεί και σε περίπτωση που κάποιος προσπαθεί να «αλλοιώσει» ένα αρχείο κειμένου αλλάζοντας την κατάληξή του από .doc σε .jpg.

5.8.2 Έλεγχος Περιεχομένου σε Επίπεδο Server

Ένα άλλο μέτρο προστασίας, είναι ο περιορισμός του μεγέθους του περιεχομένου που μπορεί να παρέχεται στο myphoto. Για αυτό τον λόγο, στο αρχείο **default-ssl.conf** προσθέτονται οι γραμμές που φαίνονται στην εικόνα 186. Με τον τρόπο αυτό, διασφαλίζεται ότι οποιαδήποτε εικόνα θα ανέβει στο myphoto (δηλαδή θα αποθηκευτεί στον φάκελο **media_cdn** δεν θα ξεπεράσει τα 10.000.000 byte.

```
<Directory /var/www/thesisproject/media_cdn>
    LimitRequestBody 10000000
</Directory>
```

Εικόνα 186: Περιορισμός επιτρεπόμενου μεγέθους αρχείων στο φάκελο media_cdn

5.9 Προστασία SECRET_KEY και Ευαίσθητων Πληροφοριών

Ένα σημαντικό τμήμα της ασφάλειας, όπως αναφέρθηκε και στην παράγραφο 4.6.1 είναι η προστασία του **SECRET_KEY** που χρησιμοποιείται για να πιστοποιήσει την ταυτότητα του myphoto, όπως και τυχόν υπόλοιπων κωδικών.

Για λόγους προσασίας, η έκθεση οποιωνδήποτε ευαίσθητων πληροφοριών στο αρχείο **settings.py** πρέπει να αποφεύγεται.

Στο myphoto συγκεκριμένα,που χρησιμοποιείται ο mail server της google για την αποστολή e – mail (ενεργοποίησης λογαριασμού και αλλαγής κωδικού), εκτός από το SECRET_KEY ευαίσθητες πληροφορίες αποτελούν και τα διαπιστευτήρια τα οποία χρησιμοποιούνται για τη σύνδεση με τον συγκεκριμένο mail server.

Προκειμένου να αντιμετωπιστεί αυτό το ζήτημα, οι ευαίσθητες πληροφορίες προστίθενται ως **environmental variables** του apache στο αρχείο /**etc/apache2/envvars,** γράφοντας τις γραμμές που φαίνονται στην εικόνα 187.

```
export SECRET_KEY='$4cd#s%e*%m)=a-=v(r)!@pkh%5v90(k0prwgat0db%-umkl72'
export EMAIL_HOST_USER='myphoto.team@gmail.com'
export EMAIL_HOST_PASSWORD='myphotothesis2017'
```

Εικόνα 187: Καθορισμός ευαίσθητων πληροφοριών ως environmental variables του apache

Έτσι, προκειμένου να ανακτηθούν από τη διαδικτυακή εφαρμογή myphoto το μόνο που χρειάζεται είναι το τμήμα του κώδικα στο αρχείο **settings.py**, που φαίνεται στις εικόνες 188 και 189.

```
23  # SECURITY WARNING: keep the secret key used in production secret!
24  SECRET_KEY = os.environ["SECRET_KEY"]
```

Εικόνα 188: Ανάκτηση του SECRET_KEY από τις environmental variables

```
EMAIL_USE_TLS=True
EMAIL_HOST_USER=os.environ["EMAIL_HOST_USER"]
EMAIL_HOST='smtp.gmail.com'
EMAIL_HOST_PASSWORD=os.environ["EMAIL_HOST_PASSWORD"]
EMAIL_PORT=587
```

Εικόνα 189: Ανάκτηση των διαπιστευτηρίων που χρησιμοποιούνται για σύνδεση με τον google mail server από τις environmental variables του apache

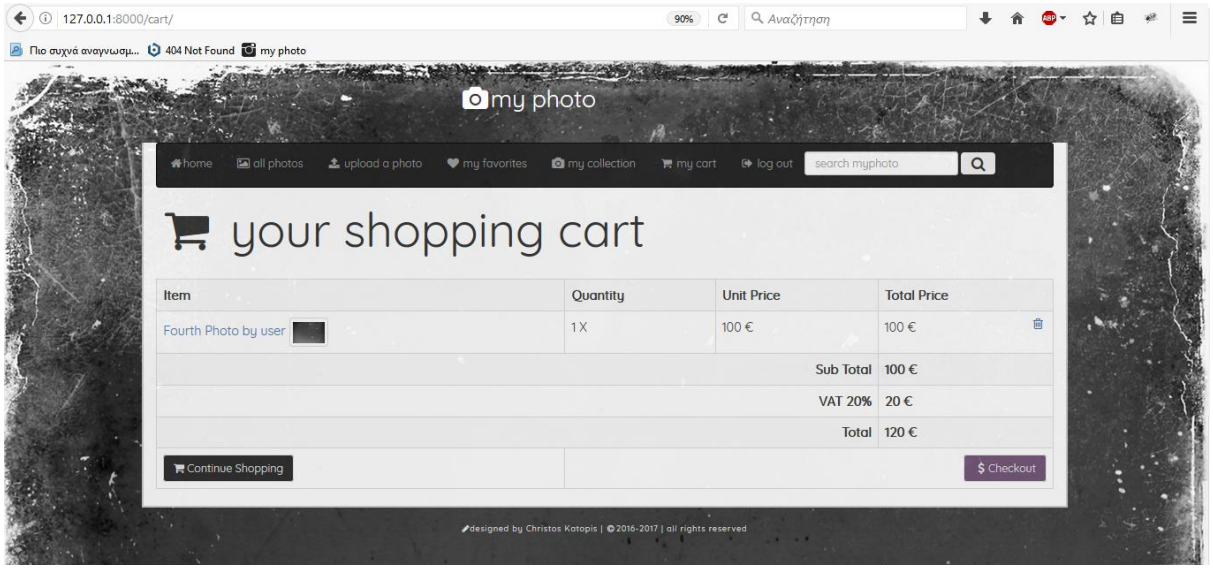
5.10 Άλλα Ζητήματα Ασφάλειας

Εκτός από την προστασία από βασικές ευπάθειες, για τη λειτουργία του myphoto υλοποιούνται κάποια δευτερεύοντα μέτρα ασφάλειας μέσω του κώδικα της διαδικτυακής εφαρμογής αλλά και των ρυθμίσεων του apache web server που την εξυπηρετεί. Για να είναι πιο ευνόητα, τα ζητήματα αυτά παρουσιάζονται βάσει της αρχής ασφάλειας στην οποία εμπίπτουν.

5.10.1 Ζητήματα Ασφάλειας στην Διαδικτυακή Εφαρμογή myphoto

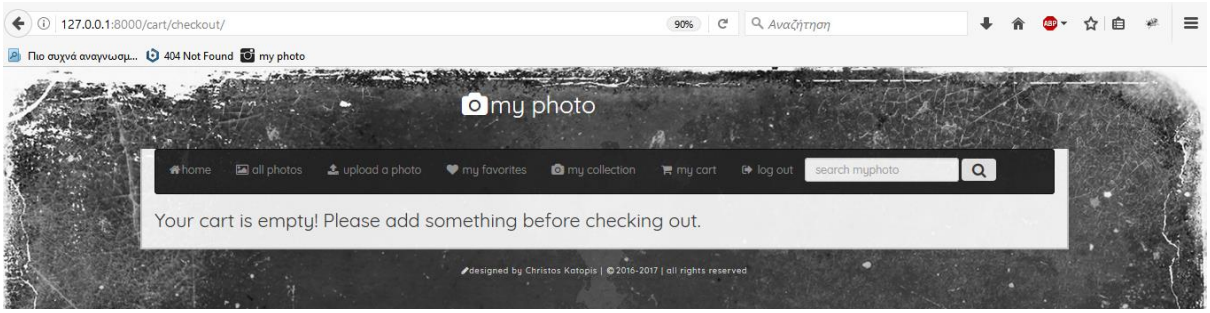
5.10.1.1 Ακεραιότητα

Το πρώτο μέτρο ασφάλειας που αφορά στη διαδικτυακή εφαρμογή σχετίζεται με τις παραγγελίες φωτογραφιών. Σε περίπτωση που ένας αυθεντικοποιημένος χρήστης επιλέξει να ολοκληρώσει την αγορά του (κάνοντας κλικ στο κουμπί Checkout όπως φαίνεται στην εικόνα 190), τότε καταρχήν το myphoto ελέγχει εάν ο χρήστης έχει προσθέσει κάτι στο καλάθι του.



Εικόνα 190: Προβολή καλαθιού χρήστη του myphoto

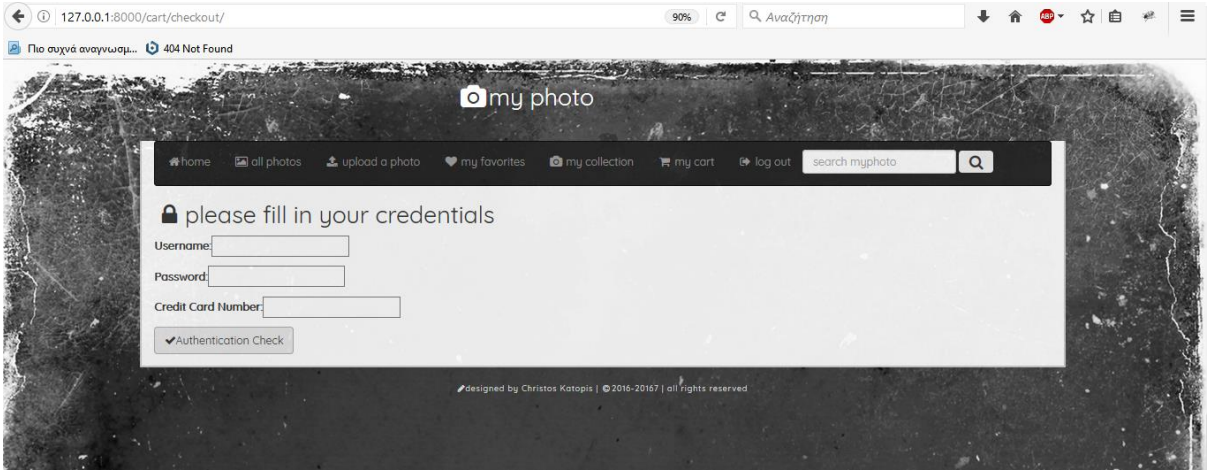
Σε περίπτωση που δεν έχει προστεθεί κάτι στον χρήστη επιστρέφεται μήνυμα που τον ενημερώνει σχετικά. Με αυτό τον τρόπο διασφαλίζεται η τήρηση της επιχειρησιακής λογικής και αποφεύγεται η αποθήκευση μηδενικών παραγγελιών στη βάση δεδομένων.



Εικόνα 191: Μήνυμα σε περίπτωση που το καλάθι χρήση είναι άδειο

Σε περίπτωση που το καλάθι του χρήστη δεν είναι άδειο, στον χρήστη εμφανίζεται μια οθόνη στην οποία καλείται να εισάγει τα διαπιστευτήριά του (username και password). Μετά την εισαγωγή τους, το myphoto τα συγκρίνει με το username και password που αντιστοιχεί στην τρέχουσα σύνοδο. Με αυτό τον τρόπο αποφεύγεται ο κίνδυνος να πραγματοποιηθεί αγορά από κακόβουλο χρήστη με στοιχεία άλλου, σε περίπτωση υποκλοπής του session cookie.

Το γεγονός ότι ο αριθμός πιστωτικής κάρτας ζητείται για πρώτη φορά σε αυτό το σημείο και δεν αποθηκεύεται στη βάση δεδομένων μειώνει τον κίνδυνο διαρροής των αριθμών των πιστωτικών καρτών και διασφαλίζει ότι ακόμη και σε περίπτωση υποκλοπής των διαπιστευτηρίων του χρήστη, ένας κακόβουλος χρήστης πρέπει να γνωρίζει και τον αριθμό της πιστωτικής κάρτας του προκειμένου να πραγματοποιήσει αγορά εις βάρος του.

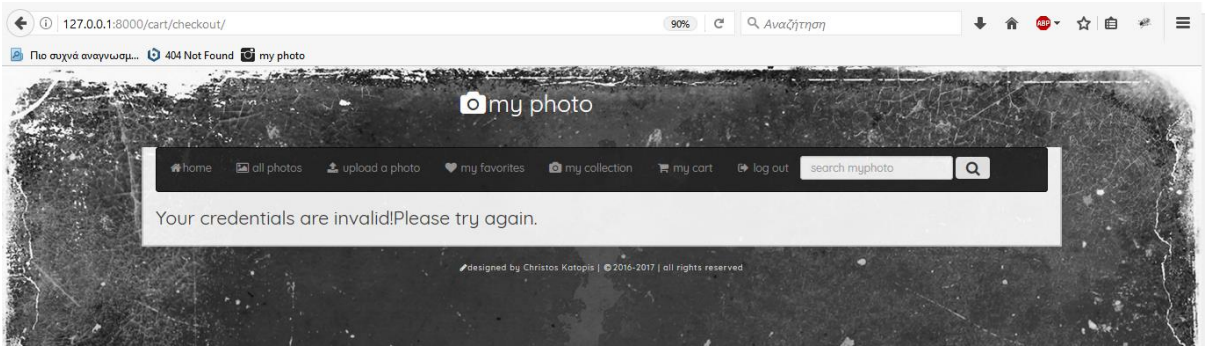


Εικόνα 192: Οθόνη επαλήθευσης των στοιχείων του χρήστη

***Σημείωση**: Για τις ανάγκες της εργασίας, ο τρόπος με τον οποίο πιστοποιείται ο αριθμός πιστωτικής κάρτας και ολοκληρώνεται μια αγορά δεν έχει υλοποιηθεί. Σε πραγματικές συνθήκες λειτουργίας μιας εφαρμογής,ο πιο αποδεκτός τρόπος θα ήταν η ανακατεύθυνση σε ιστότοπο*

τράπεζας στην οποία θα ολοκληρωνόταν η χρέωση της αντίστοιχης πιστωτικής κάρτας, κάτι που δεν θεωρείται απαραίτητο για τους σκοπούς που εξυπηρετεί η παρούσα εργασία.

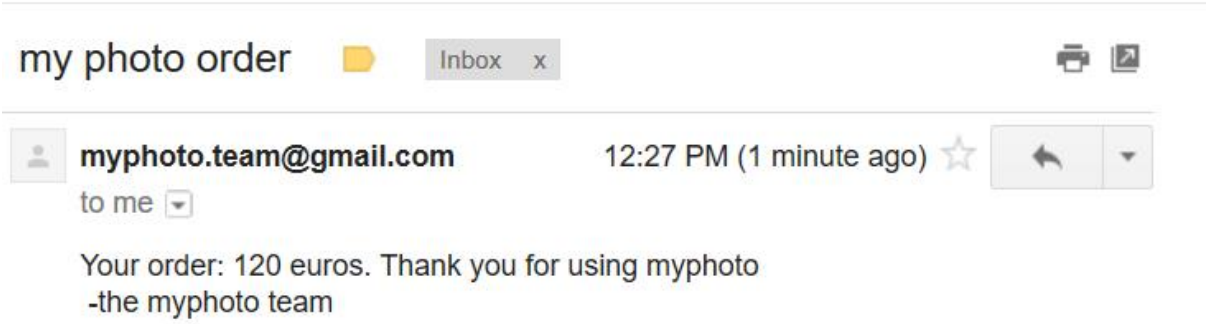
Σε περίπτωση που ο χρήστης εισάγει λάθος διαπιστευτήρια, εμφανίζεται σχετικό μήνυμα όπως φαίνεται στην εικόνα 193.



