ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΗΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής εργασίας	Εφαρμογή για χαμένα ζώα υλοποιούμενη σε Java και Spring Boot Backend Web App for lost animals implemented in Java and Spring Boot Backend
Ονοματεπώνυμο Φοιτητή	Βασιλάκης Νικόλαος
Πατρώνυμο	Εμμανουήλ
Αριθμός Μητρώου	Π17013
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης Φεβρουάριος 2025

Copyright ©

Ευχαριστίες

Θα να δώσω τις ευχρηστίες σε όσους με βοήθησαν να φτάσω σε αυτό το σημείο της καριέρας μου . Θα ήθελα να δώσω και μεγάλες Ευχαριστίες στον κ. Ευθύμιο Αλέπη που είχε την πρόθεση να με αναλάβει ως επίλεκτο φοιτητή για την εξέταση της εργασίας αυτής με το συγκεκριμένο θέμα.

Περίληψη

Το θέμα της εργασίας είναι η υλοποίηση ενός πλήρους περιβάλλοντος και λειτουργικότητας μίας ιστοσελίδας που αναλαμβάνει την εύρεση χαμένων ζώων μέσω την επικοινωνία χρηστών. Η βασική λογική της ιστοσελίδας μας θα είναι να μπορούν να γίνονται αναρτήσεις για χαμένα ζώα ή ζώα που βρεθήκανε στον δρόμο. Έπειτα θα μπορούν επαγγελματίες χρήστες (από ιδρύματα ή κτηνίατρους) να μπορούν να προστατέψουν αυτά τα ζώα μέχρι να βρεθεί ο παλιός , ή ένας νέος ιδιοκτήτης τους. Για να γίνει αυτό θα πρέπει να χτίσουμε μία βασική μεθοδολογία αποστολής και παραλαβής δεδομένων. Μέσω της τεχνολογίας (framework) Spring Boot και άλλων βοηθημάτων του μπορούμε να εξασφαλίσουμε μία τέτοια λειτουργεία έτσι ώστε να υπάρχει μια λογική και ασφαλής μεταφορά δεδομένων μεταξύ χρηστών αλλά και η διαχείριση τους από εξιδεικευμένους χρήστες του site. Η ιστοσελίδα (ή αλλιώς web app για υπολογιστές) χωρίζεται σε 3 μέρη , το frontend ,το backend και το database. Το Spring boot βρίσκεται στο backend και μέσω αυτού θα προσπαθήσουμε να καλούμε δεδομένα από την βάση (database) και να τα στέλνουμε σε χρήστες. Η σχεδίαση του backend μας είναι κρίσιμη γιατί με αυτή θα πρέπει να ορίσουμε τι δεδομένα θα μπορεί να βλέπει ένας χρήστης και να διαχειρίζεται, για παράδειγμα δεν πρέπει ένας απλός χρήστης να μπορεί να διαγράφει άλλους χρήστες. Στόχος της εργασίας είναι να ενώσουμε και τα 3 μέρη για να γίνει η σωστή υλοποίηση του site.

Λέξεις κλειδιά: Spring Boot, Backend , Frontend , Database , Framework

Abstract

The project subject is to implement a complete and functional environment for a website which endeavors the finding of lost animals with user communication. The basic logic of our website is the creation of posts of lost animals or found animals somewhere outside. Then, vets or institution employees could protect these animals until a new owner, or the old one takes them back. To succeed this, we need to build a basic system of sending and receiving messages, With the technology (framework) of Spring Boot and other tools we can ensure of an implementation of a methodology like this so that there is a logical and safe way to transfer data between multiple users and the management of them by specialized users. The website (or otherwise a web app for computers) is divided in 3 parts , the frontend , the backend and the database. Spring boot is used in the backend and with it we will retrieve data from the database and send it to other users. Or backend design is important because with it we will decide what data a user should receive and manage, for example a common user should not have the ability to delete other users. Our goal is to connect the 3 parts of our project correctly to succeed the right implementation of the website.

Key Words: Spring Boot, Backend , Frontend , Database , Framework

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Εισαγωγή

Στην εργασία θα προσπαθήσουμε να φτιάξουμε μία εφαρμογή με την οποία θα μπορούν χαμένα ζώα να επιστρέφουν πίσω στον ιδιοκτήτη τους. Θα μπορούμε επίσης να ανεβάσουμε ευρισκόμενα ζώα στην ιστοσελίδα τα οποία μπορούμε να μεταφέρουμε σε άλλους χρήστες. Μέσω των τεχνολογιών της Spring Boot θα μπορούμε να διαχειριστούμε όλα τα δεδομένα που χρειάζεται ο κάθε χρήστης και του επιτρέπεται να αποκτήσει. Για το Spring Boot θα χρειαστεί μία ανάλογη μεθοδολογία για να μπορούν να μεταφέρονται τα δεδομένα μας σωστά , και να υπάρχει ασφάλεια. Στόχος μας είναι να δημιουργήσουμε ένα βοηθητικό περιβάλλον στον χρήστη που θα τον βοηθήσει στο πρόβλημα του (συγκεκριμένα στο να βρει ένα ζώο) και να του παρέχουμε τις βασικές ανάγκες που μπορεί να χρειαστεί. Θα πρέπει να υπάρχει μία σωστή δομή που θα βοηθάει τον χρήστη και αργότερα να το προτείνει και σε άλλους για την εικόνα του. Επίσης θα είναι απαραίτητο η εμφάνιση της ιστοσελίδας και η λειτουργικότητα του frontend να είναι ελκυστική στον χρήστη και να του παρέχει όλες τις υπηρεσίες με εύκολο τρόπο. Στην εργασία θα χρησιμοποιήσουμε και άλλα εργαλεία όπως την Angular για το frontend, το Docker για την συγκέντρωση της βάσης δεδομένων και το Postman για να επικοινωνήσουμε με το backend.

