Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Τίτλος**
Ονοματεπώνυμο	**Σπύρος Σταυρόπουλος**
Πατρώνυμο	**Θανάσης**
Αριθμός Μητρώου	**ΜΠΠΛ 12053**
Επιβλέπων	**Δουληγέρης Χρήστος, Καθηγητής**
Συνεπιβλέπων	**Βασίλης Μενεκλής, Διδάκτορας**

Ημερομηνία Παράδοσης **Μάρτιος 2016**

Τριμελής Εξεταστική Επιτροπή

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
Δουληγέρης Χρήστος Καθηγητής	Βέργαδος Δημήτριος Αναπληρωτής Καθηγητής	Κωνσταντόπουλος Χ. Επίκουρος Καθηγητής

Θα ήθελα να ευχαριστήσω πρώτα απ' όλα «τους ανθρώπους» που πάντα ήταν εκεί και τους οφείλω ότι είμαι μέχρι σήμερα. Το διδάκτορα Βασίλη Μενεκλή για την ανεκτίμητη βοήθεια και γνώση που προσέφερε και στην οποία οφείλω σε μεγάλο βαθμό το νέο επαγγελματικό μου ξεκίνημα.

Περιεχόμενα

Περίληψη

Τα μέσα κοινωνική δικτύωσης (social media) είναι προϊόν που δημιουργήθηκε και αναπτύχθηκε μέσα από την συνεχώς αυξανόμενη χρήση του διαδικτύου τα οποία και κατακτούν με εκπληκτικά αυξανόμενους ρυθμούς όλο και περισσότερους χρήστες. Οι δυνατότητες τους και υπηρεσίες που προσφέρουν στους χρήστες είναι πολλές, καθώς πρόκειται για την πιο σύγχρονη αντίληψη και εξέλιξη στο χώρο του διαδικτύου που επιτρέπει, υποστηρίζει και στηρίζεται στην ενεργό συμμετοχή και την αλληλεπίδραση μεταξύ χρηστών.
Στην παρούσα διπλωματική εργασία παρουσιάζεται ο τρόπος υλοποίησης ενός μέσου κοινωνικής δικτύωσης. Προσφέρει την δυνατότατα στο χρήστη να μοιραστεί τα ενδιαφέροντά του δημόσια εντός της εφαρμογής με την μορφή βαθμολογημένης δημοσίευσης και προαιρετικού σχολιασμού, να αναπτύξει σχέσεις βάση του σχεσιακού μοντέλου μονόδρομης σχέσης (τύπου Twitter) με άλλους χρήστες, να λαμβάνει ενημερώσεις πραγματικού χρόνου.

Abstract

Social media (social media) is a product created and developed through the continuously growing usage of Internet which conquer with amazingly increasing rate more and more users.
Their features and services that offer to users are many, as they are the most modern conception and development of the Internet that enables, supports and builds the active participation and interaction between users.
This thesis shows how implementing a social media. It offers the possibility to the user to share his interests to the public into the application in the form of a graduated publication and optional commentary, develop relationships based on relational-way model (like twitter) with other users, and receive real-time updates.

Κεφάλαιο 1ο

Εισαγωγή

Ζούμε χωρίς αμφιβολία σε μια εποχή που χαρακτηρίζεται από ραγδαία τεχνολογική και επικοινωνιακή ανάπτυξη και το διαδίκτυο έχει γίνει αναπόσπαστο κομμάτι της ζωής μας αφού κατέχει πρωταγωνιστικό ρόλο στην πορεία αυτή. Όταν λέμε διαδίκτυο εννοούμε ένα σύνολο υπολογιστών και δικτύων που συνδέονται μεταξύ τους σε ένα παγκόσμιο δίκτυο και κάτω από συγκεκριμένους κανόνες έτσι ώστε να μπορούν να επικοινωνούν και να μοιράζονται πληροφορίες. Η ολοένα και μεγαλύτερη χρήση του διαδικτύου και η τάση για επικοινωνία μέσω αυτού είχαν ως αποτέλεσμα να ανθίσουν τα μέσα ηλεκτρονικής κοινωνικής δικτύωσης (social media).

Τι είναι τα social media; Αν προσπαθούσαμε να δώσουμε ένα ορισμό θα λέγαμε ότι είναι διαδικτυακοί τόποι που παρέχουν ως υπηρεσίες την επικοινωνία, την ενημέρωση και τη δημοσίευση περιεχομένου με σκοπό την αλληλεπίδραση μεταξύ των χρηστών που συμμετέχουν σε αυτά.

Στο διαδικτυακό κόσμο υπάρχουν αρκετά μέσα κοινωνικής δικτύωσης και ορισμένα από αυτά συνέβαλαν στη σύλληψη της ιδέας της παρούσας εργασίας. Ως χρήστες λοιπόν μέσων κοινωνικής δικτύωσης, γοητευμένοι από τον τρόπο λειτουργίας, από τις δυνατότητες και από το μεγάλο όγκο χρηστών μαζί με την σκέψη ότι μπορούμε να δημιουργήσουμε κάτι δικό μας ξεκινήσαμε αυτή την εργασία. Ήταν δεδομένο μέσα στην σκέψη μας ότι θα χρησιμοποιούσαμε κάποια χαρακτηρίστηκα από τις ήδη υπάρχουσες εφαρμογές. Το twitter και ο τρόπος με τον οποίο αναπτύσσονται οι σχέσεις μεταξύ χρηστών. Το pinterest και οι εικόνες, το χαρακτηριστικό των δημοσιεύσεων του. Το IMDB και το τρόπος βαθμολόγησης. Αυτά μαζί με τις ιδέες που είχαμε ως ομάδα όπως η προβολή διαδικτυακών τόπων μέσω URL είχαν ως αποτέλεσμα τη δικιά μας εφαρμογή και που παρουσιάζεται παρακάτω.

Το θέμα με το οποίο θα ασχοληθούμε στην παρούσα διπλωματική εργασία είναι η παρουσίαση και η υλοποίηση μίας εφαρμογής κοινωνικής δικτύωσης. Πρόκειται για μία εφαρμογή την οποία μπορεί να χρησιμοποιήσει οποιοσδήποτε χρήστης κατόπιν. Παρέχει υπηρεσίες όπως η δυνατότητα δημοσίευσης διαδικτυακών τόπων παντός τύπου, η βαθμολόγησή τους και ο προαιρετικός σχολιασμός, η κατηγοριοποίηση των δημοσιεύσεων με βάση τη βαθμολογία τους και την ανάπτυξη μονόδρομων σχέσεων μεταξύ χρηστών (follow relations).

Λόγω του ότι η ιδέα υλοποιήθηκε από τρεις φοιτητές[i] στην εργασία αυτή θα παρουσιαστεί το κομμάτι που έχει να κάνει με τη λειτουργία εγγραφής – εισαγωγής του χρήστη στην εφαρμογή, τη διαχείριση του προφίλ του, τη λειτουργία εύρεσης χρήστη, τις σχέσεις (σε ακολουθώ – δε σε ακολουθώ) μεταξύ χρηστών, την παρουσίαση χρηστών που με ακολουθούν - ακολουθώ και τέλος τη χρήση υπηρεσιών διαδικτύου για την αποθήκευση δεδομένων εικόνας σε αποθηκευτικό νέφος (cloud storage).

Στην συνέχεια της εργασίας θα παρουσιαστεί εύρεση σχετικής βιβλιογραφίας, υλοποιημένων διαδικτυακών εφαρμογών σχετικές με την παρούσα εργασία και θα γίνει μια σύντομη σύγκριση μεταξύ τους παρουσιάζοντας ομοιότητες και διαφορές.

Θα γίνει αναφορά των τεχνολογιών που επιλέχθηκαν να χρησιμοποιηθούν για την κατασκευή της εφαρμογής.

Παρουσίαση της αρχιτεκτονικής βάση της οποίας υλοποιήθηκε η εφαρμογή μας. Αρχιτεκτονική όπως τα προγραμματιστικά πρότυπα.

Τέλος τα συμπεράσματα τα οποία προέκυψαν κατά την εκπόνηση της εργασίας.

Κεφάλαιο 2ο

Σχετική Βιβλιογραφία

Κατά την διαδικασία της έρευνας που πραγματοποιήθηκε για την ανάλυση και την υλοποίησης της εφαρμογής βρέθηκαν βιβλιογραφικές πηγές οι οποίες καθόρισαν τις τελικές επιλογές μας σχεδιασμού και ανάπτυξης της εφαρμογής. Ο σκοπός αυτού του κεφαλαίου είναι η παρουσίαση αυτών και η αναφορά στον τρόπο με τον οποίο καθεμία επηρέασε την παρούσα εργασία.

Codebright (Rees D.)

Αυτό το σύγγραμμα έχει να κάνει με το πλαίσιο εργασία (php framework) Laravel που τελικά επιλέξαμε για να υλοποιήσουμε την εφαρμογή μας ή πιο απλά για να γράψουμε τον κώδικά μας. Δίνει την δυνατότητα στο χρήστη ξεκινώντας από το μηδέν να μπορέσει να αντιληφθεί πλήρως τι είναι το Laravel, να γνωρίσει τα εργαλεία που διαθέτει και να μπορέσει να τα εφαρμόσει για να υλοποιήσει μια δίκια του εφαρμογή. Όλα αυτά παραπάνω παρουσιάζονται στο χρήστη με πλήρως κατανοητά παραδείγματα.

Little Redis Book (Seguin K. 2014)

Αυτή η βιβλιογραφία συνέβαλε στο να κάνουμε τα πρώτα μας βήματα πάνω στην τεχνολογία των μη σχεσιακών βάσεων δεδομένων. Μας έδωσε πληροφορίες για το πώς θα εγκαταστήσουμε την βάση δεδομένων στα λειτουργικά συστήματα που χρησιμοποιούσαμε, μας έδωσε πληροφορίες για τις δομές δεδομένων που χρησιμοποιεί η βάση και εντολές διαχείρισης αυτών όπως και εντολές διαχείρισης και παραμετροποίησης της ίδιας της βάσης.

ProGit (Chacon S.)

Το παρών βιβλίο αναφέρεται στα συστήματα ελέγχου ροής αρχείων (version control system). Γίνεται αναφορά στο λόγο ύπαρξης και χρησιμότητας αυτών των συστημάτων καθώς επίσης και τον τρόπο εγκατάστασης, χρήσης και αναφορά χαρακτηριστικών του. Περιγράφονται αναλυτικά οι διαδικασίες εφαρμογής συνοδευόμενες από αναλυτικά παραδείγματα τα οποία δίνουν στο χρήστη να κατανοήσει πλήρως τις διαδικασίες. Στη παρούσα εργασία ήταν ένα χρήσιμο εργαλείο σε επίπεδο software development λόγο την συμμέτοχη τριών ατόμων στο ίδιο έργο.

Handbook of Cloud Computing (Borko Furht, Armando Escalante)

Αυτή είναι μία από τις πηγές που χρησιμοποιήθηκαν για να γίνει αντιληπτό ή καλύτερα να αποκτηθούνε γενικές γνώσεις για το τι είναι η τεχνολογία του cloud computing. Γίνεται αναφορά γενικά στο μοντέλο του cloud computing, αναφέρονται οι υπηρεσίες που προσφέρονται στο καταναλωτικό κοινό, αναφέρονται οι τύποι αυτού και τα οφέλη που έχουμε κατά την χρήση της τεχνολογίας. Επίσης γίνεται αναφορά των χαρακτηριστικών, των κανόνων ασφαλείας που πρέπει να λαμβάνει ο χρήστης κατά την υλοποίηση αυτής της τεχνολογίας και τέλος μια συνοπτική αναφορά για τις εταιρίες και τις πλατφόρμες που προσφέρουν στο καταναλωτικό κοινό καθώς και γενικές πληροφορίες για τον τρόπο χρέωσης.

Evolution of Cloud Storage as Cloud Computing Infrastructure Service (IOSR Journal of Computer Engineering)

Μία ακόμα πολύ βασική αναφορά για την τεχνολογία cloud computing είναι και αυτό το άρθρο το οποίο αναφέρει πληροφορίες για το είναι το cloud computing, τους τύπους για τα μοντέλα υπηρεσιών. Το εν λόγο άρθρο εστιάζει στην υπηρεσία Storage as a service (StaaS) ή αλλιώς cloud storage. Είναι η υπηρεσία κατά την οποία δύνεται στο χρήστη η δυνατότητα ενοικίασης απομακρυσμένου αποθηκευτικού χώρου. Παρουσιάζεται το μοντέλο της υπηρεσίας cloud storage, τα οφέλη της χρήσης του και γίνεται μια αναφορά στην δυνατότητα χρήσης της υπηρεσίας μέσω διεπαφής προγραμματισμού εφαρμογών (API). Στην ουσία αυτή η υπηρεσία δίνει την δυνατότητα στο χρήστη να επικοινωνήσει και να κάνει χρήση αυτής μέσω της εφαρμογή του.

Προσέγγιση Twitter (Twitter clone) (Sanfilippo S. 2011)

Η συγκεκριμένη εφαρμογή έχει ως κύριο θέμα την προσεγγιστική υλοποίηση του μέσου κοινωνικής δικτύωσης Twitter σε γλώσσα προγραμματισμού PHP και βάση δεδομένων REDIS. Το Retwis όπως αναφέρεται στο επίσημο site της Redis (redis.io/topics/twitter-clone) είναι μια διασύνδεση χρήστη (user interface) εντός της οποίας δημιουργούνται μονόδρομες σχέσεις μεταξύ χρηστών (follow relations) και στο οποίο ο χρήστης έχει την δυνατότητα να δημιουργεί δημοσιεύσεις απλού κειμένου. Αυτές παρουσιάζονται στην σελίδα του και στην σελίδα χρόνο-ροής (timeline) του όπου εκεί εμφανίζονται μαζί με τις δημοσιεύσεις χρηστών που ακολουθεί.

Οι ομοιότητες του retwis είναι αρκετές αφού όπως προαναφέραμε στην εισαγωγή το twitter ήταν ένας από τους κύριους εμπνευστές μας. Κάθε χρήστης μπορεί να εκτελεί την ενέργεια σε ακολουθώ σε ένα άλλο. Αυτό δίνει την δυνατότητα σε αυτόν κάνει την ενέργεια να παρακολουθεί τις δημοσιεύσεις του χρήστη που ακολουθεί. Οι σχέσεις δεν είναι αμφίδρομες δηλαδή όταν ακολουθώ ένα χρήστη δεν το κάνει αυτόματα και αυτός. Ο τρόπος ή καλύτερα η φιλοσοφία με την οποία υλοποιήθηκε η πιστοποίηση χρήστη. Στην εφαρμογή μας όπως και στο retwis χρησιμοποιήθηκε ένας μοναδικός αριθμός βάση του οποίου ο χρήστης γίνεται μοναδικός κάθε φορά που εισέρχεται στην εφαρμογή. Ο αριθμός αυτός αποθηκεύεται στα στοιχεία του χρήστη στην βάση δεδομένων και στο φυλλομετρητή του και σχεδόν σε κάθε ενέργεια που κάνει πραγματοποιείται η διαδικασία σύγκρισης των δύο αριθμών. Οι δημοσιεύσεις οι οποίες έχουν την ίδια λογική καταχώρησης και παρουσίασης καθώς και στις δύο εφαρμογές χρησιμοποιούνται τύποι δεδομένων hashes για την αποθήκευσή ενώ λίστα για την χρονική τοποθέτηση του id των δημοσιεύσεων και την ανάλογη παρουσίασή τους.

Οι διαφορές είναι ότι κατά την διαδικασία εγγραφής στην εφαρμογή μας στέλνεται ένα μήνυμα επαλήθευσης email στο χρήστη που την πραγματοποιεί και που του δίνει δικαιώματα χρήσης ανάλογα με το αν έχει απαντήσει ή όχι. Το κύριο αντικείμενο της εφαρμογής μας είναι το RATE κάτι πολύ πιο σύνθετο από την απλή δημοσίευση κειμένου του retwis αφού δίνει την δυνατότητα βαθμολόγησης, παρουσίασης εικόνας και σχολιασμού. Όπως επίσης και την δυνατότητα παρουσίασης στατιστικών των RATE και η παρουσίαση τους ανά κατηγορίες (most rated κτλ). Ακόμα μία από τις διαφορές είναι ότι το retwis δεν διαθέτει ενημερώσεις (notification) τη δυνατότητα δηλαδή της εφαρμογής να ενημερώνει το χρήστη σε κατάσταση πραγματικού χρόνου για το τι συμβαίνει στο timeline του αν προστέθηκε κάποια δημοσίευση αν έγινε από κάποιον follow.

Κεφάλαιο 3ο

Τεχνολογίες

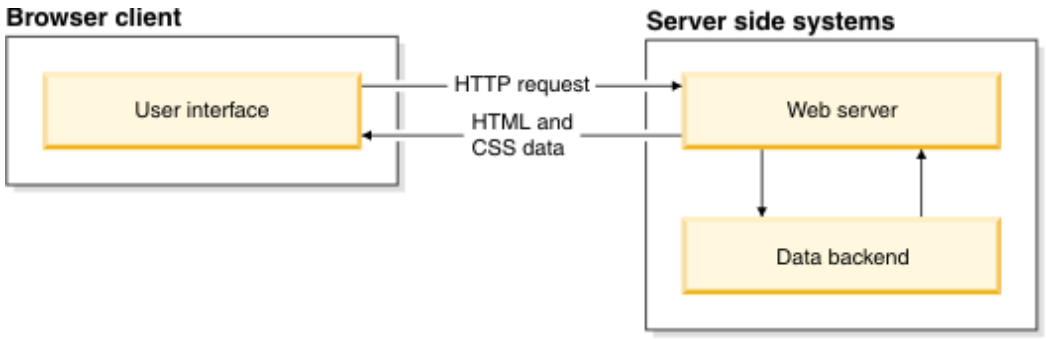
Στο παρόν κεφάλαιο γίνεται αναφορά στις τεχνολογίες που επιλέχθηκαν και χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Με τον όρο τεχνολογίες εννοούνται όλα εκείνα τα εργαλεία - υπηρεσίες client side ή server side τα οποία παρέχονται από τον παγκόσμιο ιστό ή είναι συμβατά με αυτόν και καθιστούν εφικτή την αποτελεσματική σχεδίαση & ανάπτυξη διαδικτυακών εφαρμογών με χαρακτηριστικά την διαδραστικότητα και τη χρηστικότητα.

AJAX

Τα αρχικά προέρχονται από τις λέξεις **A**synchronous **J**avaScript **And X**ML. Συνδυάζει υπάρχουσες τεχνολογίες ώστε να καταστήσει την επικοινωνία μεταξύ client (φυλλομετρητή) και server πιο άμεση και τις σελίδες που το χρησιμοποιούν πιο ζωντανές. Το κύριο χαρακτηριστικό μιας διαδικτυακής σελίδας που χρησιμοποιεί A.J.A.X. είναι η ενημέρωση της με νέο περιεχόμενο που προέρχεται από το server χωρίς αυτή να χρειάζεται να ξαναφορτωθεί εξ ολοκλήρου.

Οι τεχνολογίες που συνδυάζει το AJAX πιο συγκεκριμένα είναι XHTML ή XML και CSS για σχεδιαστικούς λόγους, το Document Object Model ή αλλιώς DOM μέσω της JavaScript για τη δυναμική αναπαράσταση των πληροφοριών και τέλος το αντικείμενο XMLHttpRequest για την ασύγχρονη επικοινωνία του client με τον server.

Πως όμως λειτουργούν όλα αυτά; Αρχικά γίνεται η σύλληψη συμβάντος - γεγονότος από τον ανάλογο ακροατή συμβάντων τοπικά στο φυλλομετρητή του χρήστη. Δημιουργείται ένα αντικείμενο XMLHttpRequest το οποίο χρησιμοποιείται για την ανταλλαγή δεδομένων από τον client προς το server και αντίθετα. Μέσω του αντικειμένου αποστέλλεται μια ζήτηση – HttpRequest. Ο server λαμβάνει τη ζήτηση και προχωράει στην επεξεργασία – υλοποίηση της. Όταν έχει ολοκληρώσει την εργασία του στέλνει απάντηση στον client με τα δεδομένα που του ζητήθηκαν και ο client ενημερώνει με το νέο περιεχόμενο την σελίδα.



Σχήμα ροής.

Στο κομμάτι της εφαρμογής που παρουσιάζεται στην παρούσα εργασία η τεχνολογία AJAX χρησιμοποιήθηκε για την παρουσίαση αποτελεσμάτων μετά την ενέργεια εύρεσης χρήστη και πιο συγκεκριμένα στην σελιδοποίηση τους. Εδώ ουσιαστικά η τεχνολογία AJAX μας δίνει την δυνατότητα να προσθέσουμε το περιεχόμενα την νέας σελίδας χωρίς να χρειαστεί να ανανεωθεί η σελίδα στο φυλλομετρητή, απλά με javascript προσθέτει τα αποτελέσματα της καινούριας ζήτησης. Στην ενέργεια follow – unfollow δηλαδή την αντικατάστασης της εικόνας που παρουσιάζει την σχέση follow μεταξύ χρηστών. Εδώ κάθε φορά που κάποιος χρήστης αλλάζει την σχέση του με κάποιον άλλο το κουμπί που πραγματοποιεί την ενέργεια δείχνει την υπάρχουσα σχέση οπότε κατά την αλλαγή σχέσης αλλάζει και το τι αναγράφεται πάνω στο κουμπί. Αυτό γίνεται όπως και πριν με javascript χωρίς να ανανεωθεί η σελίδα στο φυλλομετρητή.

Redis

Η Redis είναι η βάση δεδομένων που επιλέχθηκε για την δημιουργία του data layer της εφαρμογής μας. Όπως αναφέρει ο L. Carlson (2013) πρόκειται για μία μη - σχεσιακή ή πιο συγκεκριμένα μια βάση της λογικής τιμή – κλειδί (key-value). Όταν λέμε κλειδί (key) εννοούμε το τύπο δεδομένων στον οποίου ο χρήστης αποθηκεύει δεδομένα και τιμή (value) είναι η τιμή που καταχωρείται σε αυτό.

Οι τύποι δεδομένων στους οποίους μπορούμε να αποθηκεύσουμε δεδομένα είναι strings, hashes, lists, sets, sorted sets, bitmaps and hyperloglogs και συνδέονται άμεσα με τις θεμελιώδεις δομές δεδομένων. Οι τύποι δεδομένων που χρησιμοποιήθηκαν στην εφαρμογή μας είναι lists, sets, sorted sets και hashes.

Τα strings είναι οι τύποι δεδομένων που αποθηκεύουν συμβολοσειρές. Είναι οι πιο απλοί τύποι δεδομένων και έχουν την μορφή κλειδί - τιμή

```
redis>  GET nonexisting
(nil)
redis>  SET mykey "Hello"
OK
redis>  GET mykey
"Hello"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

Οι λίστες είναι τύποι δεδομένων στους οποίους αποθηκεύονται στοιχεία σαν σειριακή ακολουθία. Η προσθήκη στοιχείων μπορεί να γίνει από την αρχή και από το τέλος της λίστας. Σχετικές εντολές:

LPUSH mylist "world" προσθέτει στην αρχή της λίστας mylist την συμβολοσειρά world

RPUSH mylist "world" προσθέτει στο τέλος της λίστας mylist την συμβολοσειρά world

LRANGE mylist 0 -1 επιστρέφει το περιεχόμενο της λίστας από την αρχή έως το τέλος

```
redis>  LPUSH mylist "world"
(integer) 1
redis>  LPUSH mylist "hello"
(integer) 2
redis>  LRANGE mylist 0 -1
1) "hello"
2) "world"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

Τα sets μοιάζουν με τις λίστες με την διαφορά ότι τα στοιχεία τοποθετούνται χωρίς συγκεκριμένη σειρά. Σχετικές εντολές:

SADD myset "world" προσθέτει στο Set myset την συμβολοσειρά world

SMEMBERS myset επιστρέφει το περιεχόμενο του Set

SISMEMBER myset "world" ελέγχει αν η συβλολοσειρά είναι στοιχείο του Set

```
redis>  SADD myset "Hello"
(integer) 1
redis>  SADD myset "World"
(integer) 1
redis>  SADD myset "World"
(integer) 0
redis>  SMEMBERS myset
1) "Hello"
2) "World"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

```
redis>  SADD myset "one"
(integer) 1
redis>  SISMEMBER myset "one"
(integer) 1
redis>  SISMEMBER myset "two"
(integer) 0
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

Τα sorted sets είναι sets με την διαφορά ότι σε κάθε εγγραφή υπάρχει και ένα πεδίο που ονομάζεται score και δίνει την δυνατότητα τοποθέτησης των εγγραφών σε σειρά βάση της τιμής του, για παράδειγμα ακέραιος αύξων αριθμός . Σε περίπτωση που υπάρχουν καταχωρίσεις του ιδίου score η τοποθέτηση γίνεται με γνώμονα την αλφαβητική σειρά. Σχετικές εντολές:

ZADD myzset 1 "world" προσθέτει στο Sorted Set την λέξη world με score 1

ZRANGE myzset 0 -1 επιστρέφει το περιεχόμενο του Sorted Set

```
redis>  ZADD myzset 1 "one"
(integer) 1
redis>  ZADD myzset 1 "uno"
(integer) 1
redis>  ZADD myzset 2 "two" 3 "three"
(integer) 2
redis>  ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "uno"
4) "1"
5) "two"
6) "2"
7) "three"
8) "3"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

Τέλος τα hashes είναι τύποι δεδομένων που φέρουν την μορφή key – filed - value όπου key το όνομα του κλειδιού η αλλιώς του τύπου δεδομένων, value η τιμή της κάθε εγγραφής και filed το πεδίο το οποίο δίνει περιγραφή στη εκάστοτε εγγραφή. Σχετικές εντολές:

HSET myhash field1 "world" προσθέτει στο hash myhash στο πεδίο field1 την συμβολοσειρά world

HGET myhash field1 επιστρέφει την τιμή του πεδίου του field1 του hash my hash

```
redis>  HSET myhash field1 "Hello"
(integer) 1
redis>  HGET myhash field1
"Hello"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

HMSET myhash field1 "hello" field2 "world" πολλαπλή προσθήκη στο hash myhash στο πεδίο field1 την συμβολοσειρά world και στο πεδίο field2 την συμβολοσειρά world

```
redis>  HMSET myhash field1 "Hello" field2 "World"
OK
redis>  HGET myhash field1
"Hello"
redis>  HGET myhash field2
"World"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

HGETALL myhash μαζική ανάκτηση, επιστρέφει όλα τα πεδία και τις τιμές αυτών του hash myhash