Εικόνα 193: Μήνυμα σε περίπτωση εισαγωγής λάθος διαπιστευτηρίων

5.10.1.2 Καταγραφή

Σε περίπτωση επιτυχίας οποιαδήποτε αγοράς,ένα e – mail που αναγράφει το χρηματικό ποσοτικό της παραγγελίας αποστέλλεται στη διεύθυνση e – mail που αντιστοιχεί στον χρήστη που την πραγματοποίησε. Αυτό γίνεται ώστε ο χρήστης να ενημερώνεται κάθε φορά που πραγματοποιείται μια αγορά, μειώνοντας τους κινδύνους πραγματοποίησης αγοράς ακόμη και μετά την υποκλοπή των διαπιστευτηρίων και του αριθμού πιστωτικής κάρτας από έναν κακόβουλο χρήστη.



Εικόνα 194: E – mail που αποστέλλεται σε περίπτωση επιτυχημένης ολοκλήρωσης αγοράς από το myphoto

5.10.1. 3 Εμπιστευτικότητα

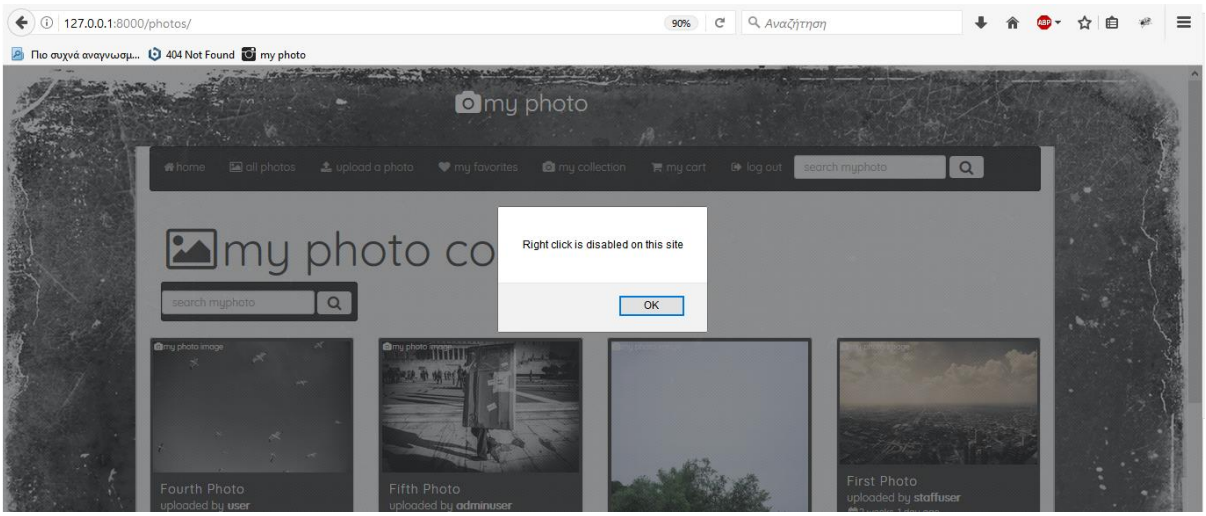
Μία άλλη μέθοδος που είναι χρήσιμη σε μια διαδικτυακή εφαρμογή που επιτελεί τον ρόλο online gallery είναι η αποτροπή των χρηστών από την δωρεάν λήψη των φωτογραφιών. Για το λόγο

αυτό, χρησιμοποιείται κώδικας Javascript ο οποίος αποτρέπει έναν χρήστη από το να κάνει δεξί κλικ σε οποιοδήποτε σημείο του myphoto. Ο κώδικας αυτός φαίνεται στην εικόνα 195. .

```
var message="Right click is disabled on this site";

function clickIE4(){
if (event.button==2){
alert(message);
return false;
}
}

function clickNS4(e){
if (document.layers||document.getElementById&&!document.all){
if (e.which==2||e.which==3){
alert(message);
return false;
}
}
}

if (document.layers){
document.captureEvents(Event.MOUSEDOWN);
document.onmousedown=clickNS4;
}
else if (document.all&&!document.getElementById){
document.onmousedown=clickIE4;
}

document.oncontextmenu=new Function("alert(message);return false")

```

Εικόνα 195: Κώδικας που αποτρέπει το δεξί κλικ σε οποιοδήποτε σημείο του site του myphoto

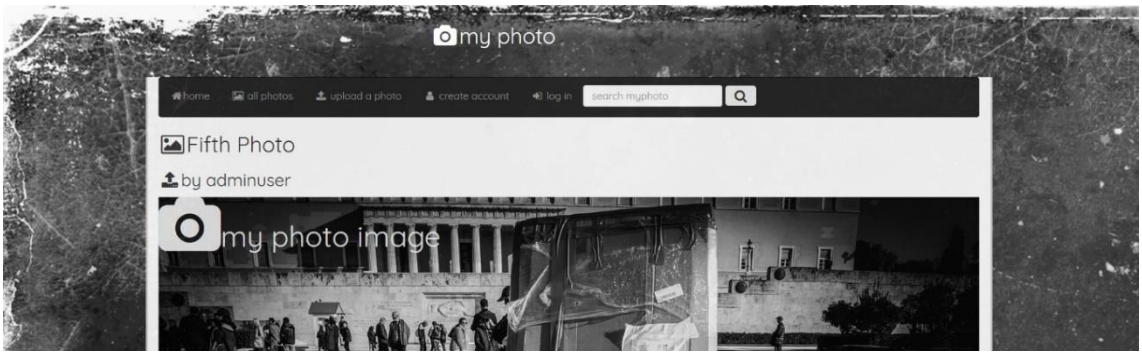
Όπως φαίνεται στην εικόνα 196, σε περίπτωση που ένας χρήστης κάνει δεξί κλικ εμφανίζεται το μήνυμα “**Right click is disabled on this site”** :



Εικόνα 196: Μήνυμα που εμφανίζεται σε περίπτωση που ο χρήστης επιχειρήσει να κάνει δεξί κλικ σε σημείο του myphoto

Με αυτόν τον τρόπο ένας χρήστης δεν μπορεί να κατεβάσει μια εικόνα που βλέπει, ούτε και να προβάλλει τον πηγαίο κώδικα της σελίδας ώστε να δει το URL στο οποίο βρίσκεται η φωτογραφία και να κατευθυνθεί σε αυτό, προβάλλοντάς τη.

Την ίδια στιγμή με χρήση κατάλληλου κώδικα css, σε κάθε φωτογραφία σε σημείο πάνω αριστερά υπάρχει «υδατογράφημα» (συγκεκριμένα η φράση **«my photo image»**, πράγμα που αποτρέπει τον χρήστη να υποκλέψει ολόκληρη την εικόνα με άλλους τρόπους (screenshot κ.ο.κ).



Εικόνα 197:Υδατογράφημα που υπάρχει σε κάθε φωτογραφία του myphoto

5.10.2 Ζητήματα Ασφάλειας στον Web Server

Μία ακόμη μέθοδος ασφάλειας είναι η απόκρυψη όσων περισσότερων πληροφοριών είναι δυνατό. Ο web server apache εκ των προτέρων, σε περίπτωση που ένας χρήστης πραγματοποιήσει πρόσβαση σε ένα URL που περιέχει αρχεία, παρουσιάζει τα αρχεία αυτά στον χρήστη (directory listing). Με αυτό τον τρόπο ένας κακόβουλος χρήστης έχει τη δυνατότητα να

πραγματοποιήσει λήψη των αρχείων αυτών ή να αντλήσει περισσότερες πληροφορίες για τη δομή ενός site.

Για να αποφευχθεί κάτι τέτοιο, απενεργοποιούμε το directory listing του apache web server προσθέτοντας τις παρακάτω γραμμές στο αρχείο **default-ssl.conf**:

```
<Directory /var/www>
    Options -Indexes
</Directory>
```

Εικόνα 198: Απενεργοποίηση directory listing στον apache web server

Πράγματι, παρατηρείται ότι με πρόσβαση στο URL **https:/192.168.1.100/static/** ο apache εμφανίζει στον χρήστη μήνυμα **403** αντί να παραθέσει τα αρχεία που βρίσκονται στο συγκεκριμένο directory.

Εικόνα 199: Μήνυμα που εμφανίζεται σε περίπτωση προβολής των περιεχομένων του φάκελου static

Κεφάλαιο 6: Έλεγχος Ασφάλειας της Διαδικτυακής Εφαρμογής με Χρήση του Εργαλείου Ανάλυσης Ευπαθειών Nessus

Στο Κεφάλαιο 5 περιγράφηκε ο ασφαλής σχεδιασμός μιας εφαρμογής βάσει των βασικών αρχών ασφάλειας, των πιο βασικών ευπαθειών ασφάλειας και των προτεινόμενων τρόπων αντιμετώπισής τους σύμφωνα με τον OWASP (και παρατέθηκαν στο Κεφάλαιο 2). Αυτό έγινε εφικτό με την αξιοποίηση των εργαλειών ασφάλειας που προσφέρει το προγραμματιστικό πλαίσιο Django (και παρουσιάστηκαν αναλυτικά στο Κεφάλαιο 4).

Πέρα από την τήρηση των βασικών αρχών ασφάλειας και των οδηγιών που προτείνει ο οργανισμός OWASP,για την αξιολόγηση της αποτελεσματικότητας της χρήσης του Django θα χρησιμοποιηθεί η μέθοδος της ανάλυσης ευπαθειών. Για την υλοποίηση της μεθόδου αυτής, θα αξιοποιηθεί ένα από τα πιο γνωστά και αξιόπιστα εργαλεία ανάλυσης και αξιολόγησης ευπαθειών, το **Nessus Vulnerability Scanner**.

6.1 Ανάλυση Ευπαθειών

Η ανάλυση ευπαθειών (**vulnerability analysis**) είναι μία διαδικασία εύρεσης, αναγνώρισης και αξιολόγησης ευπαθειών και κενών ασφάλειας ενός συστήματος με βάση τις συνέπειες που θα μπορούσε να έχει η αξιοποίησή τους.

Η αξιολόγηση ευπαθειών μιας διαδικτυακής εφαρμογής συγκεκριμένα, αφορά στις συνέπειες που θα είχε η ενδεχόμενη εκμετάλλευσή τους από κακόβουλο χρήστη [73].

6.2 Εισαγωγή στο Nessus Vulnerability Scanner

Το Nessus είναι ένα open source client - server εργαλείο της **Tenable Network Security** που υλοποιεί την ανάλυση ευπαθειών με αυτοματοποιημένο τρόπο. Πρόκειται για ένα από τα πιο ευρέως διαδεδομένα εργαλεία ανάλυσης ευπαθειών που χρησιμοποιούνται για εταιρικούς σκοπούς. Ήδη το 2005, πάνω από 75.000 εταρείες έκαναν χρήση του Nessus για εντοπισμό ευπαθειών που αφορούσαν σε δίκτυα και διαδικτυακές εφαρμογές [73]. Το Nessus αξιοποιεί πάνω από 70.000 plugins -τα οποία είναι προγράμματα που εκτελούν επιθέσεις ή άλλου τύπου επικοινωνία- προκειμένου να εντοπίσει τυχόν ευπάθειες [74].

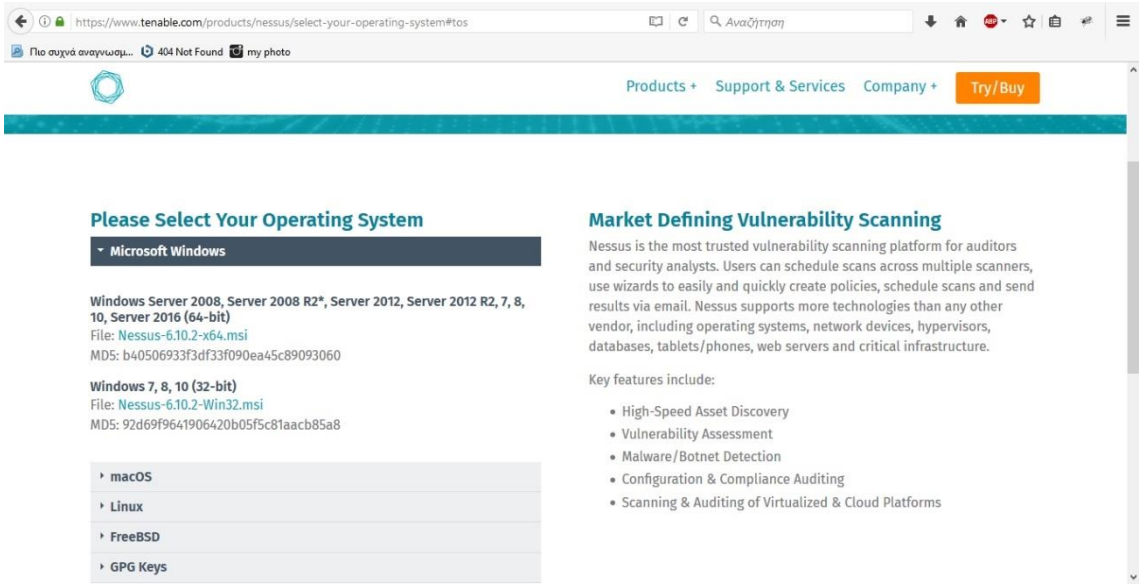
Ύστερα, βάσει ήδη καταχωρημένων CWE (Common Weakness Enumeration, βάσης δεδομένων αδυναμιών ασφάλειας που έχει δημιουργηθεί από ανοιχτή κοινότητα [75]) standards και ευπαθειών που έχουν βρεθεί και καταχωρηθεί στην βάση δεδομένων του Nessus, το Nessus τις αξιολογεί βάσει του CVSS (Common Vulnerability Scorring System, βάσης δεδομένων αξιολόγησης ευπαθειών που έχει δημιουργηθεί από ανοιχτή κοινότητα [76]) και τις συγκεντρώνει σε μια ενιαία αναφορά, που προβάλλεται στον χρήση.

Ένα σημαντικό πλεονέκτημα του Nessus έναντι άλλων εργαλειών ανάλυσης ευπαθειών που κυκλοφορούν είναι η συνεχής ενημέρωσή του με νέα ήδη ευπαθειών που ανακαλύπτονται [73],[74].

6.3 Εγκατάσταση του Nessus Vulnerability Scanner

Για τις ανάγκες της εργασίας, θα χρησιμοποιηθεί η πιο ενημερωμένη έκδοση του Nessus την περίοδο συγγραφής της εργασίας, το **Nessus 6.10**. Η χρήση του θα γίνει στο λειτουργικό σύστημα Windows 10 με στόχο τον έλεγχο του server (ο οποίος βρίσκεται στο εικονικό λειτουργικό σύστημα Ubuntu 14.04).