1. Ανάλυση Θέματος Εργασίας

1.1 Πρωτεύων ζητήματα

Το ζητούμενο της εργασίας αυτής είναι να δημιουργηθεί ένα πλήρες εικονικό περιβάλλον μέσω διαδικτύου όπου ο χρήστης θα μπορεί να βρει ένα χαμένο ζώο που έχασε. Για να γίνει αυτό θα αναλογιστούμε ότι το site θα έχει μεθόδους επικοινωνίας με τους οποίους θα μπορεί εύκολα και αποτελεσματικά να αποκτά ενημερώσεις για την τυχόν εύρεση ενός ζώου του. Επίσης θα πρέπει να υπάρχει η δυνατότητα υποστήριξης για πολλούς χρήστες που έχουν χάσει τα ζώα τους.

Μέσω αυτών των δεδομένων που αναφέραμε παραπάνω μπορούμε να φτιάξουμε μία λειτουργεία δημιουργίας αναρτήσεων όπου ο κάθε χρήστης θα μπορεί να ανεβάσει το πρόβλημα του και να το φανερώσει σε πολλούς άλλους χρήστες. Οι αναρτήσεις θα είναι πολλαπλών ειδών και θα υποστηρίζουν αρκετές πληροφορίες που θα βοηθήσουν κάποιον να βρει το χαμένο ζώο (όπως όνομα κολλάρου , τοποθεσία που χάθηκε, κλπ.).Θα μπορεί επίσης να βρει και την ανάρτηση που θέλει μέσω φίλτρων (για ημερομηνία και ώρα και αναζήτηση τίτλου) που θα βρίσκει την ανάρτηση που θέλει εύκολα και γρήγορα.

Θα χρειαστεί όμως και ένας άλλος χρήστης να μπορεί να εκφραστεί στο ζήτημα αυτό. Όταν δημοσιοποιηθεί μία ανάρτηση στο διαδίκτυο μπορεί ένας από τους χρήστες να έχει πληροφορίες σχετικά με το ζήτημα , να έχει δει το ζώο κάπου για παράδειγμα. Έτσι θα είναι απαραίτητη η δημιουργία σχολείων στην ανάρτηση αυτή και πολλοί χρήστες να συνεργαστούν μεταξύ τους να συγκεντρώσουν πληροφορίες για την εύρεση αυτού του ζώου.

Επιπλέον , είναι σημαντικό να μπορούν να μιλήσουν 2 χρήστες μεταξύ τους μέσω προσωπικών μηνυμάτων για την συγκέντρωση πληροφοριών. Αν κάποιος χρήστης έχει βρει το ζώο ή έχει αρκετές πληροφορίες για αυτό (πχ αν έψαξε στην περιοχή του) είναι απαραίτητο να γίνει η επικοινωνία μέσω άμεσων μηνυμάτων . Μπορεί επίσης να έχει βρει το ζώο και να θέλει να το επιστρέψει , για να γίνει αυτό θα πρέπει κάπως να κανονίσουν οι 2 αυτοί χρήστες να βρεθούν , που θα γίνει μέσω άμεσων μηνυμάτων. Έτσι θα υπάρχει η δυνατότητα δύο χρήστες να μπορούν να μιλήσουν , μέσω προσωπικών μηνυμάτων , για την ανάκτηση πληροφοριών.

Δεν αρκεί όμως να υπάρχουν μόνο αναρτήσεις για χαμένα ζώα. Μπορεί κάποιος να χάσει ένα ζώο και να μην έχει την δυνατότητα να βρεθεί στο συγκεκριμένο site για να αναφερθεί σε κάποιον που μπορεί να το βρει. Έτσι θα πρέπει να υπάρχει και μία σελίδα για τα ευρισκόμενα ζώα. Εκεί θα είναι τα ζώα που έχουν βρεθεί και φαίνεται ότι ανήκουν σε κάποιον ιδιοκτήτη (από κάποιο κολάρο , κλπ.). Έτσι άμα κάποιος που έχασε ένα ζώο δεν προλάβει να το ανεβάσει στο συγκεκριμένο site και δει ότι κάποιος το βρει, μπορεί με προσωπικά μηνύματα να του μιλήσει για να το πάρει πίσω. Αυτή την λειτουργεία θα την χρησιμοποιήσουμε και για ζώα σε κρίσιμη κατάσταση υγείας που θα αναφέρουμε παρακάτω.

1.2 Θεραπεία ζώων

Πέρα από την εύρεση των ζώων θα πρέπει να ασχοληθούμε και με άλλες αναφορές που μπορεί να είναι έκτακτες κατά την κατοχή αυτών των ζώων. Μπορεί ένα ζώο άμα βρεθεί να είναι σε κρίσιμη κατάσταση υγεία (να είναι τραυματισμένο ή να είχε να φάει πολύ καιρό) και να μην μπορεί ένας απλός ιδιοκτήτης να το αναθρέψει μόνος του. Για αυτό το stie θα έχει και την επιλογή ανάκτησης επαγγέλματος για χρήστες οι οποίοι είναι γιατροί και θέλουν να βοηθήσουν τα ανυπεράσπιστα ζώα που βρίσκονται από εθελοντισμό και μόνο (δεν θα υπάρχει ανταμοιβή).