```
redis>  HSET myhash field1 "Hello"
(integer) 1
redis>  HSET myhash field2 "World"
(integer) 1
redis>  HGETALL myhash
1) "field1"
2) "Hello"
3) "field2"
4) "World"
redis>
```

Σχήμα εκτέλεσης εντολών σε κονσόλα (cli).

Στην εφαρμογή μας τα hashes χρησιμοποιήθηκαν για την αποθήκευση δεδομένων χρήστη κατά την εγγραφή του, καταχώριση rating. Τα sets και sorted sets στην καταχώριση σχέσεων μεταξύ χρηστών και ισότοπων που έχει βαθμολογήσει ένας χρήστης. Οι list για την καταχώριση βαθμολογήσεων χρήστη και βαθμολογήσεων που εμφανίζονται στο χρονολόγιο του.

Το κύριο πλεονέκτημά της σε σχέση με τις άλλες βάσεις είναι η ταχύτητα που παρέχει κατά την αποθήκευση και την ανάκτηση δεδομένων. Αυτό γίνεται γιατί η REDIS αποθηκεύει πρώτα στη μνήμη και μετά στο δίσκο. Η αποθήκευση στο δίσκο γίνεται με δύο τρόπους, ο πρώτος ονομάζεται snapshotting μεταφέρει ασύγχρονα τα δεδομένα με αρχεία τύπου RDB από την μνήμη στο δίσκο ανά μικρά χρονικά διαστήματα των οποίων ο χρόνος ορίζεται από τον διαχειριστή της βάσης. Ο τρόπου που προαναφέραμε μας επιτρέπει να αρχειοθετήσουμε RDB αρχεία κάθε ώρα και κατ επέκταση να μπορούμε να επαναφέρουμε εύκολα διαφορετικά στιγμιότυπα σε περίπτωση καταστροφών. Όπως εύκολα μπορεί κάνεις να σκεφτεί ο συγκεκριμένος τρόπος δεν είναι και ο πλέον ασφαλής αφού αν στο ενδιάμεσο των χρονικών διαστημάτων που προαναφέραμε συμβεί κάποιο λάθος όπως διακοπή ρεύματος ή συστημικό πρόβλημα του server τα δεδομένα στην μνήμη θα χαθούν. Εδώ λοιπόν έρχεται ο δεύτερος τρόπος κατά το οποίο κάθε φορά που η REDIS δέχεται μια εντολή η οποία θα προκαλεί αλλαγές στην βάση η εντολή αυτή γράφεται σε ένα αρχείο AOF append – only – file και όταν ο server κάνει επανεκκίνηση πρώτα θα τρέξει το AOF αρχείο και θα ανακτήσει τα δεδομένα που χάθηκαν. Όπως παρατηρούμε και οι δύο τρόποι που προαναφέραμε είναι σημαντικοί και αλληλοσυμπληρώνονται. Στις τελευταίες εκδόσεις της REDIS δίνεται η δυνατότητα για την κοινή χρήση των δύο.

Άλλο ένα πλεονέκτημά είναι η λειτουργία Master – Slave replication και ο εξαιρετικά ο απλός τρόπος εφαρμογής του σε σχέση με τη σχεσιακές βάσεις δεδομένων. Εδώ ουσιαστικά η Redis μας δίνει την δυνατότητα να εγκατασταθεί σε παραπάνω από ένα εξυπηρετητή. Έτσι γίνεται καταμερισμός εργασιών και φόρτου εργασίας και αυτό δίνει την δυνατότητα να επικοινωνούμε με την βάση γρήγορα και χωρίς μεγάλη επιβάρυνση του εξυπηρετητή.

Πριν την επιλογή της βάση δεδομένων της εφαρμογής μας έγινε και μία σχετική έρευνα και σύγκριση με άλλες βάσεις δεδομένων σχεσιακές και μη. Ξεκινήσαμε την έρευνα και την σύγκριση με την κατηγορία των σχεσιακών βάσεων και πιο συγκεκριμένα την mySQL. Ο πιο σημαντικός παράγοντας ήταν η ταχύτητα και η ασφάλεια των δεδομένων. Η redis υπερισχύει όσον αφορά την ταχύτητα κατά πολύ και αυτό γιατί όπως είπαμε καταχωρεί τα δεδομένα πρώτα στην μνήμη και μετά ασύγχρονα στο δίσκο. Προσφέρει υψηλά στάνταρ επεκτασιμότητας δηλαδή την δυνατότητα διαμοιρασμού της σε διαφορετικές μηχανές.

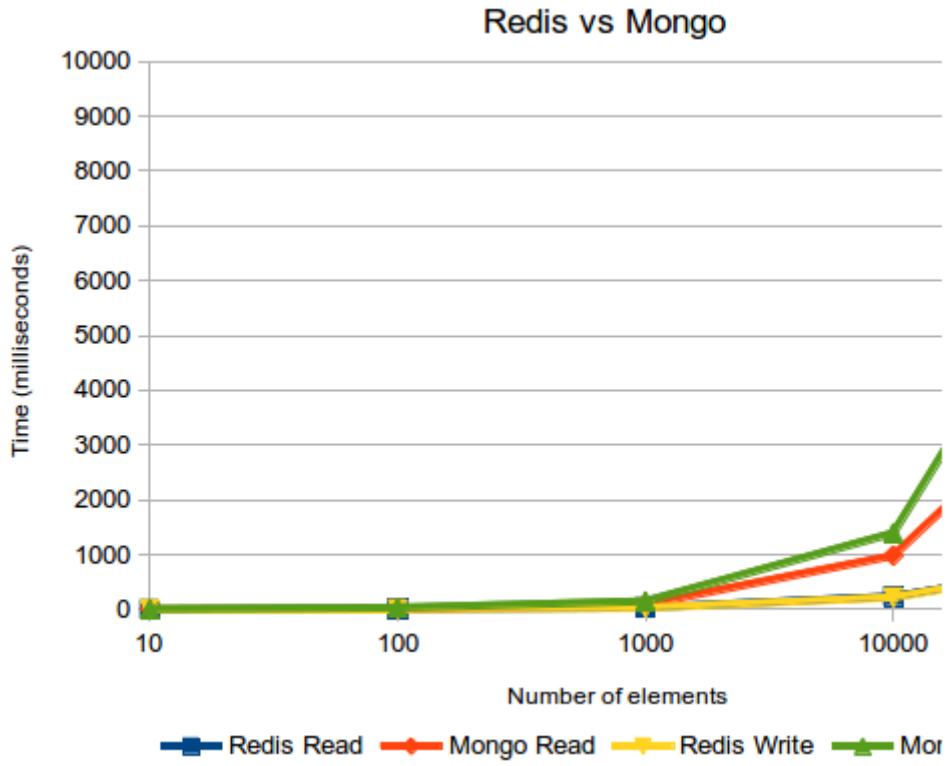
Στην κατηγορία των μη σχεσιακών βάσεων δεδομένων υπήρχε και η επιλογή της mongoDB μια document - oriented βάση δεδομένων η οποία ήταν γνωστή και πιο διαδεδομένη σε σχέση με την Redis λόγο της παλαιότητας της. Κάνοντας σχετική έρευνα καταλήξαμε ότι η Redis υπερισχύει της mongoDB όπως αναφέρεται και σε σχετικό test που αναρτήθηκε από τον Tarek Salah (2013). Το τεστ έγινε σε γλώσσα node.js, ουσιαστικά έγινε καταχώρηση και ανάκτηση απλών δεδομένων με μια εντολή επανάληψης for.

Τα χαρακτηρίστηκα του υπολογιστή στον οποίο εκτελέστηκε το τεστ ήταν επεξεργαστής 4x Intel(R) Core(TM) i5-3230M CPU στα 2.60GHz, μνήμη 3892MB, λειτουργικό σύστημα Ubuntu 13.04. Το τεστ έγινε για πλήθος εγγραφών 10, 100, 1000, 10000, 50000 όπως παρατίθεται και στον πίνακα που ακολουθεί.

	Redis Read	Mongo Read	Redis Write	Mongo Write
10	2	5	5	8
100	13	11	8	34
1,000	38	93	31	153
10,000	238	980	220	1394
50,000	958	5218	979	8713

Πίνακας παρουσίασης αποτελεσμάτων σε milliseconds.

Όπως φαίνεται από το πίνακα και από το διάγραμμα που ακολουθεί το αποτέλεσμα ήταν ότι και οι δύο βάσεις έχουν σχεδόν κοινή συμπεριφορά για όγκο δεδομένων από 10 έως και 100 εγγραφές. Από εκεί και πάνω βλέπουμε ότι όσο αυξάνεται ο αριθμός εγγραφών η Redis υπερισχύει σε ταχύτητα της mondoDB αφού χρειάζεται πολύ λιγότερο χρόνο.

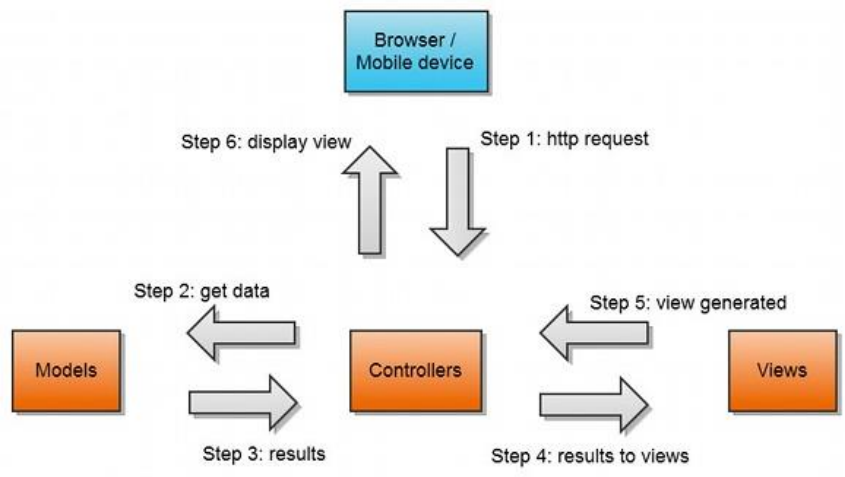


Διάγραμμα παρουσίασης αποτελεσμάτων σε milliseconds.

Laravel

Το Laravel είναι ένα PHP framework πάνω στο οποίο γράφτηκε ο κώδικάς της πτυχιακής. Είναι ένα εργαλείο ελεύθερου λογισμικού βασισμένο στο αρχιτεκτονικό πρότυπο υλοποίησης κώδικα M.V.C. (Μοντέλο Παρουσίαση Ελεγκτής).

Το MVC σύμφωνα με τον Hardik Dangar (2013) είναι ένα αρχιτεκτονικό πρότυπο με κύριο χαρακτηριστικό το διαχωρισμός λογικής κατά την γραφή και τοποθέτηση κώδικα. Ουσιαστικά ο κώδικας που γράφετε διαχωρίζεται βάση των τριών οντοτήτων που ακολουθούν. Τα Models, είναι το στρώμα που μεσολαβεί ανάμεσα στο χρήστη και στα δεδομένα, τη βάση δεδομένων. Τα Views είναι η οπτική απεικόνιση της εφαρμογής, το στρώμα στο οποίο ανήκουν αρχεία τα οποία εμφανίζονται στο φυλλομετρητή και παράγουν το εικαστικό αποτέλεσμα. Οι Controllers είναι το στρώμα σύνδεσης μεταξύ Models και Views. Πρωταρχικός ρόλος τους είναι χρήση των request που έρχονται από τα Views και η μεταφορά δεδομένων στα Models και αντίστροφα.



Σχήμα απόδοσης MVC.

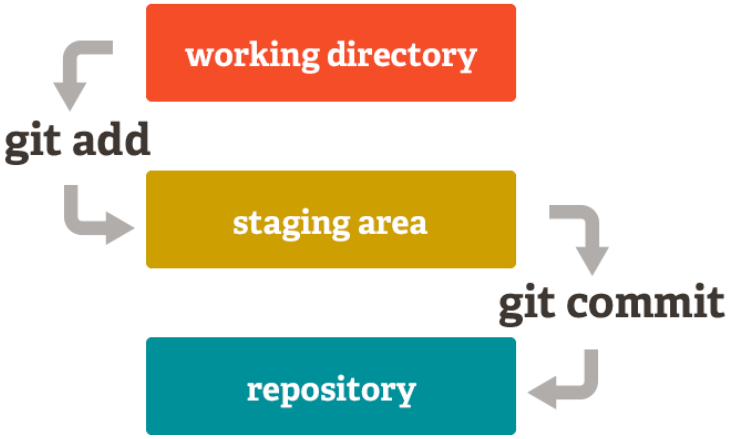
Το πλαισιο εργασίας παρέχει στους χρήστες την δυνατότητα να υλοποιούν εύκολα και γρήγορα ολοκληρωμένες εφαρμογές χρησιμοποιώντας τις βιβλιοθήκες και τα έτοιμα εργαλεία που διαθέτει καθώς και να προσθέτει νέα ανάλογα με τις απαιτήσεις της εφαρμογής. Παρέχει ασφάλεια σε κακόβουλες επιθέσεις SQL injection και Cross Site Scripting (XSS). Επιθέσεις SQL injection είναι οι εντολές SQL που εισάγονται σε πεδία εισόδου στον ιστότοπο μας με σκοπό την κακόβουλη ανάκτηση, αλλαγή, διαγραφή δεδομένων από την βάση. Οι XSS επιθέσεις είναι ο κώδικας που εισάγεται στα πεδία διαδικτυακής φόρμας μιας ιστοσελίδας μέσω του οποίου οι κακόβουλοι χρήστες μπορούν να έχουν πρόσβαση στα cookies, session και άλλα απόρρητα δεδομένα στο φυλλομετρητής (browser) του χρήστη.

Ξεκινώντας την εφαρμογή είχαμε να επιλέξουμε ανάμεσα σε δύο framework το Laravel και το CodeIgniter. Επιλέξαμε το laravel γιατί είναι ότι πιο νέο υπάρχει σε PHP πλαισιο εργασίας. Υποστηρίζει την τελευταία έκδοση PHP ενώ το CodeIgniter έως την PHP 5.2. Διαθέτει μεθόδους και εργαλεία που στο CodeIgniter πρέπει να κατασκευαστούν η να προστεθούν εκ νέου όπως για παράδειγμα την απομνημόνευση του URL που βρίσκεται ο χρήστης κάθε φορά που αλλάζει πεδίο στο διαδικτυακό χώρο μια εφαρμογής και την αυτόματη μεταφορά του σ' αυτό σε περίπτωση τυχαίας αποσύνδεσης και επανασύνδεσης του. Διαθέτει ένα από τα πιο σύγχρονα Object Relational Mapper (O.R.M) το Eloquent. Είναι προγραμματίστηκες τεχνικές μέσω των οποίων οι χρήστες μπορούν να διαχειριστούν την βάση δεδομένων της εφαρμογής τους με αντικείμενα (objects) τα οποία αντιστοιχούν στους πίνακες της βάσης. Έχει ενσωματωμένη πιστοποίηση χρήστη (user authentication), διαθέτει την «μηχανή» παραγωγής Blade templates ουσιαστικά παρέχει κώδικα βασισμένο στην γλώσσα php ο οποίος γράφεται στα αρχεία των Views και παρέχει καλύτερη λειτουργικότητα και διαχειρισμό των αρχείων. Τέλος είναι συμβατό με την βάση δεδομένων REDIS που επιλέξαμε αφού περιέχει βιβλιοθήκες και εντολές που κάνουν την σύνδεση και τη διαχείριση της βάσης πολύ απλή.

Version Control Git

Τα version control είναι συστήματα που διαχειρίζονται αλλαγές αρχείων που κατά την πλειοψηφία τους έχουν να κάνουν με κώδικα γλωσσών προγραμματισμού. Κάθε αλλαγή που πραγματοποιείτε στα αρχεία συνδέεται με μία χρονοσήμανση (timestamp) και με αυτόν που την έκανε. Έτσι διατηρείται ιστορικό των αλλαγών και οι χρήστες μπορούν ανά πάσα στιγμή να ανακτήσουν όποιο σημείο του ιστορικού επιθυμούν αποφεύγοντας έτσι προγραμματιστικά λάθη. Τέλος μετά από κάθε αλλαγή τα version control παρέχουν την δυνατότητα σύγκρισης αρχείων που έχουν υποστεί αλλαγές ή και συγχώνευσης τους.

Είναι ευκόλως εννοούμενο λοιπόν ότι ένα τέτοιο συστήματα ενδείκνυται για ομαδική εργασία και στην περίπτωση μας ήταν ένα πολύτιμο εργαλείο. Για να χρησιμοποιήσουμε το git ήταν απαραίτητη η χρήση του bitbucket μια διαδικτυακή υπηρεσία που παρέχει αποθηκευτικό χώρο και χρησιμοποιεί τα version control συστήματα Git και Mercurial. Ο χώρος στο οποίο αποθηκεύονται τα δεδομένα ονομάζεται repository το οποίο προσφέρει το bitbucket. Στο repository μπορούν να συνδεθούν έως οκτώ χρήστες, στους οποίους δίνονται δικαιώματα χρήσης από το άτομο που το δημιούργησε. Κάθε χρήστης για την ασφάλεια της σύνδεσης του με το repository χρησιμοποιεί δημόσιο και ιδιωτικό κλειδί το οποίο παράγεται από το bitbucket. Όταν ολοκληρωθεί η σύνδεση με το repository και αφού ο χρήστης κατεβάσει (pull) από το repository στο υπολογιστή του το project μπορεί πλέον να προγραμματίζει τοπικά. Κάθε φορά που τελειώνει μια εργασία θα πρέπει να πραγματοποιείται η ενέργεια ‘πρόσθεσε στο ιστορικό’ (commit). Στο commit ο χρήστης μπορεί να προσθέσει σχόλια και link που συνδέονται με υποφακέλους στο bitbucket - repository και χρησιμοποιούνται για συγγραφή θεωρίας (wiki) ή αναφορών (tasks, bugs, issues). Τέλος η εντολή push ανεβάζει τις αλλαγές ή τα καινούρια κομμάτια κώδικα στο repository.

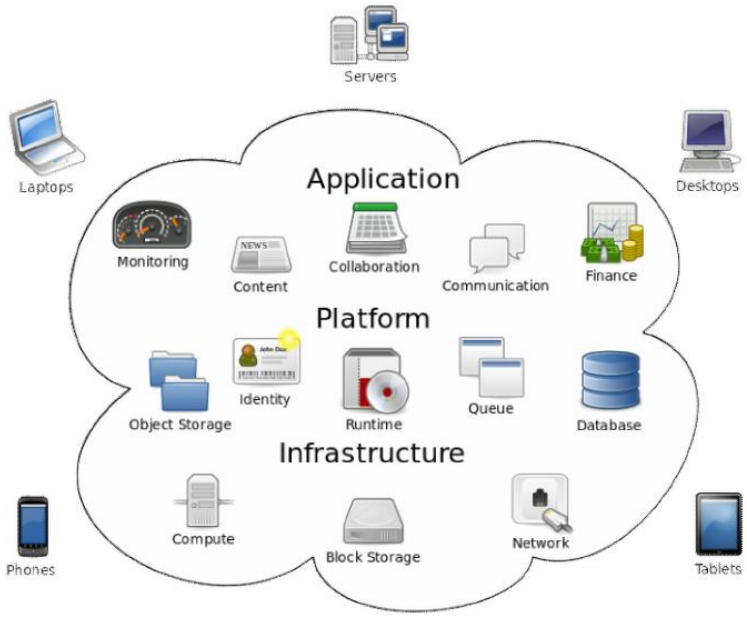


Σχήμα απόδοσης απλής ροής.

Στην περίπτωσή μας δημιουργήθηκε ένα repository στο bitbucket, όπου υπήρχε ο κώδικας της εφαρμογής μας και οι τρείς φοιτητές συνδεθήκαμε με αυτό. Τοπικά στους υπολογιστές μας υπήρχε αντίγραφο της εφαρμογής μετά την διαδικασία pull. Κάθε φορά που πραγματοποιούσαμε μία εργασία που μας είχε ανατεθεί κάναμε τις ενέργειες commit και push ενημερώνοντας έτσι το ιστορικό και τον κώδικα στο repository. Το bitbucket όπως προαναφέραμε παρέχει την δυνατότητα συγγραφής wiki και αναφορών. Για ότι εργασία μας ανατέθηκε πριν ξεκινήσουμε κάναμε μια αναφορά του θέματος της και μετά τη διεκπεραίωση ακολουθούσε η συγγραφή θεωρίας και επεξήγησης, δίνοντας έτσι σε όλους την δυνατότητα να κατανοήσουν πλήρως τον τρόπο υλοποίησης. Τα link των wikis και των αναφορών θέματος μπαίνανε στο ανάλογο commit. Όλα τα παραπάνω γινόντουσαν υπό την επίβλεψη του καθηγητή.

Cloud Storage

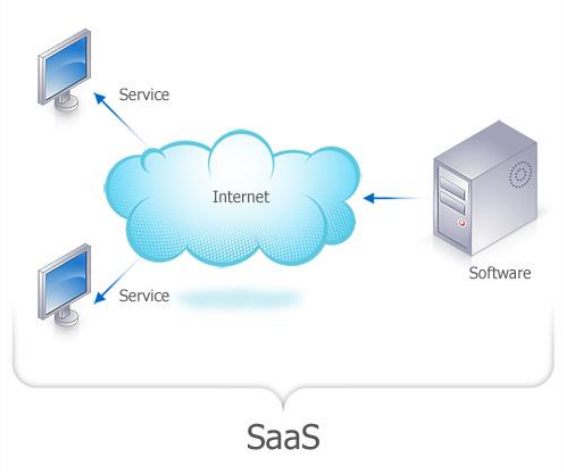
Υπολογιστικό νέφος ή αλλιώς νέφος. Σαν ορισμό θα χρησιμοποιήσουμε αυτόν του National Institute of Standards and Technology (2011). Το cloud computing είναι ένα μοντέλο που επιτρέπει ευέλικτη, χρεώσιμη δικτυακή πρόσβαση σε ένα κοινόχρηστο σύνολο υπολογιστικών πόρων όπως δίκτυα, εξυπηρετητές, αποθηκευτικό χώρο, εφαρμογές και υπηρεσίες το οποίο μπορεί να τροφοδοτηθεί γρήγορα και να είναι διαθέσιμο με την ελάχιστη προσπάθεια διαχείρισης και αλληλεπίδρασης από τον πάροχο της συγκεκριμένης υπηρεσίας.



Εικόνα 3. Υπολογιστικό νέφος.

Σε γενικές γραμμές θα λέγαμε ότι το «νέφος» αποτελεί ένα νέο επιχειρηματικό μοντέλο παροχής υπηρεσιών το οποίο παρέχει υπηρεσίες υπολογισμού, λογισμικού, διαμοιρασμού πόρων, πρόσβασης σε πληροφορίες και αποθήκευσης δεδομένων χωρίς να απαιτείται η γνώση της γεωγραφικής θέσης και της διαμόρφωσης του συστήματος από τον τελικό χρήστη. Το νέφος συνιστά μια τεράστια αλλαγή στον τρόπο με τον οποίο παρέχονται οι υπολογιστικοί πόροι, καθώς επιτρέπει την ανάκτηση πληροφοριών, την επεξεργασία και την αποθήκευση δεδομένων και τη χρήση εφαρμογών όπου και αν βρισκόμαστε χωρίς την απαίτηση για αγορά και εγκατάσταση των συγκεκριμένων υπηρεσιών. Η λειτουργία του παρουσιάζει αρκετές ομοιότητες με αυτή του δικτύου ηλεκτρικής ενέργειας, όπου οι τελικοί χρήστες καταναλώνουν ενέργεια δίχως να χρειάζεται να κατανοήσουν τις υποδομές που απαιτούνται για την παροχή της υπηρεσίας.

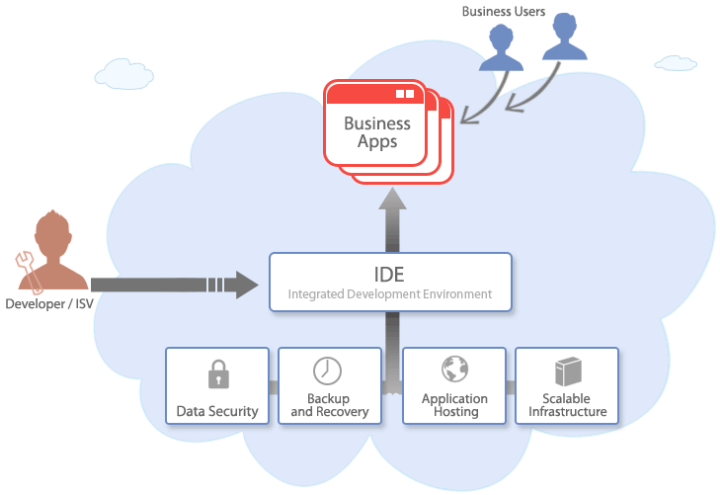
Software as Service (SasS)



Σε αυτό τον τύπο υπάρχει ένα application το οποίο βρίσκεται σε ένα cloud server και ο χρήστης μπορεί να έχει πρόσβαση σε αυτό μέσω μίας απλή σύνδεσης στο ιντερνέτ. Το software αυτό ανήκει σε κάποιον κατασκευή και ο χρήστης το πληρώνει ανάλογα με την χρήση που του κάνει και τους πόρους που χρειάζεται. Το βασικό πλεονέκτημα του μοντέλου «software as service» είναι ότι ο κατασκευαστής αναλαμβάνει τα έξοδα συντήρησης του software καθώς και τη φιλοξενία του σε κάποιον cloud server. Ο χρήστης πληρώνει μόνο την χρήση που κάνε (αν και υπάρχουν και cloud applications που είναι δωρεάν). Επίσης το μοντέλο SasS είναι δημιουργημένο με βασικό γνώμονα τη σωστή λειτουργία του software με χρήση browser. Όσον αφορά την ασφάλεια των διαφόρων εφαρμογών, συνήθως χρησιμοποιείται SSL (Secure Sockets Layer) το οποίο είναι παγκοσμίως αναγνωρισμένο. Έτσι , οι χρήστες μπορούν με ασφάλεια να χρησιμοποιήσουν το cloud application.

Platform as Service (PasS)

Αυτό το μοντέλο μοιάζει πολύ με το προηγούμενο. Το βασικό του στοιχείο είναι ότι παρέχει την πλατφόρμα την οποία χρησιμοποιεί ένας χρήστης για να δημιουργήσει κάτι, για παράδειγμα ένα web application, χωρίς να εγκαταστήσει τίποτα. Το «platform as service» μοντέλο χρησιμοποιείται πιο πολύ για δημιουργία web interfaces, web εφαρμογών κλπ. Ένα σημαντικό πρόβλημα που υπάρχει με αυτό το μοντέλο είναι ότι αυτή η εφαρμογή που δημιουργούμε βασίζεται σε ένα συγκεκριμένο framework και υπάρχει πιθανότητα αν θελήσουμε να την μεταφέρουμε σε άλλο παροχέα cloud υπηρεσιών αυτή να μη λειτουργεί σωστά.



Storage as a service (StaaS)

Στο μοντέλο αυτό υπάρχει κάποιος πάροχος αποθηκευτικού χώρου online ο οποίος στην ουσία τον νοικιάζει έναντι κάποιας αμοιβής.

Hardware as Service (HaaS)

Εδώ ο προμηθευτής αυτής της cloud υπηρεσίας παρέχει στον χρήστη έναντι «ενοικίου»-αμοιβής το hardware που χρειάζεται όπως web servers, μνήμη CPU ,αποθηκευτικό χώρο και ότι άλλο χρειάζεται ο χρήστης σε επίπεδο hardware. Τα χρήματα που πληρώνει κάποιος στο HaaS είναι αντίστοιχα της χρήσεως των πόρων του συστήματος που κάνει.

Database as Service (DaaS)

Σε αυτό το μοντέλο υπάρχει μία υπηρεσία online παρέχει την βάση δεδομένων την οποία μπορούμε να χρησιμοποιήσουμε με κάποιο web application. Σε αυτό το μοντέλο το βασικό πλεονέκτημα είναι ότι πληρώνουμε ανάλογα με την χρήση. Ουσιαστικά όσο πιο πολύ κόσμος χρησιμοποιεί την εφαρμογή μας τόσο περισσότερα χρήματα πληρώνουμε.

Τα πέντε βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι τα ακόλουθα.

On-demand self-service: Παρέχονται στον καταναλωτή μονομερώς υπολογιστικοί πόροι, όπως ο χρόνος χρήσης του server και το μέγεθος του αποθηκευτικού χώρου, οι οποίοι δεσμεύονται μέσω του δικτύου αυτόματα.

Broad network access: Οι υπολογιστικοί πόροι είναι διαθέσιμοι μέσω του δικτύου και μπορούν να χρησιμοποιηθούν από τερματικές συσκευών όπως κινητά τηλέφωνα, tablets, υπολογιστές.

Resource pooling: Οι υπολογιστικοί πόροι του παρόχου χρησιμοποιούνται για να εξυπηρετήσουν παράλληλα πολλούς καταναλωτές ακολουθώντας το μοντέλο πολλαπλών εκμισθωτών (multi - tenant). Δίνεται μια αίσθηση ανεξαρτησίας όσον αφορά τη φυσική τοποθεσία, καθώς ο καταναλωτής δεν έχει κανέναν έλεγχο ή γνώση σχετικά με την ακριβή τοποθεσία των παρεχόμενων πόρων. Παραδείγματα παρεχόμενων πόρων αποτελούν η επεξεργαστική ισχύς, η μνήμη, το εύρος ζώνης (bandwidth) του δικτύου και οι εικονικές μηχανές (virtual machines).

Rapid elasticity: Οι πόροι μπορούν να δεσμευτούν προς χρήση γρήγορα με ευέλικτο τρόπο, έτσι ώστε να εμφανίζονται στον καταναλωτή ως απεριόριστοι και διαθέσιμοι προς αγορά ανά πάσα στιγμή.

Measured service: Τα συστήματα cloud έχουν τη δυνατότητα να ελέγχουν και να βελτιστοποιούν αυτόματα τη χρήση των υπολογιστικών πόρων. Η χρήση των πόρων μπορεί να ελέγχεται παρέχοντας διαφάνεια και στις δύο πλευρές, τόσο του καταναλωτή-χρήστη όσο και του παρόχου της υπηρεσίας.

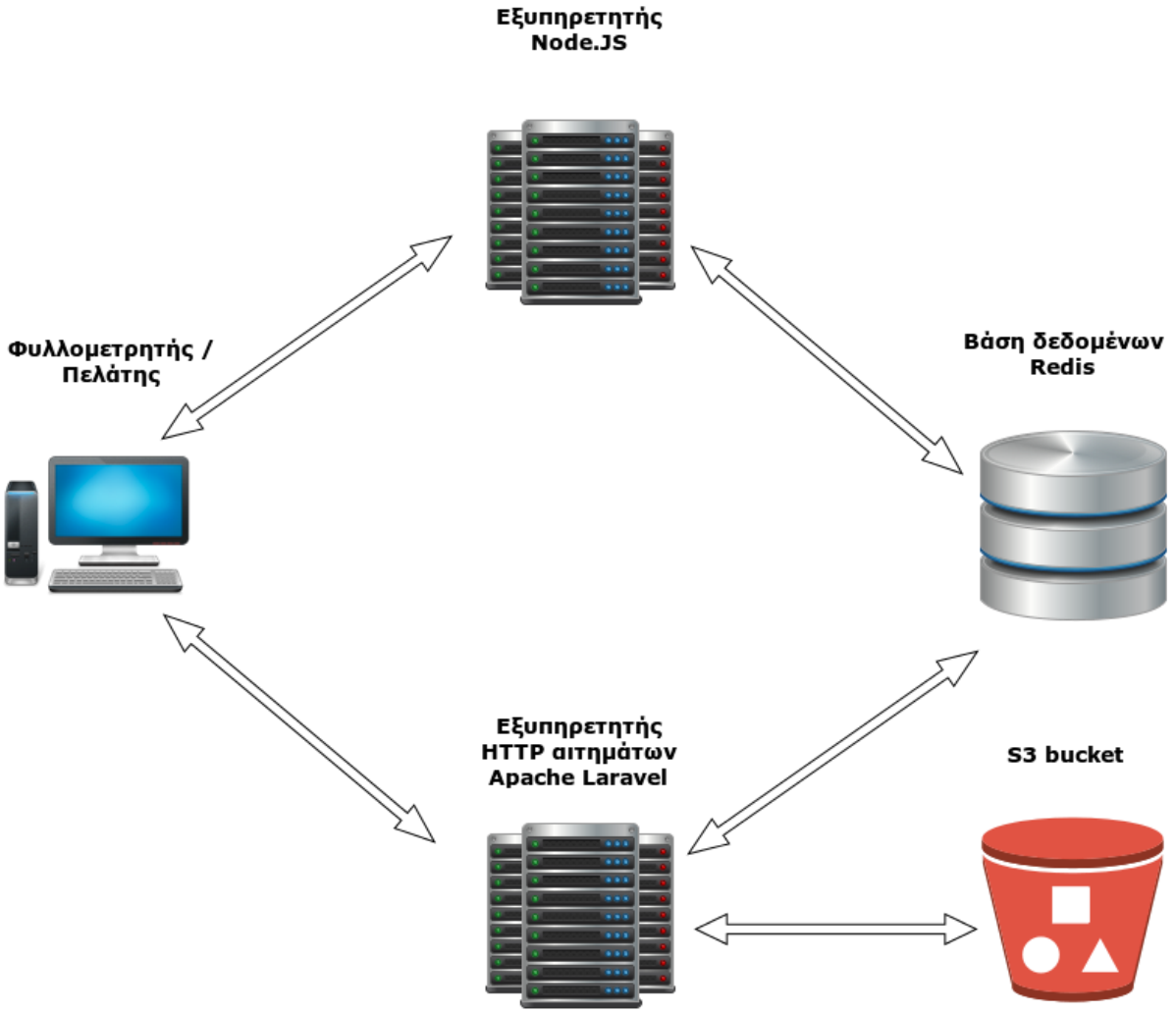
Το s3 της Amazon είναι μία online διαδικτυακή υπηρεσία αποθήκευσης αρχείων η οποία προσφέρεται από τα Amazon Web Services. Το S3 παρέχει ασφαλή και ανθεκτική δυνατότητα αποθήκευσης αρχείων με την δημιουργία αντιγράφων, χαμηλό κόστος αποθήκευσης, δυνατότητα χρήσης αρχείων που αποθηκεύονται σε αυτό από οποιοδήποτε μέρος εάν και εφόσον είναι εφικτή η σύνδεση στο διαδίκτυο και τέλος περιβάλλον διασύνδεσης χρηστών που κάνουν την διαχείριση του εξαιρετικά εύκολη. Παρέχει στους χρήστες πακέτο λογισμικού ανάλογα με την γλώσσα προγραμματισμού που έχει χρησιμοποιηθεί για την υλοποίηση της εκάστοτε εφαρμογής. Η σύνδεση με το S3 όπως και με κάθε τέτοια υπηρεσία γίνεται με την χρήση κλειδιών - κωδικών που παρέχονται από το πάροχο της υπηρεσίας.

Στην περίπτωση της παρούσας εργασίας επιλέχθηκε το S3 για την αποθήκευση εικόνων (thumbnails) των Rates και των χρηστών (avatars). Όπως αναφέραμε στην εισαγωγή κάθε φορά που πραγματοποιείται ένα νέο Rate το σύστημα βγάζει μια φωτογραφία σε μορφή εικονιδίου της αρχικής σελίδας του ιστότοπου η οποία προβάλλεται κάθε φορά που παρουσιάζεται το συγκεκριμένο Rate. Επίσης οι χρήστες που χρησιμοποιούν την εφαρμογή μπορούν να προσθέσουν στο προφίλ τους μια φωτογραφία που τους αντιπροσωπεύει. Η διαδικασία της εγκατάστασης του S3 είχε ως εξής. Πρώτα εγκαταστάθηκε στο laravel το πακέτο (SDK) που προσφέρει η Amazon για την γλώσσα προγραμματισμού PHP. Μετά από την εγκατάσταση στον ισότοπο την Amazon έγινε εγγραφή και επιλέχτηκε η ανάλογη υπηρεσία μέσω περιβάλλοντος διασύνδεσης (User Interface) όπου και κατοχυρώνεται ο αποθηκευτικός χώρος ανάλογα το πακέτο επιλογής. Έτσι κάθε χρήστης έχει το δικαίωμα διαχείρισης του αποθηκευτικού του χώρου. Εκεί δημιουργούνται και τα κλειδιά που χρειάζονται για την σύνδεση. Το κλειδί που αντιστοιχεί στο χρήστη πρέπει να τοποθετήθηκε σε ανάλογο πεδίο στον εφαρμογή και στην συγκεκριμένη περίπτωση στο laravel. Το S3 για την αποθήκευση αρχείων προϋποθέτει δημιουργία υπό φακέλων στο σύννεφό μας τα λεγόμενα καλάθια (buckets). Δημιουργήσαμε δύο όσα και τα είδη εικονιδίων που θα αποθηκεύσουμε Rates και Avatars. Πλέον μπορούμε να αποθηκεύουμε στο cloud τα αρχεία της επιλογής μας.

Κεφάλαιο 4ο

Αρχιτεκτονική

Η αρχιτεκτονική βασίζεται σε αυτή των τριών επιπέδων. Πιο συγκεκριμένα ξεκινώντας από το χαμηλότερο επίπεδο έχουμε το επίπεδο δεδομένων (data layer), επίπεδο εξυπηρετητή (web server) και επίπεδο πελάτη (front end).



Αρχιτεκτονική της εφαρμογής.

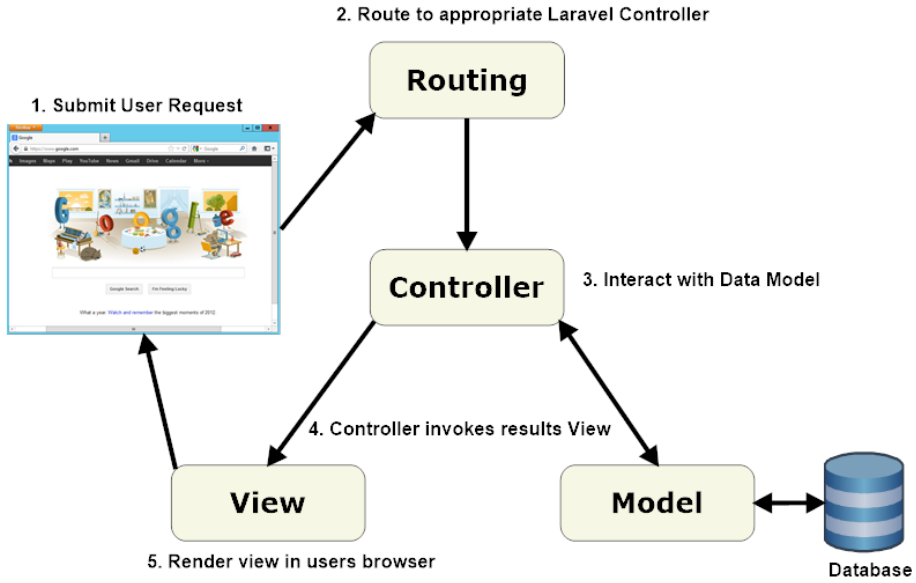
Το εργαλείο που χρησιμοποιήθηκε για να υλοποιηθούν προγραμματιστικά όλες οι λειτουργίες της εφαρμογής είναι το laravel το οποίο βασίζεται στο αρχιτεκτονικό πρότυπο MVC όπως αναφέρθηκε και στο κεφάλαιο με τις τεχνολογίες. Έτσι για κάθε κατηγορία – κομμάτι που αναφέραμε δημιουργήθηκαν και οι ανάλογοι ελεγκτές (controllers) και μοντέλα (models). Αναφορικά τα μοντέλα ασχολούνται με την βάση και ότι ενέργειες γίνονται από και προς αυτή και οι controllers μεταφέρουν και επεξεργάζονται τις πληροφορίες που έρχονται από τα views (αρχεία client side) προς τα μοντέλα και αντίστροφα.