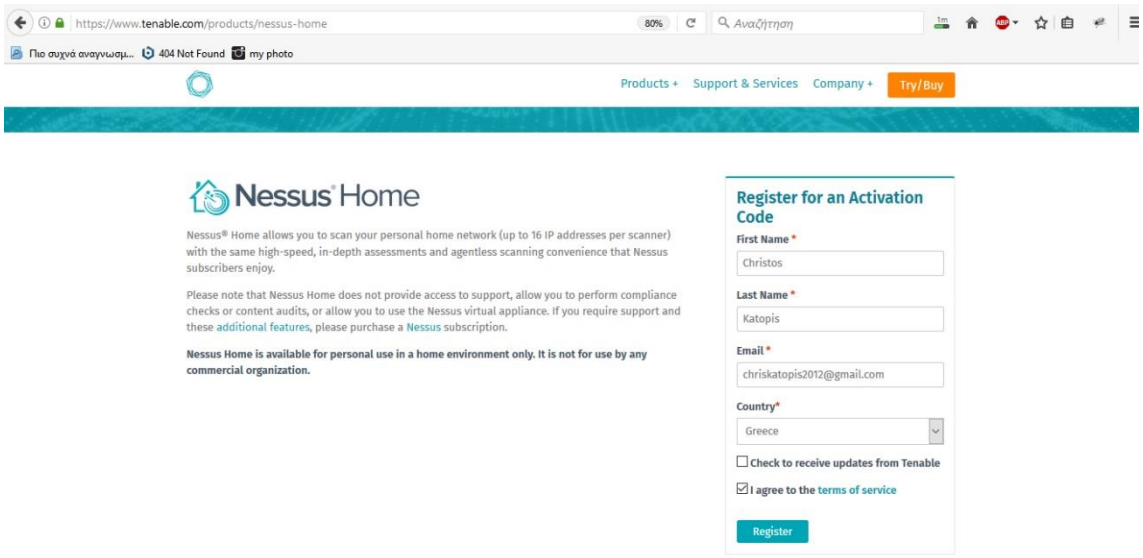
Για την εγκατάσταση του Nessus πραγματοποιείται πρόσβαση στην σελίδα **https://www.tenable.com/products/nessus/select-your-operating-system**. Ύστερα, επιλέγεται η έκδοση "**Microsoft Windows**" και πραγματοποιείται η λήψη του προγράμματος.



Εικόνα 200: Σελίδα λήψης του Nessus Vulnerability Scanner

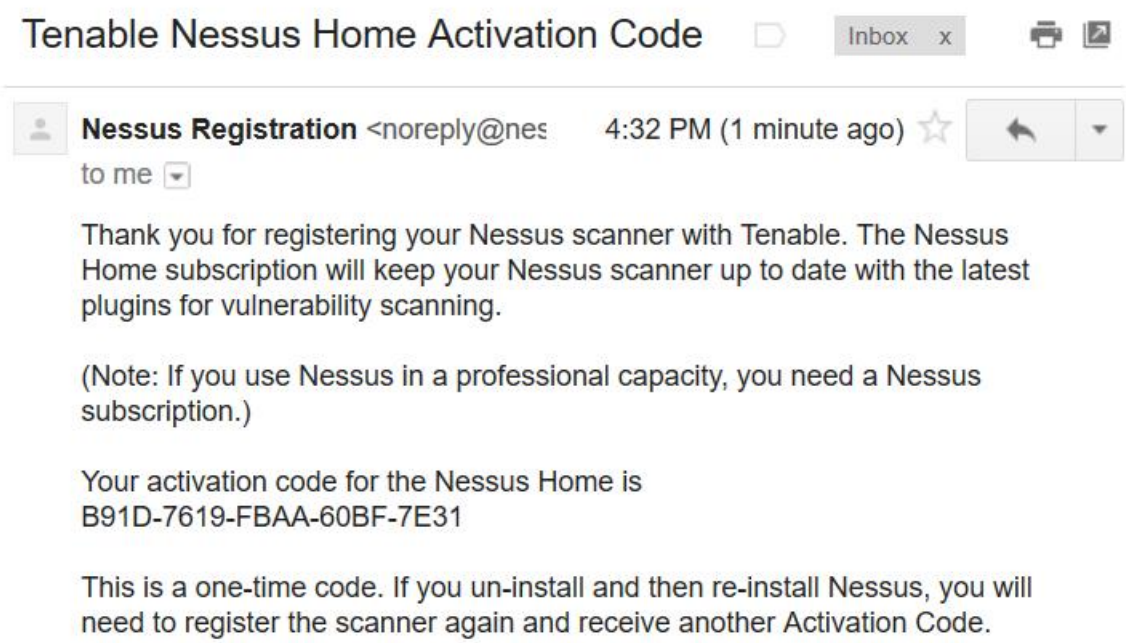
Για την χρήση του Nessus απαιτείται μοναδικός κωδικός ενεργοποίησης που παρέχεται μέσω e - mail. Σε περίπτωση χρήσης του Nessus για μη – εμπορικούς σκοπούς, ο κωδικός αυτός παρέχεται δωρεάν μέσω λήψης της Home έκδοσης που δίνει τη δυνατότητα ελέγχου δικτύου που αποτελείται μέχρι από 1 μέχρι 16 IP.

Με πρόσβαση στη σελίδα **https://www.tenable.com/products/nessus-home** συμπληρώνουμε τα κατάλληλα στοιχεία, προκειμένου να αποσταλεί ο κωδικός ενεργοποίησης, όπως φαίνεται στην εικόνα 201.



Εικόνα 201: Εισαγωγή στοιχείων για αποστολή κωδικού ενεργοποίησης

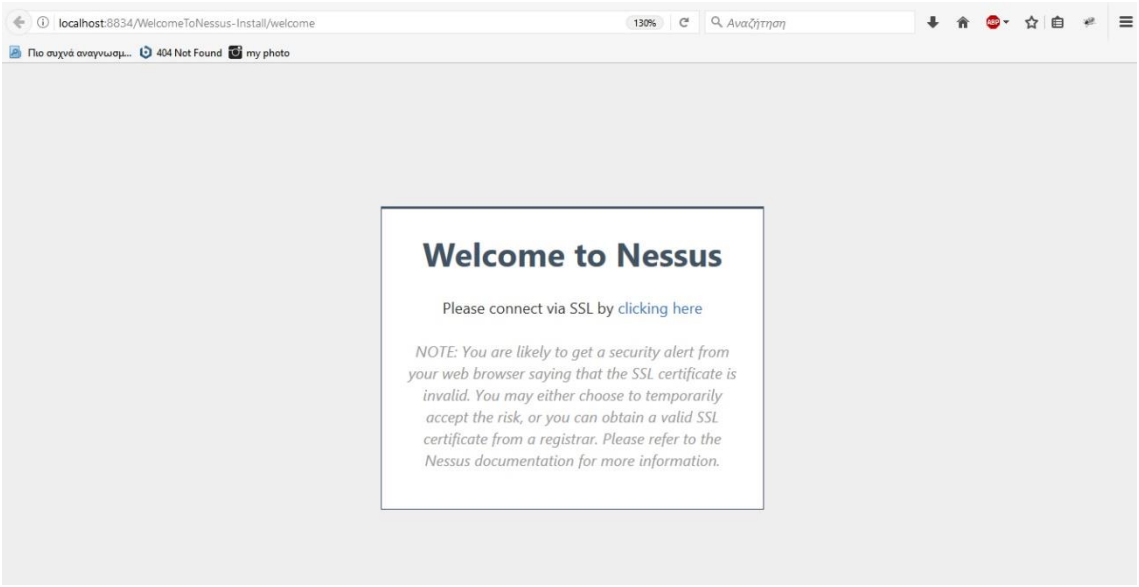
Μετά την εισαγωγή των κατάλληλων στοιχείων, παρατηρείται ότι έχει αποσταλεί e – mail με τον κωδικό ενεργοποίησης στη διεύθυνση e – mail που έχει εισαχθεί.



Εικόνα 202: E – mail που περιέχει τον κωδικό ενεργοποίησης

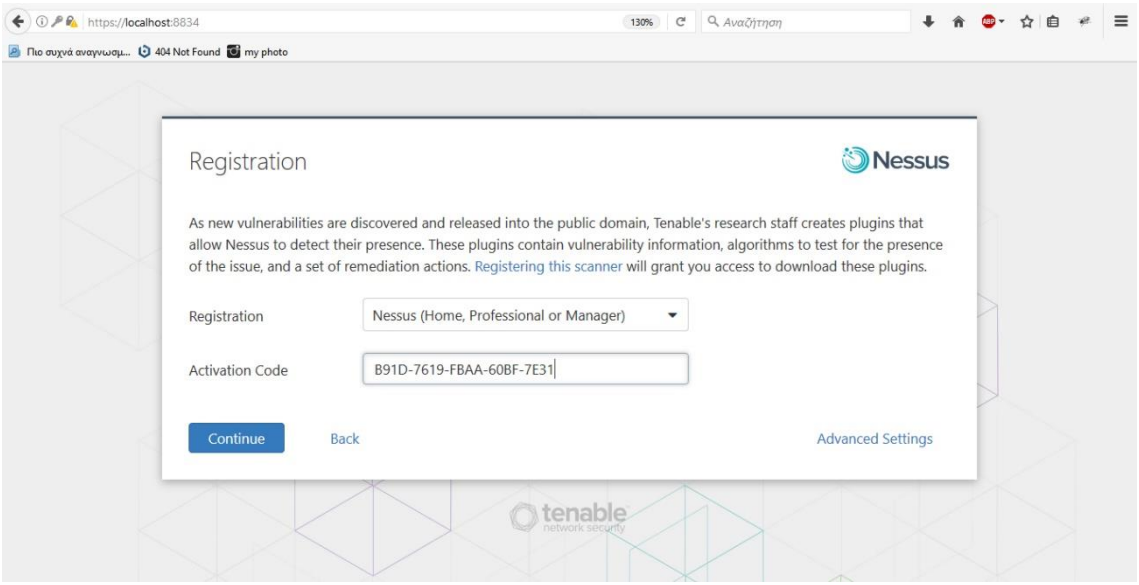
Για την εκκίνηση του Nessus πραγματοποιείται πρόσβαση στη σελίδα **https://localhost:8834**.Την πρώτη φορά που θα πραγματοποιηθεί η πρόσβαση, επιλέγουμε

την προσθήκη εξαίρεσης στην οθόνη που εμφανίζει μήνυμα για αδυναμία αναγνώρισης του πιστοποιητικού SSL.



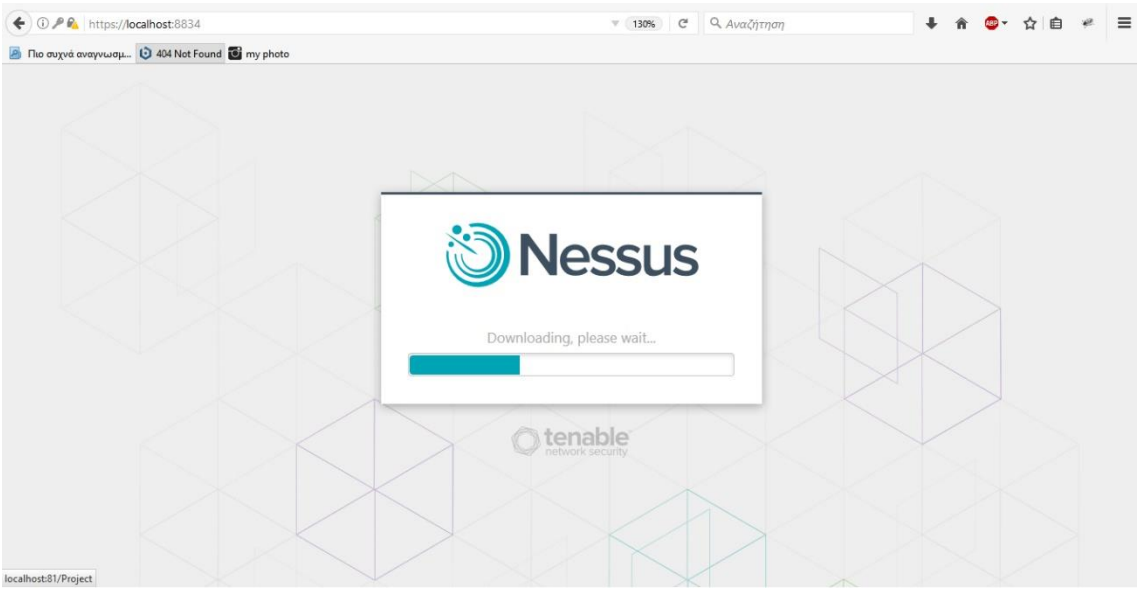
Εικόνα 203: Μήνυμα έναρξης Nessus

Ύστερα εισάγεται ο κωδικός ενεργοποίησης, όπως φαίνεται στην εικόνα 204.



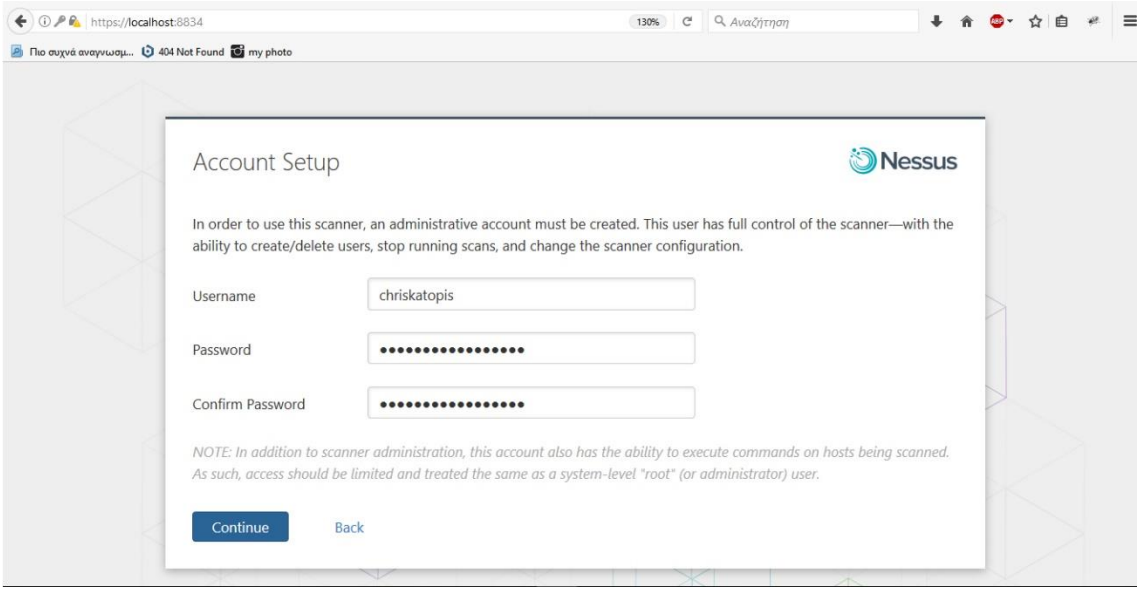
Εικόνα 204: Εισαγωγή κωδικού ενεργοποίησης του Nessus

Μετά την εισαγωγή του κωδικού ενεργοποίησης, γίνεται λήψη των plugins και όλων των δεδομένων που αξιοποιεί το Nessus για να πραγματοποιήσει την ανάλυση ευπαθειών.