Πολλοί χρήστες για κάθε μορφή επαγγέλματος που αναθρέφει ζώα (κτηνίατροι , χειρούργοι ,κλπ.) θα έχουν την δυνατότητα να αναφέρουν το επάγγελμά τους στους διαχειριστές του site για να τον δείξουν ως αυτού του είδους το επάγγελμα και να ζητάει ο κόσμος βοήθεια από αυτούς. Δεν θα πρέπει να μπορεί να το κάνει ο καθένας από μόνος του όμως , θα πρέπει όλοι οι χρήστες που έχουν αυτού του είδους το επάγγελμα να μπορούν να εγκρίνονται από τους διαχειριστές και μετά αυτοί να τους εγκρίνουν ως επαγγελματίες για περισσότερη ασφάλεια.

Κάθε χρήστης θα πρέπει να εγκρίνεται για την εμφάνιση του επαγγέλματός του. Ο τρόπος αυτός θα γίνεται μέσω τηλεφώνου ,ή email , ή με προσωπική επαφή από ένα κατάστημα. Για τα πρώτα 2 δεν χρειάζεται να κάνουμε κάτι περεταίρω διότι δεν είναι λειτουργείς του site , μπορούμε μόνο να αναφέρουμε τα δεδομένα που θα κάνουν την επικοινωνία αυτή (email,τηλέφωνο).Για την προσωπική επαφή μπορούμε να φτιάξουμε την δημιουργία ραντεβού όπου ο χρήστης θα βλέπει ημερομηνία και ώρα με την οποία θα ενημερώνεται και θα πηγαίνει.

Αρχικά , για να εγκριθεί ένας χρήστης θα πρέπει να κάνει μια αίτηση στο site. Την αίτηση αυτή θα την βλέπουν ανάλογοι διαχειριστές που έχουν αυτήν την δουλειά. Έπειτα όπως αναφέραμε παραπάνω μπούμε σε κάθε αίτηση να τοποθετήσουμε μία ημερομηνία για ραντεβού και να εγκρίνουμε ή να ακυρώσουμε μία αίτηση. Άμα εγκριθούν οι χρήστες θα μπορούν μετά να εμφανίζονται στο site με το επάγγελμά τους.

Όταν οι χρήστες αυτοί επαληθευτούν ως γιατροί θα μπορούν μετά να παίρνουν ζώα από άλλους χρήστες για τυχών ανάγκες του ζώου. Θα βοηθούν δηλαδή τους χρήστες να θεραπεύσουν τυχόν τραυματισμένα ευρισκόμενα ζώα. Για να γίνει αυτό θα βάλλω μία επιλογή με την δημιουργία κάθε ανάκτησης ως «έκτακτη» όπου θα στέλνεται μήνυμα στους επαγγελματίες γιατρούς να μιλήσουν για να θεραπεύσουν τα ζώα αυτά. Θα μπορούμε να φιλτράρουμε και τις αναρτήσεις στο ποιες είναι έκτακτες και ποιες όχι.

1.2.1 Ανατροφή ζώων

Εκτός από τους γιατρούς θα χρειαστούμε και ένα άλλος είδος επαγγελματιών που ασχολούνται με τα ζώα. Υπάρχουν περιπτώσεις που κάποιοι χρήστες δεν θα μπορούν να συντηρήσουν ένα ζώο για διαφόρους λόγους (ή επειδή δεν μπορούν / δεν έχουν τον χώρο ή

έχουν βρει ένα φίδι , κλπ.). Γι' αυτό θα πρέπει να μπορούν και ιδικά ιδρύματα ζώων να μπορούν να αποκτήσουν αυτά τα ζώα και να τα συντηρήσουν μέχρι να βρεθεί ο ιδιοκτήτης τους να τα πάρει.

Η εφαρμογή θα υποστηρίζει την αποθήκευση ιδρυμάτων . Εκεί θα υπάρχουν όλα τα ιδρύματα που έχουν φτιαχτεί με τις τοποθεσίες τους και το προσωπικό τους έτσι ώστε να υπάρχει η ευκολία επικοινωνίας μεταξύ τους. Σε κάθε ίδρυμα θα μπορεί ένας χρήστης να εγκρίνεται για την εργασία του, όπως παραπάνω με τους γιατρούς. Οι υπάλληλοι στα ιδρύματα θα δουλεύουν ομαδικά και θα αποθηκεύουν πολλά ζώα που είναι για μεταφορά ή υιοθέτηση , θα έχουν τις κατάλληλες συνθήκες για να μεγαλώσει ένα ζώο.

Αν ένας χρήστης βρει ένα ζώο και για διαφόρους λόγους και δεν μπορεί να το μεγαλώσει , μπορεί να το παραδώσει σε ένα ίδρυμα. Αυτό θα γίνει με το να επικοινωνήσει με έναν επίλεκτο από το ίδρυμα αυτό που μπορεί να βρει στο site με άμεσα μηνύματα ή με άλλους τρόπους που αναφέραμε παραπάνω όπως τηλέφωνο και email που θα παρέχει στο site. Έπειτα μέσω του site θα υπάρχει μια λειτουργεία που θα μεταφέρονται τα ζώα και θα δείχνει ποιος είναι ο τωρινός ιδιοκτήτης τους.

1.2.2 Μεταφορά ζώων

Όπως αναφέραμε παραπάνω , θα πρέπει να μπορούν αν μεταφέρονται τα ζώα μεταξύ χρηστών για λόγους ανάγκης του ζώου αυτού. Για να γίνει αυτό , εκτός από την λειτουργεία των άμεσων μηνυμάτων θα υπάρχει και η λειτουργεία μεταφοράς ζώου. Εκεί θα μπορεί ο χρήστης , όταν παραδώσει το ζώο αυτό και για να ενημερώσει τους άλλους χρήστες του site , να στείλει μία αίτηση μεταφοράς ζώου.