Όλοι οι ελεγκτές και τα μοντέλα της εφαρμογής είναι κλάσεις και οι λειτουργίες που εκτελούνται από αυτές είναι μέθοδοι. Το Laravel έχει μία συγκεκριμένη δομή και κάθε φορά που υλοποιείται μία εφαρμογή απλά επεκτείνεται η δομή αυτή με βάση τις εκάστοτε απαιτήσεις. Για να γίνουμε πιο συγκεκριμένοι το laravel διαθέτει μια κλάση την Controller και ένα ελεγκτή το BaseController που κάνει extend την αρχική κλάση και υιοθετεί όλες τις ιδιότητες και μεθόδους αυτής. Μέχρι την στιγμή που εγκαθίσταται ένα project laravel ο BaseController συμβάλει στην λειτουργία παραγωγής των views. Όλοι οι ελεγκτές που δημιουργούνται επεκτείνονται με βάση αυτόν ή αλλιώς

κάνουν extend το BaseController και κληρονομούν με την σειρά τους τα χαρακτηριστικά της κλάσης Controller.

Μία λειτουργία που χρησιμοποιήθηκε είναι ο validator. Είναι μια κλάση που διαθέτει το laravel και δίνει την δυνατότητα ελέγχου δεδομένων που έρχονται από τις φόρμες στις html σελίδες. Όλη η λειτουργικότητα είναι έτοιμη και πολύ εύκολη προς χρήση και εκτός από την μεγάλη ποικιλία που προσφέρει σε κανόνες ελέγχου υπάρχει η δυνατότητα δημιουργίας καινούριων. Όπως και με τους κανόνες το laravel μας δίνει την δυνατότατα δημιουργίας βιβλιοθηκών με καινούριες κλάσεις όπως τους ελεγκτές, τα μοντέλα και το validator.

Τέλος υπάρχει και η κλάση Route η οποία διευθετεί τα HTTP requests ή αλλιώς ασχολείται με τα URLs της εφαρμογής. Υπάρχει ένα αρχείο όπου μέσα εκεί γράφονται τα λεγόμενα Route rules τα οποία λαμβάνουν τα HTTP requests της εφαρμογής και εκτελούν την διαδικασία που αντιστοιχεί σε αυτά. Μία διαδικασία για παράδειγμα μπορεί να είναι η μέθοδος ενός ελεγκτή της εφαρμογής μας.



Εικόνα 4.0

Στην αναφορά που έγινε στο κεφάλαιο τεχνολογιών για το laravel μιλήσαμε για τις βιβλιοθήκες της βάση δεδομένων Redis που διαθέτει το framework. Κάθε φορά που θέλουμε να συνδεθούμε με την βάση καλούμε την στατική μέθοδο Redis::connection η οποία και επιστρέφει αντικείμενο της σύνδεσης με την βάση. Λόγο του ότι στην εφαρμογή μας συχνά χρειάζεται να διαχειριστούμε την βάση και η διαδικασία αυτή θα γινόταν κατ επανάληψη και σε πολλά διαφορετικά σημεία τοποθετήσαμε την στατική αυτή μέθοδο στον κατασκευαστή του BaseController και BaseModel και αποθηκεύσαμε το αντικείμενο που επιστρέφει σε χαρακτηρίστηκα αυτών. Οι κλάσεις γίνονται extend από όλους του αντίστοιχους Controllers και Models. Έτσι οποία μέθοδος εξ αυτών κληθεί κληρονομεί και το αντίστοιχο χαρακτηριστικό της σύνδεσης με την βάση και μπορεί σε οποιοδήποτε σημείο στους Controllers ή στα Models να διαχειριστούμε την βάσης.

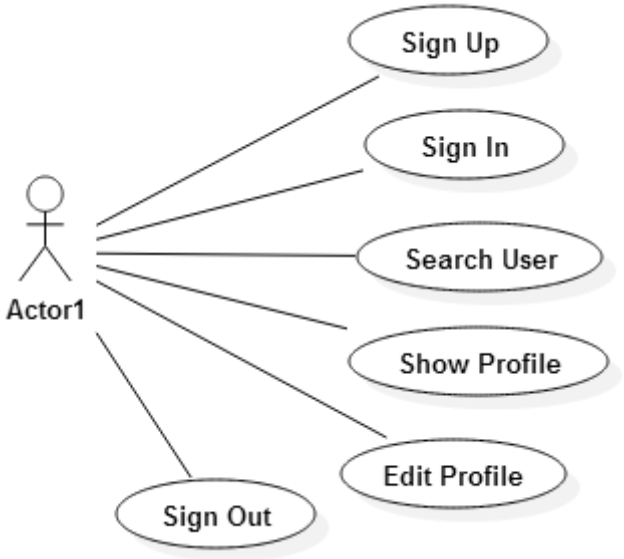
Με μια σύντομη περιγραφή θα λέγαμε ότι η εργασία μας είναι μια διαδικτυακή εφαρμογή την οποία χρήστες μπορούν να χρησιμοποιούν κατόπιν εγγραφής δημιουργώντας λογαριασμό. Εισερχόμενοι στην εφαρμογή μπορούν να δημοσιεύουν διαδικτυακούς ιστότοπους, να βαθμολογούν τις δικές τους ή και δημοσιεύσεις άλλων και να δημιουργούν μονόδρομες σχέσεις με χρήστες της εφαρμογής. Σαν επέκταση των λειτουργιών προστέθηκαν οι λειτουργίες αποθήκευση αρχείων σε αποθηκευτικό νέφος και η προγραμματιστική τεχνική comet βάσει της οποίας λειτουργούν οι ειδοποιήσεις. Ειδοποιήσεις είναι οι ενημερώσεις που προσφέρει το σύστημα σε κάθε χρήστη σε κατάσταση πραγματικού χρόνου(real time).

Βάσει της σύντομης περιγραφής που προηγήθηκε παρατηρούμε ότι η εφαρμογή χωρίζεται σε τρία βασικά μέρη. Το κομμάτι που ασχολείται με τον χρήστη (User), το κομμάτι που ασχολείται με τις δημοσιεύσεις (Rate) και το κομμάτι που ασχολείται με τις σχέσεις μεταξύ χρηστών (Follow Relations). Αυτά τα τρία κομμάτια ήταν ο γνώμονας βάση του οποίου έγινε και ο καταμερισμός εργασίας στους τρεις φοιτητές που υλοποίησαν την εφαρμογή. Σε αυτήν την εργασία ασχοληθήκαμε με τα κομμάτια

User, Follow και Cloud Storage και είναι αυτά τα οποία θα παρουσιάσουμε και στα οποία θα εμβαθύνουμε στο παρόν κεφάλαιο.

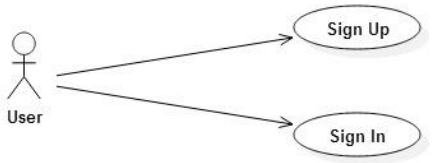
User

Εδώ εντάσσονται λειτουργίες που έχουν να κάνουν με την εισαγωγή του χρήστη στην εφαρμογή όπως φόρμα εγγραφής, φόρμα εισόδου, αποστολή μηνύματος επαλήθευσης email κατά την εγγραφή του χρήστη. Επίσης έλεγχος αυθεντικότητας, λειτουργία εύρεσης χρήστη και η σελιδοποίηση των αποτελεσμάτων, εμφάνιση προφίλ χρήστη, διαχείριση προφίλ χρήστη, λειτουργία εξόδου από την εφαρμογή.



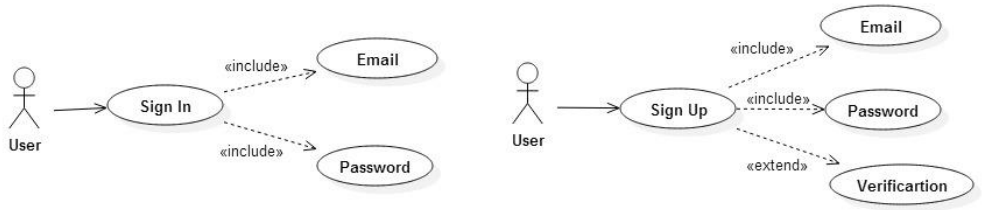
Εικόνα 4.1. Ενέργειες χρήστη.

Ξεκινώντας από την αρχή ο χρήστης όταν φτάσει στον ισότοπο της εφαρμογής συναντάει την φόρμα εισόδου και την φόρμα εγγραφής.



Εικόνα 4.2. Εγγραφή, εισαγωγή χρήστη.

Επιλέγει την φόρμα βάση της κατάστασης στην οποία βρίσκεται, νέος χρήστης Sign Up, ήδη εγγεγραμμένος Sign In και συμπληρώνει τα πεδία email και password.

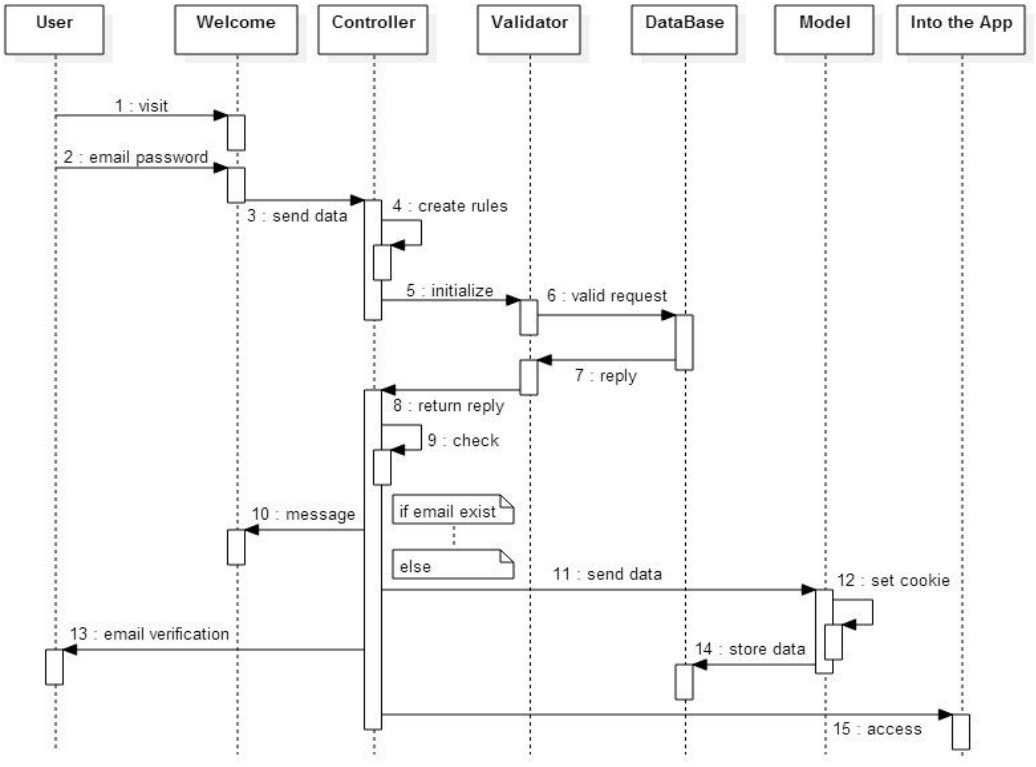


Εικόνα 4.3. Εγγραφή, εισαγωγή χρήστη.

Πριν συνεχίσουμε θα γίνει μια αναφορά στον αλγόριθμο double metaphone. Το metaphone είναι ένας φωνητικός αλγόριθμος ο οποίος μετατρέπει μια γραμματοσειρά σε μια νέα γραμματοσειρά

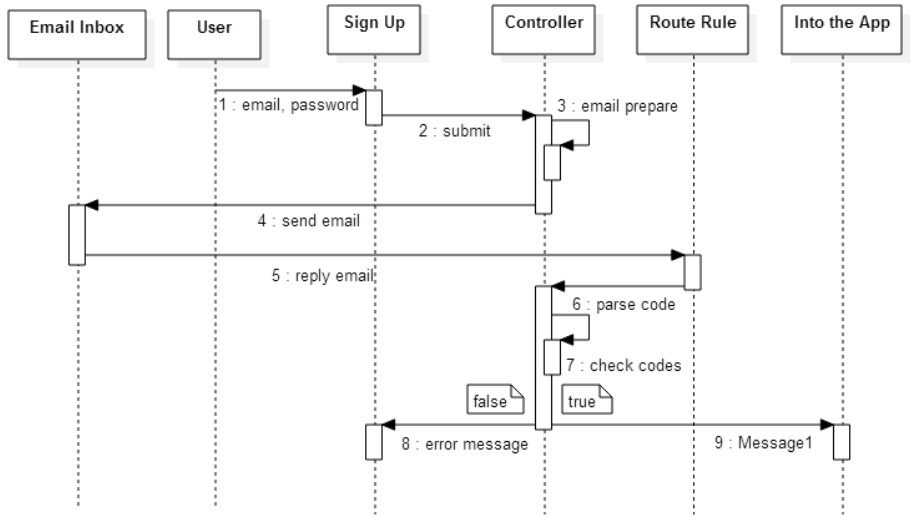
σύντμησης γραμμάτων. Για παράδειγμα η λέξη spiros μετά από την διαδικασία μετατροπής της σε metaphone γίνεται sprs. Ο λόγος που χρησιμοποιήθηκε ο συγκεκριμένος αλγόριθμος ήταν για να αποκτήσει μεγαλύτερη ευφυΐα η λειτουργία αναζήτηση βάση γραμματοσειράς. Ας εξηγήσαμε πως γίνεται αυτό. Όπως αναφέραμε ο αλγόριθμος για την λέξη spiros παράγει το metaphone sprs. Το ίδιο metaphone παράγεται και για τις λέξεις spuros ή sparos. Καταλαβαίνουμε λοιπόν ότι όταν κάποιος χρήστης θα μπει στην διαδικασία της αναζήτησης και για παράδειγμα δεν θυμάται ακριβώς το όνομα του χρήστη που ψάχνει spiros ή spuros η έξυπνη αναζήτηση θα του δώσει αποτελέσματα και τις δύο λέξεις. Επίσης σε περίπτωση αναγραμματισμού όπως για παράδειγμα sporos αντί για spiros πάλι η αναζήτηση θα έχει τα σωστά αποτελέσματα.

Όταν ένας νέος χρήστης επισκέπτεται την εφαρμογή συμπληρώνει τα στοιχεία του στην φόρμα εγγραφής. Αν έχει ξαναχρησιμοποιήσει το email στην εφαρμογή τότε το σύστημα επιστρέφει μήνυμα λάθους αλλιώς εισέρχεται. Παράλληλα το σύστημα καταγράφει στην βάση δεδομένων τα στοιχεία που έδωσε ο χρήστης. Τα στοιχεία που καταχωρούνται είναι το email, το password και το username το οποίο δημιουργείται αυτόματα από το σύστημα κρατώντας τη γραμματοσειρά μπροστά από το @ σύμβολο από το email που έδωσε ο χρήστης. Αυτό συμβαίνει για να απλουστέψουμε την διαδικασία εγγραφής και να γίνει όσο το δυνατόν πιο φιλική προς το χρήστη. Τέλος δημιουργείται και το metaphone του username και καταχωρείται στην βάση με το πρόθεμα “user:”. Το πρόθεμα αυτό προστίθεται για να κατηγοριοποιηθούν τα κλειδιά στα οποία αποθηκεύονται τα metaphone και να μπορούν εύκολα να διαχειριστούν από το command line interface της REDIS. Το σύστημα στην συνέχεια πρέπει να εξασφαλίσει την μοναδικότητα του χρήστη στην εφαρμογή και για όσο αυτός είναι ενεργός. Δημιουργεί ένα πενταψήφιο τυχαίο αριθμό τον οποίο και σε πρώτη φάση κωδικοποιεί με την μέθοδο κωδικοποίησης md5 της P.H.P. Μετά κωδικοποιεί τον αριθμό με ένα πανίσχυρο εργαλείο που προσφέρει το laravel. Είναι μια μέθοδος η οποία υλοποιεί μαθηματικούς αλγόριθμους οι οποίοι δίνουν την δυνατότητα κάθε φορά που εφαρμόζονται σε ένα αριθμό να τον παράγουν με διαφορετική κωδικοποίηση. Αυτός ο αριθμός καταχωρείται στα στοιχεία του χρήστη στην βάση και σαν cookie στον φυλλομετρητή του. Όσο ο χρήστης είναι ενεργός αυτοί οι δύο αριθμοί είναι ίδιοι και σηματοδοτούν την αυθεντικότητα του. Μαζί με το αριθμό στην βάση δημιουργείται και ένα κλειδί τύπου hash που έχει την μορφή key, field, value δηλαδή όνομα κλειδιού auth, πεδίο το κωδικοποιημένο πενταψήφιο αριθμό και τιμή το id του χρήστη. Εκεί προστίθενται όλοι οι χρήστες που είναι εντός της εφαρμογής και τον λόγο ύπαρξης θα τον αναφέρουμε στην παράγραφο που ακολουθεί. Προβάλουμε το διάγραμμα αλληλουχίας των ενεργειών που αναφέραμε με βάση το χρόνο.



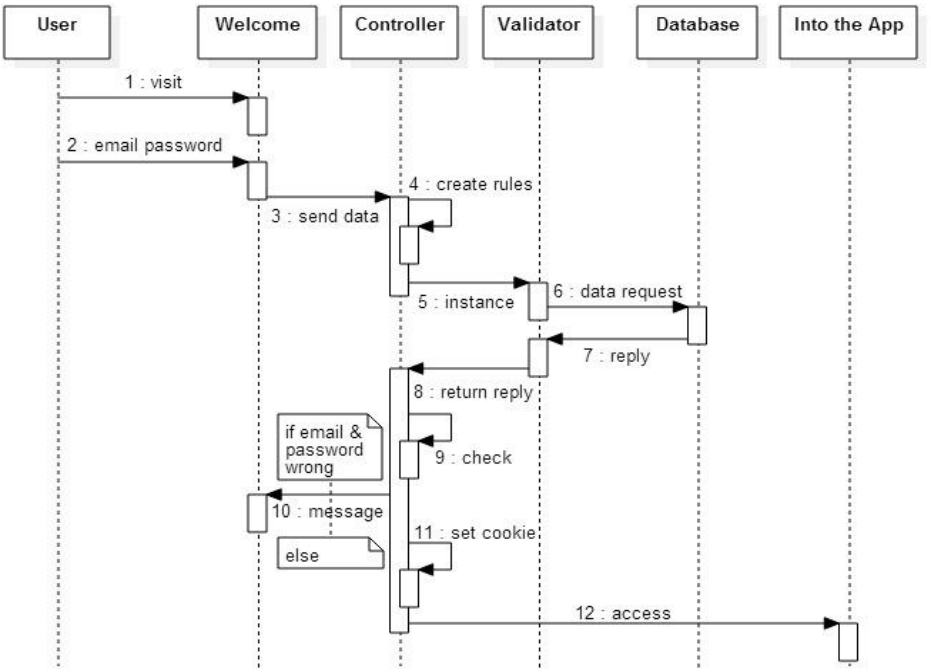
Εικόνα 4.4. Εγγραφή χρήστη.

Το σύστημα διαθέτει λειτουργία επιβεβαίωσης της διεύθυνσης ηλεκτρονικού ταχυδρομείου (email) που έδωσε ο χρήστης κατά την εγγραφή του στην εφαρμογή η αλλιώς την ενεργοποίηση του λογαριασμού του. Η λειτουργικότητα αυτή τοποθετήθηκε στον User Controller. Δημιουργήθηκε ένα route rule το οποίο είναι υπεύθυνο για την λήψη της απάντησης του χρήστη στο μήνυμα επιβεβαίωσης, η μέθοδος confirm που θα καλέσει το route rule, και ένα αρχείο html. Κατά την εγγραφή του χρήστη μαζί με τα στοιχεία του στη βάση αποθηκεύεται ο αριθμός verified, ένας κωδικοποιημένος αριθμός ο verifiedcode ένα στιγμιότυπο χρονοσήμανσης της στιγμής της εγγραφής και ένα κλειδί τύπου hash με όνομα κλειδιού το «verifyCode» πεδίο το κωδικοποιημένο αριθμό και τιμή το id του χρήστη. Ο αριθμός verified παίρνει την τιμή μηδέν ή ένα και σηματοδοτεί τον αν ο χρήστης έχει ενεργοποιήσει το λογαριασμό του την χρήση. Για τα υπόλοιπα δεδομένα θα μιλήσουμε στην συνέχεια της παραγράφου ενώ για το στιγμιότυπο χρονοσήμανσης στην παράγραφο που ακολουθεί. Στην συνέχεια το σύστημα αποστέλλει μήνυμα στο email του χρήστη με περιεχόμενο το html αρχείο το οποίο φέρει ένα σύνδεσμο που περιέχει προορισμό την διεύθυνση του route rule και παράμετρο τον αριθμό verifiedcode. Όταν ο χρήστης πατήσει το σύνδεσμο μεταφέρεται στον ιστότοπο της εφαρμογής και πιο συγκεκριμένα στο route rule το οποίο λαμβάνει το http request μαζί με την παράμετρο την οποία και περνάει στην μέθοδο confirm. Η μέθοδος ελέγχει στο hash verifyCode αν υπάρχει πεδίο ίδιο με το κωδικοποιημένο αριθμό. Αν δεν βρεθεί κάτι το σύστημα ανακατευθύνει το χρήστη στην αρχική σελίδα της εφαρμογής αλλιώς ανακτάτε το id του χρήστη. Στη συνέχεια αλλάζει η τιμή του πεδίου verified από μηδέν σε ένα, σβήνει το στιγμιότυπο χρονοσήμανσης και σβήνει το ανάλογο πεδίο από το κλειδί verifyCode. Ο χρήστης πλέον έχει ενεργοποιήσει το λογαριασμό του.



Εικόνα 4.4. Εισαγωγή χρήστη.

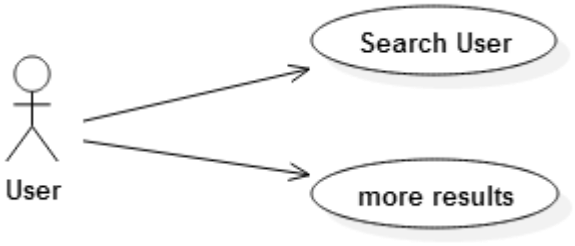
Όταν ο χρήστης είναι ήδη εγγεγραμμένος απλά συμπληρώνει το email και το password που επέλεξε κατά την εγγραφή του, πραγματοποιείτε η διαδικασία δημιουργίας cookie και εισέρχεται στην εφαρμογή.



Εικόνα 4.4. Εισαγωγή χρήστη.

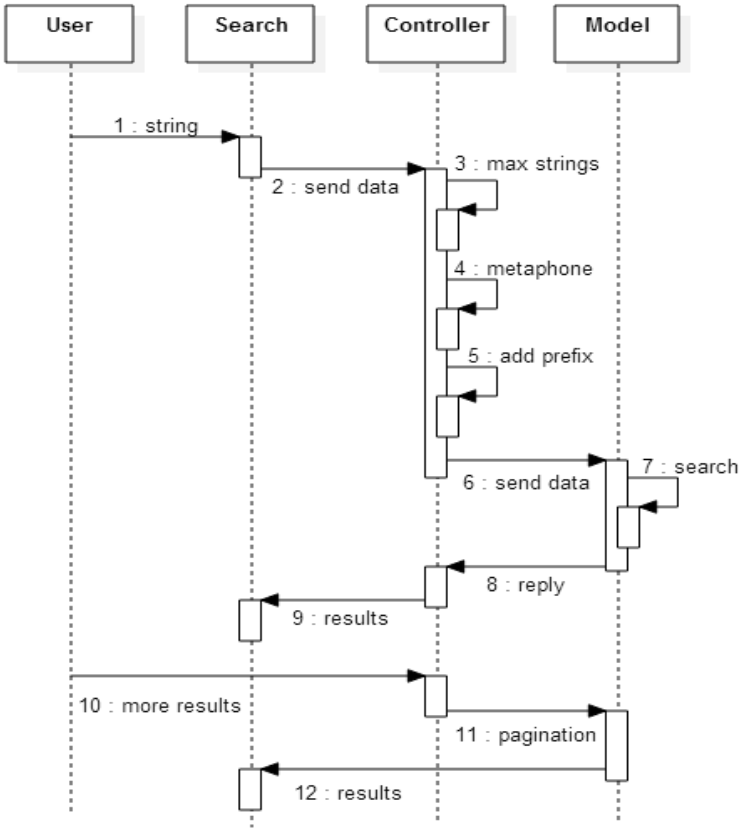
Μετά την διαδικασία εγγραφής χρήστη και χωρίς την ενεργοποίησης του λογαριασμού του μπορεί να κάνει περιορισμένη χρήση της εφαρμογής και αυτό για τις πρώτες εικοσιτέσσερις ώρες. Έτσι κάθε φορά που θα εισέλθει στην εφαρμογή το σύστημα πρέπει να γνωρίζει αν ο χρήστης έχει ενεργοποιήσει το λογαριασμό του. Επίσης σε κάθε ενέργεια του χρήστη πρέπει να εξακριβώνεται η αυθεντικότητα του. Αυτοί οι δύο έλεγχοι τοποθετήθηκαν στη μέθοδο isSignedIn στον BaseController η οποία και επιστρέφει true αν όλα βάνουν καλώς αλλιώς false όπου και το σύστημα ανακατευθύνει τον χρήστη στην αρχική σελίδα της εφαρμογής. Ο λόγος ήταν γιατί όπως αναφέραμε όλα τα route rule καλούν μεθόδους Controller οι οποίοι με την σειρά τους κάνουν extend τον Base κληρονομώντας τα χαρακτηριστικά του. Έτσι στο κατασκευαστή του εκάστοτε Controller καλείται η μέθοδος isSignedIn του Base κάθε φορά που ένα route rule καλεί μία μέθοδο. Πιο αναλυτικά όταν καλείται η μέθοδος isSignedIn το σύστημα ελέγχει αν υπάρχει το cookie στο φυλλομετρητή και στην συνέχεια το χρησιμοποιεί για να ανακτήσει το id του χρήστη με βάση το ανάλογο πεδίο στο hash auth. Αν δεν βρεθεί πεδίο ίδιο με την τιμή του cookie η μέθοδος επιστρέφει την τιμή false αλλιώς ανακτάτε το id του χρήστη και με την σειρά του τα στοιχεία του όπως η τιμή του πεδίου verified και το στιγμιότυπο χρονοσήμανσης. Αν ο χρήστης δεν έχει ενεργοποιήσει τον λογαριασμό του τότε το σύστημα παίρνει στιγμιότυπο χρονοσήμανσης εκείνη την στιγμή και με βάση αυτό από τη βάση υπολογίζει τον χρόνο που έχει περάσει από την εγγραφή του. Αν είναι μεγαλύτερος από εικοσιτέσσερις ώρες επιστρέφει την τιμή false αλλιώς συγκρίνει το cookie με τον αντίστοιχο αριθμό από τα στοιχεία του χρήστη στη βάση. Αν δεν είναι ίδιοι τότε η isSignedIn επιστρέφει false αλλιώς ανακτούνται τα στοιχεία του χρήστη και πραγματοποιείται η διαδικασία αυθεντικότητας.

Η εφαρμογή δίνει την δυνατότητα αναζήτησης χρήστη με βάσει το username, το όνομα, το επώνυμο. Εφόσον βρεθούν αποτελέσματα και ο αριθμός τους είναι μεγαλύτερος από τον προκαθορισμένο για την εμφάνιση εντός της σελίδας η εμφάνιση τους γίνεται με την μέθοδο της σελιδοποίησης.



Εικόνα 4.5. Εύρεση χρήστη.

Ο χρήστης δεν έχει παρά να συμπληρώσει το πεδίο αναζήτησης και να πατήσει το κουμπί εκτέλεσης. Το σύστημα λαμβάνει την γραμματοσειρά από το πεδίο αναζήτησης και ελέγχει πόσες λέξεις έχει δώσει ο χρήστης. Έχει οριστεί να πραγματοποιείται αναζήτηση με μέγιστο πλήθος δύο λέξεων. Οι επιπλέων λέξεις απλά δεν λαμβάνονται υπόψη. Στην συνέχεια παράγεται το αντίστοιχο metaphone και τοποθετείτε το πρόθεμα “user:” μπροστά από κάθε λέξη για να πάρουν μορφή ίδια με αυτήν που δημιουργείται όταν καταχωρούνται στην βάση τα στοιχεία του χρήστη όπως username, όνομα, επώνυμο για να μπορεί να γίνει η σύγκριση των metaphones.



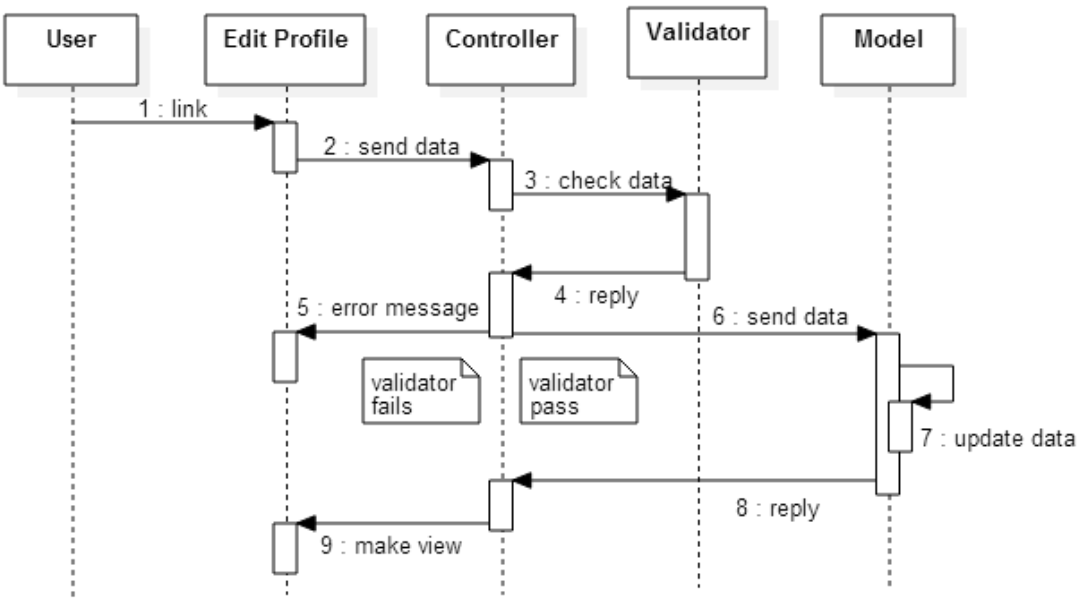
Εικόνα 4.6. Εύρεση χρήστη.

Τέλος εκτελείται στην βάση μια εντολή αναζήτησης, ένωσης, και αποθήκευσης των αποτελεσμάτων (zunionstore). Αν υπάρξουν αποτελέσματα τότε εμφανίζονται στον χρήστη σε διαφορετική περίπτωση λαμβάνει μήνυμα αδυναμίας εύρεσης. Λόγω του ότι τα αποτελέσματα ανά αναζήτηση μπορεί να είναι αρκετά και κάτι τέτοιο θα έκανε την λειτουργία μας να καθυστερήσει για αυτό το λόγο χρησιμοποιούμε την μέθοδο της σελιδοποίησης(pagination). Μας δίνει την δυνατότητα ανάκτησης μικρού όγκου δεδομένων από τη βάση κάθε φορά που θα ζητηθεί από το χρήστη. Για παράδειγμα ο χρήστης εκτελεί την ενέργεια αναζήτησης και αποστέλλει το αίτημα του στον user controller. Εκεί γίνεται το φιλτράρισμα που αναφέραμε στην προηγούμενη παράγραφο και στέλνονται τα δεδομένα στο User μοντέλο. Το μοντέλο εκτελεί τη εντολή zunionstore, τα αποτελέσματα αποθηκεύονται σε ένα προσωρινή κλειδί - λίστα στην βάση και με ανάλογη εντολή ανακτά πέντε εγγραφές. Αν ο αριθμός στο πρώτο αίτημα είναι μικρότερος από πέντε τότε οι εγγραφές απλά επιστρέφονται και η διαδικασία τερματίζεται. Σε διαφορετική περίπτωση επιστρέφονται τα πρώτα αποτελέσματα μαζί με τον αριθμό που σταμάτησε ο δείκτης κατά την ανάκτηση των αποτελεσμάτων από την λίστα. Έτσι το σύστημα ξέρει από που να ξεκινήσει την νέα ανάκτηση αν ο χρήστης αιτηθεί και άλλα αποτελέσματα. Η διαδικασία τελειώνει όταν τα τελευταία αποτελέσματα εγγραφών είναι λιγότερα από πέντε.

Σε κάθε σημείο της εφαρμογή που εμφανίζεται όνομα χρήστη υπάρχει η δυνατότητα επιλογής του. Το κάθε όνομα είναι σύνδεσμος που οδηγεί στην σελίδα προφίλ χρήστη. Για τις λειτουργίες που έχουν να κάνουν με τα προφίλ των χρηστών είναι υπεύθυνος ο profile controller ενώ για τις ενέργειες της βάσης απευθύνεται στο User μοντέλο.

Κάθε φορά που το σύστημα εμφανίζει μια σελίδα στο φυλλομετρητή και υπάρχει κάποιος χρήστης, στον κώδικα του συνδέσμου τοποθετείται και το ανάλογο id. Ο σύνδεσμος οδηγεί στο route rule και αυτό με την σειρά του στην μέθοδο εμφάνισης προφίλ χρήστη του profile controller. Εκεί γίνεται έλεγχος αν το εισερχόμενο id είναι ίδιο με αυτό του χρήστη που έκανε την ζήτηση εμφάνισης προφίλ και εφόσον διαχωριστεί καλείται το μοντέλο. Στο μοντέλο υπάρχει μία μέθοδος που χρησιμοποιείται για την ανάκτηση δεδομένων χρήστη. Δέχεται το id και επιστρέφει τα δεδομένα του ανάλογου χρήστη. Τα δεδομένα αυτά επιστρέφουν στον controller και αυτός με την σειρά του παράγει την σελίδα προφίλ.

Στα δεδομένα που στέλνει ο controller profile στα views για να εμφανιστούν τα αποτελέσματα του προφίλ υπάρχει και μία μεταβλητή η sameuser που υποδηλώνει αν αυτό το προφίλ που θα εμφανιστεί είναι του χρήστη που έκανε την ζήτηση ή οποιουδήποτε άλλου και ενημερώνεται μετά την σύγκριση των id που αναφέραμε. Αυτή η μεταβλητή χρησιμοποιείται για να δώσει την πληροφορία κατόχου προφίλ και κατά συνέπεια να εμφανιστεί η επιλογή διαχείρισης προφίλ (edit). Έτσι κάθε χρήστης έχει την δυνατότητα να διαχειριστεί το προφίλ πατώντας την επιλογή edit που παρουσιάζεται δίπλα στο όνομα του στην σελίδα του προφίλ του. Τα πεδία που μπορεί να διαχειριστεί είναι το όνομα, το επώνυμο, η ημερομηνία γέννησης, το φύλλο και το password. Συμπληρώνει τα πεδία και πατάει αποθήκευση. Τα δεδομένα πάνε στον profile controller στην μέθοδο διαχείρισης. Ο έλεγχος που γίνεται εκεί είναι για το αν έχει συμπληρωθεί κάποιο από τα πεδία του password, παλιό, νέο και επιβεβαίωση. Αν έχει συμβεί τότε καλείται ο validator και ελέγχει τα πεδία πιο συγκεκριμένα. Πρώτα πρέπει να είναι όλα τα πεδία συμπληρωμένα. Μετά το νέο password θα πρέπει να είναι τουλάχιστον τέσσερεις χαρακτήρες, τα πεδία νέο password και η επιβεβαίωση του να είναι ίδια και τέλος το παρόν παλιό πρέπει να είναι ίδιο με αυτό που είναι καταχωρημένο στη βάση δεδομένων. Αν κάποιος κανόνας δεν εκπληρωθεί τότε επιστρέφεται το ανάλογο μήνυμα λάθους. Σε διαφορετική περίπτωση τα δεδομένα πάνε στο μοντέλο. Εκεί το μοντέλο ανακτά τα παλαιά δεδομένα του χρήστη και ξεκινάει να ελέγχει αν υπάρχουν αλλαγές. Τα πεδία που ελέγχει είναι το όνομα, το επώνυμο και τέλος το username. Μετά το πρώτο έλεγχο, ελέγχει αν τα παλιά πεδία είναι κενά. Αυτό γίνεται γιατί ακολουθεί η μετατροπή των πεδίων σε metaphone όπου η μέθοδος επιστρέφει λάθος αν λάβει κενό όρισμα. Η μετατροπή γίνεται γιατί κατά την αλλαγή των πεδίων πρέπει να σβηστούν τα παλιά metaphone από την λίστα και να προστεθούν τα νέα. Αφού τελειώσει η αλλαγή των πεδίων username, όνομα, επώνυμο το σύστημα προχωράει στην αλλαγή των πεδίων ημερομηνία γέννησης, φύλλο και password. Αφού πραγματοποιηθούν οι αλλαγές επιστρέφονται οι νέες τιμές οι οποίες εμφανίζονται στη σελίδα προφίλ του χρήστη.



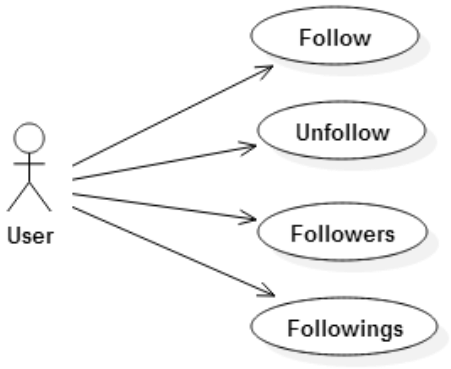
Εικόνα 4.6. Διαχείριση προφίλ.

Τέλος σε αυτό το κομμάτι ανήκει και η λειτουργικότητα της αποσύνδεσης του χρήστη από την εφαρμογή. Η επιλογή αποσύνδεσης αντιστοιχεί σε route rule το οποίο καλεί την συνάρτηση

αποσύνδεσης του User Controller. Κατά την είσοδο του αναφέραμε ότι το σύστημα δημιουργεί τον αριθμό αυθεντικότητας auth τον οποίο χρησιμοποιεί για να ορίσει την αυθεντικότητα του. Τον αποθηκεύει στην βάση σε ένα πεδίο auth στα στοιχεία του χρήστη και σε ένα κλειδί τύπου hash με όνομα κλειδιού «auth», πεδίο τον αριθμό αυθεντικότητας και τιμή το id του χρήστη. Ουσιαστικά αυτό που ενδιαφέρει το σύστημα για να γνωρίζει την μοναδικότητα του χρήστη είναι ο αριθμός στα στοιχεία του τον οποίο και συγκρίνει με το cookie στο φυλλομετρητή κάθε φορά που θέλει να κάνει τον έλεγχο. Κατά την αποσύνδεση του από την εφαρμογή το σύστημα διαγράφει τον αριθμό από τα στοιχεία του και το ανάλογο πεδίο από το hash auth αφού η πληροφορία είναι περιττή από την στιγμή που ο χρήστης δεν είναι πια ενεργός.

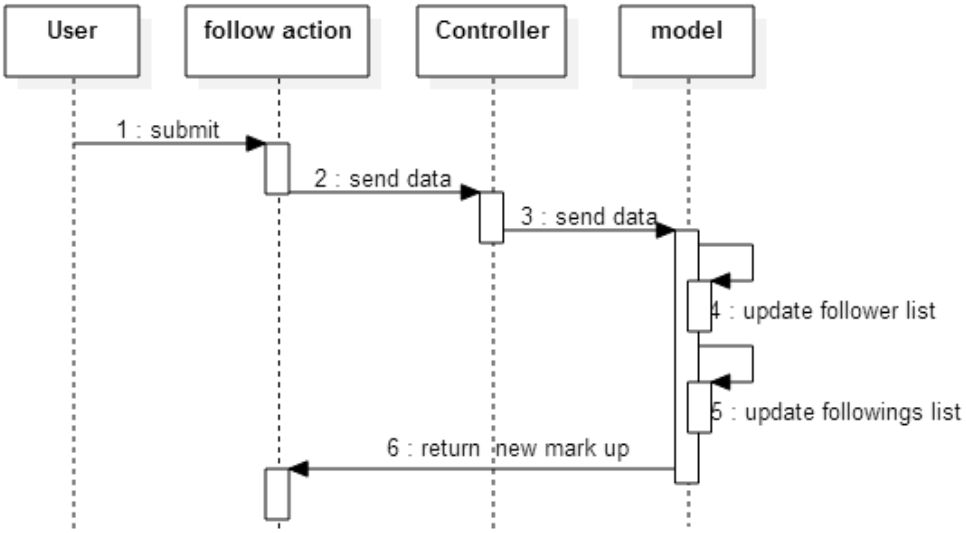
Follow

Είναι όλες οι λειτουργίες που έχουν να κάνουν με τις σχέσεις μεταξύ χρηστών(follow relations). Σε ακολουθώ (follow), σταματάω να σε ακολουθώ (unfollow), παρουσίαση αυτών που με ακολουθούν (followers), παρουσίαση αυτών που ακολουθώ (followings) και η σελιδοποίηση των αποτελεσμάτων παρουσίασης. Ο ελεγκτής που είναι υπεύθυνος για τις συγκεκριμένες λειτουργίες είναι ο FollowerGraphController και FollowGraph το ανάλογο μοντέλο.

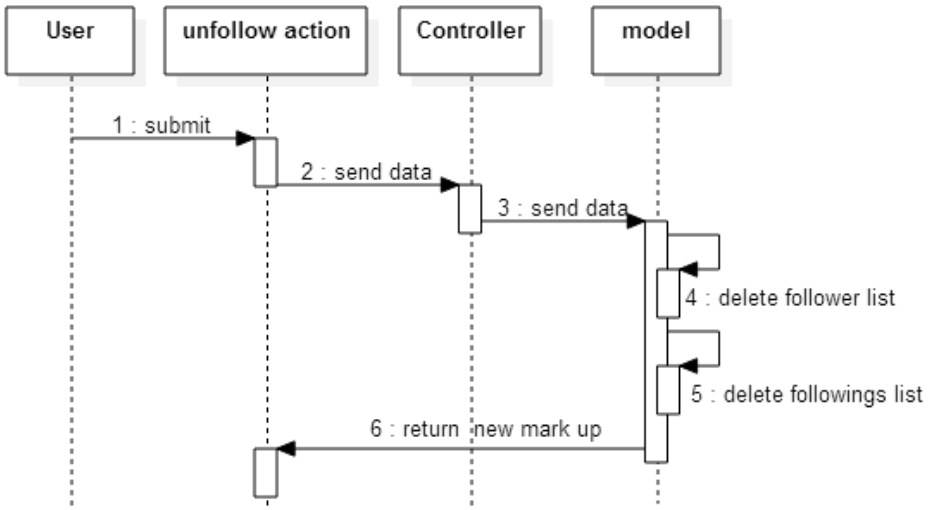