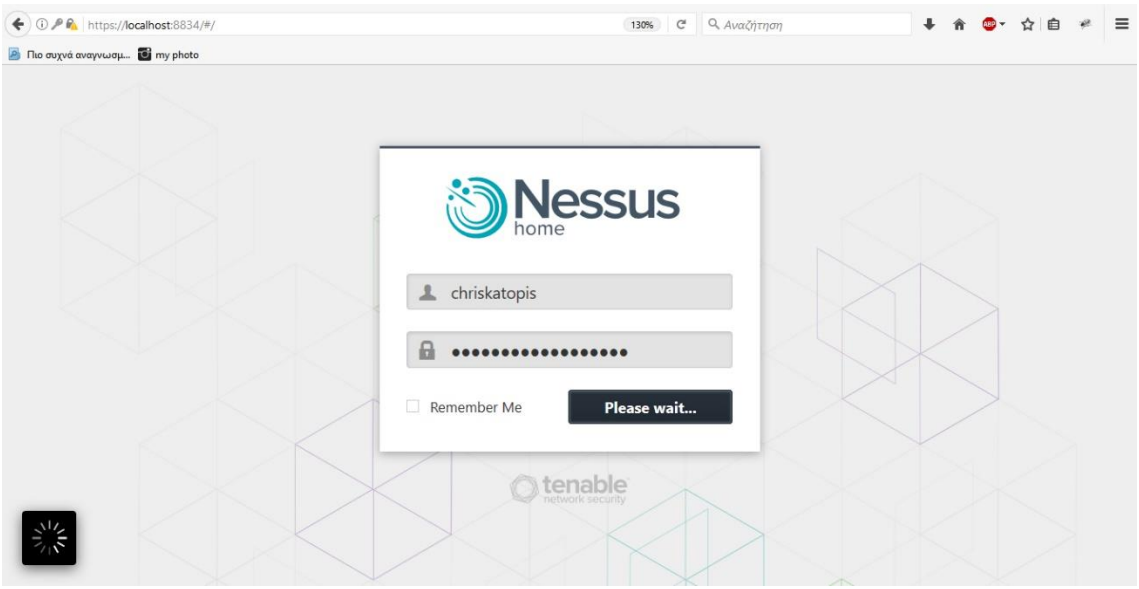
Εικόνα 205: Λήψη των απαραίτητων δεδομένων για τη λειτουργία του Nessus

Το επόμενο βήμα μετά την ολοκλήρωση της λήψης των απαραίτητων δεδομένων είναι η εισαγωγή διαπιστευτηρίων για δημιουργία λογαριασμού χρήστη της τοπικής εγκατάστασης του Nessus.



Εικόνα 206: Δημιουργία χρήστη του Nessus

Μετά τη δημιουργία λογαριασμού, πραγματοποείται έισοδος σε αυτόν με χρήση του ζεύγους διαπιστευτηρίων που έχει δημιουργηθεί.



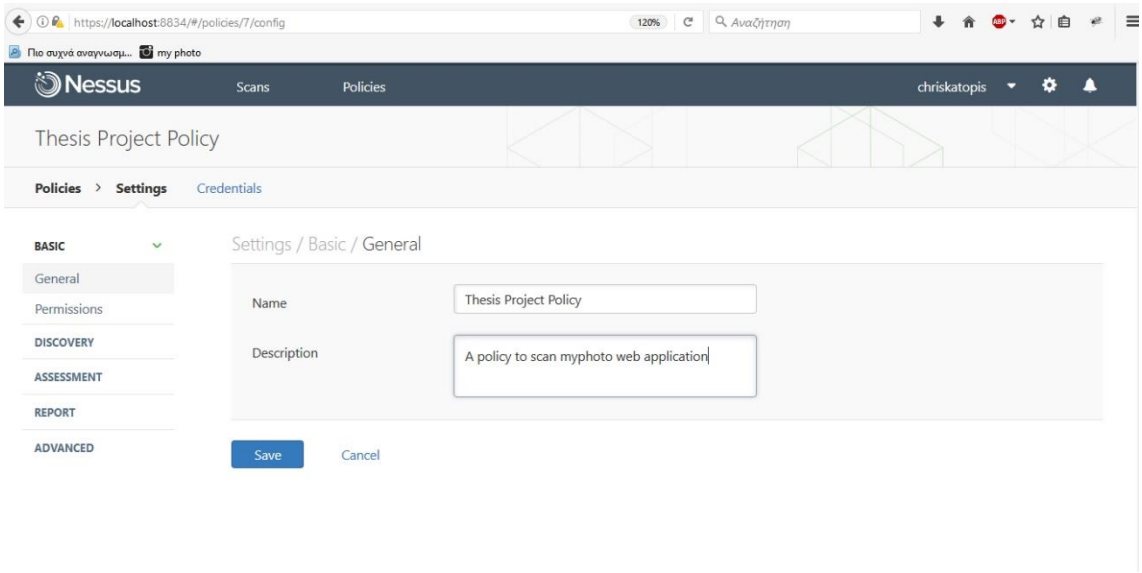
Εικόνα 207: Σύνδεση χρήστη του Nessus

Μετά την εισαγωγή κατάλληλων διαπιστευρίων, πραγματοποιείται έισοδος στην κεντρική σελίδα του Nessus. Το επόμενο βήμα είναι η πραγματοποίηση της σάρωσης της διαδικτυακής εφαρμογής myphoto για έλεγχο ευπαθειών. Η κατάλληλη διαδικασία περιγράφεται αναλυτικά στις επόμενες παραγράφους.

6.4 Δημιουργία Πολιτικής

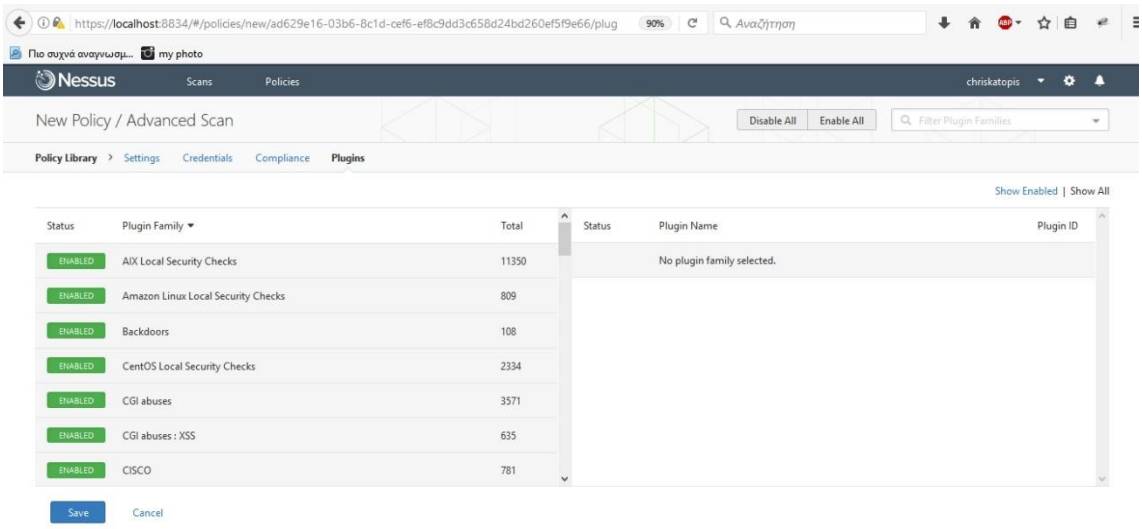
Όπως έχει αναφερθεί προηγουμένως, το Nessus επιτρέπει την ανάλυση ευπαθειών σε δίκτυα , web servers, διαδικτυακές εφαρμογές κ.α. Ο στόχος και ο τρόπος που θα πραγματοποιηθεί η ανάλυση αυτή εξαρτάται από την πολιτική (**policy**) που θα χρησιμοποιηθεί για κάθε σάρωση, τη στιγμή που θα επιλεγεί το **New Scan**. Το Nessus προσφέρει μία σειρά πολιτικών οι οποίες αφορούν σε διαφορετικού τύπου σαρώσεις (**Basic Network Scan, Web Application Scan κ.α.**) και διαφέρουν μεταξύ τους στα plugins και στις ρυθμίσεις που αξιοποιούνται.

Για τις ανάγκες της εργασίας, δημιουργείται νέα πολιτική επιλέγοντας το **Policies** και ύστερα το **New Policy**.



Εικόνα 208: Δημιουργία νέας πολιτικής σάρωσης

Το Nessus, αξιοποιεί εκ των προτέρων όλα τα plugins που έχουν καταχωρηθεί, δηλαδή όλα τα CGI scripts που εκτελούν επιθέσεις αλλά και αφορούν σε ευπάθειες web servers, λειτουργικών συστημάτων κ.ο.κ όπως φαίνεται στην εικόνα 209.



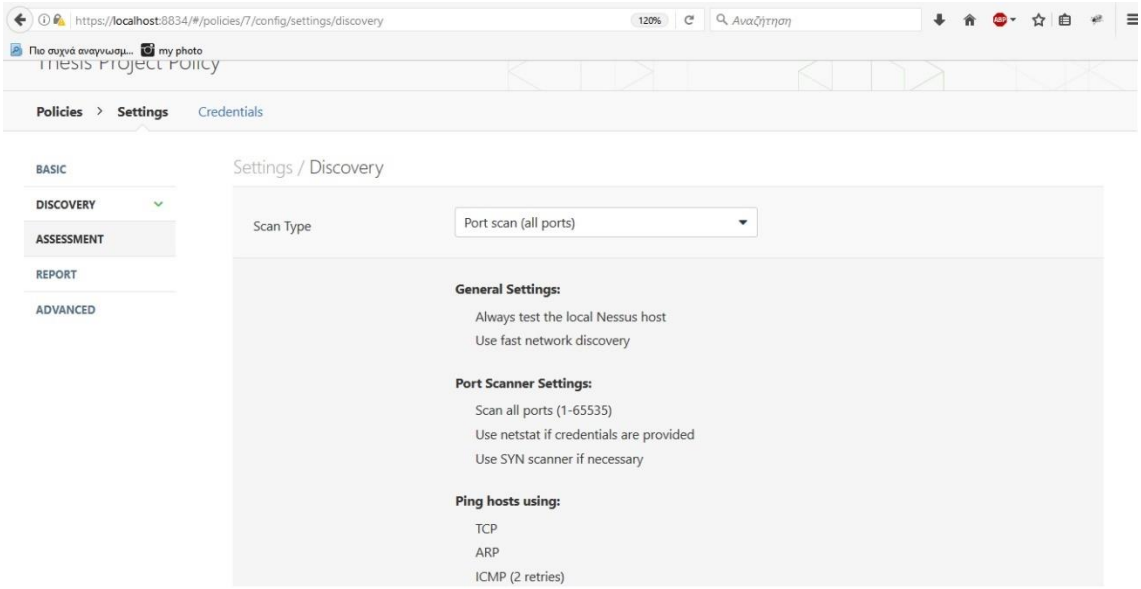
Εικόνα 209: Εγκατεστημένα και ενεργοποιημένα plugins του Nessus

Ο ορισμός πολιτικής προϋποθέτει τον καθορισμό ρυθμίσεων που θα ορίσουν τον τρόπο που θα πραγματοποιηθεί ο έλεγχος της διαδικτυακής εφαρμογής myphoto για ευπάθειες ασφάλειας.

Στο μενού αριστερά, υπάρχουν οι κατηγορίες των ρυθμίσεων της πολιτικής: **Discovery,**

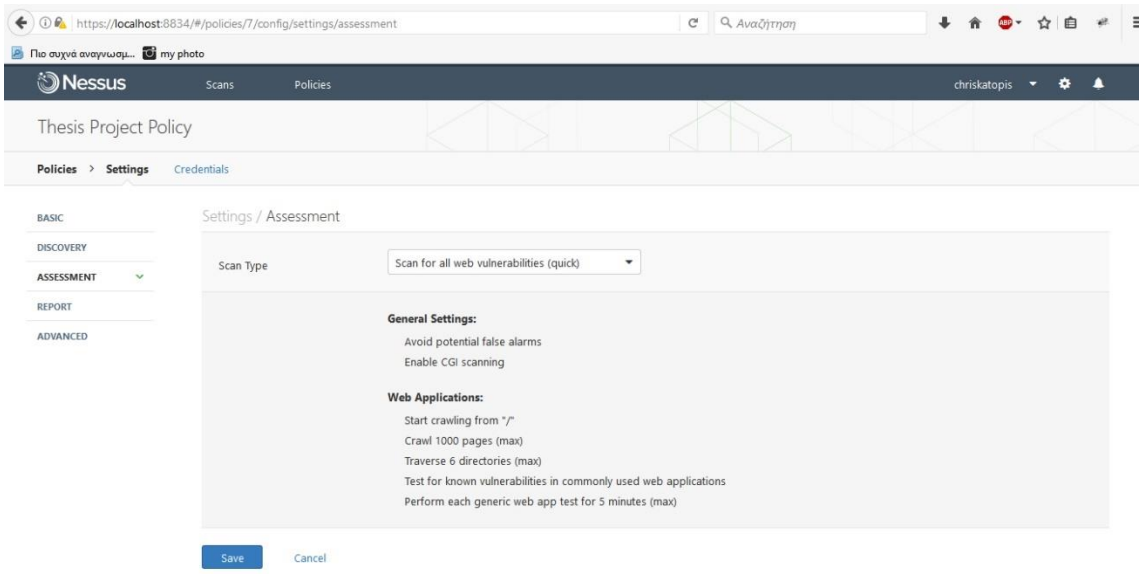
Assesment, Report και **Advanced**.

Στην κατηγορία **Advanced** γίνεται η επιλογή **Port Scan (all ports)**, η οποία διασφαλίζει ότι θα γίνει επικοινωνία με όλες τις ports του host ή των hosts που θα συμπεριληφθούν στον έλεγχο (συγκεκριμένα 1 – 65535) με πακέτα TCP, ARP αλλά και ICMP προκειμένου να διαπιστωθεί ποιές είναι ανοιχτές.



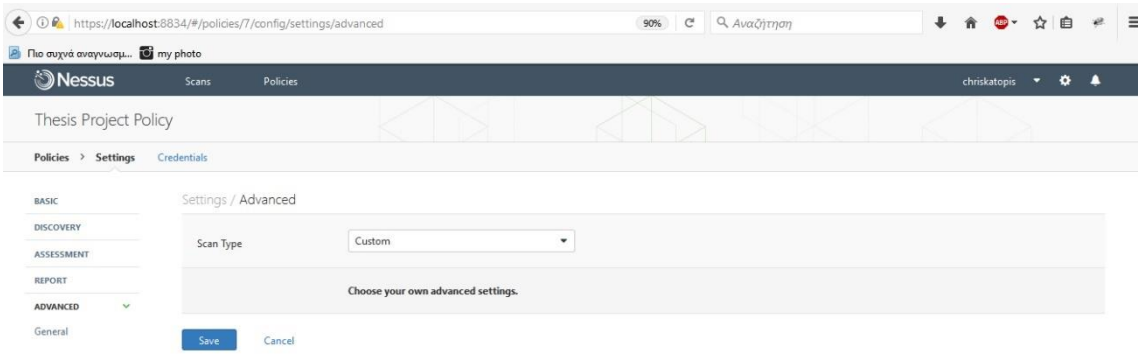
Εικόνα 210: Ρυθμίσεις Discovery

Στην επόμενη κατηγορία, **Assessment**, επιλέγεται το **Scan for all web vulnerabilites (quick)**, επιλογή που διασφαλίζει ότι θα γίνουν οι έλεγχοι για όλους τους πιθανούς τύπους ευπαθειών στον host που θα επιλεγεί για τη σάρωση, φροντίζοντας ότι η διάρκεια κάθε ελέγχου θα περιοριστεί τόσο ώστε όλων των ειδών οι έλεγχοι να ολοκληρωθούν επιτυχώς. (μιας και η μεγαλύτερη χρονική διάρκεια έχει τον κίνδυνο ο έλεγχος για κάποιου τύπου ευπάθεια να μην ολοκληρωθεί επιτυχώς και να κάνει timeout).