Η αίτηση θα μεταφέρεται σε ένα πλαίσιο ενημερώσεων του συγκεκριμένου χρήστη στο site και θα μπορεί να κάνει αποδοχή και ακύρωση αίτησης. Μόλις κάνει αποδοχή , η ανάρτηση του ευρισκόμενου ζώου θα αλλάζει κάτοχο και θα δείχνει τον τωρινό του. Έτσι άμα κάποιος χρειαστεί όπως αναφέραμε παραπάνω δώσει το ζώο σε έναν γιατρό ή ίδρυμα θα το μεταφέρει με αυτόν τον τρόπο. Επίσης άμα βρεθεί ο κάτοχος του ζώου μπορεί να το μεταφέρει έτσι και να το θέσει ο κάτοχός του ως «ΕΠΙΣΤΡΑΜΕΝΟ» για μελλοντική ενημέρωση.

Δεν θα γίνεται η μεταφορά σε ειδικούς μόνο. Θα υπάρχει μία επιλογή όταν επιλέγεται ένας χρήστης με ποιόν τρόπο θέλει να το πάρει (ως γιατρός , ως απλός χρήστης ή για να το μεταφέρει στο ίδρυμα).Με αυτόν τον τρόπο μπορεί κάποιος ως γιατρός να μεταφέρει ένα δικό του ζώο στο σπίτι του ή σε κάποιο ίδρυμα που θα είναι μέλος από μόνος του.

1.3 Βασικά βοηθήματα χρήστη

1.3.1 Ειδοποιήσεις και Ανακοινώσεις

Εκτός από το βασικό site , ένας χρήστης θα είναι καλό να μπορεί να ενημερώνεται μέσω αυτού για το αν υπάρχει ένα κρίσιμο νέο που υπάρχει σε αυτό. Θα μπορεί να μεταφερθεί στα ανάλογα μέρη που θα υπάρχουν για το αν υπάρχει κάτι που τον απασχολεί και μπορεί να πάρει δράση σε αυτό.

Θα υπάρχουν ειδοποιήσεις με τις οποίες θα ενημερώνεται ο χρήστης για αλλαγές που τον απασχολούν προσωπικά. Αν για παράδειγμα είναι γιατρός και ένα νέο ζώο μπει σε ανάρτηση στα επείγοντα θα ειδοποιούνται όλοι οι χρήστες που είναι γιατροί και με ένα link θα βλέπουν την ανάρτηση.

Θα υπάρχουν επίσης και ανακοινώσεις , όπου θα ενημερώνονται οι χρήστες για γενικές ενημερώσεις που γίνονται στο site. Αυτές οι ενημερώσεις θα περιλαμβάνουν γενικές αλλαγές στο site που μπορεί να είναι για την συντήρηση του (πχ ότι θα κλείσει για λίγες ώρες , κλπ.) ή θα είναι για αναρτήσεις σημαντικές που τελείωσαν και είχαν καλό τέλος , οι οποίες θα έχουν ένα link στο οποίο θα πηγαίνει σε αυτήν την ανάρτησης. Οι ανακοινώσεις για αναρτήσεις θα είναι κυρίως για ένα ζώο που χάθηκε για καιρό και βρέθηκε ο ιδιοκτήτης του , ή γενικά για δημοφιλή αναρτήσεις που είχαν καλό τέλος.

1.3.2 Εμφάνιση Στοιχείων

Κάθε χρήστης θα έχει την δικιά του σελίδα όπου θα εμφανίζονται όλα τα στοιχεία του για τυχόν ενημερώσει. Εκεί θα υπάρχουν κουμπιά στα οποία ο χρήστης θα μπορεί να επικοινωνήσει μαζί του ή να δώσει ένα ζώο.

2. Σχεδίαση

2.1 Γενική ιδέα

2.1.1 Εμφάνιση

Η ιστοσελίδα μας θα πρέπει να είναι οικεία προς τον χρήστη. Θα πρέπει όλες οι λειτουργείες που υπάρχουν σε αυτήν όπως τα link και τα κουμπιά και να βρίσκονται σε σημεία τα οποία ο χρήστης μπορεί να βρει εύκολα και να είναι σε εμφανές σημείο. Είναι καλό να υπάρχει ένα πλαίσιο (στο πάνω μέρος της ιστοσελίδας , δηλαδή header) το οποίο θα περιέχει όλες τις βασικές σελίδες που μπορεί να δει. Στο πλαίσιο αυτό μπορούμε να βάλουμε και τα στοιχεία του χρήστη (όπως ονοματεπώνυμο). Μπορούμε λιγότερο βασικές σελίδες να τις τοποθετήσουμε σε σημεία που ανοίγουν με το πάτημα ενός κουμπιού (όπως όταν πατάμε την εικόνα του χρήστη μας να μας εμφανίζει τις ρυθμίσεις του λογαριασμού αυτού).

Οι σελίδες θα είναι απλές και περιεκτικές , δεν θα χρειάζεται ο χρήστης να ψάχνει πολύ για να βρει αυτό που θέλει , θα βρίσκει ότι θέλει σε συγκεκριμένα σημεία και θα φαίνονται με την πρώτη ματιά. Όλες οι σελίδες όταν ανοιχθούν θα έχουν τα στοιχεία τους με μεγάλα αντικείμενα και θα δίνουν μια ξεκούραστη αίσθηση στον χρήστη. Θα υπάρχουν και χρώματα που θα ταιριάζουν μεταξύ τους στην δημιουργία του site.