Εικόνα 4.7. Ενέργειες follow.

Κάθε φορά που ένας χρήστης Β εμφανίζεται στην οθόνη του χρήστη Α που πλοηγείται στην εφαρμογή μαζί με το όνομα του φανερώνεται και η σχέση (follow relation) που έχει μαζί του. Εικαστικά υπάρχει ένα κουμπί το οποίο την αναγράφει. Για παράδειγμα αν ο χρήστης Α συναντήσει τον χρήστη Β τον οποίο και ακολουθεί το κουμπί που θα συνοδεύει το χρήστη Β θα αναγράφει την λέξη Unfollow σε διαφορετική περίπτωση Follow. Έτσι δίνεται η δυνατότητα στους χρήστες να αναπτύσσουν ή και να τερματίζουν σχέσεις μεταξύ τους. Όταν ο χρήστης προβεί στην ενέργεια follow ουσιαστικά πραγματοποιεί την υποβολή μια φόρμας η οποία έχει στοιχεία το id του χρήστη που υποβάλει την ενέργεια, το id του χρήστη που δέχεται την ενέργεια και όλα αυτά σαν παράμετροι στο όνομα του route rule που θα καλέσει τον ελεγκτή και την ανάλογη μέθοδο. Έτσι το σύστημα έχει όλα τα στοιχεία που χρειάζεται για να πραγματοποιήσει την ζήτηση. Για το χρήστη που υποβάλει προσθέτει στην λίστα με τους followings του το χρήστη που δέχεται ενώ για τον χρήστη που δέχεται προσθέτει στην λίστα με τους followers του το χρήστη που υποβάλει. Έτσι κατοχυρώνεται η σχέση follow. Το ίδιο αλλά με την διαδικασία της διαγραφής από τις λίστες που προαναφέραμε πραγματοποιείται η ενέργεια unfollow. Αυτά τα δεδομένα χρησιμοποιούνται κάθε φορά που εμφανίζεται κάποιος χρήστης με την ανάλογη σχέση follow relation.



Εικόνα 4.8. Ενέργεια follow.

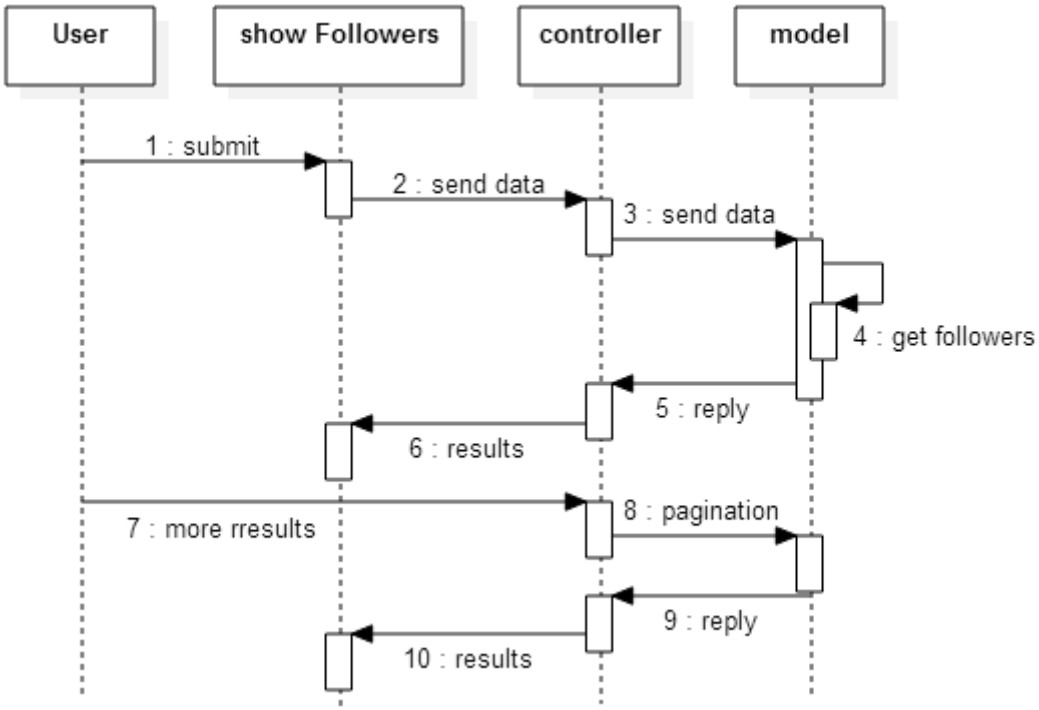


Εικόνα 4.9. Ενέργεια unfollow.

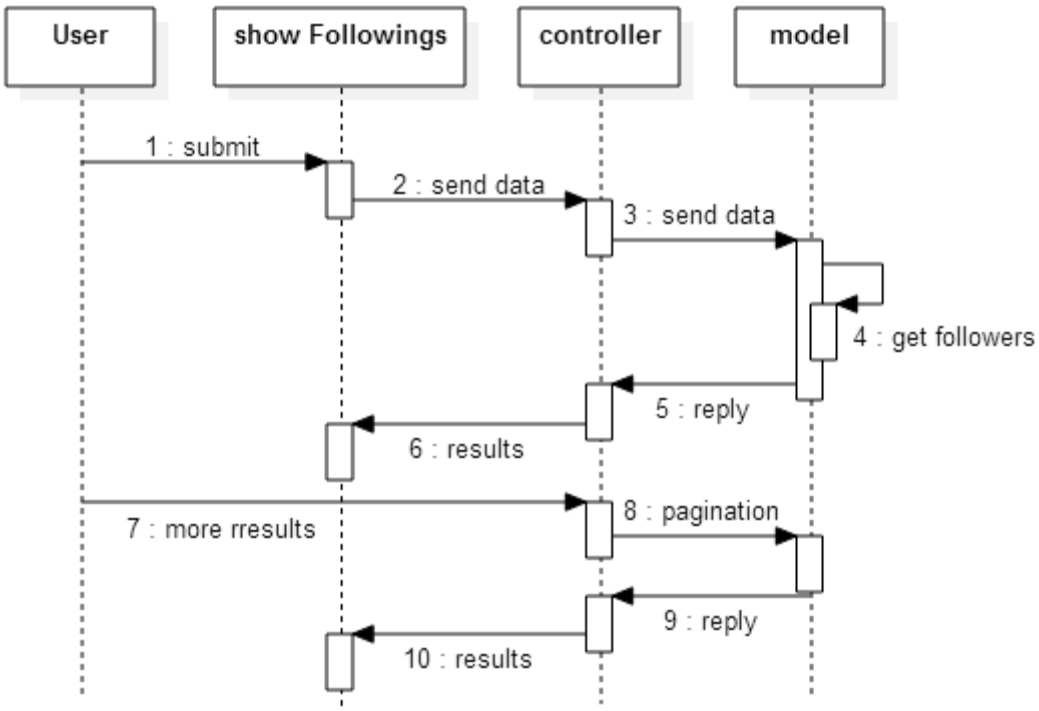
Σαν επιλογή υπάρχει και αυτή της εμφάνισης των ακολούθων (followers). Έχει την δυνατότητα να δει όσο τους δικούς τόσο και κάποιου άλλου χρήστη. Η φιλοσοφία είναι ίδια με αυτή της εμφάνισης προφίλ. Η επιλογή followers είναι ένας σύνδεσμος με το όνομα του route rule και χωρίς παράμετρο αν πρόκειται για τον ίδιο τον χρήστη ή με παράμετρο id αν πρόκειται για οπουδήποτε άλλο. Ο χρήστης εκτελεί την ενέργεια και καλείται η μέθοδος του ελεγκτή. Εκεί υπάρχουν δύο μεταβλητές. Στην περίπτωση που ο χρήστης αιτηθεί να δει τους ακόλουθους του όπως αναφέραμε ο σύνδεσμος δεν περιέχει id και ο controller τοποθετεί και στις δύο μεταβλητές το id του το οποίο και ανακτά από τα προσωπικά του δεδομένα. Σε διαφορετική περίπτωση στην μία μεταβλητή τοποθετείται το id του και στην άλλη το id που έρχεται από τον σύνδεσμο. Το μοντέλο με την σειρά του χρησιμοποιεί τα δεδομένα του έρχονται από τον controller και ανακτά τους followers. Για την εμφάνιση του κάθε follower το μοντέλο πρέπει να φροντίσει να ενημερωθούν οι μεταβλητές isFollowing και sameuser. Η πρώτη δίνει την πληροφορία αν ο χρήστης που εκτελεί την ενέργεια ακολουθεί ή όχι τον χρήστη που θα εμφανιστεί σαν follower. Το σύστημα πρέπει να το γνωρίζει για να εμφανίσει και την ανάλογη σχέση – κουμπί δίπλα στο όνομα του κάθε follower. Η δεύτερη υποδηλώνει αν θα εμφανιστεί η σχέση – κουμπί στα αποτελέσματα των followers όπου στην περίπτωση που ο χρήσης που εκτελεί την ενέργεια ζητάει να δει του δικούς τους παίρνει την τιμή false και η σχέση – κουμπί εμφανίζεται πάντα αλλιώς υπάρχει περίπτωση στην λίστα των followers κάποιου άλλου χρήστη να υπάρχει και το όνομα του. Εκεί παίρνει την τιμή true γιατί θα ήταν λάθος να μπορώ να κάνω follow ή unfollow τον εαυτό μου.

Η ίδια διαδικασία συμβαίνει και για την ενέργεια εμφάνισε αυτούς που ακολουθώ(followings). Η διαφορά είναι στην ενημέρωση των μεταβλητών και πιο συγκεκριμένα μόνο στην isFollowing. Η μεταβλητή παίρνει πάντα την τιμή true και εικαστικά εμφανίζεται μόνο το κουμπί unfollow γιατί όλοι οι χρήστες προήλθαν από την λίστα των followings.

Όπως και με την εύρεση χρήστη έτσι και εδώ τα αποτελέσματα μπορεί να είναι αρκετά σε αριθμό. Για το λόγο αυτό εφαρμόσαμε την μέθοδο της σελιδοποίησης. Όταν ξεκινήσει η διαδικασία εμφάνισης για παράδειγμα των followers και καλείται το μοντέλο μαζί με τα id στέλνεται και μία μεταβλητή που υποδηλώνει τον δείκτη που θα χρησιμοποιήσει το σύστημα για την σελιδοποίηση. Στην πρώτη κλίση του μοντέλου στέλνει σαν δείκτη την τιμή μηδέν αφού ξεκινάει η ανάκτηση των followers από την αρχή. Στην τιμή του δείκτη εντός του μοντέλου το σύστημα προσθέτει τον αριθμό πέντε που αντιστοιχεί στο πλήθος των χρηστών που θέλουμε να εμφανίσουμε κατά την σελιδοποίηση. Έτσι έχουμε δύο μεταβλητές τη start και την semistop και το σύστημα γνωρίζει κάθε φορά πόσους και ποιους χρήστης θα εμφανίσει. Επίσης το σύστημα γνωρίζει με ανάλογη εντολή στην βάση τον αριθμό του συνόλου των followers. Η μεταβλητή που περιέχει τον αριθμό συγκρίνεται κάθε φορά με την τιμή semistop. Όταν οι τιμές γίνουν ίσες η διαδικασία της σελιδοποίησης σταματάει.



Εικόνα 4.10. Εμφάνιση followers.



Εικόνα 4.11. Εμφάνιση followings

Αποθηκευτικό Νέφος

Όπως αναφέραμε και στο τρίτο κεφάλαιο χρησιμοποιήθηκε και η τεχνολογία αποθήκευσης αρχείων σε αποθηκευτικό νέφος. Τα αχρεία που επιλέχθηκαν να αποθηκευτούν εκεί ήταν οι εικόνες χρηστών(AVATAR) και δημοσιεύσεων(RATES). Για την υλοποίηση αυτής της λειτουργικότητας δημιουργήθηκε μια καινούρια κλάση στις βιβλιοθήκες του laravel η StorageFactory. Χτίσθηκε βάση του σχεδιαστικού προτύπου(design pattern) factory.

Τα σχεδιαστικά πρότυπα είναι μέθοδοι υλοποίησης κώδικα με σκοπό την ελαχιστοποίηση της πολυπλοκότητας του και της επίλυσης κοινών προβλημάτων που παρουσιάζονται κατά την υλοποίηση εφαρμογών. Ας κάνουμε μια γρήγορη περιγραφή του σχεδιαστικού πρότυπου που επιλέχθηκε. Αποτελείται από ένα σύνολο κλάσεων οι οποίες υλοποιούν διαφορετικές λειτουργίες ανάλογα με τις ανάγκες της εφαρμογής. Επιπλέον υπάρχει και μια κλάση η factory μέσω της οποίας γίνεται επιλογή μιας εκ των συνόλου κλάσεων.

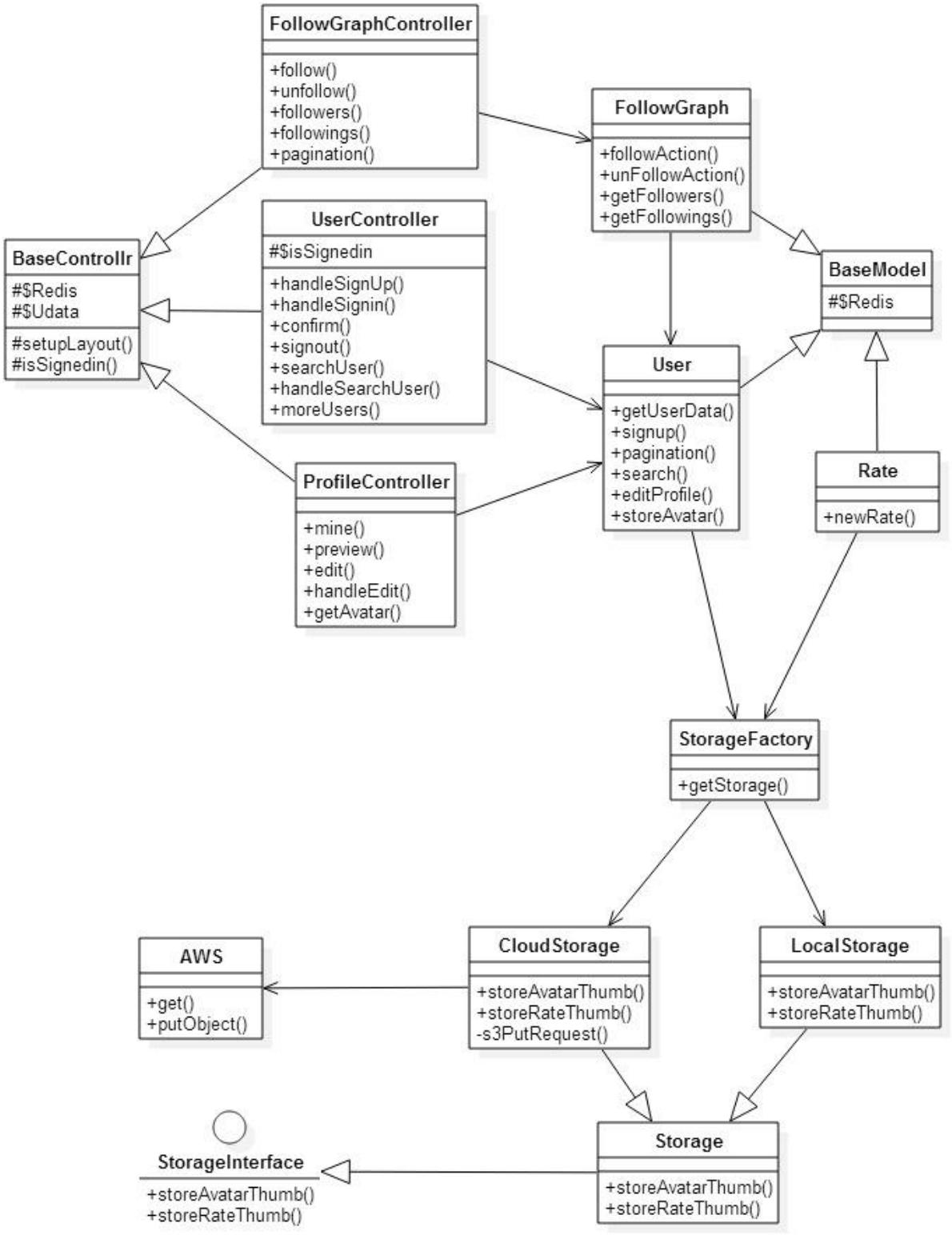
Στην περίπτωσή μας το σύνολο των κλάσεων αποτελείται από την κλάση που υλοποιεί την λειτουργία αποθήκευση εικόνων προφίλ και δημοσιεύσεων στον εξυπηρετητή και η κλάση που αποθηκεύει τα αρχεία εικόνων στο νέφος. Η επιλογή γίνεται με βάση μεταβλητής του συστήματος του laravel. Ο διαχειριστής του συστήματος αλλάζει τιμή ανάλογα με το που θέλει να αποθηκεύονται οι εικόνες. Οι επιλογές μας είναι στον server ή στο νέφος. Ο έλεγχος αυτός τοποθετείται στον κατασκευαστή της κλάσης factory. Υπάρχει μία συνθήκη που ελέγχει την τιμή της μεταβλητής και αρχικοποιεί την ανάλογη κλάση ανάλογα με την τιμή της μεταβλητής. Αυτόματα λοιπόν όταν αρχικοποιηθεί η κλάσης factory επιστρέφεται και το ανάλογο αντικείμενο. Μέσω αυτού του αντικειμένου καλούνται και οι μέθοδοι που μας ενδιαφέρουν. Η κλήση του αντικειμένου factory γίνεται είτε στον profile controller αν πρόκειται για αποθήκευση εικόνας χρήστη είτε στο rate μοντέλο αν πρόκειται για την αποθήκευση εικόνας RATE.

Αποθήκευση εικόνας χρήστη. Κατά την διαχείριση προφίλ χρήστη του δίνεται η δυνατότητα να καταχωρήσει στο σύστημα μια εικόνα που θα εμφανίζεται σαν εικόνα προφίλ. Επιλέγει μέσω ενός παραθύρου διαλόγου την εικόνα επιλογής του και αποστέλλει τα δεδομένα στον controller. Εκεί όπως και σε όλες τις φόρμες της εφαρμογής καλείται ο validator. Ο έλεγχος που εκτελείται είναι αν το αχρείο που επέλεξε ο χρήσης είναι εικόνα και αν το μέγεθος της είναι έως και 1MB. Όταν περάσει από τον έλεγχο του validator ο controller δημιουργεί ένα πίνακα μεταβλητών με το όνομα του αρχείου το είδος

του και την τοποθεσία που είναι προσωρινά τοποθετημένο το αρχείο όσο αυτό είναι υπό επεξεργασία. Τέλος καλείται η μέθοδος αποθήκευση του στην τοποθεσία που ορίζει ο διαχειριστής μέσω της μεταβλητής config item. Στην επιλογή αποθήκευσης στον εξυπηρετητή μεταφέρεται η εικόνα σε ανάλογη τοποθεσία και αποθηκεύεται η διαδρομή της τοποθεσίας στην βάση στα στοιχεία του χρήστη. Στην επιλογή αποθήκευσης στο νέφος καλείται την μέθοδο s3PutRequest η οποία δημιουργεί και επιστρέφει αντικείμενο της κλάσης Amazon Web Services που υπάρχει στο SDK του S3 το οποίο και εγκαταστάθηκε στην εφαρμογή μας. Μέσω αυτού του αντικειμένου καλείται η μέθοδος putObject που τοποθετεί στο νέφος το αρχείο και επιστρέφει την διεύθυνση που προβάλει το νέφος την εικόνα. Η διεύθυνση αποθηκεύεται στην βάση στα στοιχεία του χρήστη. Και στις δύο περιπτώσει λοιπόν το σύστημα γνωρίζει που βρίσκεται η εικόνα του χρήστη εφόσον υπάρχει και εμφανίζει την εικόνα.

Αποθήκευση εικόνας rate. Η διαδικασία ξεκινάει όταν δημιουργείται ένα νέο rate. ο Η εφαρμογή μας διαθέτει ένα εργαλείο το οποίο όταν ο χρήστης δώσει το URL του ιστότοπου για το οποίο θέλει να κάνει rate δημιουργεί εικόνα από την αρχική σελίδα αυτού και την αποθηκεύει. Αυτό συμβαίνει στον rate controller και από εκεί το εικονίδιο μαζί με όλα τα στοιχεία που συγκεντρώνονται για το νέο rate στέλνονται στο rate μοντέλο. Εκεί γίνεται έλεγχος αν το σύστημα έχει τραβήξει φωτογραφία του ισοτόπου του νέου rate. Αν όχι απλά αποθηκεύεται στα στοιχειά του χρήστη το μονοπάτι συγκεκριμένης εικόνας που αντιπροσωπεύει την αδυναμία δημιουργία εικόνας. Σε διαφορετική περίπτωση δημιουργείται αντικείμενο της κλάσης factory και καλείται η μέθοδος αποθήκευσης εικόνας. Αν το config item ορίσει την αποθήκευση αρχείου να γίνει στο server τότε το σύστημα αφήνει την εικόνα αποθηκευμένη εκεί που αποθηκευτικέ κατά την δημιουργία της και απλά καταχωρεί την τοποθεσία αυτή στη βάση στα στοιχεία του χρήστη. Στη περίπτωση που η εικόνα θα αποθηκευτεί στο νέφος το σύστημα ακολουθεί την ίδια διαδικασία με αυτή της εικόνας χρήστη στην ανάλογή περίπτωση καλείται η μέθοδός putObject και το url που επιστρέφει το αποθηκεύσει στα στοιχεία του χρήστη.

Κλείνοντας αυτό το κεφάλαιο παραθέτουμε το διάγραμμα κλάσεων και των σχέσεων που αναπτύσσονται μεταξύ αυτών. Όπως αναφέραμε στην αρχή οι κλάσεις controllers κάνουν επεκτείνουν (extend) τον BaseController. Από την συγκεκριμένη ενέργεια κληρονομούν τις ιδιότητες της και στην περίπτωση μας το αντικείμενο $Redis την σύνδεσης με την βάση δεδομένων, και την μέθοδο isSignedIn η οποία κάνει τον έλεγχο αυθεντικότητας του χρήστη. Επίσης οι controllers δημιουργούν αντικείμενα των μοντέλων και χρησιμοποιούν τις μεθόδους που απαιτούνται για την εκάστοτε λειτουργία. Οι κλάσεις μοντέλα με την σειρά τους κάνουν extend το BaseModel και κληρονομούν το αντικείμενο $Redis την σύνδεσης με την βάση δεδομένων. Οι κλάση StorageFactory και πιο συγκριμένα η στατική της μέθοδος getStorage καλείται από τα μοντέλα User και Rate και δημιουργεί ανάλογα με το config item αντικείμενο της κλάσης cloudStorage ή localStorage. Μετά την δημιουργία αντικειμένου τα μοντέλα χρησιμοποιούν τις αντίστοιχες μεθόδους για την αποθήκευση εικονιδίων. Η κλάση cloudStorage δημιουργεί αντικείμενο της κλάσης AWS και καλεί την στατική μέθοδο get για την σύνδεση της εφαρμογής με το νέφος και την μέθοδο putObject και την αποθήκευση εικονιδίου σε αυτό. Οι κλάσεις cloudStorage και localStorage κάνουν extend την Storage και αυτή με την σειρά της υλοποιεί το StorageInterface.



Εικόνα 4.12. Διάγραμμα κλάσεων εφαρμογής.

Κεφάλαιο 5ο

Συμπεράσματα

Αυτό το κεφάλαιο παρουσιάζει τα συμπεράσματα που βγήκαν κατά την μελέτη και την εκπόνηση της εργασίας. Πιο αναλυτικά θα γίνει αναφορά στις γνώσεις που λάβαμε κατά την διάρκεια της εργασίας, στις προοπτικές που έχει η εφαρμογή και στις βελτιώσεις που θα μπορούσαν να γίνουν.

Από τα πρώτα μας βήματα μάθαμε πώς γίνεται το project planning μιας εφαρμογής. Ουσιαστικά την ανάλυση των απαιτήσεων μιας εφαρμογής, την επιλογή των εργαλείων – τεχνολογιών, το καταμερισμό των προς υλοποίηση θεμάτων και την σειρά της υλοποίησης. Έτσι στην περίπτωση μας πρώτα έγινε η συγκέντρωση και η ανάλυση των απαιτήσεων. Εκεί ουσιαστικά καταγράψαμε το τι θέλουμε να κάνει η εφαρμογή μας, επιλέξαμε το τι εργαλεία θα χρησιμοποιήσουμε(Redis, Laravel), έγινε ο καταμερισμός εργασιών δηλαδή με τι θα ασχοληθεί ο κάθε φοιτητής και ξεκίνησε η υλοποίηση από την δημιουργία της βάσης δεδομένων ένα από το πιο σημαντικά κομμάτια στης εφαρμογής μας. Είναι πολύ σημαντικό να αναφέρουμε ότι παρόλο που πριν ξεκινήσουμε την υλοποίηση κάναμε την μελέτη για το data layout κατά στην διάρκεια την υλοποίησης της εφαρμογή τα θέματα υλοποίησης που προέκυπταν μας ανάγκασαν να αλλάξουμε την αρχική μας μελέτη ώστε να μπορέσουμε να συνεχίσουμε ομαλά. Αυτό δείχνει ότι σε μια τόσο πολύπλοκη και συνεργατική εφαρμογής ποτέ δεν μπορείς να προβλέψεις τα πάντα.

Στη συνέχεια γνωρίσαμε καινούριες για τα δεδομένα μας τεχνολογίες. Τα php framework και πιο συγκεκριμένα το laravel. Είναι πολύτιμα εργαλεία με έτοιμες βιβλιοθήκες και λειτουργικότητες που βοηθάνε στην υλοποίηση μεγάλων και πολύπλοκων εφαρμογών. Τα version control συστήματα που διαχειρίζονται αλλαγές αρχείων και δίνουν την δυνατότητα αποθήκευσης τους σε αποθηκευτικό νέφος. Μία ακόμα πολύ σημαντική τεχνολογία αφού δουλέψαμε σαν ομάδα και κάτι τέτοιο ήταν απαραίτητο αφού θα ήταν δύσκολο να ενώνονται κάθε φορά τα αρχεία και ο κώδικας που δημιουργούσε ο κάθε φοιτητής τοπικά στον υπολογιστή του. Το αποθηκευτικό νέφος, τεχνολογία που μας έλυσε τα χέρια όσον αφορά το κομμάτι του διαθέσιμου χώρου αποθήκευσης που θα χρειαζόμασταν να έχουμε για την εφαρμογή μας. Όπως αναφέραμε η προοπτική της εφαρμογής σαν social media ήταν να φιλοξενήσει μεγάλο όγκο δεδομένων και αρχείων. Αυτό παραπέμπει στην ανάγκη απόκτησης μεγάλου αποθηκευτικού χώρου, ασύμφορο για τα ακαδημαϊκά πλαίσια που κυμάνθηκε η εργασία μας. Το νέφος ουσιαστικά μας έδωσε την δυνατότητα να το πραγματοποιήσουμε και σε χαμηλό κόστος.

Σε επίπεδο software development μάθαμε και εφαρμόσαμε έννοιες όπως δομημένος κώδικας, βελτιστοποίηση κώδικα, γνωρίσαμε το αρχιτεκτονικό πρότυπο MVC και το σχεδιαστικό πρότυπο υλοποίησης κώδικα design pattern factory.

Δομημένος κώδικας είναι ο τρόπος με τον οποίο γράφουμε και ο οποίος κάνει τον κώδικα ευανάγνωστο και εύκολα κατανοητό.

Βελτιστοποίηση κώδικα είναι η διαδικασία κατά την οποία ο κώδικάς μας γίνεται πιο αποτελεσματικός πιο γρήγορος και δεσμεύοντας λιγότερη μνήμη.

Το αρχιτεκτονικό πρότυπο MVC χρησιμοποιείται κατά κόρων από τα php πλαίσιο εργασίας. Βάση αυτού ο κώδικας της εφαρμογής δομείτε και διαχωρίζεται με βάση την κύρια λειτουργία που εκτελεί.

Το σχεδιαστικό πρότυπο factory πρόσθεσε ευκολία και απλοποίηση στην υλοποίηση της λειτουργίας αποθήκευσης εικονιδίων.

Η εφαρμογή υλοποιήθηκε από τρεις φοιτητές οι οποίοι είχαμε τις βασικές προγραμματιστικές γνώσεις που όμως δεν ήταν αρκετές για την εκπόνηση μιας τέτοιας εργασίας. Οι απαραίτητες γνώσεις δεν είναι μόνο αυτές που αναφέραμε μέχρι τώρα σε αυτό το κεφάλαιο. Υπάρχουν και παράμετροι όπως η σωστή συνεργασία μιας ομάδας η οποία όπως αποδείχτηκε ήταν ένας σημαντικός παράγοντας για την υλοποίηση της εφαρμογής. Σωστή συνεργασία με βάση την ανταλλαγή απόψεων και γνώσεων και μετάδοσης της γνώσης. Για παράδειγμα όταν κάποιος φοιτητής υλοποιούσε κάτι το οποίο οι υπόλοιποι δεν γνώριζαν οι υπόλοιποι προσέφερε ανάλογο κείμενο επεξήγησης το οποίο έδινε την δυνατότητα στους λοιπούς να αποκτήσουν εύκολα και γρήγορα τις απαραίτητες γνώσεις. Θέματα προς υλοποίηση τέθηκαν υπό συζήτηση με σκοπό την βέλτιστη και πιο αποτελεσματική εκτέλεση τους. Έτσι αλληλοσυμπλήρωνε ο ένας τον άλλο με αποτέλεσμα να εξελισσόμαστε πιο γρήγορα ατομικά και συνολικά κατά την διάρκεια εκπόνησης της εφαρμογής. Εκτός από τη μετάδοση

γνώσης υπήρχαν πολλές φορές οι περιπτώσεις αντιπαραθέσεις για θέματα της εφαρμογής που όμως με την σωστή διαχείριση τους γινόντουσαν απλά θέματα προς συζήτηση.

Πολλές φορές οι εφαρμογές έχουν αδύνατα σημεία τα οποία είτε έχουν εντοπιστεί κατά την διαδικασία της υλοποίησης και απλά κατ επιλογή παρέμειναν έτσι είτε παρατηρήθηκαν κατά την χρήση τους. Εκτός από τα αδύνατα σημεία υπάρχουν και περιθώρια βελτίωσης τους. Ένα από αυτά ήταν η λειτουργία εύρεσης χρήστη. Το μειονέκτημα ήταν ουσιαστικά η αδυναμία της βάσης δεδομένων να διαχειριστεί συμβολοσειρές (strings). Αυτή η αδυναμία είχε σαν αποτέλεσμα ότι όταν κάποιος χρήστης εκτελούσε την ενέργεια αναζήτησης τα αποτελέσματα δεν ερχόντουσαν με την επιθυμητή σειρά αλλά με τυχαία και ίσως μετά από πολλά αποτελέσματα κάτι που έκανε της λειτουργία της αναζήτησης μη εξυπηρετική. Η λύση του προβλήματος ήταν η χρήση της σχεσιακής βάσης δεδομένων MySQL η οποία θα πρόσφερε την δυνατότητα αυτή. Τα προς υλοποίηση θέματα είναι ότι θα μπορούσε να γίνει πιο λεπτομερής η λειτουργικότητα ως προς το κομμάτι του follow relations. Για παράδειγμα θα μπορούσαν να προστεθούν φίλτρα τα οποία θα έδιναν την δυνατότητα παρουσίασης κοινών follower – followings όπως επίσης θα μπορούσε να δίνεται η δυνατότητα απόκρυψης αυτών. Επιπλέων η πρόσθεση λειτουργίας εύρεσης followers και followings υθα έκανε την εύρεση χρήστη πιο λειτουργική και ευέλικτη.

Βιβλιογραφία

redis.io (2009) *Tutorial: Design and implementation of a simple Twitter clone*. [Online] Available from: http://redis.io/topics/twitter-clone [Accessed: 19 June 2014].

Seguin K. (2012) The Little Redis Book. [Online] Available from: http://openmymind.net/redis.pdf [Accessed: 19 June 2014].

oldblog.antirez.com/ (2010) *Auto Complete with Redis.* [Online] Available from: http://oldblog.antirez.com/post/autocomplete-with-redis.html [Accessed: 28 July 2014].

Josiah L. Carlson (2013*) Redis in Action-Manning.* Shelter Island N.Y: Manning Publications Co.

Tiago Macedo and Fred Oliveira (2011) *Redis Cookbook*. O'Reilly MediaFinal

openmymind.net Karl Seguin (2012) *The Little Redis Book.* [Online] Available from: http://openmymind.net/2012/1/23/The-Little-Redis-Book/

badrit.com (2013) *Redis vs. MongoDB Performance* [Online] Available from: http://www.badrit.com/blog/2013/11/18/redis-vs-mongodb-performance#.VRPzZPysXuM [Accessed: 03 May 2015].

Laravel.com (2008) Laravel, official website documentation. [Online] Available from: http://laravel.com/docs/4.2 [Accessed: 15 June 2014].

Hardik Dangar (2013*) Learning Laravel 4 Application Development.* Birmingham U.K: Packt Publishing Ltd.

Ravishankar Somasundaram (2013) *Git: Version Control for Everyone Beginner's Guid.* Birmingham U.K: Packt Publishing Ltd.

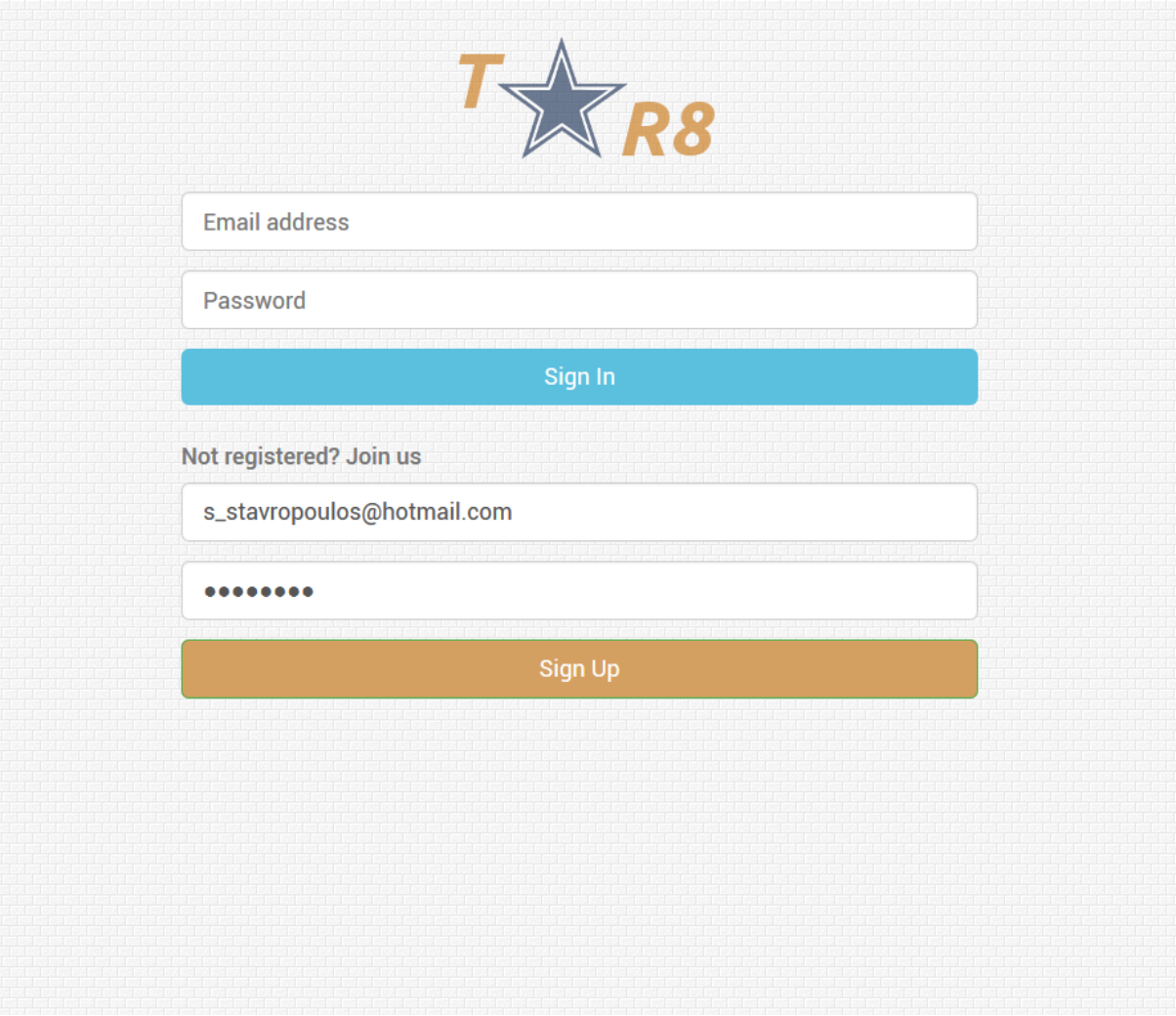
Borko Furht, Armando Escalante (2010) *Handbook of Cloud Computing* Springer. N.Y Springer Science+Business Media, LLC

Peter Mell, Timothy Grance. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication 800-145*. [Online] Available from: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf [Accessed: 15 May 2015].

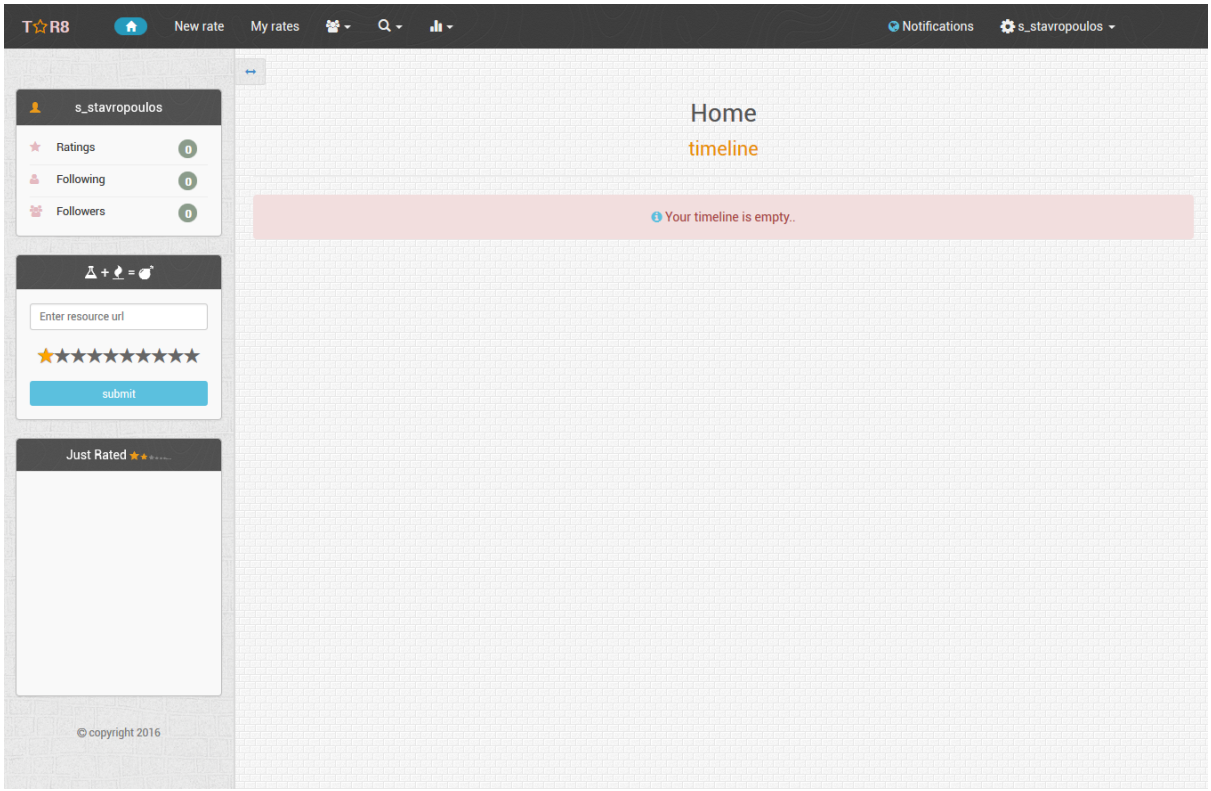
Β. Γερογιάννης, Β. Κακαρότζας, Α. Καμέας, Γ. Σταμέλος, Π. Φιτσιλής (2007) *Αντικειμενοστραφής Ανάπτυξη λογισμικού με την UML.* Αθήνα: Εκδόσεις Κλειδάριθμος.

Screenshots Εφαρμογής

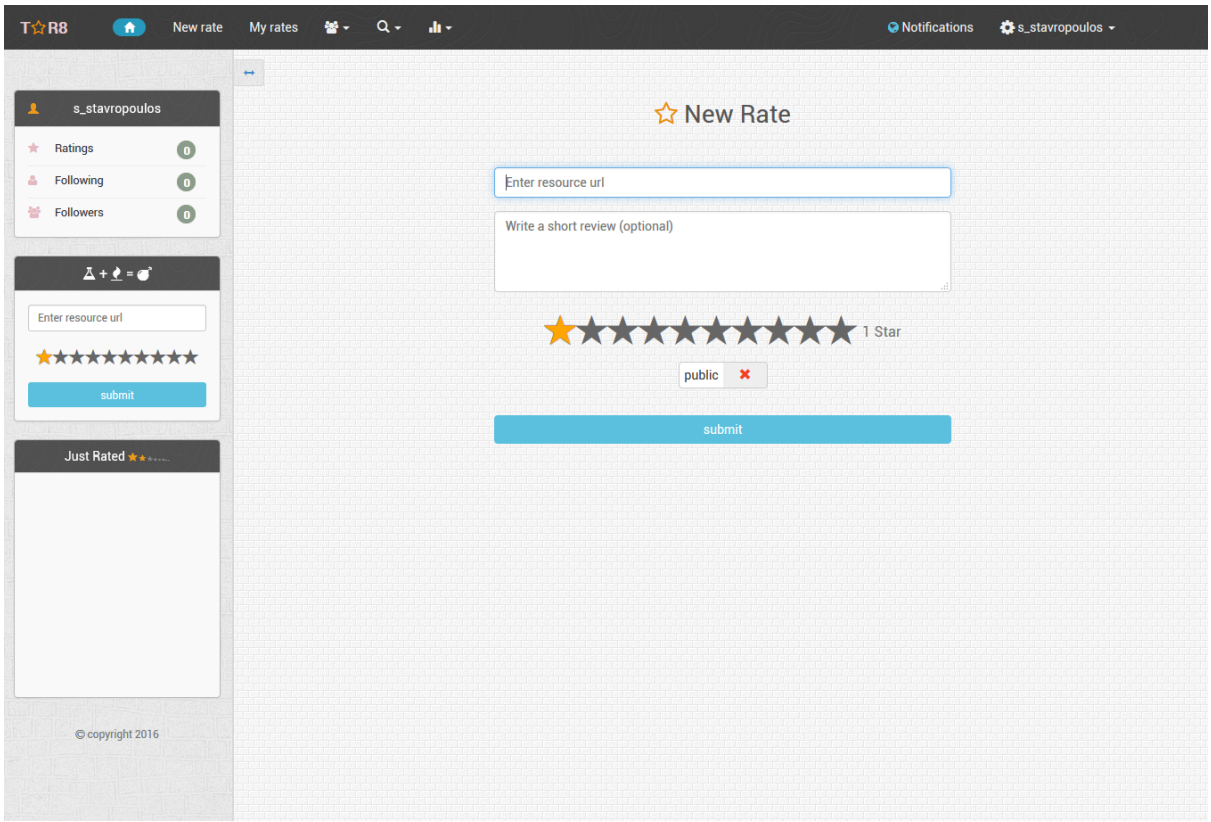
Sign Up – Sign in



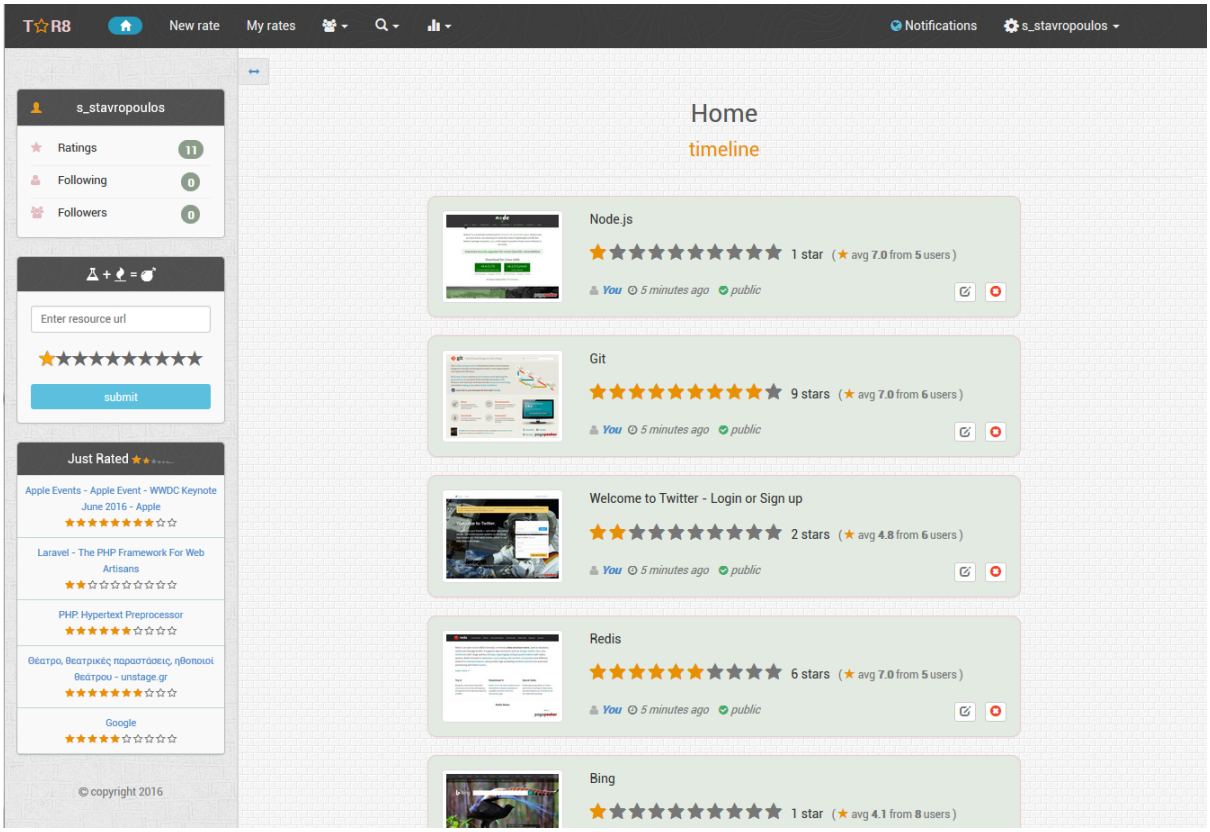
Empty Timeline



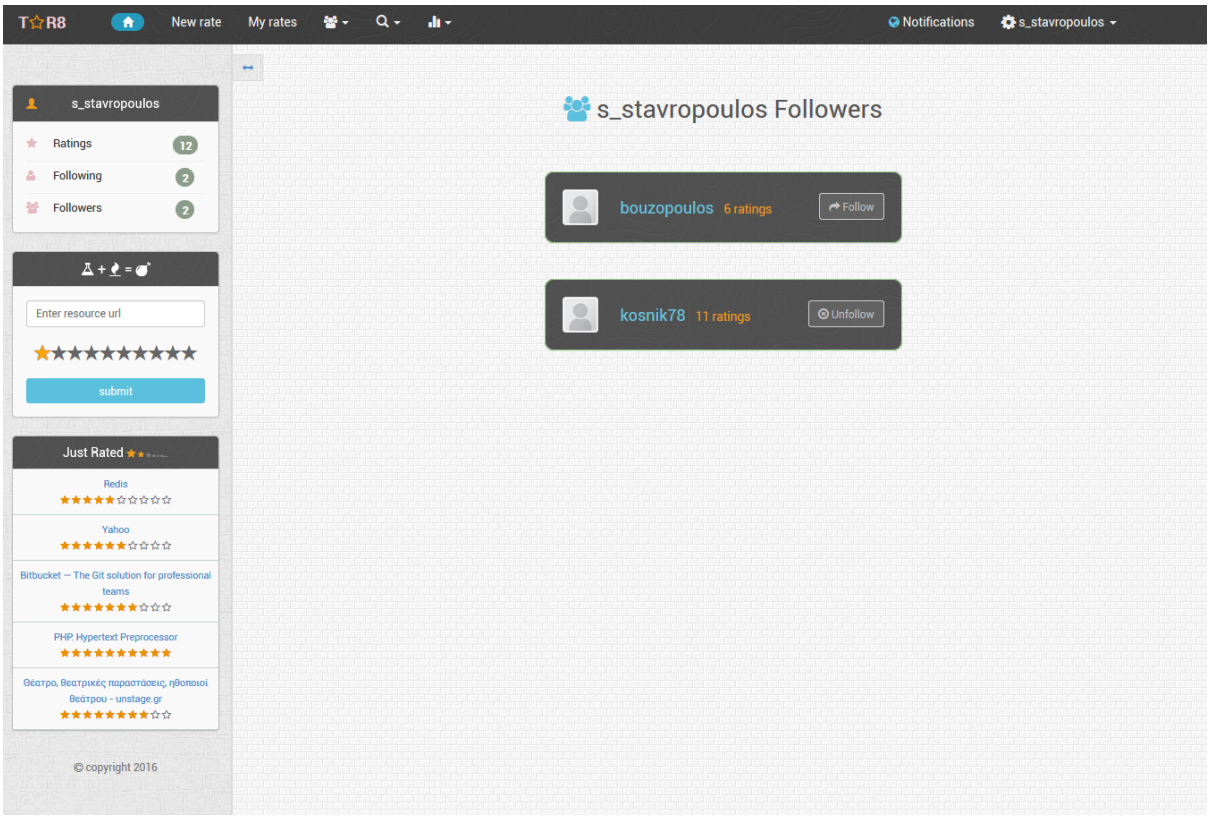
New Rate



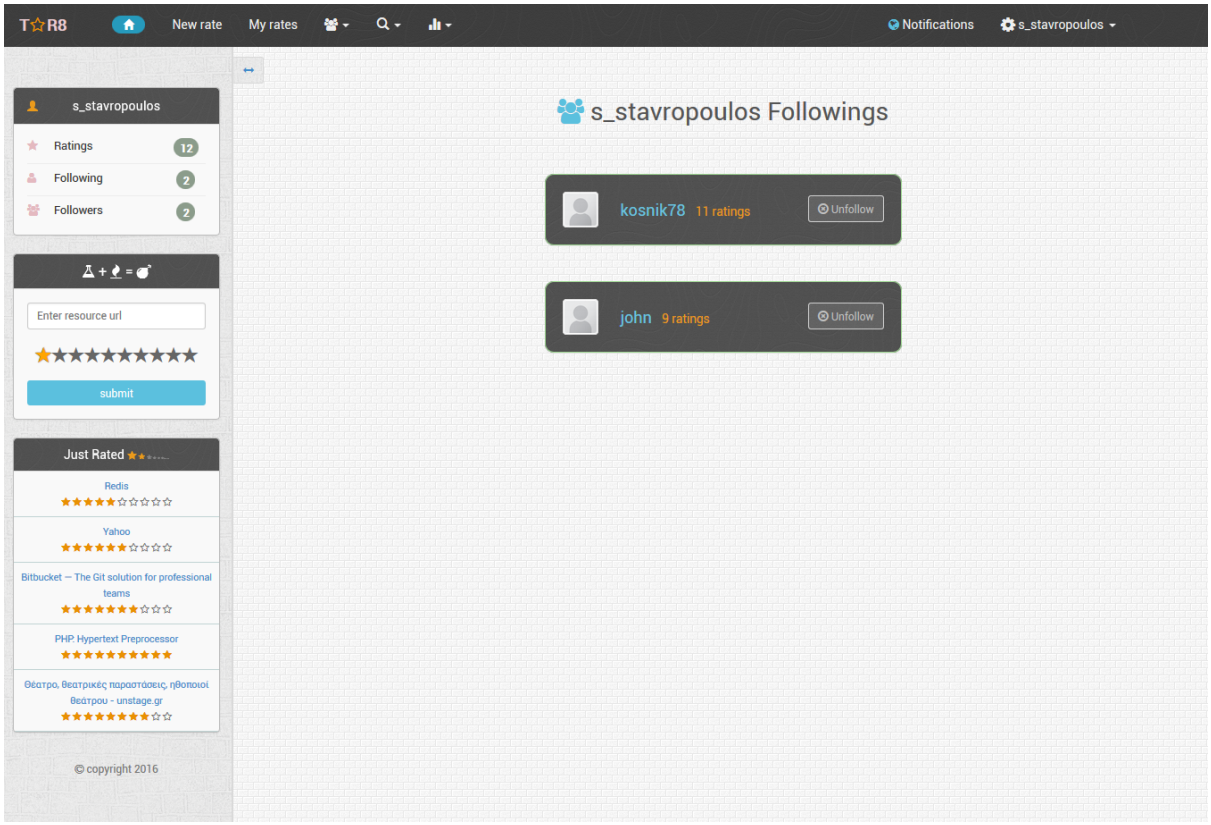
Timeline



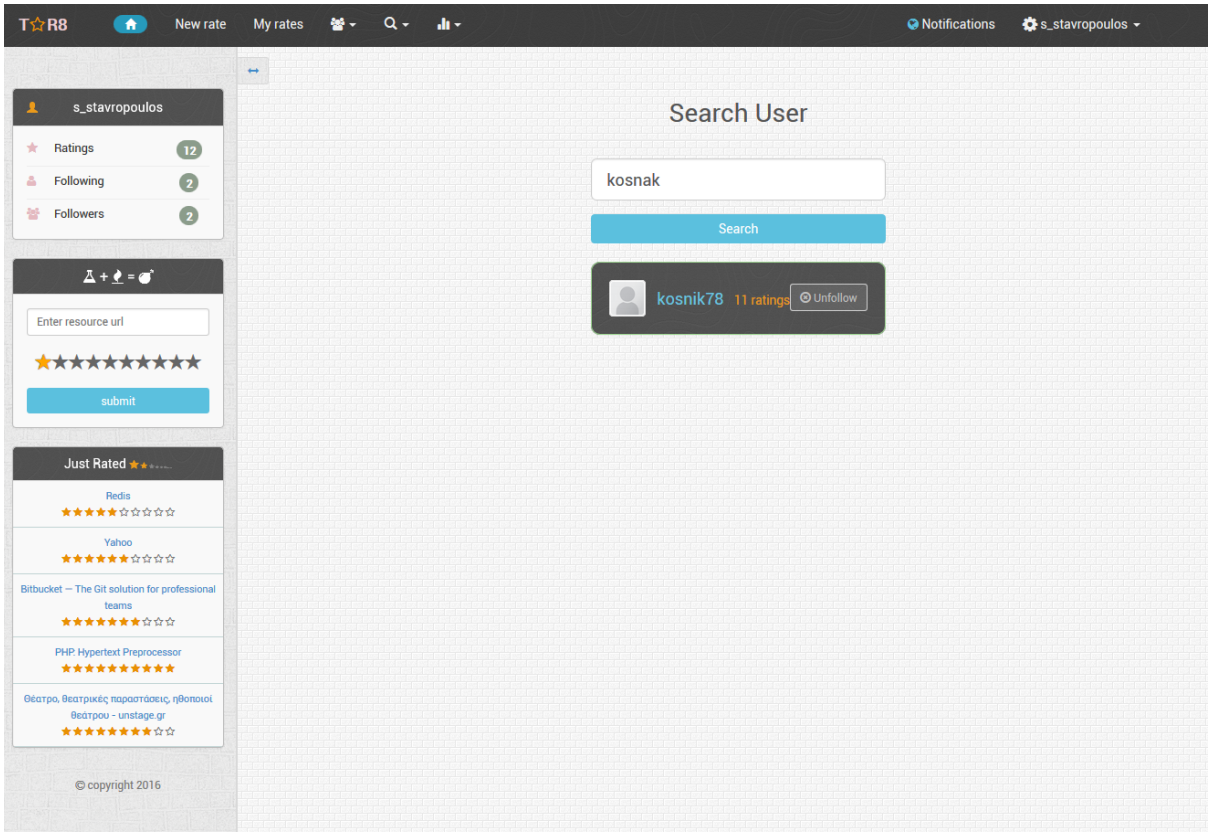
My Followers



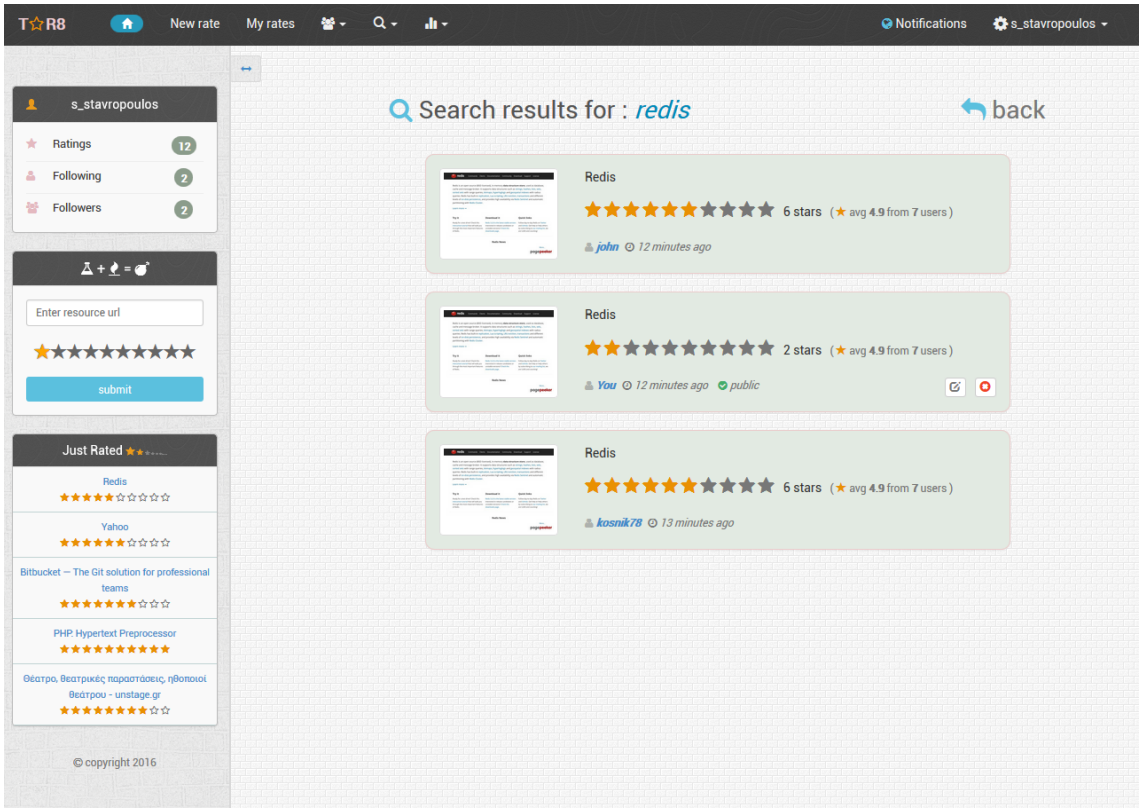
My Followings



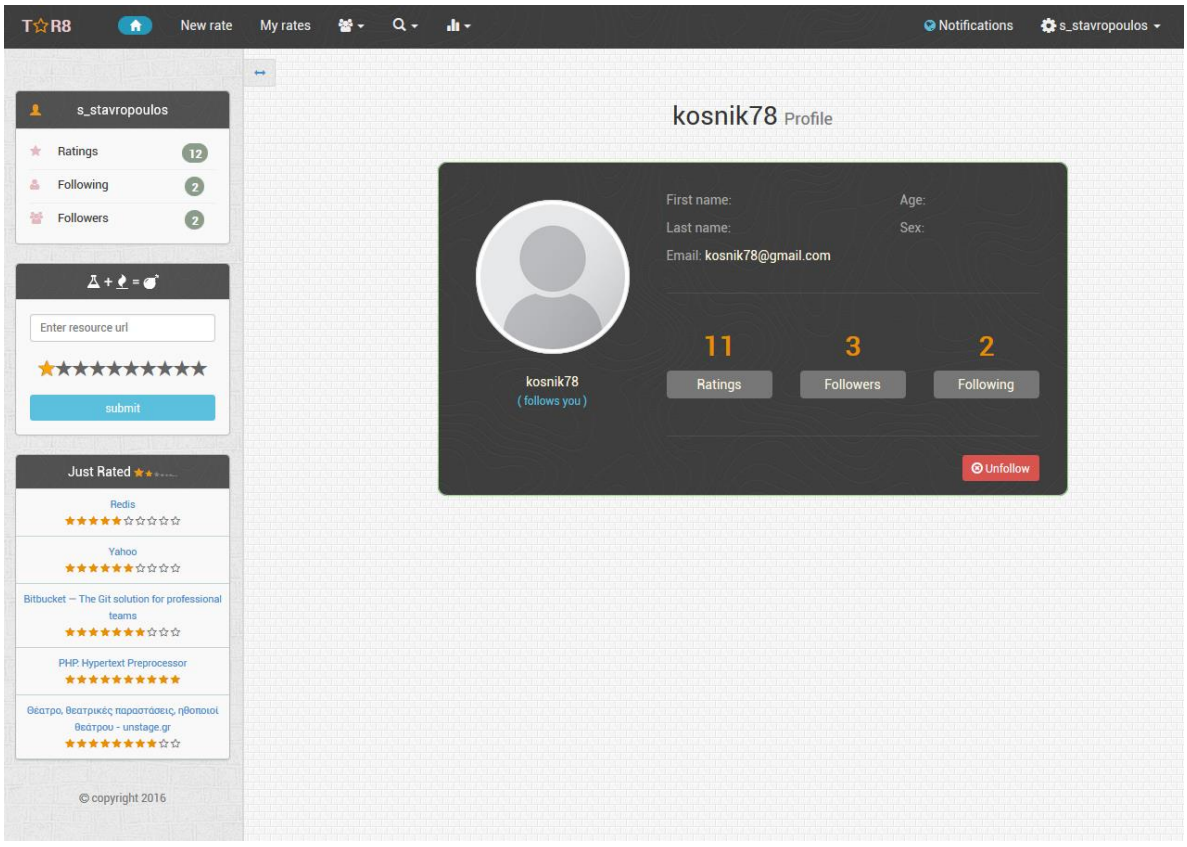
My Followers



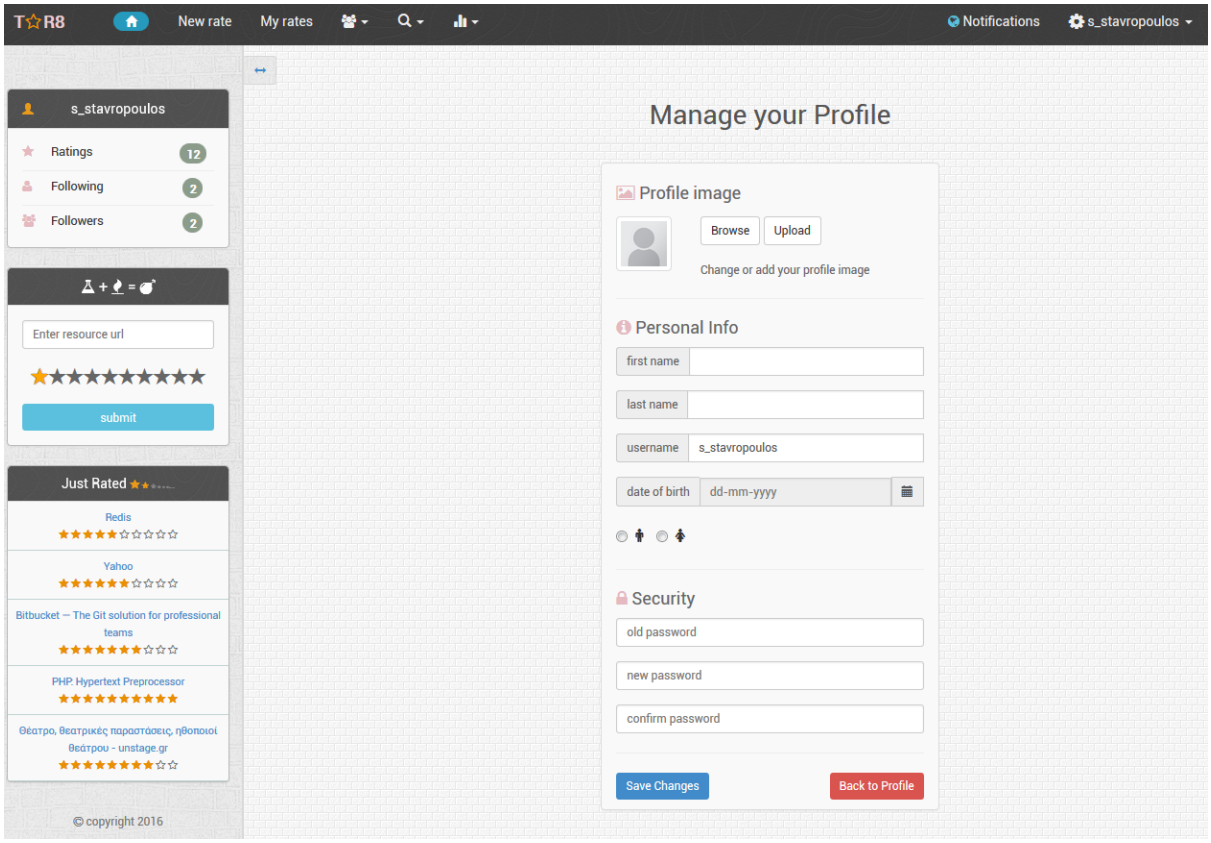
Search Rate



User Profile



Mine Profile



Παράρτημα κώδικα

Controllers

BaseController.php