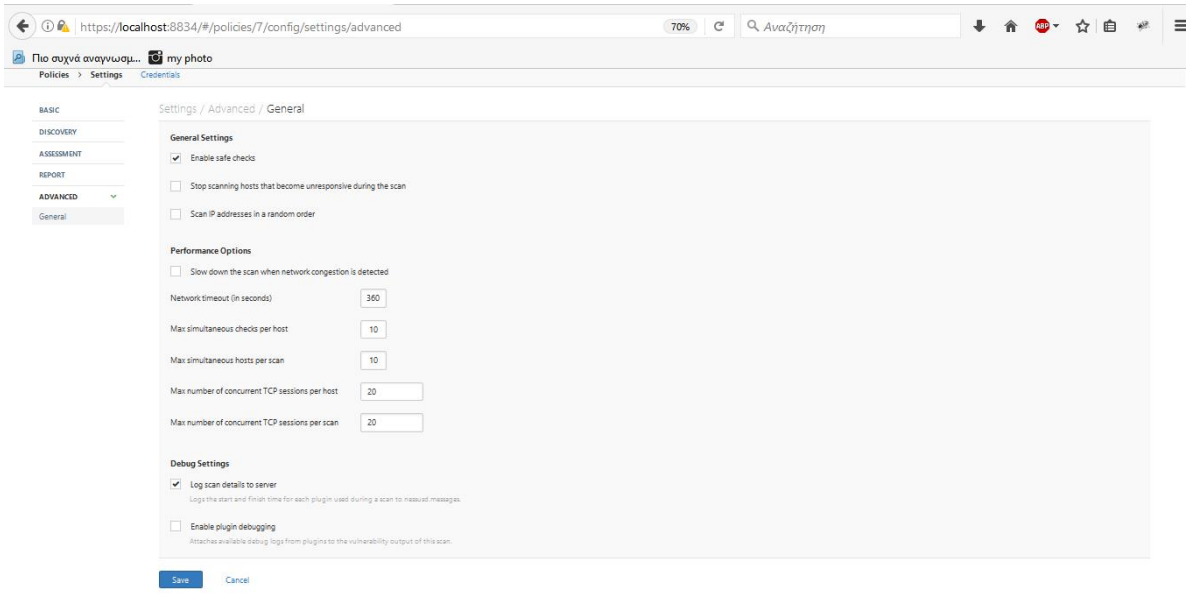
Εικόνα 211: Ρυθμίσεις Assessment

Στην επόμενη κατηγορία, το **Report**, γίνεται η επιλογή **Custom** γεγονός που επιτρέπει στον χρήστη να καθορίσει στην κατηγορία **Advanced** συγκεκριμένες ρυθμίσεις σχετικά με κάθε σάρωση που θα ακολουθήσει την πολιτική που ορίζεται.



Εικόνα 212: Ρυθμίσεις Report #1

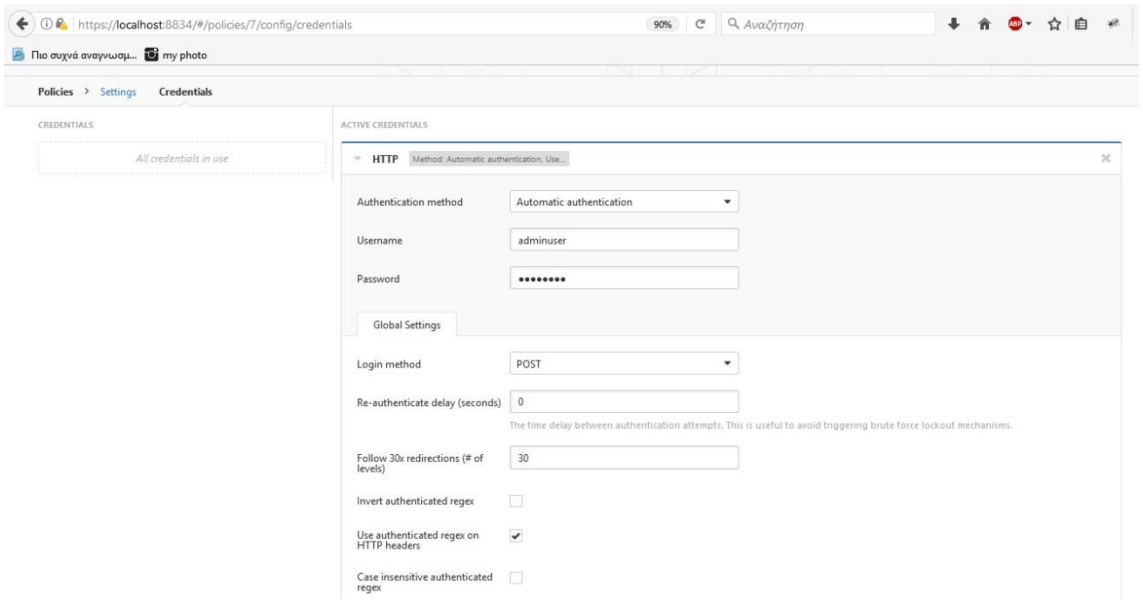
Μετά την ολοκλήρωση αυτής της επιλογής στην υποκατηγορία **General** της κατηγορίας **Advanced** γίνονται επιλογές σε ρυθμίσεις που αφορούν στον τρόπο του ελέγχου αλλά και στην αποθήκευση και στην προβολή της αναφοράς που θα προκύψει.



Εικόνα 213: Ρυθμίσεις Report #2

Όπως φαίνεται και στην εικόνα 213 επιλέγεται μεγάλο χρονικό διάστημα **Network Timeout**, προκειμένου να μην προκύψει κάποιο ζήτημα σε περίπτωση καθυστερημένης απόκρισης του host, αλλά και μεγάλος αριθμός **TCP συνόδων** ανά host που ελέγχεται.

Τέλος, έχοντας ολοκληρώσει τις ρυθμίσεις όλων των κατηγοριών του Settings, επιλέγουμε την κατηγορία **Credentials**, η οποία αφορά στις ρυθμίσεις που απαιτούνται για τις ενδεχόμενες σελίδες ενός server ενός host που θα ελεγθεί, στις οποίες απαιτείται είσοδος χρήστη.

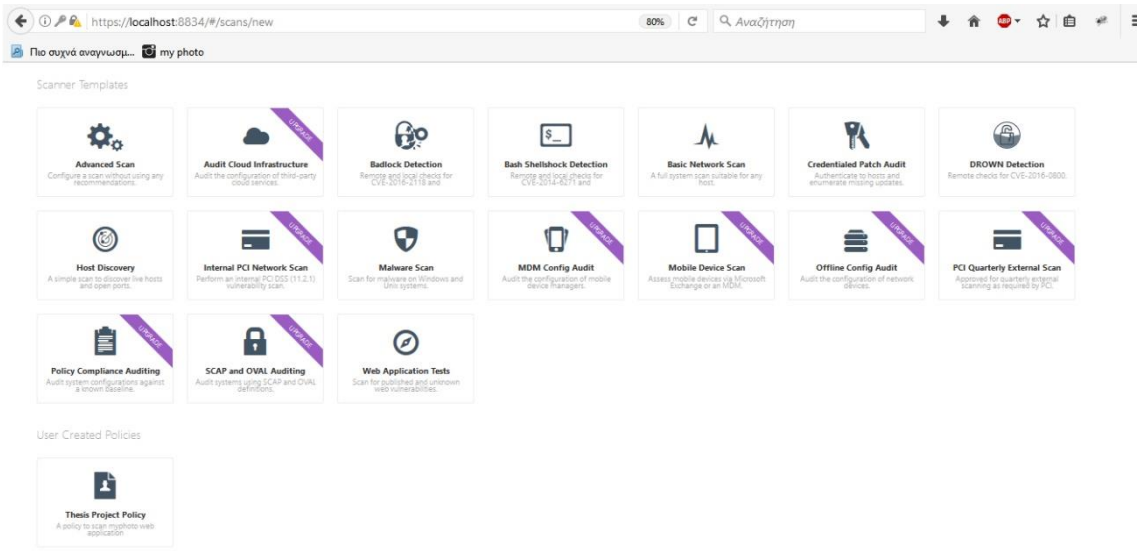


Εικόνα 214: Ρυθμίσεις Credentials

Όπως φαίνεται στην εικόνα, πραγματοποιείται η επιλογή **Automatic authentication** και εισάγεται ζεύγος διαπιστευτηρίων χρήστη του myphoto. Σε αυτό το σημείο, η νέα πολιτική με τίτλο **"Thesis Project Policy"** έχει δημιουργηθεί, και πλέον μπορεί να χρησιμοποιηθεί για σάρωση.

6.5 Έλεγχος της Διαδικτυακής Εφαρμογής για Ευπάθειες Ασφάλειας

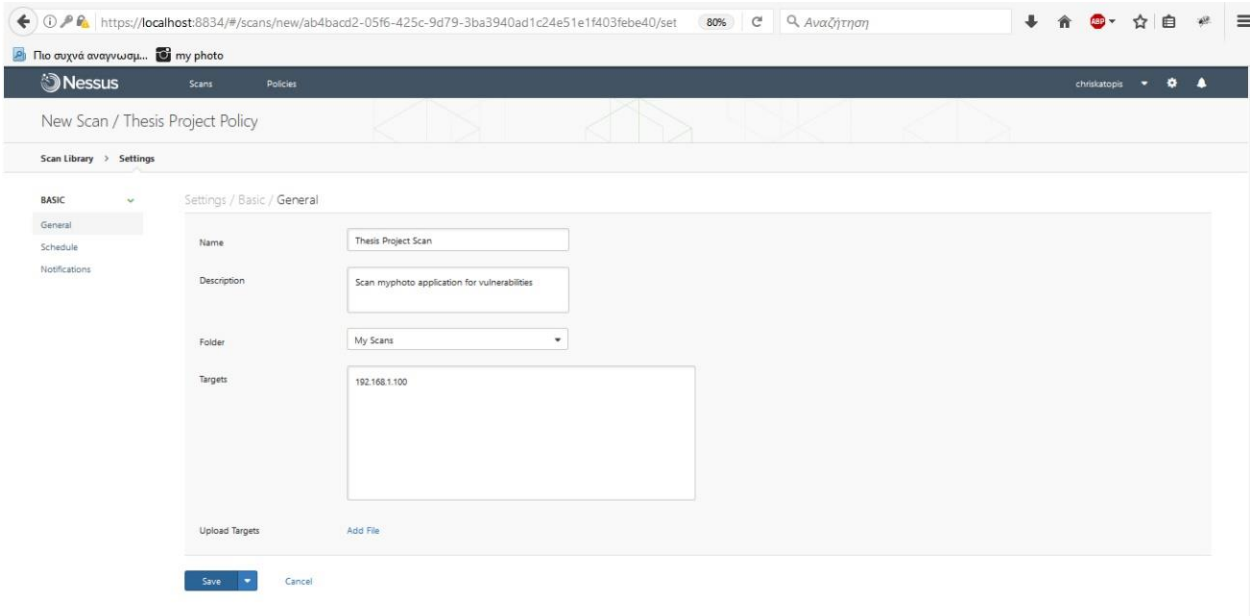
Έχοντας ολοκληρώσει τη δημιουργία της πολιτικής, επιλέγουμε **New Scan** και ύστερα το **Thesis Project Policy** από τη λίστα πολιτικών του Nessus που εμφανίζεται.



Εικόνα 215: Επιλογή πολιτικής σάρωσης

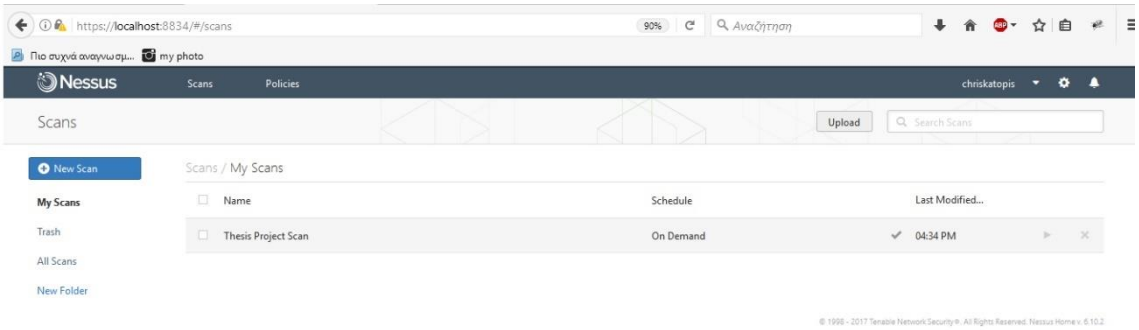
Στην οθόνη που προκύπτει ορίζονται ρυθμίσεις όπως το όνομα της σάρωσης αλλά και ποιές IP θα ελεγχούν. Σε αυτή τη ρύθμιση (**Targets**) εισάγουμε την IP **192.168.1.100** που αντιστοιχεί στο εικονικό λειτουργικό σύστημα Ubuntu που φιλοξενεί τον apache web server, ο οποίος εξυπηρετεί τη διαδικτυακή εφαρμογή myphoto.

Όπως φαίνεται στην εικόνα 216, στις ρυθμίσεις της σάρωσης μπορεί να καθοριστούν επιπλέον ημερομηνίες και ώρες που θα πραγματοποιείται ο έλεγχος (**Schedule**) ή επιπλέον επιλογές για αποστολή μηνυμάτων σχετικά με τα αποτελέσματα του ελέγχου (**Notifications**).



Εικόνα 216: Επιλογές του ελέγχου ευπαθειών

Μετά την ολοκλήρωση των ρυθμίσεων της σάρωσης, ο νέος τύπος σάρωσης που δημιουργήθηκε, με τίτλο **“Thesis Project Scan”** εμφανίζεται στην κεντρική σελίδα του Nessus.