Οι σελίδες θα χωρίζονται στις βασικές κατηγορίες που αναφέραμε στην ανάλυση. Η κύριες σελίδες μας είναι των αναρτήσεων για ζώα (χαμένων και ευρισκόμενων ζώων). Η σελίδες αυτές θα έχουν τις αναρτήσεις σε μεγάλα πλαίσια και θα έχουν αρκετές πληροφορίες πάνω. Στα πλαίσια αυτά θα εμφανίζονται οι πιο πολλές πληροφορίες για την ανάρτηση έτσι ώστε να καταλαβαίνει ο χρήστης το περιστατικό αυτό. Θα είναι καλό να υπάρχει και ένας χάρτης όπου ο χρήστης που ανεβάζει την ανάρτηση να δείχνει με ακρίβεια σε ποιο σημείο έχασε το ζώο του για να μπορέσει αν εντοπιστεί πιο εύκολα. Θα υπάρχουν και φίλτρα στα αριστερά των αναρτήσεων σε μία μπάρα (sidebar) όπου ο χρήστης θα μπορεί να επιλέξει στην ημερομηνία που χάθηκε το ζώο και αν είναι σε έκτακτη κατάσταση (χρειάζεται γιατρό).

Για να δημιουργηθεί μία ανάρτηση όπως και με τα πιο πολλά δεδομένα θα υπάρχει μία φόρμα. Στην φόρμα της ανάρτησης θα μπορεί ο χρήστης να επιλέξει το μέρος που χάθηκε το ζώο με κλικ σε ένα χάρτη χωρίς να ξέρει ακριβή συντεταγμένες (θα μπορεί να βάλει συντεταγμένες άμα θέλει και θα του εμφανιστούν στον χάρτη).Αυτό θα γίνεται επίσης αν ένας χρήστης ανεβάσει την εργασία του (το πού δουλεύει , το κτηνιατρείο του) ή αν ανεβεί ένα νέο ίδρυμα ζώων , θα μπορεί με τον ίδιο τρόπο να επιλέξει και να εμφανίσει που βρίσκεται το κάθε ένα στον χάρτη.

Θα υπάρχει και μία αρχική οθόνη όπου θα είναι η αρχική της ιστοσελίδας μας. Επίσης θα υπάρχουν γραφικά εμφανισιακά , αλλά κάπως απλά για να μην τον μπερδεύουν όταν κάνει τις συναλλαγές του.

2.1.2 Δεδομένα

Θα πρέπει να σκεφτούμε και τι δεδομένα θα υποστηρίζει η ιστοσελίδα. Αυτά τα δεδομένα θα μεταφέρονται σε χρήστες και θα τροποποιούνται από αυτούς. Τα δεδομένα αυτά θα είναι βασικών κατηγοριών τα οποία θα υποστηρίζουν μία λειτουργία στην ιστοσελίδα.

Αρχικά θα έχουμε τα δεδομένα για τις αναρτήσεις. Για να έχουμε ικανοποιητικές αναρτήσεις είναι βασικό να έχουμε αρκετά δεδομένα όπου θα βοηθούν τον χρήστη να βρει με ευκολία ένα ζώο (ή να βρεθεί το αφεντικό σε ένα ευρισκόμενο).Οι πληροφορίες αυτές θα είναι η τοποθεσία , το όνομα του ζώου , ο τύπος ζώου με την ράτσα του , τα στοιχεία κολάρο του , τον κάτοχο (ή αυτόν που έχασε το ζώο) μία περιγραφή και η κατάσταση υγείας του , έτσι ώστε να ξέρουμε σε τί κατάσταση είναι το ζώο και τι πρέπει να το ταΐσουμε άμα το βρούμε (άμα χρειάζεται να ξέρουμε , αν είναι αλλεργικό για παράδειγμα).

Έπειτα θα έχουμε τις πληροφορίες των επαγγελματιών κάποιων εθελοντικών χρηστών στην ιστοσελίδα. Αρχικά θα μπορούν επαγγελματίες κτηνίατροι , μετά από έγκριση να

προσθέτουν την τοποθεσία της δουλειάς τους και τον τύπο της (κτηνίατρος χειρούργος κλπ.).Ακόμη , μετά από την δημιουργία ιδρυμάτων , που θα έχουν την τοποθεσία και το όνομα τους, θα μπορούν επαγγελματίες σε αυτά τα ιδρύματα να εγκριθούν και να υπάσχουν στην ιστοσελίδα ότι δουλεύουν σε ένα συγκεκριμένο ίδρυμα.

Πέρα από αυτά θα υπάρχουν κατηγορίες για ανακοινώσεις, μηνύματα , πληροφορίες χρηστών , πληροφορίες για αιτήσεις επαγγελμάτων , αιτήσεις για ανταλλαγή ζώων , ειδοποιήσεις , σχόλια και ενεργές συζητήσεις.

Όλα αυτά τα δεδομένα που αναφέραμε θα τα υποστηρίζει μία βάση δεδομένων όπου κάθε κατηγορία δεδομένων θα είναι ένας πίνακας. Κάθε πίνακας έχει πληροφορίες όπου θα επιστρέφονται ανάλογα από το backend για κάποιες αναζητήσεις που θα κάνουμε στην ιστοσελίδα. Κάθε πληροφορία του πίνακα θα αποθηκεύεται σε μία στήλη και όλες οι στήλες θα κάνουν τον πίνακα αυτό. Στο κεφάλαιο της υλοποίησης θα μιλήσουμε με ακρίβεια για τα δεδομένα αυτά και πως θα μεταφέρονται.

2.1.3 Λειτουργικότητα και ασφάλεια

Η λειτουργικότητα στης σελίδας μας θα πρέπει να είναι αυστηρή. Θα πρέπει όλα τα δεδομένα να μεταφέρονται με κατάλληλο τρόπο χωρίς να μπορεί να δει ένας χρήστης δεδομένα που δεν πρέπει.