```php
<?php

class BaseController extends Controller
{

    protected $Redis;
    // declaration of redis variable
    protected $Udata;
    // declaration of user data

    public function __construct()
    {
        $this -> Redis = Redis::connection();
    }

    /**
     * Setup the layout used by the controller.
     *
     * @return void
     */

    protected function setupLayout()
    {
        if (!is_null($this -> layout))
        {
            $this -> layout = View::make($this -> layout);
        }
    }

    /*
     * Method to confirm if user is signed in.
     * Check if user is signed in
     */
    protected function isSignedin()
    {
        if (isset($_COOKIE['auth']))
        {
            //laravel send cookie with encrypt format
            $cookie = Crypt::decrypt($_COOKIE['auth']);

            // get user id below data set

            $uid        = $this -> Redis -> hget ("auth", $cookie);

            // get all user data
            $fname      = $this -> Redis -> hget ('uid:'.$uid, 'fname');
            $lname      = $this -> Redis -> hget ('uid:'.$uid, 'lname');
            $email      = $this -> Redis -> hget ('uid:'.$uid, 'email');
            $sex        = $this -> Redis -> hget ('uid:'.$uid, 'sex');
            $age        = $this -> Redis -> hget ('uid:'.$uid, 'age');
            $thumb      = $this -> Redis -> hget ('uid:'.$uid, 'thumb');
            $username   = $this -> Redis -> hget ('uid:'.$uid, 'username');
            $uid_auth   = $this -> Redis -> hget ('uid:'.$uid, 'auth');
            $isActive   = $this -> Redis -> hget ('uid:'.$uid, 'verified');

            //check if auth filed and cookie are not equal
            if ($uid_auth != $cookie)
            {
                return FALSE;
            }

            //if verify code is false means that user doesn't have reply to verification email
            if (!$isActive)
            {
                //get current and signed up datetime
                $currentDT = microtime(true);
                $signedupDT = $this -> Redis -> hget('uid:' . $uid, 'timestamp');
                $result = $currentDT - $signedupDT;

                // check if this period is rather than 24 hours
                // and if is true force him sign out
                if ($result > 86400)
                {
                    $this -> Redis -> hdel('uid:' . $uid, 'auth');
                    $this -> Redis -> hdel('auth', $uid_auth);

                    return FALSE;
                }
            }

            $this->Udata = array(
```

```
82
83                 'uid'       => $uid,
84                 'fname'     => $fname,
85                 'lname'     => $lname,
86                 'email'     => $email,
87                 'age'       => $age,
88                 'sex'       => $sex,
89                 'thumb'     => $thumb,
90                 'username'  => $username,
91                 'verified'  => $isActive
92             );
93
94             // Share with every view a notification counter and a hidden field value
95             $notificationsCount = $this -> Redis -> get('uncheckedNotifications:' . $this -> Udata['uid']);
96             if($notificationsCount==0)$notificationsCount=null;
97             View::share(array(
98                 'myUid' => $uid,
99                 'notificationsCount' => $notificationsCount
100            ));
101
102            return TRUE;
103        }
104        return FALSE;
105    }
106
107 }
108
```

HomeController.php

```php
<?php

class HomeController extends BaseController
{

    public function __construct()
    {
        parent::__construct();
        // check login authentication always on class instantiation
        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                return View::make('hello');
                // show signin/signup options
            });
        }
    }

    /*
     * If user is signedin ( checked in the constructor),show homepage  (timeline)
     * When the view is loaded, jquery takes action to show the timeline (does pagination if needed)
     */
    public function showHome()
    {
        //count the total rates of the timeline
        $totalRids = $this -> Redis -> llen("timeline_ratings:" . $this -> Udata['uid']);
        if (!$totalRids)//no rates to show
        {
            return View::make('home', array(
                'username' => $this -> Udata['username'],
                'verified' => $this -> Udata['verified'],
                'message' => '<i class="fa fa-info-circle text-info"></i> Your timeline is empty..',
                'uid'      => $this ->Udata['uid']
            ));
        }

        //get the first 5 rids(change this to the number of rates we want to paginate)
        $rids = $this -> Redis -> lrange("timeline_ratings:" . $this -> Udata['uid'], 0, 4);
        $i = 0;
        foreach ($rids as $rid)
        {
            //get all the rid hash info
            $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
            //add rid, avg score,rater's username and total # of ratings('count') on that resource
            $rates[$i]['rid'] = $rid;
            $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
            $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
            $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
            //check if I have rated it (to show 'rate it' btn  in rate markup)
            if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
            {
                $rates[$i]['metoo'] = true;
            }
            $i = $i + 1;
        }
        // //compute the pages needed for pagination
        // $items = 5;  //how many rates to paginate
        // $pages = ceil($totalRids/$items); //needed for jquery on home view
        //show home
        return View::make('home', array(
            'username' => $this -> Udata['username'],
            'verified' => $this -> Udata['verified'],
            'uid' => $this -> Udata['uid'],
            'rates' => $rates,
            'totalRids' => $totalRids
        ));
    }

}
```

UserController.php

```php
<?php

class UserController extends BaseController {

    private $isSignedin; // stored result of isSignedin method

    public function __construct()
    {
        parent::__construct();
        //current constructor inherit parent from BaseController.