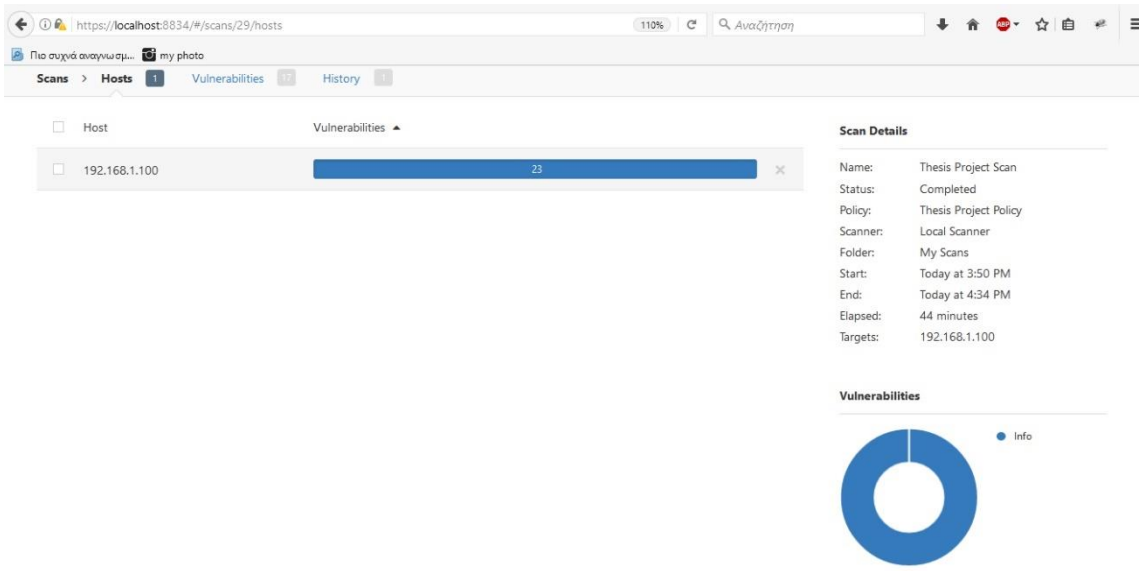


Εικόνα 217: Έναρξη ελέγχου ευπαθειών

Επιλέγουμε το «Thesis Project Scan» που μόλις δημιουργήσαμε και περιμένουμε την ολοκλήρωσή του, για την εμφάνιση των αποτελεσμάτων.

6.6 Αποτελέσματα Ανάλυσης Ευπαθειών

Μετά την ολοκλήρωσή του ελέγχου, η συνολική εικόνα των αποτελεσμάτων εμεφανίζεται στην κεντρική σελίδα του Nessus, όπως φαίνεται στην εικόνα 218.



Εικόνα 218: Συνολική εικόνα αποτελεσμάτων του ελέγχου ευπαθειών

Όπως αποκαλύπτεται και στην εικόνα, στον host που έχει ελεγχθεί έχουν εμφανιστεί 23 ευπάθειες ασφάλειας, το 100% των οποίων ανήκουν στην κατηγορία Info.

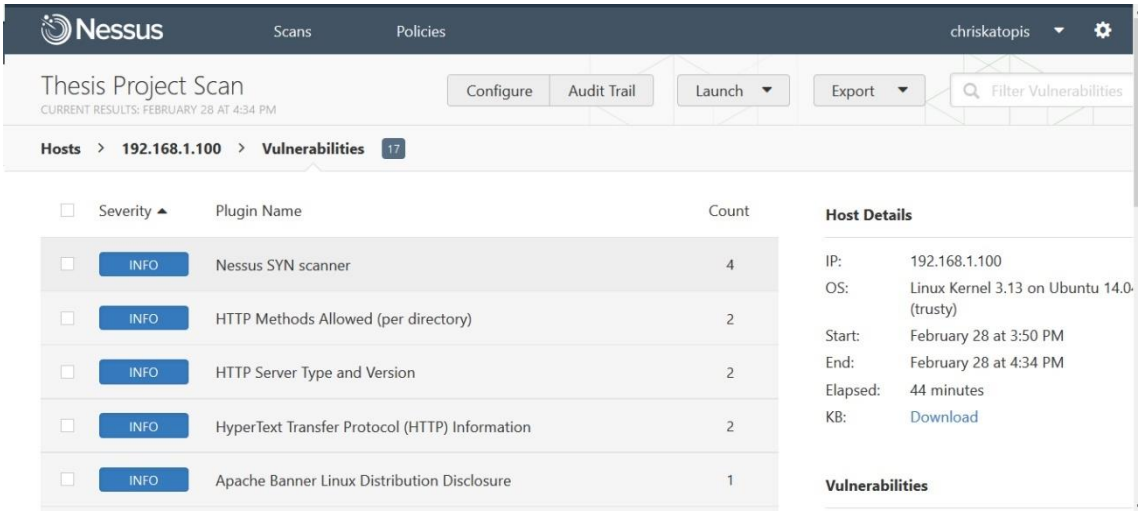
Συγκεκριμένα, το Nessus βάσει της επικινδυνότητας (δηλαδή της συνέπειας που μπορέι να έχει η εκμετάλλευσή της) μιας ευπάθειας και της βαθμολογίας CVSS που έχει η κατηγορία στην οποία ανήκει χωρίζει τα αποτελέσματα των ελέγχων που πραγματοποιεί σε τέσσερις κατηγορίες**: Info, Low, Medium και High**.

Στην κατηγορία Info τοποθετούνται πληροφορίες που εξάγει το Nessus από τον έλεγχο και δεν έχουν άμεσες συνέπειες: στην πραγματικότητα, μπορεί να μην πρόκειται καν για ευπάθειες, αλλά απλώς για πληροφορίες τις οποίες ανακτά το Nessus και σχετίζονται με τον τρόπο λειτουργίας του web server και της εφαρμογής που βρίσκονται στο σύστημα που αντιστοιχεί IP η οποία ελέγχεται.

Κάνοντας κλικ στην αναφορά, αποκαλύπτεται μια πιο λεπτομερής αναφορά των αποτελεσμάτων, που παρουσιάζεται στις εικόνες 219,220,222,223.

Όπως φαίνεται παρακάτω, το Nessus κατηγοριοποιεί τις ευπάθειες ανά είδος, και ταυτόχρονα ταυτόχρονα αναγράφει το πλήθος των ευπαθειών που σχετίζονται με την κάθε κατηγορία. Συγκεκριμένα, τα είδη των αποτελεσμάτων, σύμφωνα με την ταξινόμηση του Nessus, όπως φαίνεται και στην εικόνα 219 είναι 17.

Αναλυτικότερα, τα αποτελέσματα που εμφανίζει το Nessus είναι τα εξής:



Εικόνα 219: Αναλυτικά αποτελέσματα ελέγχου ευπαθειών #1

α) Nessus SYN Scanner (4): Το συγκεκριμένο αποτέλεσμα αναφέρει τις ports που είναι ανοιχτές. Συγκεκριμένα, εμφανίζει την 80 που χρησιμοποιείται για την επικοινωνία μέσω HTTP, την 443 που χρησιμοποιείται για την επικοινωνία μέσω HTTPS και τις 21 και 22 που χρησιμοποιούνται για την μεταφόρτωση αρχείων.

β) HTTP Methods Allowed (per directory) (2): Το συγκεκριμένο αποτέλεσμα αναφέρει τις HTTP μεθόδους που επιτρέπονται μέσω HTTP και HTTPS σύνδεσης.

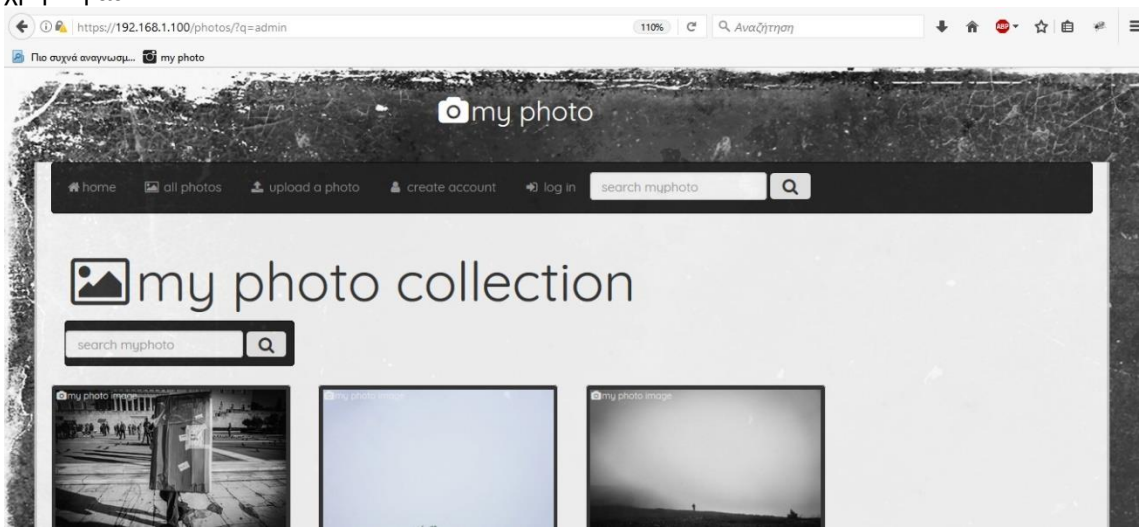
γ) HyperText Tranfer Protocol (HTTP) Information (2): Το συγκεκριμένο αποτέλεσμα αναφέρει τα πρωτόκολλα που χρησιμοποιούνται για την επικοινωνία με τον web server που βρίσκεται στο εικονικό ηλεκτρονικό σύστημα και συγκεκριμένα το HTTP και HTTPS.

δ) Apache Banner Linux Distribution Disclosure (1): Το συγκεκριμένο αποτέλεσμα αναφέρει ότι στις HTTP responses που επιστρέφει ο web server αποκαλύπτεται η έκδοση του web server (Apache 2.4.7) αλλά και του λειτουργικού συστήματος (Ubuntu 14.04), γεγονός που μπορεί να δώσει πληροφορίες που να αξιοποιηθούν από κακόβουλο χρήστη.

Εικόνα 220: Αναλυτικά αποτελέσματα ελέγχου ευπαθειών #2

ε) CGI Generic Injectable Parameter (1): Το συγκεκριμένο αποτέλεσμα αναφέρεται στην μεταβλητή **q**, που χρησιμοποιείται για την αναζήτηση αποτελεσμάτων στο myphoto. Συγκεκριμένα, αν στο myphoto πραγματοποιηθεί μια αναζήτηση, τότε το string το οποίο αναζητήθηκε, εμφανίζεται στο URL, ως τιμή της μεταβλητής **q**.

Για παράδειγμα, στην εικόνα αναζητούνται οι φωτογραφίες που έχουν μεταφορτωθεί από τον χρήστη admin:



Εικόνα 221: Αναζήτηση φωτογραφιών στο myphoto

Όπως φαίνεται, το URL που αντιστοιχεί στο αποτέλεσμα της αναζήτησης είναι το **https://192.168.1.100/photos/?q=admin**. Το γεγονός αυτό επιτρέπει την χρήση τιμών για την μεταβλητή **q** από χρήστες και την επιστροφή αντίστοιχων αποτελεσμάτων, κάτι που θα μπορούσε να καταστήσει δυνατή την έγχυση κώδικα SQL. Αυτό το γεγονός επισημαίνει το Nessus ως πιθανή ευπάθεια, κάτι που δεν ισχύει, μιας και το Django ORM διασφαλίζει τον έλεγχο κάθε query πριν την εκτέλεσή του, αποτρέποντας την εκτέλεση κακόβουλων queries.

στ) CGI Generic Test Load Estimation (all tests) (1): Το συγκεκριμένο αποτέλεσμα αναφέρεται στον αριθμό των requests και των ελέγχων που πραγματοποιήθηκαν σε σχέση με κάθε είδους επίθεση στη διαδικτυακή εφαρμογή (**SQL Injection, XSS Scripting, Directory Traversal κ.ο.κ**) που εξυπηρετείται σε web server στην IP που ελέγχθηκε. Τα αποτελέσματα, παρουσιάζονται, ενδεικτικά, αναλυτικότερα στην εικόνα 222.

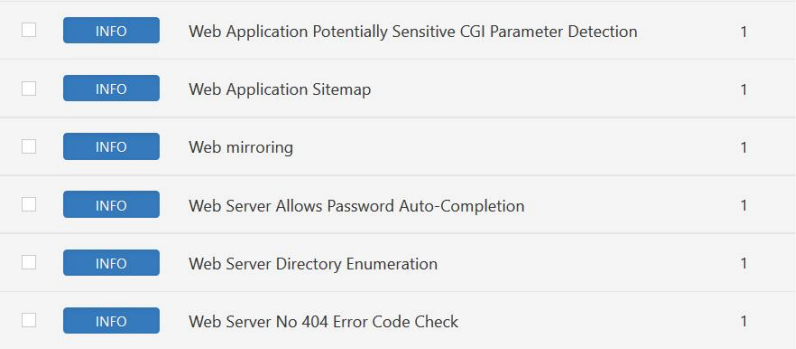
```
All tests                                    : S=5068      SP=13216     AP=13216     SC=54732     AC=54732

Here are the estimated number of requests in miscellaneous modes
for both methods (GET and POST) :
[Single / Some Pairs / All Pairs / Some Combinations / All Combinations]

cross-site scripting (comprehensive test): S=208       SP=544       AP=544       SC=2256      AC=2256
persistent XSS                           : S=208       SP=544       AP=544       SC=2256      AC=2256
arbitrary command execution              : S=832       SP=2176      AP=2176      SC=9024      AC=9024
web code injection                       : S=52        SP=136       AP=136       SC=564       AC=564
HTML injection                           : S=10        SP=10        AP=10        SC=10        AC=10
arbitrary command execution (time based) : S=312       SP=816       AP=816       SC=3384      AC=3384
script injection                         : S=2         SP=2         AP=2         SC=2         AC=2
XML injection                            : S=52        SP=136       AP=136       SC=564       AC=564
unseen parameters                        : S=1820      SP=4760      AP=4760      SC=19740     AC=19740
directory traversal (write access)       : S=104       SP=272       AP=272       SC=1128      AC=1128
SQL injection (2nd order)                : S=52        SP=136       AP=136       SC=564       AC=564
on site request forgery                  : S=2         SP=2         AP=2         SC=2         AC=2
blind SQL injection (4 requests)         : S=208       SP=544       AP=544       SC=2256      AC=2256
HTTP response splitting                  : S=18        SP=18        AP=18        SC=18        AC=18
directory traversal (extended test)      : S=2652      SP=6936      AP=6936      SC=28764     AC=28764
header injection                         : S=4         SP=4         AP=4         SC=4         AC=4
injectable parameter                     : S=104       SP=272       AP=272       SC=1128      AC=1128
directory traversal                      : S=1300      SP=3400      AP=3400      SC=14100     AC=14100
local file inclusion                     : S=52        SP=136       AP=136       SC=564       AC=564
cross-site scripting (extended patterns) : S=12        SP=12        AP=12        SC=12        AC=12
blind SQL injection                      : S=624       SP=1632      AP=1632      SC=6768      AC=6768
SQL injection                            : S=1248      SP=3264      AP=3264      SC=13536     AC=13536
SSI injection                            : S=156       SP=408       AP=408       SC=1692      AC=1692
format string                            : S=104       SP=272       AP=272       SC=1128      AC=1128

All tests                                : S=10136     SP=26432     AP=26432     SC=109464    AC=109464

Your mode : single, GET or POST.
Maximum number of requests : 5068
```

Εικόνα 222: Αναλυτικά αποτελέσματα όλων των τύπων επιθέσεων

ζ) External URLs (1): Το συγκεκριμένο αποτέλεσμα αναφέρει την εύρεση χρήσης εξωτερικού URL. Συγκεκριμένα, πρόκειται για το URL **https://fonts.googleapis.com/css?family=Bungee+Hairline|Cuprum|Fjalla+One|Oswald|Poiret+One|Quicksand** το οποίο χρησιμοποιείται προκειμένου να γίνει η χρήση φόντου της Google για τις ανάγκες της εμφάνισης της διαδικτυακής εφαρμογής.