Θα πρέπει μέσω του backend μας να διορίσουμε την σωστή δομή που θα πρέπει να έχουν οι πληροφορίες μας. Κάθε πληροφορία που θα ανεβαίνει στον εξυπηρετητή (server) θα πρέπει να διαχωρίζεται ως έγκυρο ή όχι με βάση τα δεδομένα που έστειλε ο χρήστης. Η έγκριση θα γίνεται κυρίως σε τομής όπως το μέγεθος κάποιου ονόματος (αν είναι πολύ μικρό ή μεγάλο) , αν έχει προσθέσει όλα τα αναγκαία στοιχεία που θέλει (να μην είναι κενό δηλαδή κάποιο δεδομένο) και τον τύπο των δεδομένων (αν για παράδειγμα έχει εισάγει email ο χρήστης). Το Backend μετά που διαβάζει τα αρχεία μας θα πρέπει να ξεχωρίσει ποια θα αποθηκεύονται και ποια όχι.

Θα πρέπει να υπάρχει και η ασφάλεια των δεδομένων. Για την σωστή ροή του site , όπου όλοι οι χρήστες θα εμπιστεύονται τα προσωπικά τους αντικείμενα και να μην διαχειρίζονται από τον οποιονδήποτε χρήστη θα πρέπει να υπάρχουν οι κατάλληλοι ρόλοι όπου σε κάθε ρόλο ο χρήστης θα έχει τις δικιές του δυνατότητες. Δεν θα πρέπει τα δεδομένα να μπορεί ένας χρήστης να ανακτά από το backend μας προσωπικά δεδομένα άλλου ή δεδομένα που δεν πρέπει να δει (λόγο του ρόλου του). Επίσης πρέπει οι χρήστες να μην μπορούν να επεξεργαστούν δεδομένα που δεν τους επιτρέπετε (όπως ένας απλός χρήστης να διαγράψει άλλον χρήστη).

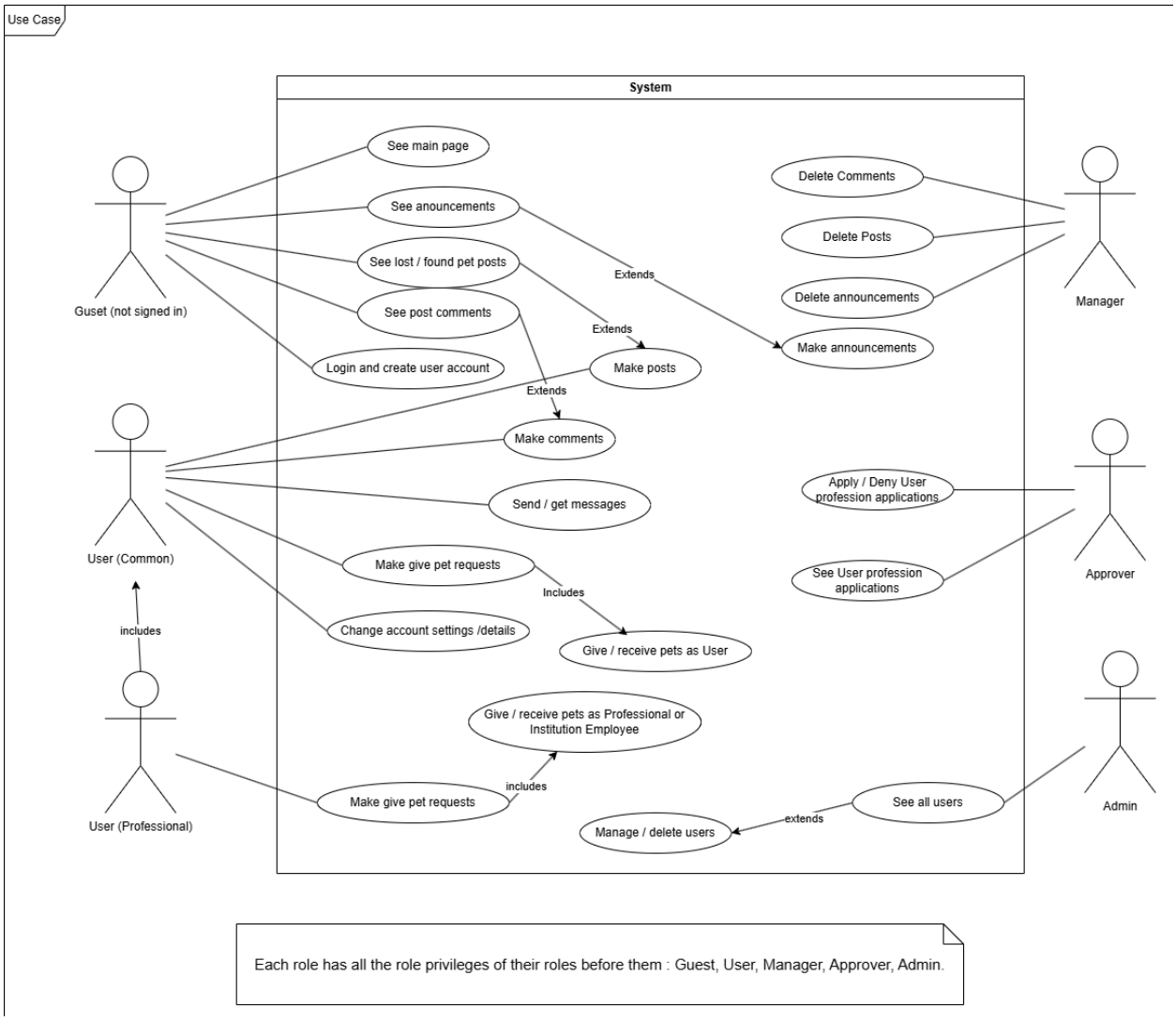
Κάθε χρήστης όταν συνδέεται θα έχει μία σύνοδο (session) στην οποία θα μπορεί με το να συνδέεται να έχει ένα κλειδί (token) με το οποίο για ένα χρονικό διάστημα να μπορεί να κάνει κάποιες ενέργειες στην ιστοσελίδα . Το κλειδί αυτό θα κάνει τις ενέργειες που θα πρέπει μόνο ένας συνδεδεμένος χρήστης να κάνει και με αυτό θα εγκρίνεται ότι είναι συνδεδεμένος.