        // exist hear because need to use return value and user data: id, username
        // in signout, searchUser, handleSearchUser methods
        $this->isSignedin = $this->isSignedin();
    }

    /**
     * Retrieves values from form fields and creates new user
     * Calls signupUser() method in model app/models/User.php
     *
     */
    public function handleSignup()
    {
        // Fetch all request data.
        $formData = Input::all();

        // Build the validation constraint set.
        $rules = array(

            'emailUp'     => 'required|email|already_exists',
            'passwordUp'  => 'required|min:4'
        );

        // Create a new validator instance
        $validator = Validator::make($formData, $rules);

        if ($validator -> fails())
        {
            // Validator fails, form fields dont comply with the rules
            return Redirect::to('/') -> withErrors($validator) -> withInput(Input::only('emailUp'));
        }
        //  Calculate the md5 hash of the string
        $formData['password']   = md5($formData['passwordUp']);
        $formData['email']      = $formData['emailUp'];
        $formData['verifyCode'] = md5(rand());
        $formData['auth']       = md5(rand());

        // Call method for user creation
        $User = new User;
        $User -> signup($formData);

        // send email and use views/emails/verify.blade.php file
        Mail::send('emails.verify', $formData, function($message)
        {
            //$message->from('triple@gmail.com', 'Laravel' );
            $message->to(Input::get('emailUp') , 'Recipient')->subject('Welcome to the Laravel 4 App!');
        });

        // Successful signup
        return Redirect::to('/');
    }

    /*
     * Get the values from sign in form and confirm if
     * the email and the password exist in database
     */
    public function handleSignin() {

        $formData = Input::all();

        $rules = array(

            'emailIn'       => 'required|exist_email',
            'passwordIn'    => 'required|exist_password:'.$formData['emailIn'].','.$formData['passwordIn']
            );

        //create instance of validator and pass form data and the rules
        $validator = Validator::make($formData, $rules);

        if ($validator -> fails())
        {
```

```
82              // Return to Sign In form with Errors and the email been given
83              return Redirect::to('/') -> withErrors($validator) -> withInput(Input::only('emailIn'));
84          }
85
86          $currentUid = $this -> Redis -> hget('email', $formData['emailIn']);
87
88          $formData['uid']      = $currentUid;
89          $formData['auth']     = md5(rand()); //create a md5 number for authentication code
90
91          //Set cookie auth and parse to the next response
92          Cookie::queue('auth', $formData['auth'], time() + 3600 * 24 * 365);
93
94          //Create new field in user hash named auth with value a random md5 number.
95          //Store in a new hash with field the md5 number the value of user id.
96          $this -> Redis -> hset('uid:' . $formData['uid'], 'auth', $formData['auth']);
97          $this -> Redis -> hset('auth', $formData['auth'], $formData['uid']);
98
99          //Redirect to url that was brefore the system redirect user to sign in. Else to default (profile)
100         return Redirect::intended('/');
101     }
102
103     /*
104      * Method reliable for the email verification.
105      * Get the verification code that comes from the reply
106      * email and proceed to the account activation.
107      */
108     public function confirm($verifyCodeSent)
109     {
110         if (empty($verifyCodeSent))
111         {
112             return Redirect::to('/') -> withErrors('Error with account verification');
113         }
114
115         $uid =  $this -> Redis -> hget('verifyCode', $verifyCodeSent);
116
117         $verifyCodeStored = $this -> Redis -> hget('uid:'.$uid, 'verifyCode');
118
119         if ($verifyCodeStored != $verifyCodeSent)
120         {
121             return Redirect::to('/') -> withErrors('Error with account verification');
122         }
123
124         //$dbAuth = $this -> Redis -> hget('uid:'.$uid, 'auth');
125         //$this -> Redis -> hdel('uid:'.$uid, 'auth');
126         //$this -> Redis -> hdel('auth', $dbAuth);
127         $this -> Redis -> hdel('uid:'.$uid, 'verifyCode');
128         $this -> Redis -> hdel('uid:'.$uid, 'timestamp');
129         $this -> Redis -> hdel('verifyCode', $verifyCodeStored);
130         $this -> Redis -> hset('uid:'.$uid, 'verified', 1);
131
132         return Redirect::to('/') -> withErrors('Your account has been activated!');
133     }
134
135     /*
136      * Sign out Method
137      */
138     public function signout()
139     {
140         if (!$this->isSignedin)
141         {
142             return Redirect::to('/');
143         }
144
145         $cookie = Crypt::decrypt($_COOKIE['auth']);
146
147         //Retrieve user data
148         $uid     = $this -> Redis -> hget('auth', $cookie);
149         $DBAuth  = $this -> Redis -> hget('uid:' . $uid, 'auth');
150
151         //Delete md5 number from hush filed to set user sign out
152         $this -> Redis -> hdel('uid:' . $uid, 'auth');
153         $this -> Redis -> hdel('auth', $DBAuth);
154
155         return Redirect::to('/');
156     }
157
158     /*
159      * Make search user view
160      */
161     public function searchUser()
162     {
163         //parse to userSignedin.blade.php
164         return View::make('searchUser') ->with('username', $this->Udata['username'])
165                                         ->with('verified', $this->Udata['verified'])
166                                         ->with('semistop', FALSE);
```

```
167     }
168
169     /*
170      * Search user method
171      */
172     public function handleSearchUser()
173     {
174         $formdata['searchUser'] = Input::get('searchUser');
175
176         // Build the validation constraint set.
177         $rules = array('searchUser' => 'required');
178
179         // Create a new validator instance
180         $validator = Validator::make($formdata, $rules);
181
182         if ($validator -> fails())
183         {
184             if (Request::ajax())
185             {
186                 // render cause need searchUserResults.php file with validator erros for updating over AJAX
187                 $html = View::make('userBase')->withErrors($validator)->render();
188
189                 return Response::json(array(
190
191                     'status' => TRUE,
192                     'formCase' => 'searchUser',
193                     'view'   => $html
194                 ));
195             }
196
197             return Redirect::to('searchuser')->withErrors($validator);
198         }
199
200         //get current user id
201         $formdata['uid']      = $this->Udata['uid'];
202         $metaphone            = new DoubleMetaPhone;
203         $formdata['keywords'] = $metaphone->getKeywords($formdata['searchUser']);
204
205         // protect from metaphone return nothing
206         if (!count($formdata['keywords']))
207         {
208             if (!Request::ajax())
209             {
210                 return Redirect::to('searchuser')->withErrors('Nothing found..');
211             }
212
213             $html = View::make('userBase')->withErrors('Nothing found..')->render();
214
215             return Response::json(array(
216
217                 'status' => TRUE,
218                 'formCase' => 'searchUser',
219                 'view'   => $html
220             ));
221         }
222
223         // search for max two different strings from search field. Remove the rests
224         if (sizeof($formdata['keywords'] ) > 2)
225         {
226             array_splice($formdata['keywords'], 2);
227         }
228
229         // add prefix user: before every string coming from search user field
230         array_walk($formdata['keywords'], function(&$value, $key)
231         {
232             $value = 'users:'.$value;
233         });
234
235         //instance of User model and call search function
236         $User    = new User();
237         $results = $User->search($formdata);
238
239         if (!$results)
240         {
241             if (Request::ajax())
242             {
243                 // render cause need searchUserResults.php file with the results of search for updating over AJAX
244                 $html = View::make('userBase')->withErrors('Nothing found..')
245                                               ->with('semistop', FALSE)
246                                               ->render();
247
248                 $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
249
250                 return Response::json(array(
251
```

```
252                 'status' => TRUE,
253                 'formCase' => 'searchUser',
254                 'view'   => $html,
255                 'more'   => $more
256             ));
257         }
258 
259         return Redirect::to('searchuser')->withErrors('Nothing found..');
260     }
261 
262     if (Request::ajax())
263     {
264         $html = View::make('userBase')->with('results', $results['results'])
265                                       ->with('semistop', $results['semistop'])
266                                       ->render();
267 
268         $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
269 
270         return Response::json(array(
271 
272             'status' => TRUE,
273             'formCase' => 'searchUser',
274             'view'   => $html,
275             'more'   => $more
276         ));
277     }
278 
279     return View::make('searchUser')->with('results', $results['results'])
280                                    ->with('semistop', $results['semistop'])
281                                    ->with('username', $this->Udata['username'])
282                                    ->with('verified' , $this->Udata['verified']);
283 }
284 
285 /*
286  * Show more search users results every time user scroll at the
287  * bottom of the page
288  */
289 public function moreUsers()
290 {
291     $start   = Input::get('semistop');
292     $User    = new User();
293     $results = $User->pagination(++$start, $this->Udata['uid']); //++$start avoid show the last result as first
294 
295     if (Request::ajax())
296     {
297         $html = View::make('userBase')->with('results', $results['results'])
298                                       ->with('semistop', $results['semistop'])
299                                       ->render();
300 
301         $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
302 
303         return Response::json(array(
304 
305             'status' => TRUE,
306             'formCase' => 'moreUsers',
307             'view'   => $html,
308             'more'   => $more
309         ));
310     }
311 }
312 ////////////////////////////////
313 public function getSidebarProfileInfo()
314 {
315     $info['ratings'] = $this ->Redis-> llen('own_ratings:'.$this->Udata['uid']);
316     $info['followers'] = $this ->Redis-> zcard('followers:'.$this->Udata['uid']);
317     $info['followings'] = $this ->Redis-> zcard('followings:'.$this->Udata['uid']);
318 
319     return Response::json(array(
320         'numOfRatings'      => $info['ratings'],
321         'numOfFollowers'    => $info['followers'],
322         'numOfFollowings'   => $info['followings']
323     ));
324 
325 }
326 
327 }
```

ProfileController.php

```php
<?php

/*
 * User profile
 */

class ProfileController extends BaseController
{

    public function __construct()
    {
        parent::__construct();
        //current constructor inherit parent from BaseController

        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> withErrors('Sign in is required.');
            });
        }
    }

    /*
     * Show profile of signed in user
     */
    public function mine()
    {
        // Instanciate and get data from user with id the given argument
        $User       = new User();
        $this->Udata = $User->getUserData($this->Udata['uid']);

        $this->Udata['sameuser'] = TRUE;
        //if a dob was set (field 'age' in user hash, timestamp format), calculate his age
        if(!empty($this->Udata['age']))
        {
            $this->Udata['age'] = $this->calculateAge($this->Udata['age']);
        }


        return View::make('userPage')->with('user', $this->Udata)
                                     ->with('username', $this->Udata['username'])
                                     ->with('verified', $this->Udata['verified']);
    }

    /*
     * Show profile of foreign user
     */
    public function preview($uid)
    {
        if ($uid == $this->Udata['uid'])
        {
            return Redirect::to('profile');
        }

        // Instanciate and get data from user with id the given argument
        $User     = new User();
        $uid_data = $User->getUserData($uid);
        $uid_data['sameuser'] = FALSE;
        //if a dob was set (field 'age' in user hash, timestamp format), calculate his age
        if(!empty($uid_data['age']))
        {
            $uid_data['age'] = $this->calculateAge($uid_data['age']);
        }
        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $uid_data['isFollower'] = $this->Redis->zrank('followers:'.$this->Udata['uid'], $uid);
        $uid_data['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $uid);

        return View::make('userPage')->with('user', $uid_data)
                                     ->with('username', $this->Udata['username'])
                                     ->with('verified', $uid_data['verified']);
    }

    /*
     * Show edit profile view of signed in user
     */
    public function edit()
    {
        //if a dob was set (field 'age' in user hash, timestamp format), convert to d-m-y date
```

```
82      if(!empty($this->Udata['age']))
83      {
84          $this->Udata['age'] = date('d-m-Y',$this->Udata['age']);
85      }
86
87      return View::make('editProfile')->with('user', $this->Udata)
88                                      ->with('username', $this->Udata['username'])
89                                      ->with('verified', $this->Udata['verified']);
90  }
91
92  /*
93   * get post variables from edit profile view and proceed to changes
94   */
95  public function handleEdit()
96  {
97      $formData = Input::all();
98
99      $rules = array( 'username' => 'min:2' );
100
101     if (Input::has('fname'))// if first name has value add rule to validator
102     {
103         $rules['fname'] = 'min:2';
104     }
105
106     if (Input::has('lname'))
107     {
108         $rules['lname'] = 'min:2';
109     }
110
111     if(Input::has('dob'))
112     {
113         $rules['dob'] = 'date_format:"d-m-Y"';
114     }
115     $messages = array( 'min' => 'At least two characters' );// custom validator message for min rule
116
117     $validator = Validator::make($formData, $rules, $messages);
118
119     if ($validator -> fails())
120     {
121         // Validator fails, form fields dont comply with the rules
122         return Redirect::to('edit') -> withErrors($validator);
123     }
124
125     // check exist changes in security fields and parse values to validator
126     if (Input::has('oldPassword') || Input::has('newPassword') || Input::has('confirmPassword'))
127     {
128         // get user email because need it for exist_password validation rule
129         $email = $this->Redis->hget('uid:'.$this->Udata['uid'], 'email');
130
131         // Build the validation constraint set.
132         $rules = array(
133
134             'oldPassword'     => 'required|exist_password:'.$email.','.$formData['oldPassword'],
135             'newPassword'     => 'required|min:4',
136             'confirmPassword' => 'required|password_confirmed:'.$formData['newPassword'],
137
138         );
139
140         //custom validator message for exist_password rule
141         $messages = array(
142                 'exist_password' => 'Wrong Password',
143                 'dob'            => 'invalid date format'
144                  );
145
146         // Create a new validator instance
147         $validator = Validator::make($formData, $rules, $messages);
148
149         if ($validator -> fails())
150         {
151             // Validator fails, form fields dont comply with the rules
152             return Redirect::to('edit') -> withErrors($validator);
153         }
154
155         $formData['password'] = $formData['newPassword'];
156     }
157
158     $formData['uid'] = $this->Udata['uid'];
159     if(Input::has('dob'))
160     {
161         $formData['age'] = strtotime($formData['dob']);
162     }
163     //print_r($formData);return;
164     //parse data to user model and return the changes
165     $User     = new User();
166     $userData = $User->editProfile($formData);
```

```
167
168
169         return  Redirect::to('edit')->with('user'    , $userData)
170                                     ->with('username', $userData['username'])
171                                     ->with('verified', $userData['verified'])
172                                     ->with('message' , 'Changes Saved');
173     }
174
175     /*
176      * Get avatar thumb from editProfile.blade.php
177      */
178     public function getAvatar()
179     {
180         $input = array('avatar' => Input::file('avatar')); // give name to input filed
181
182         $rules = array('avatar' => 'required|image|max:1000');
183
184         $messages = array('avatar.required' => 'Select an image first');
185
186         $validator = Validator::make($input, $rules, $messages);
187
188         if ($validator -> fails())
189         {
190             // Validator fails, form fields dont comply with the rules
191             return Redirect::to('edit')->withErrors($validator);
192         }
193
194         $avatar   = Input::file('avatar'); // instance from uploaded file
195
196         $s3Data['filepath'] = $avatar->getRealPath();               // input type files temporary saved into server. use this path to skip move method
197         $s3Data['filetype'] = $avatar->getMimeType();                // get type of file
198         $fileExtension      = $avatar->getClientOriginalExtension();    // get extension
199         $s3Data['filename'] = 'avatar'.$this->Udata['uid'].'.'.$fileExtension; // create a new file name with prefix avatar plus the uid
200         $s3Data['bucket']   = 'avatarthumb';
201         $s3Data['uid']      = $this->Udata['uid'];
202
203         $User   = new User();
204         $User->storeAvatar($s3Data);
205
206         return Redirect::to('edit');
207     }
208
209
210     /**
211      * Calculate age in years based on timestamp and reference timestamp
212      * If the reference $now is set to 0, then current time is used
213      *
214      * @param int $timestamp
215      * @param int $now
216      * @return int
217      */
218     public function calculateAge($timestamp = 0, $now = 0) {
219     # default to current time when $now not given
220     if ($now == 0)
221     $now = time();
222
223     # calculate differences between timestamp and current Y/m/d
224     $yearDiff = date("Y", $now) - date("Y", $timestamp);
225     $monthDiff = date("m", $now) - date("m", $timestamp);
226     $dayDiff = date("d", $now) - date("d", $timestamp);
227
228     # check if we already had our birthday
229     if ($monthDiff < 0)
230     $yearDiff--;
231     elseif (($monthDiff == 0) && ($dayDiff < 0))
232     $yearDiff--;
233
234     # set the result: age in years
235     $result = intval($yearDiff);
236
237     # deliver the result
238     return $result;
239     }
240 }
241
```

FollowerGraphController.php

```php
<?php

class FollowerGraphController extends BaseController
{

    /*
     *  Controller of follow graph actions
     */

    public function __construct()
    {
        parent::__construct(); //inherit variable of redis connection from parent

        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> withErros('You are not signed in...');
            });
        }
    }


    public function follow()
    {
        $formdata = Input::all(); // get foreign user id
        $formdata['current_uid'] = $this->Udata['uid']; // get current user id

        $follow   = new FollowerGraph();
        $userData = $follow->followAction($formdata);
        $page     = false;

        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $userData['isFollower']  = $this->Redis->zrank('followers:'.$this->Udata['uid'], $userData['uid']);
        $userData['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $userData['uid']);

        //get (the new) number of your followings to update sidebar's mini profile panel info
        $numOfFollowings = $this->Redis->zcard('followings:'.$this->Udata['uid']);

        if(array_key_exists('page', $formdata))  //form submitted from userProfile view
        {
            $page = $formdata['page']; //will return with the json response
            //render the userProfileMarkup that will replace the current (will chang the 'follow' btn to 'unfollow' and the # of followings info)
            $html = View::make('userProfileMarkup')->with('user',$userData)->with('username', $this->Udata["username"])->render();
        }
        else
        {
            // render cause need file with updated results for updating over AJAX
            $html = View::make('userMarkup')->with('user', $userData)->render();
        }

        $message = 'You are now following '.$userData['username'];

        return Response::json(array(

            'status'    => TRUE,
            'formCase'  => 'followerGraph',
            'view'      => $html,
            'uid'       => $formdata['foreign_uid'],
            'user'      => $userData,
            'numOfFollowings' => $numOfFollowings,
            'page'      => $page,
            'message'   => $message
        ));
    }

    public function unfollow()
    {
        $formdata = Input::all();
        $formdata['current_uid'] = $this->Udata['uid'];

        $unfollow = new FollowerGraph();
        $userData = $unfollow->unFollowAction($formdata);
        $page     = false;

        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $userData['isFollower'] = $this->Redis->zrank('followers:'.$this->Udata['uid'], $userData['uid']);
        $userData['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $userData['uid']);

```

```
82      //get (the new) number of your followings to update sidebar's mini profile panel info
83      $numOfFollowings = $this->Redis->zcard('followings:'.$this->Udata['uid']);
84
85      if(array_key_exists('page', $formdata))  //form submitted from userProfile view
86      {
87          $page = $formdata['page']; //will return with the json response
88          //render the userProfileMarkup that will replace the current (will chang the 'follow' btn to 'unfollow' and the # of followings info)
89          $html = View::make('userProfileMarkup')->with('user',$userData)->with('username', $this->Udata["username"])->render();
90      }
91      else
92      {
93          // render cause need file with updated results for updating over AJAX
94          $html = View::make('userMarkup')->with('user', $userData)->render();
95      }
96      $message = 'You no longer follow '.$userData['username'];
97      return Response::json(array(
98
99          'status'    => TRUE,
100         'formCase' => 'followerGraph',
101         'view'      => $html,
102         'uid'       => $formdata['foreign_uid'],
103         'user'      => $userData,
104         'page'      => $page,
105         'message'   => $message,
106         'numOfFollowings' => $numOfFollowings
107     ));
108 }
109
110 /*
111  * Show followers
112  */
113 public function followers($uid = null)
114 {
115     //get uid as argument(optional) if no uid parsed means that user who ask for followers
116     //is the signed in user(current) else..
117     if ($uid == null)
118     {
119         $uid      = $this->Udata['uid'];
120         $username = $this->Udata['username'];
121     }
122     else
123     {
124         $username = $this->Redis->hget('uid:'.$uid, 'username');
125     }
126
127     $start      = 0;
128     $followers  = new FollowerGraph();
129     $results    = $followers->getFollowers($uid, $this->Udata['uid'], $start);
130
131     if (!$results)
132     {
133         return View::make('showFollowersFollowings')->with('results'  , $results)
134                                                     ->with('username' , $this->Udata['username'])
135                                                     ->with('verified' , $this->Udata['verified'])
136                                                     ->with('fUsername', $username)
137                                                     ->with('semistop' , $results['semistop'])
138                                                     ->with('action'   , 'Followers')
139                                                     ->withErrors('Nothing found..');
140     }
141
142     return View::make('showFollowersFollowings')->with('results'  , $results['results'])
143                                                 ->with('username' , $this->Udata['username'])
144                                                 ->with('verified' , $this->Udata['verified'])
145                                                 ->with('fUsername', $username)
146                                                 ->with('semistop' , $results['semistop']) // point where the pagination stops
147                                                 ->with('uid'      , $results['uid'])
148                                                 ->with('action'   , 'Followers');
149 }
150
151 /*
152  * Show followings
153  */
154 public function followings($uid = null)
155 {
156     if ($uid == null)
157     {
158         $uid      = $this->Udata['uid'];
159         $username = $this->Udata['username'];
160     }
161     else
162     {
163         $username = $this->Redis->hget('uid:'.$uid, 'username');
164     }
165
166     $start      = 0;
```

```
167         $followings = new FollowerGraph();
168         $results    = $followings->getFollowings($uid, $this->Udata['uid'], $start);
169
170         if (!$results)
171         {
172             return View::make('showFollowersFollowings')->with('results'  , $results)
173                                                         ->with('username' , $this->Udata['username'])
174                                                         ->with('verified' , $this->Udata['verified'])
175                                                         ->with('fUsername', $username)
176                                                         ->with('semistop' , $results['semistop'])
177                                                         ->with('action'   , 'Followings')
178                                                         ->withErrors('Nothing found..');
179         }
180
181         return View::make('showFollowersFollowings')->with('results'  , $results['results'])
182                                                     ->with('username' , $this->Udata['username'])
183                                                     ->with('verified' , $this->Udata['verified'])
184                                                     ->with('fUsername', $username)
185                                                     ->with('semistop' , $results['semistop'])
186                                                     ->with('uid'      , $results['uid'])
187                                                     ->with('action'   , 'Followings');
188     }
189
190     /*
191      * Pagination on followers followings results
192      */
193     public function pagination()
194     {
195         $start      = Input::get('semistop');// where stop the previus count
196         $uid        = Input::get('uid');     // id of user whose followers - followings be presented
197         $from       = Input::get('from');    // which action come from followers or followings
198         $followers  = new FollowerGraph();
199
200         if ($from == 'Followers')
201         {
202             $results = $followers->getFollowers($uid, $this->Udata['uid'], ++$start);
203         }
204         else
205         {
206             $results = $followers->getFollowings($uid, $this->Udata['uid'], ++$start);
207         }
208
209         if (Request::ajax())
210         {
211             $html = View::make('userBase')->with('results', $results['results'])
212                                           ->render();
213
214             $more = View::make('moreUsers')->with('semistop', $results['semistop'])
215                                            ->with('uid', $results['uid'])
216                                            ->render();
217
218             return Response::json(array(
219
220                 'status' => TRUE,
221                 'formCase' => 'moreUsers',
222                 'view'   => $html,
223                 'more'   => $more
224             ));
225         }
226     }
227 }
228
229
230
```

RateController.php

```php
<?php

class RateController extends BaseController
{

    protected $uid;

    public function __construct()
    {
        parent::__construct();
        // check login authentication always on class instantiation
        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> with('message', 'You are not signed in...');
            });
        }
    }

    ////////////////////////////////////////////////////////////////////////////////////
    /*
     * Function newRate()
     * Called on 'New rate' menu button pressed
     * Shows new rate form (app/views/newRate.blade.php)
     *
     */
    public function newRate()
    {

        return View::make('newRate', array(
            'uid' => $this -> Udata['uid'],
            'username' => $this -> Udata['username'],
            'verified' => $this -> Udata['verified']
        ));
    }

    ////////////////////////////////////////////////////////////////////////////////////
    /*
     * Function handleNewRate()
     * Called on new rate form submit (From quick(sidebar) or normal (newrate view))
     * Gets form input data
     * Runs checks and validation rules
     * Calculates data needed (resource title, e.t.c.)
     * Calls appropriate method from rate model (app/models/Rate.php) to insert/update rate into db (newRate() or edit())
     */

    public function handleNewRate()
    {

        //get data from form
        $rdata = Input::all();
        // if rate is quick rate (no review input), create an empty review index and set validation rules
        if (!array_key_exists('review', $rdata))
        {
            $form = 'quickNewRate';
            //returned with json response
            $rdata['review'] = "";
            $rules = array('urlQuick' => 'required|activeurl');
            // //trim any forward slash, spaces, tabs and linebreaks before inserting to db
            // // (e.g. we want www.foo.com & www.foo.com/ to be treated as same resource)
            // $rdata['urlQuick']   = rtrim($rdata['urlQuick'], "/ \t\n\r");
            $uri = $rdata['url'];
            // The URI will be included into json response on form success.
            // The function formSuccessQuickRate(data) will use it to decide if the page should be refreshed (case page = home or myrates) or not
        }
        else// new rate from newRate view (not from sidebar's quick rate form)
        {
            $form = 'normalNewRate';
            $rules = array('urlNormal' => 'required|activeurl');
            // //trim any forward slash, spaces, tabs and linebreaks
            // $rdata['urlNormal'] = rtrim($rdata['urlNormal'], "/ \t\n\r");
            $uri = null;
            // we don't need the URI in this case (new rate from newRate view)
        }

        //set custom validation error messages:
        //if laravel's default messages are used, the form's input names are shown (e.g. 'the urlQuick field is required') - not elegant
        $messages = array(
```

```
82              'required' => 'The url field is required',
83              'activeurl' => 'The url is not a valid URL'
84          );
85
86          // client-sided (javascript) validation of the form succeeded, continue with server-sided validation
87
88          // Create a new validator instance (pass the custom messages too)
89          $validator = Validator::make($rdata, $rules, $messages);
90
91          if ($validator -> fails())
92          {
93              // Validator fails, return JSON encoded response
94              return Response::json(array(
95                  'status' => FALSE,
96                  'errors' => array($validator -> messages() -> all())
97              ));
98          }
99
100         // server-sided validation ok, continue:
101         // 'normalize' $rdata['urlNormal'] / $rdata['urlquick'] to $rdata['url'] before passing to model
102         if (array_key_exists('urlNormal', $rdata))// 'normal' form submitted
103         {
104             $rdata['url'] = $rdata['urlNormal'];
105             unset($rdata['urlNormal']);
106         }
107         else//quick form submitted
108         {
109             $rdata['url'] = $rdata['urlQuick'];
110             unset($rdata['urlQuick']);
111         }
112         //
113         //trim any forward slash, spaces, tabs and linebreaks before inserting to db
114         $rdata['url'] = rtrim($rdata['url'], "/ \t\n\r");
115         //remove 'www.' if exists
116         $rdata['url'] = str_replace("http://www.", "http://", $rdata['url']);
117         //set the id of the user-rater
118         $rdata['uid'] = $this -> Udata['uid'];
119         //check if url exists, and if so, check if it is edit case (only possible from quick rate)
120         //initialize to true and change if needed
121         $rdata['newUrl'] = true;
122         //initialize to newRate and change if needed
123         $case = 'newRate';
124         // check if case is edit:
125         // a)check if url exists in db
126         $UrlManagement = new UrlManagement;
127         if ($UrlManagement -> urlExists($rdata['url']))
128         {
129             $rdata['newUrl'] = false;
130             // b)check if edit case
131
132             if ($this -> Redis -> sismember('uid:' . $rdata['uid'] . ':rated_urls', $rdata['url']))
133             {
134                 $case = 'edit';
135
136                 if ($form == 'quickNewRate')
137                 {
138                     $rdata['public'] = 'old';
139                     //for the model to know that case is edit from quick rate form=>keep the old public/private setting
140                 }
141             }
142         }
143         //take care of the rest of the rating data (only if case = newRate, not needed for edit )
144         if ($case != 'edit')
145         {
146             // set the rest of the rate fields
147             $rdata['title'] = $this -> getTitle($rdata['url']);
148             $rdata['thumb'] = $this -> getThumb($rdata['url']);
149         }
150
151         if (empty($rdata['public']))//case quick rate or 'off' in new rate form
152         {
153             $rdata['public'] = 0;
154             //default value
155         }
156
157         // get the search keywords associated with the rating title (only if rate = new rate -new rid is created)
158         if ($case == "newRate")
159         {
160             $mp = new DoubleMetaPhone;
161             $keywords = $mp -> getKeywords($rdata['title']);
162             $rdata['keywords'] = $keywords;
163         }
164         // bootstrap switch plugin sends public as 'on' if checked, convert it to '1' (for the model to store it in the redis hash as 1)
165         if ($rdata['public'] === "on")
166         {
```

```
167             $rdata['public'] = 1;
168         }
169
170         // ready to pass new rate data to model
171         // new rate model instance  (app/models/Rate.php)
172         $Rate = new Rate;
173         //call appropriate model function
174         $rid = false;
175         switch ($case)
176         {
177             case 'newRate' :
178                 $Rate -> newRate($rdata);
179                 //just a message for the json response (or the redirect if js is disabled)
180                 $message = 'New rate inserted';
181                 break;
182             case 'edit' :
183                 $Rate -> edit($rdata);
184                 //get the rid - needed for the response (so as to know which rate to update if in timeline or my rates view)
185                 $rid = $this -> Redis -> hget('url:' . $rdata['url'], $this -> Udata['uid']);
186                 //just a message for the json response
187                 $message = 'Rate edited';
188                 break;
189         }
190
191         // rate inserted succesfully, return JSON encoded response
192
193         return Response::json(array(
194             'status' => TRUE,
195             'formCase' => $form,
196             'message' => $message,
197             'uri' => $uri,
198             'rid' => $rid
199         ));
200
201     }
202
203     ////////////////////////////////////////////////////////////////////////////
204     public function handleRateIt()
205     {
206         $data = Input::all();
207         $form = 'rateIt';
208         //returned with json response
209         $data['uid'] = $this -> Udata['uid'];
210         $data['newUrl'] = false;
211
212         if (array_key_exists('public', $data))// public is 'on' - if public switch was off, the form does not post a 'public' parameter
213         {
214             $data['public'] = 1;
215         }
216         else
217         {
218             $data['public'] = 0;
219         }
220         $data['thumb'] = $this -> getThumb($data['url']);
221         //get the metaphones , for the model to add the rid into the keyword(s) set(s)
222         $mp = new DoubleMetaPhone;
223         $keywords = $mp -> getKeywords($data['title']);
224         $data['keywords'] = $keywords;
225
226         $message = 'Rate inserted';
227         //for the json response
228
229         // ready to pass new rate data to model
230         // new rate model instance  (app/models/Rate.php)
231         $Rate = new Rate;
232         $Rate -> newRate($data);
233
234         //get the rid of the rate just created
235         $rid = $this -> Redis -> hget('url:' . $data['url'], $this -> Udata['uid']);
236         //get the new avg score
237         $newAvgScore = $this -> Redis -> get('url:' . $data['url'] . ':average_score');
238         //get the new num of raters on that resource
239         $newCount = $this -> Redis -> hlen('url:' . $data['url']);
240         //if user is on home, return the new rate markup and prepend it to the timeline/myrates
241         $newRate = false;
242         $rids = false;
243         if ($data['target'] === 'home')
244         {
245             //create the rate markup to be returned to the view
246             //get the data needed for the markup
247             $rate = $this -> Redis -> hgetall('rid:' . $rid);
248             $rate['rid'] = $rid;
249             $rate['username'] = $this -> Udata['username'];
250             $rate['average_score'] = $newAvgScore;
251             $rate['count'] = $newCount;
```