η) HyperText Transfer Protocol (HTTP) Redirection Information (1): Το συγκεκριμένο αποτέλεσμα αναφέρει ότι οι συνδέσεις με τον web server μέσω HTTP ανακατευθύνονται μέσω HTTPS, που συμβαίνει λόγω της χρήσης της μεταβλητής **SECURE_SSL_REDIRECT=True** στο αρχείο **settings.py** της διαδικτυακής εφαρμογής myphoto.

θ) Missing or Permissive Content - Security – Policy HTTP Response Header (1): Το συγκεκριμένο αποτέλεσμα αναφέρεται στην έλλειψη της HTTP κεφαλίδας Content – Security – Policy από τα HTTP responses.



Εικόνα 223: Αναλυτικά αποτελέσματα ελέγχου ευπαθειών #3

ι) Web Application Potentially Sensitive CGI Parameter Detection (1): Το συγκεκριμένο αποτέλεσμα επισημαίνει τον κίνδυνο επίθεσης τύπου λεξικού (dictionary attack) που υπάρχει στη σελίδα εισόδου χρήστη του myphoto. Όμως αυτό αντιμετωπίζεται ήδη, μιας και, για τη δημιουργία ενός νέου κωδικού χρησιμοποιείται ο validator **CommonPasswordValidator** που πιστοποιεί ότι ο κωδικός δεν ταυτίζεται με κάποιον από λίστα των 1000 πιο κοινών κωδικών που χρησιμοποιείται, ενώ η χρήση των υπόλοιπων validators αυξάνουν την πολυπλοκότητά του, αυξάνοντας παράλληλα και τη δυσκολία επιτυχίας μιας τέτοιας επίθεσης.

κ) Web Application Sitemap (1): Το συγκεκριμένο αποτέλεσμα αναφέρει ότι στις σελίδες της διαδικτυακής εφαρμογής υπάρχουν σύνδεσμοι που οδηγούν σε URLs, κάνοντας έτσι εμφανή τα URLs του site (και άρα το sitemap) σε οποιοδήποτε χρήστη, κάτι που όμως είναι αναγκαίο προκειμένου το site να είναι στοιχειωδώς εύχρηστο.

Αξίζει να σημειωθεί ότι το URL **/myphoto_root** που αντιστοιχεί στο admin site της διαδικτυακής εφαρμογής δεν υπάρχει με τη μορφή συνδέσμου σε οποιοδήποτε site εξαιρώντας το από το sitemap και κάνοντας δυσκολότερη την πρόσβαση σε αυτό.

λ) Web Mirroring (1): Το συγκεκριμένο αποτέλεσμα αναφέρει την εύρεση πρωτοκόλλου CGI (συγκεκριμένα τη χρήση HTTP μεθόδων POST και GET) σε διάφορα URLs της διαδικτυακής εφαρμογής.

μ) Web Server Allow Password Auto – Completion (1): Το συγκεκριμένο αποτέλεσμα αναφέρει ότι ο κώδικας είναι τέτοιος που επιτρέπει σε browsers να συμπληρώνουν αυτόματα , από αποτελέσματα που έχουν αποθηκεύσει στην μνήμη τους, τα διαπιστευτήρια στις φόρμες εισόδου και δημιουργίας χρήστη που χρησιμοποιούνται στο myphoto. Αυτό θα μπορούσε να προκαλέσει πρόβλημα σε περίπτωση χρήσης του ίδιου υπολογιστή από παραπάνω από έναν χρήστες, μιας και θα υπήρχε πιθανότητα χρήσης των διαπιστευτηρίων ενός χρήστη του myphoto από άλλον χρήστη.

ν) Web Sever Directory Enumeration (1): Το συγκεκριμένο αποτέλεσμα αναφέρει την ύπαρξη directories με συνήθη ονόματα στον web server, και συγκεκριμένα των directories photos και cart.

ξ) Web Server No 404 Error Code Check (1): Το συγκεκριμένο αποτέλεσμα αναφέρει ότι ο apache web browser δεν έχει ρυθμιστεί κατάλληλα ώστε να επιστρέφεται το HTTP σφάλμα 404

σε περίπτωση request για URL που δεν ορίζεται, κάτι που, όμως, υλοποιείται από το Django.

Όπως γίνεται αντιληπτό, από την αναλυτική παράθεση των αποτελεσμάτων του Nessus, η διαδικτυακή εφαρμογή δεν παρουσιάζει ουσιαστικές ευπάθειες ασφάλειας. Η συντριπτική πλειοψηφία των αποτελεσμάτων είναι ενημερωτική και σχετίζεται με την ανάκτηση πληροφοριών για τον τρόπο λειτουργίας του server και της διαδικτυακής εφαρμογής.

Ένα μικρότερο ποσοστό των αποτελεσμάτων αφορά στο γεγονός ότι οι τρέχουσες ρυθμίσεις αποκαλύπτουν πληροφορίες που μπορεί να τις αξιοποιήσει κάποιος κακόβουλος χρήστης (π.χ. η γνώση της έκδοσης του apache web server μπορεί να οδηγήσει στην αναζήτηση ευπαθειών που παρουσιάζει ο συγκεκριμένος web server), ενώ ένα ακόμη μικρότερο ποσοστό αφορά την υλοποίηση συγκεκριμένων λειτουργιών της διαδικτυακής εφαρμογής που δεν μπορεί να γίνει με διαφορετικό τρόπο (όπως η παράθεση των URLs στο κεντρικό μενού της διαδικτυακής εφαρμογής myphoto).

Αξίζει να σημειωθεί επίσης, ότι σχεδόν το σύνολο των αποτελεσμάτων αφορούν στις ρυθμίσεις του web server και όχι στη διαδικτυακή εφαρμογή αυτή καθαυτή.

6.7 Βελτίωση της Ασφάλειας της Διαδικτυακής Εφαρμογής

Η παρουσίαση των αποτελεσμάτων από τον έλεγχο του Nessus δίνει τη δυνατότητα βελτίωσης της ασφάλειας στα ζητήματα που προκύπτουν, από έναν προγραμματιστή.

Βάσει των αποτελεσμάτων του ελέγχου που πραγματοποιήθηκε στις προηγούμενες παραγράφους, η διαδικτυακή εφαρμογή myphoto και ο apache web server δεν παρουσιάζουν ουσιαστικές ευπάθειες ασφάλειας.

Εξετάζοντας τα αποτελέσματα που παρουσιάζονται στην παράγραφο 6.6, τα πιο σημαντικά ζητήματα ασφάλειας σχετίζονται με την αποκάλυψη πληροφοριών της έκδοσης του web server και του λειτουργικού συστήματος καθώς και με το password auto – completion στις φόρμες εισόδου και δημιουργίας χρήστη στο myphoto. Σε αυτή την παράγραφο θα περιγραφεί η διαδικασία «βελτίωσης» της ασφάλειας του myphoto μέσω της «διόρθωσης» αυτών των δύο ζητημάτων.

6.7.1 Απόκρυψη Πληροφοριών Λειτουργικού Συστήματος και Web Server

Προκειμένου να μην εμφανίζεται η έκδοση του apache αλλά και του λειτουργικού συστήματος το οποίο τον φιλοξενεί στα HTTP responses που επστρέφονται από τον server, μετά την πρόσβαση στο αρχείο **/etc/apache2/conf-available/security.conf** προστίθενται οι γραμμές **ServerTokens Prod** και **ServerSignature Off**, περικλείοντας ταυτόχρονα σε σχόλιο τις υπόλοιπες γραμμές που σχετίζονται με τις μεταβλητές ServerTokens και ServerSignature.

```
  GNU nano 2.2.6   File: /etc/apache2/conf-available/security.conf

# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#ServerTokens Minimal
ServerTokens Prod
#ServerTokens Full

#
# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
ServerSignature Off
#ServerSignature On
```

Εικόνα 224: Ρυθμίσεις του αρχείου security.conf

6.7.2 Απενεργοποίηση Auto – completion

Προκειμένου να μην συμπληρώνονται τα διαπιστευτήρια, βάσει αυτών που έχει αποθηκεύσει ένας browser στη μνήμη του,προσθέτουμε την κατάλληλη γραμμή κώδικα στα σημεία που υπάρχουν HTML φόρμες.

Συγκεκριμένα, στα αρχεία **create_account.html** και **login.html** (τα template αρχεία που επιστρέφονται στο χρήστη κατά τη διάρκεια δημιουργίας λογαριασμού ή εισόδου εγγεγραμένου χρήστη στο myphoto), στον φάκελο **templates** και στον φάκελο **templates/registration** αντίστοιχα, στο σημείο που υπάρχει η HTML ετικέτα **<form>** προστίθεται τη φράση **autocomplete='off'**, όπως φαίνεται στις εικόνες 225 και 226.

```
6   <form method='POST' action='' enctype='multipart/form-data' autocomplete='off'>{% csrf_token %}
7   {{ form.as_p }}
8   <button type='submit' class='btn btn-default'><i class="fa fa-user fa-fw"></i>Sign Up</button>
9   <br>
10  <p><i class="fa fa-sign-in fa-fw"></i>Already have an account? Click <a href="{% url 'auth:login' %}">here</a> to log in</p>
11  </div>
```

Εικόνα 225: Απενεργοποίηση autocompletion στο create_account.html

```
<form method='POST' action='' enctype='multipart/form-data' autocomplete='off'>{% csrf_token %}
{{ form.as_p }}
<button  type='submit' class='btn btn-default'><i class="fa fa-sign-in fa-fw"></i>Log In</i></button><br>
<p><i class="fa fa-user fa-fw"></i>Don't have an account? Click <a href="{% url 'auth:create_account' %}">here</a> to sign up</p>
<p><i class="fa fa-lock fa-fw"></i>Forgot your password? Click <a href="{% url 'auth:password_reset' %}">here</a> to change it</p>
</form>
</div>
```

Εικόνα 226: Απενεργοποίηση autocompletion στο login.html

Κεφάλαιο 7: Συμπεράσματα

Οι ευπάθειες ασφάλειας στο επίπεδο της διαδικτυακής εφαρμογής, αποτελούν, αν όχι τη βασικότερη, μία από τις βασικότερες κατηγορίες ευπαθειών ασφάλειας, μιας και η εκμετάλλευσή τους από κακόβουλους χρήστες για πραγματοποίηση κάποιας επίθεσης μπορεί να έχει συνέπειες υψηλότατου βαθμού.

Σε μια εποχή που οι υπηρεσίες διαδικτύου και οι διαδικτυακές εφαρμογές χρησιμοποιούνται καθημερινά για την κάλυψη βασικών αναγκών όπως χρηματικές συναλλαγές, αποθήκευση και ανταλλαγή προσωπικών δεδομένων κ.α., η ύπαρξη ευπαθειών ασφάλειας σε μια διαδικτυακή εφαρμογή μπορεί να οδηγήσει σε απώλεια ή υποκλοπή ευαίσθητων πληροφοριών, χρημάτων κ.ο.κ.

Είναι, λοιπόν, απολύτως εμφανές, ότι η μέριμνα για την ασφάλεια μιας διαδικτυακής εφαρμογής δεν μπορεί να αφεθεί αποκλειστικά για το στάδιο της συντήρησης. Λόγω του πλήθους των χρηστών του διαδικτύου σήμερα, αλλά και της φύσης των δεδομένων που αποθηκεύονται και ανταλάσσονται μέσω αυτού,ένας προγραμματιστής δεν μπορεί να «θυσιάσει» την ασφάλεια για χάρη της ταχύτητας ανάπτυξης μιας εφαρμογής, ή να δοκιμάσει να διαθέσει στο κοινό μια διαδικτυακή εφαρμογή που δεν ικανοποιεί βασικές αρχές ασφάλειας.

Ο ασφαλής σχεδιασμός μιας διαδικτυακής εφαρμογής είναι ίσης σημασίας με τον όσο δυνατό ταχύτερο και αποτελεσματικό σχεδιασμό της, βάσει των λειτουργικών απαιτήσεων που τίθενται κάθε φορά.

Ταυτόχρονα, οι σύχρονες απαιτήσεις σχετικά με την ταχύτητα ανάπτυξης ενός προγράμματος αλλά και σχετικά με την πολυπλοκότητα των λειτουργιών που ένα σύγχρονο πρόγραμμα καλείται να παρέχει, επιβάλλει την αξιοποίηση web frameworks, τα οποία δίνουν τη δυνατότητα στους προγραμματιστές να ενσωματώσουν και να επαναχρησιμοποιήσουν κώδικα που αναλαμβάνει την υλοποίηση των πιο συνήθων λειτουργιών μιας διαδικτυακής εφαρμογής (είσοδος και εγγραφή χρήστη κ.ο.κ).

Από τα παραπάνω προκύπτει ότι η χρήση web frameworks τα οποία περιλαμβάνουν εργαλεία και κώδικα που σχετίζεται με την διασφάλιση της ασφάλειας μιας διαδικτυακής εφαρμογής, είναι ζωτικής σημασίας για το σχεδιασμό μιας διαδικτυακής εφαρμογής.

Το Django είναι ένα web framework (τύπου MVC) σε γλώσσα Python, που, σύμφωνα με τους δημιουργούς του απευθύνεται σε “τελειομανείς με ασφυκτικούς χρονικούς περιορισμούς” (perfectionists with deadlines).

Παρέχει πλήθος εργαλειών και τμημάτων κώδικα που σχετίζονται με την ασφάλεια μιας διαδικτυακής εφαρμογής, δίνοντας στους προγραμματιστές τη δυνατότητα να δημιουργήσουν εύκολα και γρήγορα ασφαλείς διαδικτυακές εφαρμογές.

Ο κατανοητός τρόπος δομής του κώδικα του Django (με τον σαφή διαχωρισμό Models – Views – Templates) επιτρέπει την ταχεία δημιουργία διαδικτυακών εφαρμογών και την αξιοποίηση των έτοιμων εργαλείων που παρέχει, καθώς και την παραμετροποίηση αυτών ή την δημιουργία εντελώς νέων με απλό τρόπο.

Μόνο με την έναρξη ενός project με τη χρήση Django και χωρίς ιδιαίτερες απαιτήσεις κώδικα από την πλευρά του προγραμματιστή, το Django παρέχει:

1) ολοκληρωμένο admin interface.
2) Middlewares που διαχειρίζονται τις συνόδους και την προστασία από επιθέσεις CSRF, XSS και Clickjacking.
3) Τμήματα κώδικα που χρησιμοποιούνται για την κρυπτογράφηση κωδικών αλλά και για τη διαχείρισή τους.
4) Το Django ORM που επιτρέπει εύκολη σύνδεση και αναπαράσταση εγγραφών της βάσης δεδομένων με χρήση κλάσεων Python αλλά και την εκτέλεση SQL queries με ασφαλή τρόπο.
5) Την κλάση User που χρησιμοποιείται για την αναπαράσταση των χρηστών μιας διαδικτυακής εφαρμογή.
6) Συναρτήσεις που υλοποιούν την είσοδο και έξοδο, την αποστολή e – mail, την αλλαγή κωδικού χρήστη κ.ο.κ.