Επίσης το κλειδί αυτό θα αποθηκεύει και τον ρόλο του χρήστη για το τί πράγμα θα μπορεί να κάνεις την σελίδα με το που συνδεθεί. Μέσω του κλειδιού θα μπορούμε από το backend μας να τον αναγνωρίσουμε και να του επιστρέφουμε τα δεδομένα που του ανήκουν (πχ τα μηνύματά του).

2.2 UML αρχεία

2.2.1 Use case diagram

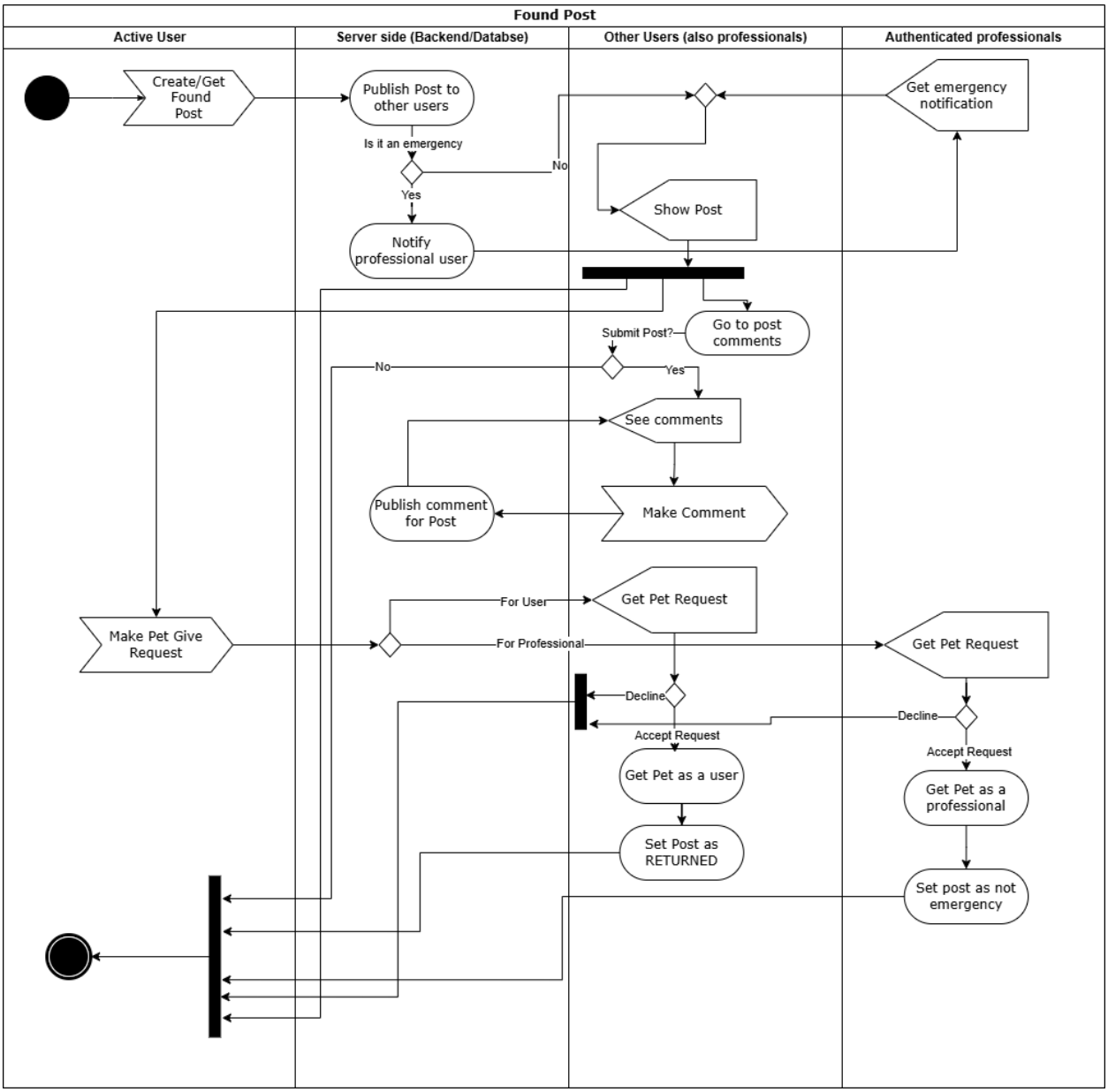
Θα χρησιμοποιηθούν 4 ρόλοι για την λειτουργεία της σελίδας , ο User , ο Manager , ο Approver και ο Admin. Μπορούμε να θέσουμε και ως ρόλο έναν επισκέπτη που δεν θα έχει συνδεθεί (Guest). Είναι σημαντικό και ένας επισκέπτης να μπορεί να δει τις αναρτήσεις για υπάρχει εμβέλεια στο πόσοι μπορούν να δουν στις αναρτήσεις που γίνονται και να βρίσκονται πιο γρήγορα τα ζώα. Ο χρήστης (User) χωρίζεται σε απλό χρήστη και χρήστη με επάγγελμα , με την μόνη διαφορά ότι όταν του δοθεί ένα ζώο θα τον δείχνει ως το επάγγελμα του ή το ίδρυμα που δουλεύει. Ο απλός χρήστης (User) θα συνδέεται και θα μπορεί να κάνει τις βασικές λειτουργείες που μπορεί να κάνει ο καθένας για να εξυπηρετηθεί. Οι υπόλοιποι θα πρέπει να ελέγχουν την σωστοί ροή του site και να τον εξυπηρετούν σωστά.