```
252             $username = $this -> Udata['username'];
253
254             $newRate = View::make('rateMarkup', array(
255                 'rate' => $rate,
256                 'username' => $username
257             )) -> render();
258
259         }
260         // Get original rid
261         $originalRid = str_replace('rid', '', $data['originalRid']);
262         //Get original uid
263         $originalUid = $this -> Redis -> hget('rid:' . $originalRid, 'uid');
264         // Increace notifications counter
265         $this -> Redis -> incr('uncheckedNotifications:' . $originalUid);
266         // Add notification to user's sorted set
267         $this -> Redis -> zadd('notifications:' . $originalUid, microtime(true), $this -> Udata['uid'] . ':rated:' . $originalRid);
268
269         $isOnline = $this -> Redis -> hget('uid:' . $originalUid, 'auth');
270         // If "original user" is online notify him
271         if (!empty($isOnline))
272         {
273             $notification = array('uid' => $originalUid, );
274             Event::fire('cometNotification', array($notification));
275         }
276         return Response::json(array(
277             'status' => TRUE,
278             'formCase' => $form,
279             'message' => $message,
280             'originalRid' => $data['originalRid'], //to know which rate's 'rate it' button was pressed
281             'newrate' => $newRate,
282             'newAvgScore' => $newAvgScore,
283             'newNumOfRaters' => $newCount
284         ));
285     }
286
287     ////////////////////////////////////////////////////////////////////////////////
288     /*
289      * Function getTitle($url)
290      * Gets the title of the rating on the given resource
291      */
292     private function getTitle($url)
293     {
294         //if url is already a rated url , just get title from an existing rate hash
295         if (($this -> Redis -> hlen('url:' . $url)) > 0)
296         {
297             // get the title hash field of the first rid hash found (for that specific url)
298             $rids = $this -> Redis -> hvals('url:' . $url);
299             $title = $this -> Redis -> hget('rid:' . $rids[0], 'title');
300             return $title;
301         }
302         //if rate is new rate on non-existing url, use the appropriate lib function to get the title
303
304         //new instance of 'UrlManagement' library class ( /libraries/UrlManagement.php)
305         $UrlManagement = new UrlManagement;
306         $title = $UrlManagement -> urlGetTitle($url);
307         return $title;
308     }
309
310     ////////////////////////////////////////////////////////////////////////////////
311
312     /*
313      * Function getThumb($url)
314      * Creates and stores a thumb of the rated resource.
315      * Returns the name (rid) of the stored thumb
316      * Functionality identical to getTitle.
317      */
318     private function getThumb($url)
319     {
320         if (($this -> Redis -> hlen('url:' . $url)) > 0)
321         {
322             $rids = $this -> Redis -> hvals('url:' . $url);
323             $thumb = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
324             return $thumb;
325         }
326
327         $UrlManagement = new UrlManagement;
328         $thumb = $UrlManagement -> urlGetThumb($url);
329         return $thumb;
330     }
331
332     ////////////////////////////////////////////////////////////////////////////
333     /*
334      * Function searchRate()
335      * Shows the search Rate form view  (/app/views/searchRate.blade.php - 'search rate' menu item)
336      */
```

```
337     public function searchRate()
338     {
339         return View::make('searchRate', array(
340             'uid' => $this -> Udata['uid'],
341             'username' => $this -> Udata['username'],
342             'verified' => $this -> Udata['verified']
343         ));
344     }
345
346     ////////////////////////////////////////////////////////////////////////
347
348     /*
349      * Function handleSearchRate()
350      * Called on search rate form submission
351      * Gets form input data
352      * Prepares the input search data to be sent to search() function in Rate model
353      */
354
355     public function handleSearchRate()
356     {
357         $formdata = Input::all();
358
359         $rules = array(
360             'searchIn' => 'required', // form's checkboxes own/followings/both  (where to search)
361             'searchQuery' => 'required'
362         );
363         //set a custom error message
364         $messages = array(
365             'searchIn.required' => 'Tell me where to search',
366             'searchQuery.required' => 'Type something'
367         );
368
369         $validator = Validator::make($formdata, $rules, $messages);
370
371         if ($validator -> fails())
372         {
373             $messages = $validator -> messages();
374             // redirect to search form view (/app/views/searchRate.blade.php) with errors messagebag
375             Input::flashonly('searchQuery', 'searchIn');
376             // keep the old inputs
377             return Redirect::to('searchrate') -> withErrors($validator);
378         }
379
380         // Validation succeeded, prepare data to be sent to model
381
382         // $criteria array will be passed to model (holding search criteria, user id and 'searchIn' option)
383         // Set 'search in' option
384         $searchIn = '"both"';
385         // assume both checkboxes are checked
386         if (sizeof($formdata['searchIn']) == 1)// only one checkbox is checked
387         {
388             // get checked checkbox's name ('own' or 'followings')
389             $searchIn = key($formdata['searchIn']);
390         }
391
392         //split the search query into keywords (based on which the search will take place)
393         $mp = new DoubleMetaPhone;
394         $keywords = $mp -> getKeywords($formdata['searchQuery']);
395         if (!count($keywords))
396         {
397             return Redirect::to('searchrate') -> with('message', 'Nothing found!') -> withInput(Input::only('searchQuery', 'searchIn'));
398         }
399         $criteria = array(
400             'uid' => $this -> Udata['uid'],
401             'searchIn' => $searchIn,
402             'keywords' => $keywords
403         );
404
405         //if user clicked advanced search, add rating score range  in $criteria array to be passed to model
406         if (array_key_exists('advanced', $formdata))
407         {
408             // sort rating range (if for example user entered score from 10 to 8 (and not from 8 to 10))
409             $scores[0] = $formdata['fromScore'];
410             $scores[1] = $formdata['toScore'];
411             sort($scores);
412             //add them to criteria array
413             $criteria['scores'] = $scores;
414         }
415
416         //new rate model instance
417         $Rate = new Rate;
418         $searchResults = $Rate -> search($criteria);
419         //print_r($searchResults);return;
420         if (!$searchResults)
421         {
```

```
422             return Redirect::to('searchrate') -> with('message', 'Nothing found!') -> withInput(Input::only('searchQuery', 'searchIn'));
423
424         }
425         return View::make('searchRateResults', array(
426             'username' => $this -> Udata['username'],
427             'verified' => $this -> Udata['verified'],
428             'searchResults' => $searchResults,
429             'searchQuery' => $formdata['searchQuery']
430         ));
431
432     }
433
434     ////////////////////////////////////////////////////////////////////////////////
435     /*
436      * Function handleDeleteRate()
437      * Called on delete rate form submission
438      * Gets form input data
439      * Sends rid and uid to delete() function in Rate model
440      * delete() functions returns true or false.
441      */
442     public function handleDeleteRate()
443     {
444         $formdata = Input::only('rid');
445         $Rate = new Rate;
446         // Status is either FALSE or contains rate array
447         $status = $Rate -> delete($formdata['rid'], $this -> Udata['uid']);
448         if ($status)
449         {
450             $message = 'Rate deleted';
451             $errors = FALSE;
452         }
453         else
454         {
455             $message = FALSE;
456             $errors = ' Something went wrong, mate...';
457         }
458         //get the number of public rates, to update if needed the info displayed on page header(e.g myRates: #total - #PUBLIC rates)
459         $numOfPublic = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
460         if (Request::ajax())
461         {
462             return Response::json(array(
463                 'status' => $status,
464                 'formCase' => 'deleteRate',
465                 'message' => $message,
466                 'errors' => $errors,
467                 'rid' => $formdata['rid'],
468                 'numOfPublic' => $numOfPublic
469             ));
470         }
471
472         return Redirect::back();
473     }
474
475     ////////////////////////////////////////////////////////////////////////////////
476
477     /*
478      * Handles the edit rate form (shown on pressing 'edit' button on a rate - NOT edit case form quick or new rate form)
479      */
480
481     public function handleEditRate()
482     {
483
484         //Get form post data:
485         $formdata = Input::all();
486
487         if (!array_key_exists('public', $formdata))//public checkbox not checked, set to 0
488         {
489             $formdata['public'] = 0;
490         }
491         else
492         {
493             $formdata['public'] = 1;
494         }
495         $formdata['uid'] = $this -> Udata['uid'];
496         //needed by the edit() method in model
497         //new model instance
498         $Rate = new Rate;
499         $status = $Rate -> edit($formdata);
500         // assume things went wrong, overriden if not
501         $message = FALSE;
502         $errors = ' Something went wrong';
503         //if edited succesfully,set $message, $errors accordingly and get the all the rating info
504         //(needed for the response, as the whole rate markup will be returned to replace the old one)
505         if ($status)
506         {
```

```
507             $rate = $this -> Redis -> hgetall('rid:' . $formdata['rid']);
508             //add the rid, username, #of raters on the resource (count) , avg score (the markup expects them)
509             $rate['rid'] = $formdata['rid'];
510             $rate['username'] = $this -> Udata['username'];
511             $rate['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rate['rid'], 'url') . ':average_score');
512             $rate['count'] = $this -> Redis -> hlen('url:' . $rate['url']);
513             $username = $this -> Udata['username'];
514             $errors = FALSE;
515             $message = "Rate edited";
516         }
517
518         //get the number of public rates, to update if needed the info displayed on page header( #total - #PUBLIC rates)
519         $numOfPublic = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
520
521         //Prepare the edited rate markup. Will be included in the response and will replace the old rate markup
522         $editedRate = View::make('rateMarkup', array(
523             'rate' => $rate,
524             'username' => $username
525         )) -> render();
526
527         return Response::json(array(
528             'status' => $status,
529             'formCase' => 'editRate',
530             'message' => $message,
531             'errors' => $errors,
532             'targetRateDiv' => $formdata['target'], //to know which rate will be replaced
533             'editedRate' => $editedRate,
534             'numOfPublic' => $numOfPublic,
535         ));
536     }
537
538     ////////////////////////////////////////////////////////////////////////////////////////////////////
539
540     public function myRates()
541     {
542         //count total and public own rates
543         $totalRids = $this -> Redis -> llen("own_ratings:" . $this -> Udata['uid']);
544         $numOfPublicRates = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
545         if (!$totalRids)//no rates to show
546         {
547             return View::make('myRates', array(
548                 'username' => $this -> Udata['username'],
549                 'verified' => $this -> Udata['verified'],
550                 'message' => 'You have no rates, mate...',
551                 'public' => $numOfPublicRates,
552                 'totalRids' => $totalRids
553             ));
554         }
555         $rids = $this -> Redis -> lrange("own_ratings:" . $this -> Udata['uid'], 0, 4);
556         if (!$rids)
557         {
558             return false;
559         }
560         $i = 0;
561         foreach ($rids as $rid)
562         {
563             //get all the rid hash info
564             $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
565             //add rid, avg score,rater's username and total # of ratings('count') on that resource
566             $rates[$i]['rid'] = $rid;
567             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
568             $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
569             $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
570             //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
571             if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
572             {
573                 $rates[$i]['metoo'] = true;
574             }
575             $i = $i + 1;
576         }
577
578         return View::make('myRates', array(
579             'username' => $this -> Udata['username'],
580             'verified' => $this -> Udata['verified'],
581             'rates' => $rates,
582             'public' => $numOfPublicRates,
583             //'pages'   => $pages,
584             'totalRids' => $totalRids
585         ));
586     }
587
588     /*
589      * Function that returnes the 10 top rated urls.
590      * Only urls that meet criteria (minimum 5 votes ) are included.
591      */
```

```
592     public function topRated()
593     {
594         $rates = null;
595         // Array with url as key and average score as value
596         $urlsAndAverageScores = $this -> Redis -> zrevrange('scoreBasedChart', 0, 9, array('withscores' => TRUE));
597         for ($i = 0; $i < count($urlsAndAverageScores); $i++)
598         {
599             // Gets all rids associated with this url
600             $rids = $this -> Redis -> hvals('url:' . $urlsAndAverageScores[$i][0]);
601             // Get title and thumbnail for this url from the first rid (random selection)
602             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
603             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
604             // Set average score and url
605             $rates[$i]['url'] = $urlsAndAverageScores[$i][0];
606             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
607             // Number of voters
608             $rates[$i]['count'] = count($rids);
609         }
610         return View::make('topRated', array(
611             'username' => $this -> Udata['username'],
612             'verified' => $this -> Udata['verified'],
613             'rates' => $rates
614         ));
615     }
616 
617     /*
618      * Function that returnes the 10 most rated urls.
619      */
620     public function mostRated()
621     {
622         $rates = null;
623         // Array with url as key and average score as value
624         $urlsAndRaters = $this -> Redis -> zrevrange('popularityBasedChart', 0, 9, array('withscores' => TRUE));
625         for ($i = 0; $i < count($urlsAndRaters); $i++)
626         {
627             // Gets all rids associated with this url
628             $rids = $this -> Redis -> hvals('url:' . $urlsAndRaters[$i][0]);
629             // Get title and thumbnail for this url from the first rid (random selection)
630             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
631             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
632             // Set average score and url
633             $rates[$i]['url'] = $urlsAndRaters[$i][0];
634             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
635             // Number of voters
636             $rates[$i]['count'] = count($rids);
637         }
638         return View::make('mostRated', array(
639             'username' => $this -> Udata['username'],
640             'verified' => $this -> Udata['verified'],
641             'rates' => $rates
642         ));
643     }
644 
645     /*
646      * Function that returnes the 10 newest rates.
647      * If rids in DB are less than 10, all of them are going to be returned.
648      */
649     public function newestRates()
650     {
651         // Response returned when newest rates are accessed from sidebar
652         if (Request::ajax())
653         {
654 
655             $rates = null;
656             $rid = $this -> Redis -> get('nextRateId');
657             $i = 0;
658             while ($rid > 0 && $i < 5)
659             {
660                 $check = $this -> Redis -> hget('rid:' . $rid, 'url');
661                 if (!empty($check))
662                 {
663                     $rates[$i]['rid'] = $rid;
664 
665                     $i++;
666                     $rid--;
667                 }
668                 else
669                 {
670                     $rid--;
671                 }
672             }
673             $rates = array_reverse($rates);
674             return $rates;
675 
676         }
```

```
677
678         $rates = null;
679         $rid = $this -> Redis -> get('nextRateId');
680         $i = 0;
681         while ($rid > 0 && $i < 10)
682         {
683             $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
684             if (!empty($rates[$i]))
685             {
686                 $rates[$i]['rid'] = $rid;
687                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
688                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
689                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
690                 $i++;
691                 $rid--;
692             }
693             else
694             {
695                 $rid--;
696             }
697         }
698         //print_r($rates[1]);return;
699         return View::make('newestRates', array(
700             'username' => $this -> Udata['username'],
701             'verified' => $this -> Udata['verified'],
702             'rates' => $rates
703         ));
704     }
705
706     /*
707      * Function that returnes the 50 worst rated urls.
708      * Only urls that meet criteria (minimum 5 votes ) are included.
709      */
710     public function worstRated()
711     {
712         $rates = null;
713         // Array with url as key and average score as value
714         $urlsAndAverageScores = $this -> Redis -> zrange('scoreBasedChart', 0, 49, array('withscores' => TRUE));
715         for ($i = 0; $i < count($urlsAndAverageScores); $i++)
716         {
717             // Gets all rids associated with this url
718             $rids = $this -> Redis -> hvals('url:' . $urlsAndAverageScores[$i][0]);
719             // Get title and thumbnail for this url from the first rid (random selection)
720             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
721             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
722             // Set average score and url
723             $rates[$i]['url'] = $urlsAndAverageScores[$i][0];
724             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
725             // Number of voters
726             $rates[$i]['count'] = count($rids);
727         }
728         return View::make('worstRated', array(
729             'username' => $this -> Udata['username'],
730             'verified' => $this -> Udata['verified'],
731             'rates' => $rates
732         ));
733     }
734
735     ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
736
737     public function showUserRates($uid)
738     {
739         //check that the user for which the rates are requested exists
740         if (!$this -> Redis -> exists('uid:' . $uid))
741         {
742             return View::make('userRates', array(
743                 'username' => $this -> Udata['username'],
744                 'verified' => $this -> Udata['verified'],
745                 'message' => 'Oops..You have requested the ratings of a user that does not exist'
746             ));
747         }
748
749         //if the requested user is the same user, show 'my Rates'
750         if ($uid == $this -> Udata['uid'])
751
752         {
753             return Redirect::action('RateController@myRates');
754
755         }
756
757         //count total and public rates
758         $totalRids = $this -> Redis -> llen("own_ratings:" . $uid);
759         $numOfPublicRates = $this -> Redis -> scard('uid:' . $uid . ':publicRids');
760
761         //create a temp redis list of the user's public rids (needed to lrange it for pagination)
```

```
762         $rates = false;
763         if ($numOfPublicRates)
764         {
765             $publicRids = $this -> Redis -> smembers('uid:' . $uid . ':publicRids');
766             sort($publicRids);
767             foreach ($publicRids as $prid)
768             {
769                 $this -> Redis -> lpush('tempList:' . $uid, $prid);
770             }
771             $list = 'tempList:' . $uid;
772 
773             $rids = $this -> Redis -> lrange($list, 0, 4);
774             if (!$rids)
775 
776             {
777                 return false;
778             }
779             $i = 0;
780             foreach ($rids as $rid)
781             {
782                 //get all the rid hash info
783                 $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
784                 //add rid, avg score,rater's username and total # of ratings('count') on that resource
785                 $rates[$i]['rid'] = $rid;
786                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
787                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
788                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
789                 //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
790                 if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
791                 {
792                     $rates[$i]['metoo'] = true;
793                 }
794                 $i = $i + 1;
795             }
796 
797         }
798 
799         return View::make('userRates', array(
800             'username' => $this -> Udata['username'], //the user that is logged in
801             'verified' => $this -> Udata['verified'],
802             'rates' => $rates,
803             'user' => $this -> Redis -> hget('uid:' . $uid, 'username'), //the user of whom the rates i want to show
804             'totalRids' => $totalRids,
805             'public' => $numOfPublicRates,
806             'uid' => $uid,
807         ));
808 
809     }
810 
811     ////////////////////////////////////////////////////////////////////////////
812     /*
813      * Pagination
814      * called with ajax, returns html markup with the rates of one 'page'
815      */
816 
817     public function paginate()
818     {
819         //get the number of page to paginate as sent with ajax
820         $currentRates = Input::get('currentRates');
821 
822         //get the view for which pagination is requested  (timeline-myRates-userRates)
823         $case = Input::get('case');
824         // redis 'lrange' command will be used to retrieve the rids we need.
825         //set the list to apply lrange for each case
826         switch ($case)
827         {
828             case 'timeline' :
829                 $list = "timeline_ratings:" . $this -> Udata['uid'];
830                 $totalRids = $this -> Redis -> llen($list);
831                 break;
832             case 'myRates' :
833                 $list = "own_ratings:" . $this -> Udata['uid'];
834                 $totalRids = $this -> Redis -> llen($list);
835                 break;
836             case 'userRates' :
837                 $userId = Input::get('userId');
838 
839                 //in this case we want to retrieve the public rates of the user, stored in a redis set (not a list as above cases)
840                 $publicSet = 'uid:' . $userId . ':publicRids';
841                 //count the public rids of the user
842                 $totalRids = $this -> Redis -> scard($publicSet);
843                 //create a temp redis list of public rids (needed to lrange it for pagination)
844                 $publicRids = $this -> Redis -> smembers($publicSet);
845                 sort($publicRids);
846                 foreach ($publicRids as $prid)
```

```
847             {
848                 $this -> Redis -> lpush('tempList:' . $userId, $prid);
849             }
850             $list = 'tempList:' . $userId;
851             break;
852         }//end of switch
853         // set num of rates per page
854         $items = 5;
855         //$pages = ceil($totalRids/$items);
856         //initialize rates array (will hold each rate's data needed for the rate markup)
857         $rates = array();
858         $page = "empty";
859 
860         if ($totalRids - $currentRates > 0)//if any more rates to show
861         {
862             //set the range of items to fetch from the ratings list
863             $start = $currentRates;
864             $end = $start + $items - 1;
865             //get the rids to show
866             $rids = $this -> Redis -> lrange($list, $start, $end);
867 
868             $i = 0;
869             foreach ($rids as $rid)
870             {
871                 //get all the rid hash info
872                 $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
873                 //add rid, avg score,rater's username and total # of ratings('count') on that resource
874                 $rates[$i]['rid'] = $rid;
875                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
876                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
877                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
878                 //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
879                 if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
880                 {
881                     $rates[$i]['metoo'] = true;
882                 }
883                 $i = $i + 1;
884             }
885             $page = "";
886             //create the html to return (a 'page' with the rates fetched)
887             foreach ($rates as $rate)
888             {
889                 $page = $page . View::make('rateMarkup', array(
890                     'rate' => $rate,
891                     'username' => $this -> Udata['username']
892                 )) -> render();
893             }
894 
895         }
896 
897         //delete the temp list (if case was userRates)
898         if ($case === 'userRates')
899         {
900             $this -> Redis -> del($list);
901         }
902         //return the page with tha rate markups
903         return $page;
904     }
905 
906     /*
907      * Returns rate markup along with the rate data.
908      * Called by AJAX in Home.
909      */
910     public function rateMarkup()
911     {
912 
913         $rid = Input::get('rate');
914         $rate = $this -> Redis -> hgetall('rid:' . $rid);
915         $rate['rid'] = $rid;
916         $rate['average_score'] = $this -> Redis -> get('url:' . $rate['url'] . ':average_score');
917         $rate['username'] = $this -> Redis -> hget('uid:' . $rate['uid'], 'username');
918         $rate['count'] = $this -> Redis -> hlen('url:' . $rate['url']);
919 
920         //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
921         if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rate['url']))
922         {
923             $rate['metoo'] = true;
924         }
925 
926         return View::make('rateMarkup', array(
927             'username' => $this -> Udata['username'],
928             'verified' => $this -> Udata['verified'],
929             'uid' => $this -> Udata['uid'],
930             'rate' => $rate
931         ));
```

```
932     }
933
934     /*
935      * Returns minified rate markup along with the rate data.
936      * Called by AJAX in signedInBase.
937      */
938     public function miniRateMarkup()
939     {
940
941         $rid = Input::get('rate');
942         $rate['url'] = $this -> Redis -> hget('rid:' . $rid, 'url');
943         $rate['score'] = $this -> Redis -> hget('rid:' . $rid, 'score');
944         $rate['title'] = $this -> Redis -> hget('rid:' . $rid, 'title');
945         $rate['rid'] = $rid;
946         return View::make('minifiedRateMarkup', array('rate' => $rate, ));
947     }
948
949     /*
950      *  Returns all user's notifications and resets unchecked notifications counter
951      */
952     public function notifications()
953     {
954         // Change notifications counter to 0
955         $notificationsCount = $this -> Redis -> set('uncheckedNotifications:' . $this -> Udata['uid'], 0);
956         // Get user's notifications
957         $notificationsPlainText = $this -> Redis -> zrevrange('notifications:' . $this -> Udata['uid'], 0, -1, array('withscores' => TRUE));
958         // Edit each result, values are of the form uid:text ( follow action ) or uid:text:rid ( rate action )
959         $notifications = false;
960         for ($i = 0; $i < count($notificationsPlainText); $i++)
961         {
962             // Seperate value to uid , text and rid ( if exists )
963             $uidAndMessage = explode(':', $notificationsPlainText[$i][0]);
964             $username = $this -> Redis -> hget('uid:' . $uidAndMessage[0], 'username');
965             $notifications[$i]['username'] = $username;
966             $notifications[$i]['uid'] = $uidAndMessage[0];
967             $notifications[$i]['text'] = $uidAndMessage[1];
968             $notifications[$i]['timestamp'] = $notificationsPlainText[$i][1];
969             // If rid exists
970             if (!empty($uidAndMessage[2]))
971             {
972                 $notifications[$i]['rid'] = $uidAndMessage[2];
973                 $notifications[$i]['title'] = $this -> Redis -> hget('rid:' . $notifications[$i]['rid'], 'title');
974                 $notifications[$i]['url'] = $this -> Redis -> hget('rid:' . $notifications[$i]['rid'], 'url');
975                 $notifications[$i]['score'] = $this -> Redis -> hget('url:' . $notifications[$i]['url'].':uscores', $notifications[$i]['uid']);
976             }
977         }
978
979         return View::make('notifications', array(
980             'username' => $this -> Udata['username'],
981             'verified' => $this -> Udata['verified'],
982             'notificationsCount' => null,
983             'notifications' => $notifications,
984         ));
985     }
986
987 } //end-of-class
988
```

BaseModel.php

```
1   <?php
2
3       /*
4        * Model for global declaration of Redis connection on models.
5        * Every model that uses redis connection must extends BaseModel.
6        */
7
8
9
10      class BaseModel
11      {
12          protected $Redis;
13
14          public function __construct() {
15
16              $this->Redis = Redis::connection();
17          }
18
19      }
20
```

Models

BaseModel.php

```php
<?php

    /*
     * Model for global declaration of Redis connection on models.
     * Every model that uses redis connection must extends BaseModel.
     */



    class BaseModel
    {
        protected $Redis;

        public function __construct() {

            $this->Redis = Redis::connection();
        }

    }

```

User.php

```php
<?php

class User extends BaseModel {

    public function __construct()
    {
        parent::__construct();
    }

    /*
     * Return user data with id the givewn argument
     */
    public function getUserData($uid)
    {
        $uid_data['uid']        = $uid;
        $uid_data['fname']      = $this->Redis->hget ('uid:'.$uid, 'fname');
        $uid_data['lname']      = $this->Redis->hget ('uid:'.$uid, 'lname');
        $uid_data['email']      = $this->Redis->hget ('uid:'.$uid, 'email');
        $uid_data['sex']        = $this->Redis->hget ('uid:'.$uid, 'sex');
        $uid_data['age']        = $this->Redis->hget ('uid:'.$uid, 'age');
        $uid_data['thumb']      = $this->Redis->hget ('uid:'.$uid, 'thumb');
        $uid_data['username']   = $this->Redis->hget ('uid:'.$uid, 'username');
        $uid_data['verified']   = $this->Redis->hget ('uid:'.$uid, 'verified');
        $uid_data['ratings']    = $this->Redis->llen ('own_ratings:'.$uid);
        $uid_data['numOfFollowers']  = $this->Redis->zcard('followers:'.$uid);
        $uid_data['numOfFollowings'] = $this->Redis->zcard('followings:'.$uid);

        return $uid_data;
    }

    // Method called from app/controllers/UserController.php
    public function signup($formData)
    {
        // Get next user id
        $id = $this -> Redis -> incr( 'next_user_id' );

        // Generate username from email
        $formData['username'] = strstr($formData['email'], '@', TRUE);

        // Build the validation constraint set.
        $rules = array(

            'username' => 'already_exists:username'
        );

        // Create a new validator instance
        $validator = Validator::make( $formData, $rules );

        if ( $validator -> fails() )
        {
        // Append username with id
            $formData['username'] .= $id;
        }

        Cookie::queue('auth', $formData['auth'], time() + 3600 * 24 * 365);

        // Save to Database
        $this -> Redis -> hmset('uid:'.$id,

            'email'         , $formData['email'],
            'password'      , $formData['password'],
            'username'      , $formData['username'],
            'verifyCode'    , $formData['verifyCode'],
            'timestamp'     , microtime(true),
            'verified'      , 0,
            'auth'          , $formData['auth']
        );

        $metaphone = new DoubleMetaPhone;
        $keywords  = $metaphone->getKeywords($formData['username']);

        // A way to retrieve uid from username, email, auth and verifyCode.
        $this -> Redis -> hset('username', $formData['username'], $id);
        $this -> Redis -> hset('email', $formData['email'], $id);
        $this -> Redis -> hset('auth', $formData['auth'], $id);
        $this -> Redis -> hset('verifyCode', $formData['verifyCode'], $id);
        //required for search user fanctionality
        $this -> Redis -> sadd('user:'.$formData['username'], $id);
    }

    /*
```

```
82      * Implement pagianation on search user. use $start
83      * argument to know form where to start the pagination
84      */
85     public function pagination($start, $currentUid)
86     {
87         $semiStop   = $start + 4; //show 5 records on every page of panination
88
89         // get from the temporary sorted set the records defined by start & semistop
90         $resultUids = $this -> Redis ->zrange('search', $start, $semiStop);
91
92         $recordsNo  = $this -> Redis ->zcard('search'); // count records
93
94         // prepare uid results to be ready for the controller
95         foreach ($resultUids as $uid)
96         {
97             //check if current user is same with search user
98             if ($uid == $currentUid)
99             {
100                 $results[$uid]['sameuser']  = TRUE; //first index of $results is the user id. Important to seperate users data.
101             }
102             else $results[$uid]['sameuser'] = FALSE;
103
104             //retrieve and strore in $results array each user data
105             $results[$uid]['uid']       = $uid;
106             $results[$uid]['username']  = $this -> Redis -> hget('uid:'.$uid, 'username');
107             $results[$uid]['lname']     = $this -> Redis -> hget('uid:'.$uid, 'lname');
108             $results[$uid]['fname']     = $this -> Redis -> hget('uid:'.$uid, 'fname');
109
110             //initialize variable for follow relationship
111             $results[$uid]['followings']  = FALSE;
112
113             //check if user from search result exist in current user followings sorted set
114             $existInFollowings = $this->Redis->zscore('followings:'.$currentUid, $uid);
115
116             //get own_ratings number of the user
117             $results[$uid]['ratings'] = $this->Redis->llen ('own_ratings:'.$uid);
118
119             //check if current user have followigns
120             if ($this->Redis->exists('followings:'.$currentUid))
121             {
122                 if (!empty($existInFollowings))
123                 {
124                     $results[$uid]['followings']  = TRUE;
125                 }
126             }
127         }
128
129         // remove or set (as mark up) the form who call pagination method.
130         if ($semiStop >= $recordsNo - 1)
131         {
132             return array(
133
134                 'results' => $results,
135                 'semistop'   => FALSE
136             );
137         }
138
139         return array(
140
141             'results' => $results,
142             'semistop'   => $semiStop
143         );
144     }
145
146     /*
147      * Serach user action
148      */
149     public function search($data)
150     {
151         $results = FALSE;
152         $count   = 0;
153         $start   = 0;
154         // zunionstore doesn't accept multidimensional array. So need to
155         // separate cases for one or two coming strings from search field
156         if (sizeof($data['keywords'] ) == 1)
157         {
158             $this -> Redis -> zunionstore('search', 1, $data['keywords'][0]);
159         }
160         else
161         {
162             $this -> Redis -> zunionstore('search', 2, $data['keywords'][0], $data['keywords'][1]);
163         }
164
165         if (!$this->Redis->exists('search') )
166         {
```

FollowerGraph.php

```php
<?php


    /*
     *  Follower Graph actions
     */

    class FollowerGraph extends BaseModel
    {
        public function __construct()
        {
            parent::__construct(); //current constructor inherit parent from BaseModel
        }


        public function followAction($data)
        {
            $this -> Redis -> zadd('followers:'.$data['foreign_uid'], microtime(true), $data['current_uid']);   //add current user in sorted set named followers:*  uid*
            $this -> Redis -> zadd('followings:'.$data['current_uid'], microtime(true), $data['foreign_uid']);  //add foreign user in sorted set named followings:* uid*

            $User     = new User();
            $userData = $User->getUserData($data['foreign_uid']);

            // give values because every view with follow/unfollow action contains variables below
            $userData['followings'] = TRUE;
            $userData['sameuser']   = FALSE;

            // Increase notifications counter
            $this -> Redis -> incr('uncheckedNotifications:'. $data['foreign_uid']);
            // Add notification to user's sorted set
            $this -> Redis -> zadd('notifications:' . $data['foreign_uid'], microtime(true), $data['current_uid'].':follows you');
            // If "foreign user" is online notify him
            if (!empty($this -> Redis -> hget('uid:' . $data['foreign_uid'], 'auth')))
            {
                $notification = array(
                    'uid' => $data['foreign_uid'],
                );
                Event::fire('cometNotification', array($notification));
            }

            return $userData;
        }


        public function unFollowAction($data)
        {
            $this -> Redis -> zrem('followers:'.$data['foreign_uid'], $data['current_uid']);  //remove current user from sorted set named followers:*  uid*
            $this -> Redis -> zrem('followings:'.$data['current_uid'], $data['foreign_uid']); //remove foreign user from sorted set named followings:*  uid*
            // Remove notification from sorted set
            $this -> Redis -> zrem('notifications:' . $data['foreign_uid'], $data['current_uid'].':follows you');

            $User     = new User();
            $userData = $User->getUserData($data['foreign_uid']);

            $userData['followings'] = FALSE;
            $userData['sameuser']   = FALSE;

            $rids = $this -> Redis -> lrange('own_ratings:'.$data['foreign_uid'], '0', '-1');   //get all rating from the followed user

            foreach ($rids as $rid)
            {
                $this->Redis->lrem('timeline_ratings:'.$data['current_uid'], '0', $rid);    //remove all ratings(above action) from follower user
            }

            return $userData;
        }

        /*
         *  Retrieve followers from database. Use $uid for each user and $currentUid
         *  for giving value to same user index e.g. When results of action "show uid followers"
         *  contains the signed in user(current) then need to give value to same user variable (hide follow/unfollow button)
         */
        public function getFollowers($uid, $currentUid, $start)
        {
            $semistop  = $start + 4;

            //get followers of $uid
            $followers = $this -> Redis -> zrange('followers:'.$uid, $start, $semistop);

            $recordsNo = $this -> Redis -> zcard('followers:'.$uid); // count records

            if (empty($followers))
```

```
82          {
83              //for error message
84              return $results = FALSE;
85          }
86
87          // instance of User model
88          $User = new User();
89
90          foreach ($followers as $fUid)
91          {
92              // method of user model. get data of user with id the given argument
93              $results[$fUid] = $User->getUserData($fUid);
94
95              $isfolloiwng    =  $this -> Redis -> zscore('followings:'.$uid, $fUid);
96
97              if (!empty($isfolloiwng))
98              {
99                  $results[$fUid]['followings'] = TRUE;
100             }
101             else
102             {
103                 $results[$fUid]['followings'] = FALSE;
104             }
105
106             if ($currentUid == $fUid)
107             {
108                 $results[$fUid]['sameuser'] = TRUE;
109             }
110             else
111             {
112                 $results[$fUid]['sameuser'] = FALSE;
113             }
114         }
115
116         if ($semistop >= $recordsNo - 1)
117         {
118             return array(
119
120                 'results'   => $results,
121                 'semistop'  => FALSE,
122                 'uid'       => $uid
123             );
124         }
125
126         return array(
127
128             'results'   => $results,
129             'semistop'  => $semistop,
130             'uid'       => $uid
131         );
132     }
133
134
135     public function getFollowings($uid, $currentUid, $start)
136     {
137         $semistop  = $start + 4;
138
139         $followings = $this -> Redis -> zrange('followings:'.$uid, $start, $semistop);
140
141         $recordsNo = $this -> Redis -> zcard('followings:'.$uid);
142
143         if (empty($followings))
144         {
145             return $results = FALSE;
146         }
147
148         $User = new User();
149
150         foreach ($followings as $fUid)
151         {
152             $results[$fUid] = $User->getUserData($fUid);
153
154             $results[$fUid]['followings'] = TRUE;
155
156             if ($currentUid == $fUid)
157             {
158                 $results[$fUid]['sameuser'] = TRUE;
159             }
160             else
161             {
162                 $results[$fUid]['sameuser'] = FALSE;
163             }
164         }
165
166         if ($semistop >= $recordsNo - 1)
```

```
167             {
168                 return array(
169 
170                     'results'   => $results,
171                     'semistop'  => FALSE,
172                     'uid'       => $uid
173                 );
174             }
175 
176             return array(
177 
178                 'results'   => $results,
179                 'semistop'  => $semistop,
180                 'uid'       => $uid
181             );
182         }
183     }
184 
185 
186 
187 
```

Rate.php