Η μελέτη περίπτωση του ασφαλή σχεδιασμού της διαδικτυακής εφαρμογής myphoto υλοποιήθηκε με την αξιοποίηση αρκετών από τα εργαλεία ασφάλειας που παρέχει το Django αλλά και με την δημιουργία νέου κώδικα (συναρτήσεις check_ip(), is_locked_out(), ενεργοποίηση λογαριασμού με χρήση e – mail κ.ο.κ), σύμφωνα με τις βασικές αρχές ασφάλειας για διαθεσιμότητα, ακεραιότητα και εμπιστευτικότητα των δεδομένων και των υπηρεσιών που παρέχει το myphoto.

Με αξιοποίηση του Django υλοποιήθηκε η αυθεντικοποίηση και εξουσιοδότηση χρηστών, η προφύλαξη από αλλοίωση παραμέτρων, η ασφαλής διαχείριση συνόδων, ο έλεγχος εισαγωγής δεδομένων και περιεχομένου από το χρήστη, η καταγραφή των ενεργειών των χρηστών.

Σημαντικά ζητήματα ασφάλειας που αφορούσαν στο web server ή επιπλέον μεθοδοι προστασίας, διευθετήθηκαν με ρυθμίσεις στον web server και χρήση κώδικα Javascript.

Ο έλεγχος και η ανάλυση ευπαθειών σε εργαστηριακό περιβάλλον παραγωγής από το Nessus Vulnerability Scanner, δεν επέστρεψε ευπάθειες αλλά κυρίως ενημερωτικά αποτελέσματα σε σχέση με τη διαδικτυακή εφαρμογή και τον apache web server που την εξυπηρετεί.

Οι όποιες βελτιώσεις ύστερα από τον έλεγχο αφορούσαν αποκλειστικά στην απόκρυψη πληροφοριών σε σχέση με τον web server, το λειτουργικό σύστημα και την πιθανή απομνημόνευση διαπιστευτηρίων χρηστών του myphoto από έναν browser, ζητήματα που έχουν έμμεση σχέση με την ύπαρξη ευπαθειών.

Αποδείχτηκε ότι με ελάχιστη παραμετροποίηση και ενσωμάτωση των εργαλείων ασφάλειας που προσφέρει το Django, οι σημαντικότεροι κίνδυνοι ασφάλειας σύμφωνα με τον OWASP αλλά και οι ευπάθειες όλων των γνωστών ειδών αντιμετωπίζονται αποτελεσματικά.

Συνοψίζοντας, η τελευταία έκδοση του Django web framework μπορεί να απευθυνθεί επιτυχημένα στο σύγχρονο πρόβλημα του γρήγορου και ασφαλούς σχεδιασμού διαδικτυακών εφαρμογών, αναλαμβάνοντας κατά ένα μεγάλο ποσοστό την προστασία μιας διαδικτυακής εφαρμογής και την αντιμετώπιση των ενδεχόμενων ευπαθειών ασφάλειας.

Βιβλιογραφία

[1] Wikipedia, Web Application [Ηλεκτρονικό],Available: https://en.wikipedia.org/wiki/Web_application [Πρόσβαση: Ιούλιος 2016]

[2] R. Hackett, Hackers steal $5 million from major bitcoin exchange [Ηλεκτρονικό], Available: http://fortune.com/2015/01/05/bitstamp- bitcoin- freeze- hack/ [Πρόσβαση: Ιούλιος 2016]

[3] M.K. McGee, Another massive health data hack: Premera blue cross is the latest victim [Ηλεκτρονικό], Available: http://www.databreachtoday.com/ another- massive- health- data-hack- a- 8026 [Πρόσβαση: Ιούλιος 2016]

[4] M. Williams, The 5 biggest data breaches of 2014 (so far) [Ηλεκτρονικό], Available: http://www. pcworld.com/article/2453400/the- biggest- data- breaches- of- 2014- so- far.htm [Πρόσβαση: Ιούλιος 2016]

[5] C. McGarry, Change your passwords, ebay users: the site was hacked [Ηλεκτρονικό],Available: http:// www.pcworld.com/article/2157604/ebay- users- change- your-passwords- the- auction- site- was- breached.htm [Πρόσβαση: Ιούλιος 2016]

[6] J. Kirk, Bitcoin exchange loses $250,0 0 0 0 after unencrypted keys stolen (2012) [Ηλεκτρονικό], Available: http://www.computerworld.com/article/2492037/cybercrime-hacking/ bitcoin- exchange- loses- - 250- 0 0 0 0- after- unencrypted- keys- stolen.html [Πρόσβαση: Ιούλιος 2016]

[7] N. Bilton, B. Stelter, Sony says playstation hacker got personal data [Ηλεκτρονικό], Available: http://www.nytimes.com/2011/04/27/technology/27playstation.html [Πρόσβαση: Ιούλιος 2016]

[8] B. Acohido, Hackers breach heartland payment credit card system [Ηλεκτρονικό], Available: http:// usatoday30.usatoday.com/money/perfi/credit/2009- 01- 20- heartland- credit- card-security- breach _ N.htm [Πρόσβαση: Ιούλιος 2016]

[9] G. Deepa, P. Santhi Thilagam, Securing web applications from injection and logic vulnerabilities: Approaches and challenges, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, India, Φεβρουάριος 2016

[10] Π. Κοτζανικολάου, Θ. Ντούσκας, Ασφάλεια Πληροφοριών: Διαχείριση Ασφάλειας Πληροφοριακών Συστημάτων, ΜΠΣ «Πληροφορική» Τμήμα Πληροφορικής Πανεπιστήμιου Πειραιώς, 2016

[11] Wikipedia, Web Application Security [Ηλεκτρονικό], Available: https://en.wikipedia.org/wiki/Web_application_security [Πρόσβαση: Ιούλιος 2016]

[12] Trustwave, 2016 Trustwave Global Security Report, Trustwave, 2016

[13] Django Documentation [Ηλεκτρονικό],Available: https://docs.djangoproject.com/ [Πρόσβαση: Ιούλιος 2016]

[14] Python 2.7.12 Documentation [Ηλεκτρονικό], Available: https://docs.python.org/2/ [Πρόσβαση: Ιούλιος 2016]

[15] The Open Web Application Security Project, OWASP Top Ten 2013, The OWASP Foundation, 2013

[16] Π. Κοτζανικολάου, Θ. Ντούσκας, Ασφάλεια Πληροφοριών: Ευπάθειες Διαδικτυακών Εφαρμογών ΜΠΣ «Πληροφορική» Τμήμα Πληροφορικής Πανεπιστήμιου Πειραιώς, 2016

[17] Brian Sullivan, Vincent Liu, Web Application Security: A Beginner's Guide, Mc Graw – Hill Companies, 2012

[18] Microsoft, THE STRIDE Threat Model [Ηλεκτρονικό], Available: https://msdn.microsoft.com/en-us/library/ee823878%28v=cs.20%29.aspx [Πρόσβαση: Ιούλιος 2016]

[19] Microsoft, Design Guidelines For Secure Web Applications [Ηλεκτρονικό], Available: https://msdn.microsoft.com/el-gr/library/aa302420.aspx [Πρόσβαση: Ιούλιος 2016]

[20] The Open Web Application Security Project, SQL Injection Prevention Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[21] The Open Web Application Security Project, Authentication Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Authentication_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[22] The Open Web Application Security Project, Session Management Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Session_Management_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[23] The Open Web Application Security Project, XSS Cross – Site Scripting Prevention Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[24] The Open Web Application Security Project, Insecure Direct Object Reference Prevention Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[25] The Open Web Application Security Project, Access Control Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Access_Control_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[26] The Open Web Application Security Project, Cross – Site Request Forgery (CSRSF) Prevention Cheat – Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[27] The Open Web Application Security Project, Unvalidated Redirects And Forwards Cheat Sheet [Ηλεκτρονικό], Available: https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet [Πρόσβαση: Ιούλιος 2016]

[28] F. Neal, Art of Java Web Development: Struts, Tapestry, Commons, Velocity, Junit, Axis, Cocoon, Internetbeans, Webwork.
Manning Publications Co., Greenwich, CT, USA,2003

[29] María del Pilar Salas-Zárate, Giner Alor-Hernández, Alejandro Rodríguez-González, Developing Lift-based Web Applications Using Best Practices, The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science,Elsevier Ltd, 2012.

[30] Dirk Riehle.Framework Design: A Role Modeling Approach.Ph.D. Thesis, No. 13509., ETH Zurich,Switzerland, 2000.

[31] Microsoft, Model – View – Controller [Ηλεκτρονικό], Available: https://msdn.microsoft.com/en-us/library/ff649643.aspx [Πρόσβαση: Νοέμβριος 2016]

[32] A Freeman, S Sanderson, Pro ASP.NET MVC 3 Framework, Apress, 2011

[33] W. J. Gilmore, Easy PHP Websites, Columbus, Ohio: W.J. Gilmore, LLC, 2009

[34] J. Galloway, P. Haack, B. Wilson și K. S. Allen, Professional ASP.NET MVC 3, John Wiley & Sons, Inc., 2011

[35] W. J. Gilmore, Easy PHP Websites, Columbus, Ohio: W.J. Gilmore, LLC, 2009

[36] J. Galloway, P. Haack, B. Wilson și K. S. Allen, Professional ASP.NET MVC 3, John Wiley & Sons, Inc., 2011

[37] E. J. O'Neil, Object/relational mapping 2008: hibernate and the entity data model (edm), Proceedings of the 2008 ACM SIGMOD
International Conference On Management Of Data,2008

[38] J. Lerman, Programming Entity Framework, O'Reilly, 2010

[39] Dragos-Paul Pop, Adam Altar, Designing an MVC Model for Rapid Web Application Development, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, Elsevier Ltd., 2014

[40] Object Relational – Mapping [Ηλεκτρονικό], Available: http://danhartshorn.com/2011/12/object-relational-mapping-wikipedia-the-free-encyclopedia [Πρόσβαση: Νοέμβριος 2016]

[41] Django Documentation FAQ [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/faq/general/#django-appears-to-be-a-mvc-framework-

but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names [Πρόσβαση: Νοέμβριος 2016]

[42] Balakumar Parameshwaran, Django Python Web Framework [Ηλεκτρονικό], Available: http://www.slideshare.net/balakumarp/django-framework [Πρόσβαση: Νοέμβριος 2016]

[43] Django Documentation, Models [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/db/models/ [Πρόσβαση: Νοέμβριος 2016]

[44] Django Documentation, Making Queries [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/db/queries/ [Πρόσβαση: Νοέμβριος 2016]

[45] Django Documentation, URL Dispatcher [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/urls/ [Πρόσβαση: Νοέμβριος 2016]

[46] Django Documentation, Writing Views [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/views/ [Πρόσβαση: Νοέμβριος 2016]

[47] Django Documentation, Class – Based Views [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/class-based-views/ [Πρόσβαση: Νοέμβριος 2016]

[48] Django Documentation, File Uploads [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/file-uploads/ [Πρόσβαση: Νοέμβριος 2016]

[49] Django Documentation, View Decorators [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/decorators/ [Πρόσβαση: Νοέμβριος 2016]

[50] Django Documentation, Django Shortcut Functions [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/shortcuts/ [Πρόσβαση: Δεκέμβριος 2016]

[51] Django Documentation, Middleware [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/middleware/ [Πρόσβαση: Δεκέμβριος 2016]

[52] Django Documentation, Templates [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/templates/ [Πρόσβαση: Δεκέμβριος 2016]

[53] Django Documentation, The Django Template Language [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/ref/templates/language/ [Πρόσβαση: Δεκέμβριος 2016]

[54] Django Documentation, Custom Template Tags & Filters [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/howto/custom-template-tags/ [Πρόσβαση: Δεκέμβριος 2016]

[55] Django Documentation, Working with Forms [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/forms/ [Πρόσβαση: Δεκέμβριος 2016]

[56] Django Documentation, The Django Admin Site [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/ref/contrib/admin/ [Πρόσβαση: Δεκέμβριος 2016]

[57] Django Documentation, Using the Authentication System [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/auth/default/ [Πρόσβαση: Δεκέμβριος 2016]

[58] Django Documentation, Password Management In Django [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/auth/passwords/ [Πρόσβαση: Δεκέμβριος 2016]

[59] Burt Kaliski, PKCS #5: Password-Based Cryptography Specification Version 2.0 [Ηλεκτρονικό], Available: https://tools.ietf.org/html/rfc2898#section-5.2 [Πρόσβαση: Δεκέμβριος 2016]

[60] Meltem Sönmez Turan, Elaine Barker, William Burr, Lily Chen, Recommendation for Password-Based Key Derivation, Computer Security Division Information Technology Laboratory, 2010

[61] Password Hashing Competion [Ηλεκτρονικό], Available: https://password-hashing.net/ [Πρόσβαση: Δεκέμβριος 2016]

[62] Django Documentation, Cross Site Request Forgery Protection [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/ref/csrf/#how-csrf-works [Πρόσβαση: Δεκέμβριος 2016]

[63] Django Documentation, Django Security [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/security/ [Πρόσβαση: Ιανουάριος 2017]

[64] Django Documentation, Clickjacking Protection [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/ref/clickjacking/ [Πρόσβαση: Ιανουάριος 2017]

[65] Django Documentation,Settings [Ηλεκτρονικό],Available: https://docs.djangoproject.com/en/1.10/ref/settings/ [Πρόσβαση: Ιανουάριος 2017]

[66] Django Documentation, How to use sessions [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/http/sessions/ [Πρόσβαση: Ιανουάριος 2017]

[67] Django Documentation, Cryptographic signing [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/topics/signing/ [Πρόσβαση: Ιανουάριος 2017]

[68] Font – Awesome [Ηλεκτρονικό],Available:http://fontawesome.io/ [Πρόσβαση: Δεκέμβριος 2016]

[69] Oracle,Virtual Machines [Ηλεκτρονικό],Available: https://www.virtualbox.org/wiki/Virtualization [Πρόσβαση: Φεβρουάριος 2017]

[70] P.J. Eby, Python Web Server Gateway Interface v1.0.1 [Ηλεκτρονικό], Available: https://www.python.org/dev/peps/pep-3333/#abstract [Πρόσβαση: Φεβρουάριος 2017]

[71] Graham Dumpleton, mod_wsgi Documentation [Ηλεκτρονικό], Available: https://modwsgi.readthedocs.io/en/develop/ [Πρόσβαση: Φεβρουάριος 2017]

[72] Django Documentation, How to use Django with Apache and mod_wsgi [Ηλεκτρονικό], Available: https://docs.djangoproject.com/en/1.10/howto/deployment/wsgi/modwsgi/

[Πρόσβαση: Φεβρουάριος 2017]

[73]Jay Beale, Russ Rogers,Nessus Network Auditing 2nd Edition, Syngress, 2008

[74]Tenable Network Security, Nessus 6.10 User Guide, Tenable Network Security Inc., 2017

[75] Common Weaknes Enumeration [Ηλεκτρονικό],Available: https://cwe.mitre.org/ [Πρόσβαση: Μάρτιος 2017]

[76] Peter Mell, Karen Scarfone Sasha Romanosky, A Complete Guide to the Common Vulnerability Scoring System Version 2.0, 2007