```
1   <?php
2 
3   class Rate extends BaseModel
4   {
5 
6       public function __construct()
7       {
8           parent::__construct();
9       }
10 
11      /* Function newRate($rdata)
12       * Inserts a new rate in db
13       * Input: $rdata-> array of new rate data
14       */
15      public function newRate($rdata)
16      {
17          //print_r($rdata);return;
18          //get unique rid
19          $rid = $this -> Redis -> incr('nextRateId');
20 
21          //if capture thumb plugin fails store into rid thumb field the below
22          if (!$rdata['thumb'])
23          {
24              $this->Redis->hset('rid:' . $rid, 'thumb', '/images/thumbnails/NoPhotoAvailable.jpg');
25          }
26          else
27          {
28              // call static method from libraries/StorageFactory.php and get
29              // the appropriate instance based on config item placed app/config/app.php
30              $storage = StorageFactory::getStorage();
31 
32              $dataR['rid']       =  $rid;
33              $dataR['thumbUrl']  =  $rdata['thumb'];
34 
35              $storage->storeRateThumb($dataR);
36          }
37 
38          //insert data to hashes and sets
39          //hash    rid:xxx   url     title  score   e.t.c.
40          $this -> Redis -> hmset('rid:' . $rid, 'url', $rdata['url'], 'title', $rdata['title'], 'timestamp', microtime(true), 'uid', $rdata['uid'], 'score', $rdata['score'], 'review', $rdata['review'], 'public', $rdata['public']);
41          //hash    url:<string>   uid  rid
42          $this -> Redis -> hset('url:' . $rdata['url'], $rdata['uid'], $rid);
43          //hash    url:<string>:rscores   rid  score
44          $this -> Redis -> hset('url:' . $rdata['url'] . ':rscores', $rid, $rdata['score']);
45          //hash    url:<string>:uscores   uid  score
46          $this -> Redis -> hset('url:' . $rdata['url'] . ':uscores', $rdata['uid'], $rdata['score']);
47          //set     uid:xxx:rated_urls    set-of-urls
48          $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':rated_urls', $rdata['url']);
49          //set     uid:xxx:rids          set of rids
50          $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':rids', $rid);
51          // add the rid into the score indexing set
52          $this -> Redis -> sadd('score:' . $rdata['score'], $rid);
53          //if rate is public, add it to the public rates set:
54          if ($rdata['public'])
55          {
56              $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':publicRids', $rid);
57              $this -> addToChart($rdata['url']);
58              Event::fire('cometNewRate', $rid);
59          }
60          //take care of the average score
61          if ($rdata['newUrl'])//new rate on non-existing url
62          {
63              //average score = score
64              $this -> Redis -> set('url:' . $rdata['url'] . ':average_score', $rdata['score']);
65          }
66          else//case = existingURL:new rate on existing url
67          {
68              //compute new average
69              $this -> average_score($rdata['url']);
70          }
71          //push rate to user's + followers lists
72          $this -> fanout($rdata['uid'], $rid, 'new', $rdata['public']);
73 
74          //add the rid to the keywords index sets)
75          if (array_key_exists('keywords', $rdata))
76          {
77              foreach ($rdata['keywords'] as $keyword)
78              {
79                  //add the rid in the set of this word
80                  $this -> Redis -> sadd('word:' . $keyword, $rid);
81              }
```

```
82          }
83
84          return TRUE;
85      }//end-of-newRate method
86
87      ////////////////////////////////////////////////////////////////////////////////////////////////
88
89      /*
90       * Function edit($data)
91       * Edits a Rate
92       * Input: array of edited rate data
93       * Updates a rate (new score,review",timestamp) and repushes it to lists (timelines + own)
94       */
95      public function edit($data)
96      {
97
98          // get rate id - if needed
99          if (empty($data['rid']))
100         {
101             $data['rid'] = $this -> Redis -> hget('url:' . $data['url'], $data['uid']);
102         }
103         if ($data['public'] === "old")// edit case from quick rate sidebar form
104         {
105             //retrieve the old value
106             $data['public'] = $this -> Redis -> hget('rid:' . $data['rid'], 'public');
107         }
108         //check if new review is given (if not, old one stays)
109         if (empty($data['review']))
110         {
111             //retrieve the old review
112             $data['review'] = $this -> Redis -> hget('rid:' . $data['rid'], 'review');
113         }
114         //get the old score
115         $oldScore = $this -> Redis -> hget('rid:' . $data['rid'], 'score');
116         //remove the rid form the old score index set:
117         $this -> Redis -> srem('score:' . $oldScore, $data['rid']);
118         //add the rid to the new score index set
119         $this -> Redis -> sadd('score:' . $data['score'], $data['rid']);
120         //update rid hash fields
121         $this -> Redis -> hmset('rid:' . $data['rid'], 'score', $data['score'], 'review', $data['review'], 'timestamp', microtime(true), 'public', $data['public']);
122         //update new score to hashes 2,3
123         $this -> Redis -> hset('url:' . $data['url'] . ':rscores', $data['rid'], $data['score']);
124         $this -> Redis -> hset('url:' . $data['url'] . ':uscores', $data['uid'], $data['score']);
125         //compute new average
126         $this -> average_score($data['url']);
127         //fanout edited rate
128         $this -> fanout($data['uid'], $data['rid'], 'edit', $data['public']);
129         //if public, add it to user's set of public rids (if already there, nothing happens)
130         if ($data['public'])
131         {
132             $this -> Redis -> sadd('uid:' . $data['uid'] . ':publicRids', $data['rid']);
133             $this -> addToChart($data['url']);
134         }
135         else//remove from public set the rid
136         {
137             $this -> Redis -> srem('uid:' . $data['uid'] . ':publicRids', $data['rid']);
138         }
139
140         return TRUE;
141     }
142
143     ////////////////////////////////////////////////////////////////////////////////////////////////
144
145     /*
146      * Function fanout($uid, $rid, $case, $public)
147      * Input: user Id, rate Id, $case -> edit or new rate(on existing or not-existing url), $public (e.g. only public rates shoyl be fanned out in others timelines)
148      * Pushes a new/edited rate to user's and followers lists
149      */
150     private function fanout($uid, $rid, $case, $public)
151     {
152
153         //push rate to user's lists
154         if ($case == 'edit')
155         {
156             //remove rate from user's lists to repush it
157             $this -> Redis -> lrem('own_ratings:' . $uid, '0', $rid);
158             $this -> Redis -> lrem('timeline_ratings:' . $uid, '0', $rid);
159         }
160         //(re)push rate
161         $this -> Redis -> lpush('own_ratings:' . $uid, $rid);
162         $this -> Redis -> lpush('timeline_ratings:' . $uid, $rid);
163
164         //Get user's followers and push rate to their timelines
165         $followers = $this -> Redis -> zrange('followers:' . $uid, '0', '-1');
166
```

```
167         if (!empty($followers))
168         {
169             switch ($case)
170             {
171                 case 'edit' :
172                     foreach ($followers as $follower_uid)
173                     {
174                         //remove rid from timeline
175                         $this -> Redis -> lrem('timeline_ratings:' . $follower_uid, '0', $rid);
176                         //if is public, repush it
177                         if ($public)
178                         {
179                             $this -> Redis -> lpush('timeline_ratings:' . $follower_uid, $rid);
180                             $isOnline = $this -> Redis -> hget('uid:' . $follower_uid, 'auth');
181                             if (!empty($isOnline))
182                             {
183                                 $update = array(
184                                     'updateTimeline' => $follower_uid,
185                                     'ridBase' => $rid
186                                 );
187                                 Event::fire('cometTimelineUpdate', array($update));
188                             }
189                         }
190 
191                     }
192                     break;
193 
194                 case 'new' :
195                     //if new rate is public, push it to all followers' timelines
196                     if ($public)
197                     {
198                         foreach ($followers as $follower_uid)
199                         {
200                             $this -> Redis -> lpush('timeline_ratings:' . $follower_uid, $rid);
201                             $isOnline = $this -> Redis -> hget('uid:' . $follower_uid, 'auth');
202                             // If follower is online update timeline
203                             if (!empty($isOnline))
204                             {
205                                 $update = array(
206                                     'updateTimeline' => $follower_uid,
207                                     'ridBase' => $rid
208                                 );
209                                 Event::fire('cometTimelineUpdate', array($update));
210                             }
211                         }
212                     }
213 
214                     break;
215             }
216 
217         }
218     }
219 
220     ////////////////////////////////////////////////////////////////////////////////////////////////
221 
222     /* Function average_score($url)
223      * Computes and updates average score for a resource
224      * (needs to run on edit rate + new rate on existing url)
225      */
226     private function average_score($url)
227     {
228         //get all the scores for the url
229         $scores = $this -> Redis -> hvals('url:' . $url . ':rscores');
230         //get the number of scores
231         $sum = $this -> Redis -> hlen('url:' . $url . ':rscores');
232         $average = array_sum($scores) / $sum;
233         $average = number_format($average, 1, '.', '');
234         $this -> Redis -> set('url:' . $url . ':average_score', $average);
235     }
236 
237     ////////////////////////////////////////////////////////////////////////////////////////////////
238 
239     /* Function search($sdata)
240      * Takes search data (criteria,uid e.t.c.)  passed from Rate Controller
241      * Returns multidimensional array $searchResults holding the rating objects (rid:xxx hashes) as arrays
242      */
243     public function search($sdata)
244     {
245         //initialize search results array (will be returned to controller)
246         $searchResults = false;
247         //$rates = false;
248         //redis keyword-index is in form : 'word:keyword {set of rids}' so we add the string 'word:' before each keyword the controller sent
249         array_walk($sdata['keywords'], function(&$value)
250         {
251             $value = 'word:' . $value;
```

```
252     });
253     //store ALL the rids associated with every keyword to a temporary redis set (will be deleted after intersection(s) needed for the search)
254     //this temp set needs to be unique (otherwise conflicts may arise when 2+ users search simultaneously)
255     $this -> Redis -> sunionstore($sdata['uid'] . ':temp1', $sdata['keywords']);
256     //e.g. 3:temp1 (set of rids) where 3=the id of the user
257     // if no rids found , return false
258     if (!$this -> Redis -> scard($sdata['uid'] . ':temp1'))
259     {
260         return false;
261     }
262     //repeat the above process to retrieve rids based on score range, IF search is advanced search
263     if (array_key_exists('scores', $sdata))
264     {
265         //e.g. if score[0]=2, score[1]=5, we want the range of scores in an array: 2,3,4,5
266         $scores = range($sdata['scores'][0], $sdata['scores'][1]);
267
268         //add the string 'score:' before each score (to get them in the form of redis index keys score:x)
269         array_walk($scores, function(&$value, $key)
270         {
271             $value = 'score:' . $value;
272         });
273         //get all the rids within score range in a temp set
274         $this -> Redis -> sunionstore($sdata['uid'] . ':temp3', $scores);
275         //intersect the 2 temp sets to get only the rids matching the keywords AND the score range (and store the result in uid:temp1 set)
276         $this -> Redis -> sinterstore($sdata['uid'] . ':temp1', $sdata['uid'] . ':temp3', $sdata['uid'] . ':temp1');
277         //delete the temp3 set
278         $this -> Redis -> del($sdata['uid'] . ':temp3');
279     }
280
281     switch ($sdata['searchIn'])
282     {
283         case '"own"' :
284             //intersect user's own rids with rids associated with search keywords (and scores if advanced search)
285             $rids = $this -> Redis -> sinter('uid:' . $sdata['uid'] . ':rids', $sdata['uid'] . ':temp1');
286             rsort($rids);
287             break;
288
289         case '"followings"' :
290             //get all followings' uids
291             $fuids = $this -> Redis -> zrange('followings:' . $sdata['uid'], 0, -1);
292
293             //convert the above uids in 'uid:xxx:publicRids' -> the key of the set with the public rids of each user
294             array_walk($fuids, function(&$value, $key)
295             {
296                 $value = 'uid:' . $value . ':publicRids';
297             });
298
299             //put all followings' public rids in a temp redis set
300             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $fuids);
301
302             //intersect followings' public rids with rids associated with search keywords
303             $rids = $this -> Redis -> sinter($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp1');
304             rsort($rids);
305
306             break;
307         case '"both"' :
308             //get all followings' uids
309             $fuids = $this -> Redis -> zrange('followings:' . $sdata['uid'], 0, -1);
310
311             //convert the above uids(xxx) in 'uid:xxx:publicRids' -> the key of the set with the public rids of each user
312             array_walk($fuids, function(&$value, $key)
313             {
314                 $value = 'uid:' . $value . ':publicRids';
315             });
316
317             //gather all  followings' public rids in a temp redis set
318             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $fuids);
319
320             //add to this set the user's own rids (public or not)
321             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp2', 'uid:' . $sdata['uid'] . ':rids');
322
323             //intersect own+followings' rids with rids associated with search keywords
324             $rids = $this -> Redis -> sinter($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp1');
325             rsort($rids);
326
327             break;
328     }//end-of-switch
329
330     //delete the remaining temp redis sets
331     $this -> Redis -> del($sdata['uid'] . ':temp1');
332     $this -> Redis -> del($sdata['uid'] . ':temp2');
333
334     $i = 0;
335
336     foreach ($rids as $rid)
```

```
337         {
338             //insert the rid object (array) into the searchResults array that will be returned to the controller
339             $searchResults[$i] = $this -> Redis -> hgetall('rid:' . $rid);
340             //print_r($searchResults[$i]);return;
341             //add rid, avg score and following's username
342             $searchResults[$i]['rid'] = $rid;
343             $searchResults[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
344             $searchResults[$i]['username'] = $this -> Redis -> hget('uid:' . $searchResults[$i]['uid'], 'username');
345             //number of total ratings on this resource
346             $searchResults[$i]['count'] = $this -> Redis -> hlen('url:' . $searchResults[$i]['url']);
347             //check if user has rated it (we need to show this info on other's ratings, e.g. 'You have rated this')
348             if ($this -> Redis -> sismember('uid:' . $sdata['uid'] . ':rated_urls', $searchResults[$i]['url']))
349             {
350                 $searchResults[$i]['metoo'] = true;
351             }
352 
353             $i++;
354         }
355         return $searchResults;
356     }
357 
358     ////////////////////////////////////////////////////////////////////////////
359     public function delete($rid, $user)
360     {
361         // Retrieve url
362         $url = $this -> Redis -> hget('rid:' . $rid, 'url');
363         // Retrieve uid
364         $uid = $this -> Redis -> hget('rid:' . $rid, 'uid');
365         // Retrieve thumb url
366         $thumb  = $this -> Redis -> hget('rid:' . $rid, 'thumb');
367 
368         $rate['title'] = $this -> Redis -> hget('rid:' . $rid, 'title');
369         $rate['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
370         // Check if user is on of the raters
371         if ($uid != $user)
372         {
373             return FALSE;
374         }
375         // count raters
376         $countRaters = $this -> Redis -> hlen('url:' . $url);
377         $rate['raters'] = $countRaters;
378         // Commands associated with Delete rate, see use cases 4 & 5 (_Data_Layout/Use_Cases - Queries.doc)
379         //if rate is public, delete it from the user's set of public rids
380         if ($this -> Redis -> hget('rid:' . $rid, 'public'))
381         {
382             $this -> Redis -> srem('uid:' . $uid . ':publicRids', $rid);
383         }
384         $redis = $this -> Redis -> pipeline();
385         $redis -> del('rid:' . $rid);
386         $redis -> hdel('url:' . $url, $uid);
387         $redis -> hdel('url:' . $url . ':rscores', $rid);
388         $redis -> hdel('url:' . $url . ':uscores', $uid);
389         $redis -> lrem('own_ratings:' . $uid, 0, $rid);
390         $redis -> lrem('timeline_ratings:' . $uid, 0, $rid);
391         $redis -> srem('uid:' . $uid . ':rated_urls', $url);
392         $redis -> srem('uid:' . $uid . ':rids', $rid);
393         $redis -> execute();
394 
395         if ($countRaters == 1)
396         {
397             $this -> Redis -> del('url:' . $url . ':average_score');
398 
399             if (strpos($thumb, 's3') !== FALSE)
400             {
401                 $thumbName = explode('com/', $thumb);
402 
403                 $s3obj  = AWS::get('s3');
404 
405                 $result = $s3obj->deleteObject(array(
406 
407                     'Bucket' => 'ratethumb',
408                     'Key'    => $thumbName[1]
409                 ));
410             }
411             else
412             {
413                 $thumbName = explode('/', $thumb);
414 
415                 unlink('images/thumbnails/'.$thumbName[3]);
416             }
417         }
418         else
419         {
420             $this -> average_score($url);
421         }
```

```
422         $followers = $this -> Redis -> zrange('followers:' . $uid, 0, -1);
423         foreach ($followers as $follower)
424         {
425             $this -> Redis -> lrem('timeline_ratings:' . $follower, 0, $rid);
426             // remove notification from users sorted set
427             $notificationRid = $this -> Redis -> hget('url:' . $url, $follower);
428             $this -> Redis -> zrem('notifications:' . $follower, $uid . ':rated:' . $notificationRid);
429         }
430         $this -> removeFromChart($url);
431         return array('rate' => $rate);
432     }
433
434     private function addToChart($url)
435     {
436         $votes = $this -> Redis -> hlen('url:' . $url . ':rscores');
437         if ($votes >= 4)
438         {
439             $average_score = $this -> Redis -> get('url:' . $url . ':average_score');
440             $this -> Redis -> zadd('scoreBasedChart', $average_score, $url);
441         }
442         $this -> Redis -> zadd('popularityBasedChart', $votes, $url);
443
444     }
445
446     private function removeFromChart($url)
447     {
448         $redis = $this -> Redis -> pipeline();
449         $redis -> zrem('scoreBasedChart', $url);
450         $redis -> zrem('popularityBasedChart', $url);
451         $redis -> execute();
452     }
453
454 }
455
```

Libraries

server.js

```
// Initialization
var express = require('express');
var redis = require('redis');
var io = require('socket.io');
var http = require('http');
var app = express();
var server = http.createServer(app).listen(3000, 'localhost');

console.log('Server successfully started...');
// Client connection
io.listen(server).on('connection', function(client)
{
    var redisClientNews = redis.createClient();
    redisClientNews.subscribe('newRates');
    var redisClientNotifications = redis.createClient();
    var redisClient = redis.createClient();

    client.on('uDataNotifications', function(data)
    {
        var subscribed = false;
        var redisClientAuth = redis.createClient();
        var uDataNotifications = data.split(',');
        redisClientAuth.hget('uid:' + uDataNotifications[0], 'auth', function(err, reply)
        {
            if (reply == uDataNotifications[1])
            {
                redisClientNotifications.subscribe('notifications:' + uDataNotifications[0]);
                subscribed = true;
            }
        });
        redisClientAuth.quit();
        redisClientAuth.on("end", function(err)
        {
            if (!subscribed)
            {
                client.disconnect();
            }
        });
    });

    // Recieve credentials and subscribe client to Timeline update channel or kill connection
    client.on('uData', function(data)
    {

        var subscribed = false;
        var redisClientAuth = redis.createClient();
        var uData = data.split(',');
        redisClientAuth.hget('uid:' + uData[0], 'auth', function(err, reply)
        {
            if (reply == uData[1])
            {
                redisClient.subscribe('timeline:' + uData[0]);
                subscribed = true;
            }
        });
        redisClientAuth.quit();
        redisClientAuth.on("end", function(err)
        {
            if (!subscribed)
            {
                client.disconnect();
            }
        });
    });
    // When a string is published on channel Timeline:xxx send it to client
    redisClient.on('message', function(channel, message)
    {
        client.emit(channel, message);
    });

    redisClientNews.on('message', function(channel, message)
    {
        client.emit(channel, message);
    });

    redisClientNotifications.on('message', function(channel, message)
    {
        client.emit(channel, message);
    });

    client.on('disconnect', function()
    {
```

```
83            redisClient.quit();
84            redisClientNews.quit();
85            redisClientNotifications.quit();
86        });
87
88    });
89
90
```

StorageFactory.php

```php
<?php

    /*
     * Design pattern factory
     * Every instance of the class return the appropriate instance
     * based on config item 'cloudStorage' placed into app/config/app.php
     */

    class StorageFactory
    {
        public static function getStorage()
        {
            if (Config::get('app.cloudStorage'))
            {
                $storage = new CloudStorage();
            }

            else
            {
                $storage = new LocalStorage();
            }

            return $storage;
        }
    }

    /*
     * Specify the method below
     */

    interface StorageInterface
    {
        public function storeAvatarThumb($dataA);
        public function storeRateThumb($dataR);
    }

    /*
     * Contains Redis instance and intreface's methods
     */

    class Storage implements StorageInterface
    {
        protected $Redis;

        public function __construct()
        {
            $this->Redis = Redis::connection();
        }

        public function storeAvatarThumb($dataA)
        {
             return;
        }

        public function storeRateThumb($dataR)
        {
             return;
        }
    }

    /*
     * Hanlde local storage of User avatar & Rate avatr
     */

    class LocalStorage extends Storage
    {
        function __construct()
        {
            parent::__construct();
        }

        public function storeAvatarThumb($dataL)
        {
            //move the uploaded file from server's temp location to 'public_path()./images/avatar/'
            Input::file('avatar')->move( public_path().'/images/avatar/', $dataL['filename']);

            // store url into user data
            $this->Redis->hset ('uid:'.$dataL['uid'], 'thumb', 'images/avatar/'.$dataL['filename']);

            return;
        }

        public function storeRateThumb($dataL)
```

```
80          {
81              // store url into rid data
82              $this->Redis->hset('rid:' . $dataL['rid'], 'thumb', $dataL['thumbUrl']);
83
84              return;
85          }
86      }
87
88      /*
89       * Hanlde cloud storage of User avatar & Rate avatar
90       */
91      class CloudStorage extends Storage
92      {
93          function __construct()
94          {
95              parent::__construct();
96          }
97
98          public function storeAvatarThumb($dataC)
99          {
100             $url = $this->s3PutRequest($dataC);
101
102             // store url in user data
103             $this->Redis->hset ('uid:'.$dataC['uid'], 'thumb', $url);
104
105             return;
106         }
107
108         public function storeRateThumb($dataC)
109         {
110             $thumbId = $this->Redis->get('nextThumbId');
111
112             //prepare data for parsing to s3PutRequest()
113             $dataC['filename']  = 'thumb'.$thumbId.'.jpg';
114             $dataC['filepath']  = public_path().$dataC['thumbUrl'];
115             $dataC['filetype']  = 'image/jpeg';
116             $dataC['bucket']    = 'ratethumb';
117
118             $url = $this->s3PutRequest($dataC);
119
120             // store url in rid data
121             $this->Redis->hset('rid:'.$dataC['rid'], 'thumb', $url);
122
123             // thumbnail captured when a new rate created and stored into server.
124             // Delete that thumb after cloud storage action
125             unlink($dataC['filepath']);
126
127             return;
128         }
129
130         private function s3PutRequest($DataS3)
131         {
132             //instance of AWS SDK
133             $s3obj = AWS::get('s3');
134
135             // put request store given thumb and return the desirable url
136             $url = $s3obj->putObject(array(
137
138                     'Bucket'     => $DataS3['bucket'],   // sub folder of s3 storage
139                     'Key'        => $DataS3['filename'], // set the name of the uploaded file
140                     'SourceFile' => $DataS3['filepath'], // where find the file
141                     'ACL'        => 'public-read',       // set access permissions
142                     'ContentType'=> $DataS3['filetype'], // set type of file
143                 ));
144
145             return $url['ObjectURL'];
146         }
147     }
148
149
150
151
152
153
```

UrlManagement.php

```php
<?php
class UrlManagement extends BaseModel
{
    public function __construct()
    {
        parent::__construct();
    }

    //class variable $mimetypes: array with all the known filetypes
    public $mimeTypes = array(
        "323" => "text/h323",
        "acx" => "application/internet-property-stream",
        "ai" => "application/postscript",
        "aif" => "audio/x-aiff",
        "aifc" => "audio/x-aiff",
        "aiff" => "audio/x-aiff",
        "asf" => "video/x-ms-asf",
        "asr" => "video/x-ms-asf",
        "asx" => "video/x-ms-asf",
        "au" => "audio/basic",
        "avi" => "video/x-msvideo",
        "axs" => "application/olescript",
        "bas" => "text/plain",
        "bcpio" => "application/x-bcpio",
        "bin" => "application/octet-stream",
        "bmp" => "image/bmp",
        "c" => "text/plain",
        "cat" => "application/vnd.ms-pkiseccat",
        "cdf" => "application/x-cdf",
        "cer" => "application/x-x509-ca-cert",
        "class" => "application/octet-stream",
        "clp" => "application/x-msclip",
        "cmx" => "image/x-cmx",
        "cod" => "image/cis-cod",
        "cpio" => "application/x-cpio",
        "crd" => "application/x-mscardfile",
        "crl" => "application/pkix-crl",
        "crt" => "application/x-x509-ca-cert",
        "csh" => "application/x-csh",
        "css" => "text/css",
        "dcr" => "application/x-director",
        "der" => "application/x-x509-ca-cert",
        "dir" => "application/x-director",
        "dll" => "application/x-msdownload",
        "dms" => "application/octet-stream",
        "doc" => "document/msword",
        "dot" => "application/msword",
        "dvi" => "application/x-dvi",
        "dxr" => "application/x-director",
        "eps" => "application/postscript",
        "etx" => "text/x-setext",
        "evy" => "application/envoy",
        "exe" => "application/octet-stream",
        "fif" => "application/fractals",
        "flr" => "x-world/x-vrml",
        "gif" => "image/gif",
        "gtar" => "application/x-gtar",
        "gz" => "application/x-gzip",
        "h" => "text/plain",
        "hdf" => "application/x-hdf",
        "hlp" => "application/winhlp",
        "hqx" => "application/mac-binhex40",
        "hta" => "application/hta",
        "htc" => "text/x-component",
        "ico" => "image/x-icon",
        "ief" => "image/ief",
        "iii" => "application/x-iphone",
        "ins" => "application/x-internet-signup",
        "isp" => "application/x-internet-signup",
        "png" => "image/png",
        "jfif" => "image/pipeg",
        "jpe" => "image/jpeg",
        "jpeg" => "image/jpeg",
        "jpg" => "image/jpeg",
        "js" => "application/x-javascript",
        "latex" => "application/x-latex",
        "lha" => "application/octet-stream",
        "lsf" => "video/x-la-asf",
        "lsx" => "video/x-la-asf",
        "lzh" => "application/octet-stream",
        "m13" => "application/x-msmediaview",
```

```
82      "m14" => "application/x-msmediaview",
83      "m3u" => "audio/x-mpegurl",
84      "man" => "application/x-troff-man",
85      "mdb" => "application/x-msaccess",
86      "me" => "application/x-troff-me",
87      "mht" => "message/rfc822",
88      "mhtml" => "message/rfc822",
89      "mid" => "audio/mid",
90      "mny" => "application/x-msmoney",
91      "mov" => "video/quicktime",
92      "movie" => "video/x-sgi-movie",
93      "mp2" => "video/mpeg",
94      "mp3" => "audio/mpeg",
95      "mpa" => "video/mpeg",
96      "mpe" => "video/mpeg",
97      "mpeg" => "video/mpeg",
98      "mpg" => "video/mpeg",
99      "mpp" => "application/vnd.ms-project",
100     "mpv2" => "video/mpeg",
101     "ms" => "application/x-troff-ms",
102     "mvb" => "application/x-msmediaview",
103     "nws" => "message/rfc822",
104     "oda" => "application/oda",
105     "p10" => "application/pkcs10",
106     "p12" => "application/x-pkcs12",
107     "p7b" => "application/x-pkcs7-certificates",
108     "p7c" => "application/x-pkcs7-mime",
109     "p7m" => "application/x-pkcs7-mime",
110     "p7r" => "application/x-pkcs7-certreqresp",
111     "p7s" => "application/x-pkcs7-signature",
112     "pbm" => "image/x-portable-bitmap",
113     "pdf" => "document/pdf",
114     "pfx" => "application/x-pkcs12",
115     "pgm" => "image/x-portable-graymap",
116     "pko" => "application/vnd.ms-pkipko",
117     "pma" => "application/x-perfmon",
118     "pmc" => "application/x-perfmon",
119     "pml" => "application/x-perfmon",
120     "pmr" => "application/x-perfmon",
121     "pmw" => "application/x-perfmon",
122     "pnm" => "image/x-portable-anymap",
123     "pot" => "application/vnd.ms-powerpoint",
124     "ppm" => "image/x-portable-pixmap",
125     "pps" => "application/vnd.ms-powerpoint",
126     "ppt" => "application/vnd.ms-powerpoint",
127     "prf" => "application/pics-rules",
128     "ps" => "application/postscript",
129     "pub" => "application/x-mspublisher",
130     "qt" => "video/quicktime",
131     "ra" => "audio/x-pn-realaudio",
132     "ram" => "audio/x-pn-realaudio",
133     "ras" => "image/x-cmu-raster",
134     "rgb" => "image/x-rgb",
135     "rmi" => "audio/mid",
136     "roff" => "application/x-troff",
137     "rtf" => "application/rtf",
138     "rtx" => "text/richtext",
139     "scd" => "application/x-msschedule",
140     "sct" => "text/scriptlet",
141     "setpay" => "application/set-payment-initiation",
142     "setreg" => "application/set-registration-initiation",
143     "sh" => "application/x-sh",
144     "shar" => "application/x-shar",
145     "sit" => "application/x-stuffit",
146     "snd" => "audio/basic",
147     "spc" => "application/x-pkcs7-certificates",
148     "spl" => "application/futuresplash",
149     "src" => "application/x-wais-source",
150     "sst" => "application/vnd.ms-pkicertstore",
151     "stl" => "application/vnd.ms-pkistl",
152     "stm" => "text/html",
153     "svg" => "image/svg+xml",
154     "sv4cpio" => "application/x-sv4cpio",
155     "sv4crc" => "application/x-sv4crc",
156     "t" => "application/x-troff",
157     "tar" => "application/x-tar",
158     "tcl" => "application/x-tcl",
159     "tex" => "application/x-tex",
160     "texi" => "application/x-texinfo",
161     "texinfo" => "application/x-texinfo",
162     "tgz" => "application/x-compressed",
163     "tif" => "image/tiff",
164     "tiff" => "image/tiff",
165     "tr" => "application/x-troff",
166     "trm" => "application/x-msterminal",
```

```
167         "tsv" => "text/tab-separated-values",
168         "txt" => "text/plain",
169         "uls" => "text/iuls",
170         "ustar" => "application/x-ustar",
171         "vcf" => "text/x-vcard",
172         "vrml" => "x-world/x-vrml",
173         "wav" => "audio/x-wav",
174         "wcm" => "application/vnd.ms-works",
175         "wdb" => "application/vnd.ms-works",
176         "wks" => "application/vnd.ms-works",
177         "wmf" => "application/x-msmetafile",
178         "wps" => "application/vnd.ms-works",
179         "wri" => "application/x-mswrite",
180         "wrl" => "x-world/x-vrml",
181         "wrz" => "x-world/x-vrml",
182         "xaf" => "x-world/x-vrml",
183         "xbm" => "image/x-xbitmap",
184         "xla" => "application/vnd.ms-excel",
185         "xlc" => "application/vnd.ms-excel",
186         "xlm" => "application/vnd.ms-excel",
187         "xls" => "application/vnd.ms-excel",
188         "xlsx" => "vnd.ms-excel",
189         "xlt" => "application/vnd.ms-excel",
190         "xlw" => "application/vnd.ms-excel",
191         "xof" => "x-world/x-vrml",
192         "xpm" => "image/x-xpixmap",
193         "xwd" => "image/x-xwindowdump",
194         "z" => "application/x-compress",
195         "zip" => "application/zip"
196     );
197
198     /*
199      * Function urlExists($url)
200      * Checks if url exists in database.
201      *
202      */
203     public static function urlExists($url)
204     {
205         $Redis = Redis::connection();
206
207         if (!$Redis -> hlen('url:' . $url))
208         {
209             return false;
210         }
211         return true;
212
213     }
214
215     ////////////////////////////////////////////////////////////////////
216
217     /*
218      * Function urlType($url)
219      * Returns the type of url: 'webpage' or filetype (image, document e.t.c)
220      */
221     public function urlType($url)
222     {
223         //get rid of variables in the url - if any
224         $url = strtok($url, '?');
225         $urlinfo = pathinfo($url);
226         //example: url="http://www.davey.com/media/1001/home-tree.png"
227         //pathinfo returns:
228         //Array ( [dirname] => http://www.davey.com/media/1001 [basename] => home-tree.png [extension] => png [filename] => home-tree )
229
230         //check if is webpage  (case url has no extension  or extension does not refer to known filetype)
231         if (empty($urlinfo['extension']) || !array_key_exists($urlinfo['extension'], $this -> mimeTypes))
232         {
233             $type = "webpage";
234             return $type;
235         }
236         // url is file, get filetype
237         $ext = Str::lower($urlinfo['extension']);
238         $type = strtok($this -> mimeTypes[$ext], '/');
239         //e.g. if $ext='jpg', $mimeTypes['jpg'] = image/jpeg, strtok gives 'image'
240         return $type;
241     }
242
243     ////////////////////////////////////////////////////////////////////////////////////
244     /*
245      * Function urlGetTitle($url)
246      * Returns a title for a rate
247      * Calls  function UrlType() to get the type of url, and:
248      * 1.If  url is a webpage,set as title the html 'title' tag of the <head> section.
249      * 2.If extension exists and is a known filetype, then title=filename
250      */
251
```

```
252     public function urlGetTitle($url)
253     {
254         $type = $this -> urlType($url);
255         if ($type == 'webpage')
256         {
257             include_once ('simple_html_dom.php');
258             // Create a DOM object
259             $html = new simple_html_dom();
260             // Load HTML from a URL
261             $html -> load_file($url);
262             //get title
263             if ($html -> find('title', 0))
264             {
265                 $title = trim($html -> find('title', 0) -> innertext);
266                 return $title;
267             }
268             // if no title retrieved, set url basename as title
269             $url = strtok($url, '?');
270             $urlinfo = pathinfo($url);
271             $title = $urlinfo['basename'];
272             return $title;
273         }
274 
275         //type is file, set title=filename
276         $url = strtok($url, '?');
277         $urlinfo = pathinfo($url);
278         $title = $urlinfo['filename'];
279         return $title;
280 
281     }
282 
283     public function urlGetThumb($url)
284     {
285         // Online Thumb generator - Download Image
286         $thumb = 'http://free.pagepeeker.com/v2/thumbs.php?size=x&url=' . $url;
287         // Online Thumb generator - Check if thumb is ready Image
288         $thumbReady = 'http://free.pagepeeker.com/v2/thumbs_ready.php?size=x&url=' . $url;
289         // Ignore HTTP error file_get_contents() function might throw
290         $context = stream_context_create(array('http' => array('ignore_errors' => true), ));
291         // Get JSON code API returns
292         $json = file_get_contents($thumbReady, false, $context);
293         // Decode JSON code
294         $response = json_decode($json);
295         $i = 0;
296         // Check for up to 30 seconds if thumb is ready
297         while (!$response -> IsReady && $i < 30)
298         {
299             sleep(1);
300             $json = file_get_contents($thumbReady, false, $context);
301             $response = json_decode($json);
302             $i++;
303         }
304         if (!$response -> IsReady)
305         {
306             return $url = FALSE;
307         }
308         else
309         {
310             // Get thumb
311             $thumb = file_get_contents($thumb, false, $context);
312             // Get next thumb ID
313             $thumbId = $this -> Redis -> incr('nextThumbId');
314             $name = 'thumb' . $thumbId . '.jpg';
315             $destinationPath = public_path() . '/images/thumbnails/' . $name;
316             // Save thumb
317             file_put_contents($destinationPath, $thumb);
318 
319             $url = '/images/thumbnails/' . $name;
320 
321             return $url;
322         }
323     }
324 }
325 
```

[i] Οι τίτλοι των άλλων δύο Μεταπτυχιακή Διατριβή:
Εφαρμογή αξιολόγησης διαδικτυακών πόρων με χαρακτηριστικά κοινωνικού δικτύου
Εφαρμογή αξιολόγησης διαδικτυακών πόρων με λειτουργικότητα ειδοποιήσεων σε πραγματικό χρόνο