Εικόνα 2.1 - Use Case Διάγραμμα.

2.2.2 Found Post Activity diagram

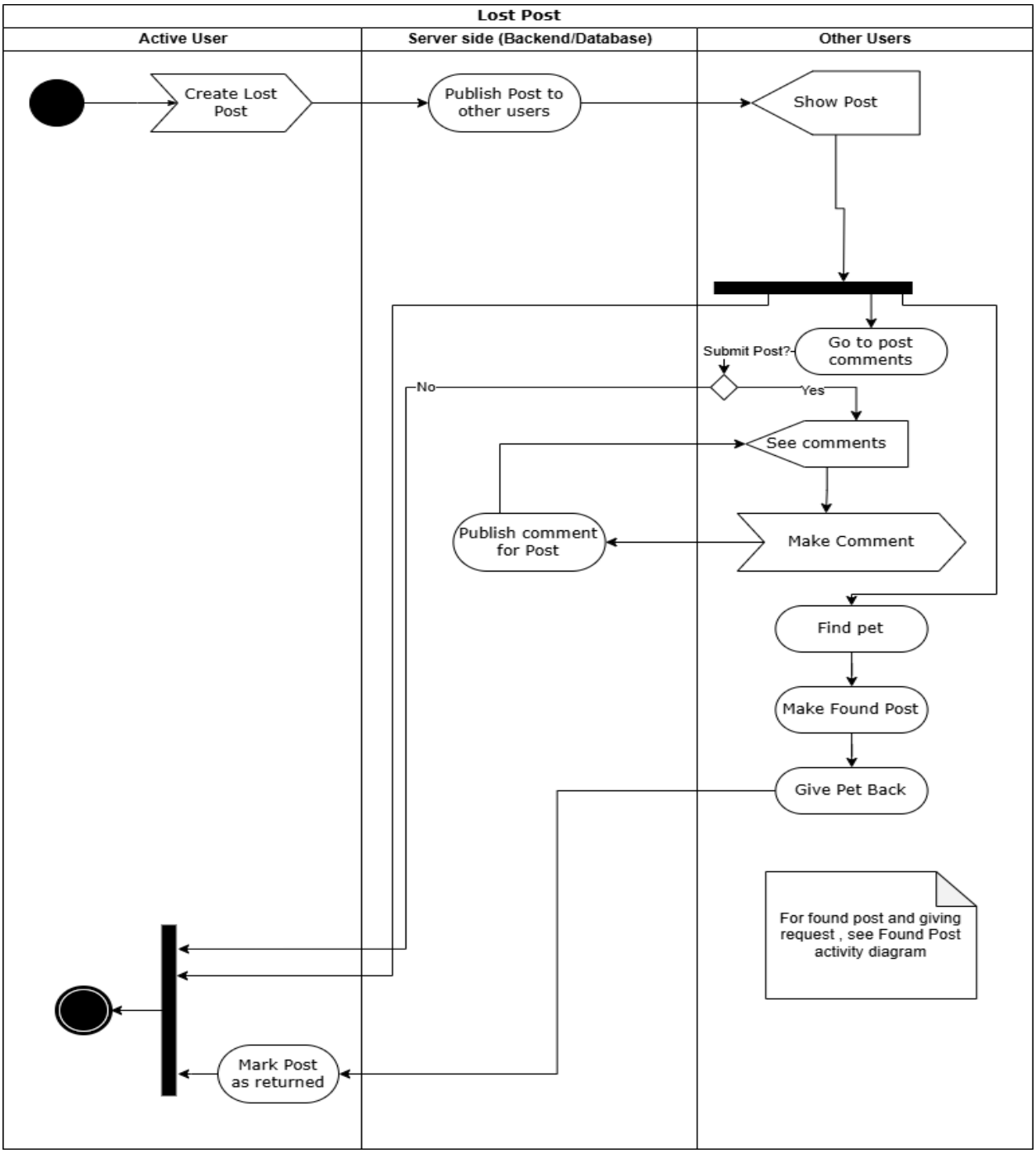
Άμα ένας χρήστης βρει ένα ζώο θα μπορούν να γίνουν οι εξής ενέργειες στην σελίδα. Όταν θα ανεβεί η ανάρτηση στην σελίδα αυτή θα μπορούν χρήστες να μιλάνε μέσω σχολείων για να εύρεση του αφεντικού του ζώου. Θα μπορεί επίσης το ζώο να μεταφερθεί σε άλλον χρήστη για λόγους ανάγκης. Επίσης αν το ζώο αυτό έχει κάποιο πρόβλημα (βρεθεί με κάποιο τραύμα) θα μπορεί να το ανεβεί ως επείγων (emergency) περιστατικό για να ειδοποιήσει τους κτηνιάτρους που θα το θεραπεύσουν.



Εικόνα 2.2 - Activity Diagram , Found Post Page.

2.2.3 Lost Posts Activity diagram

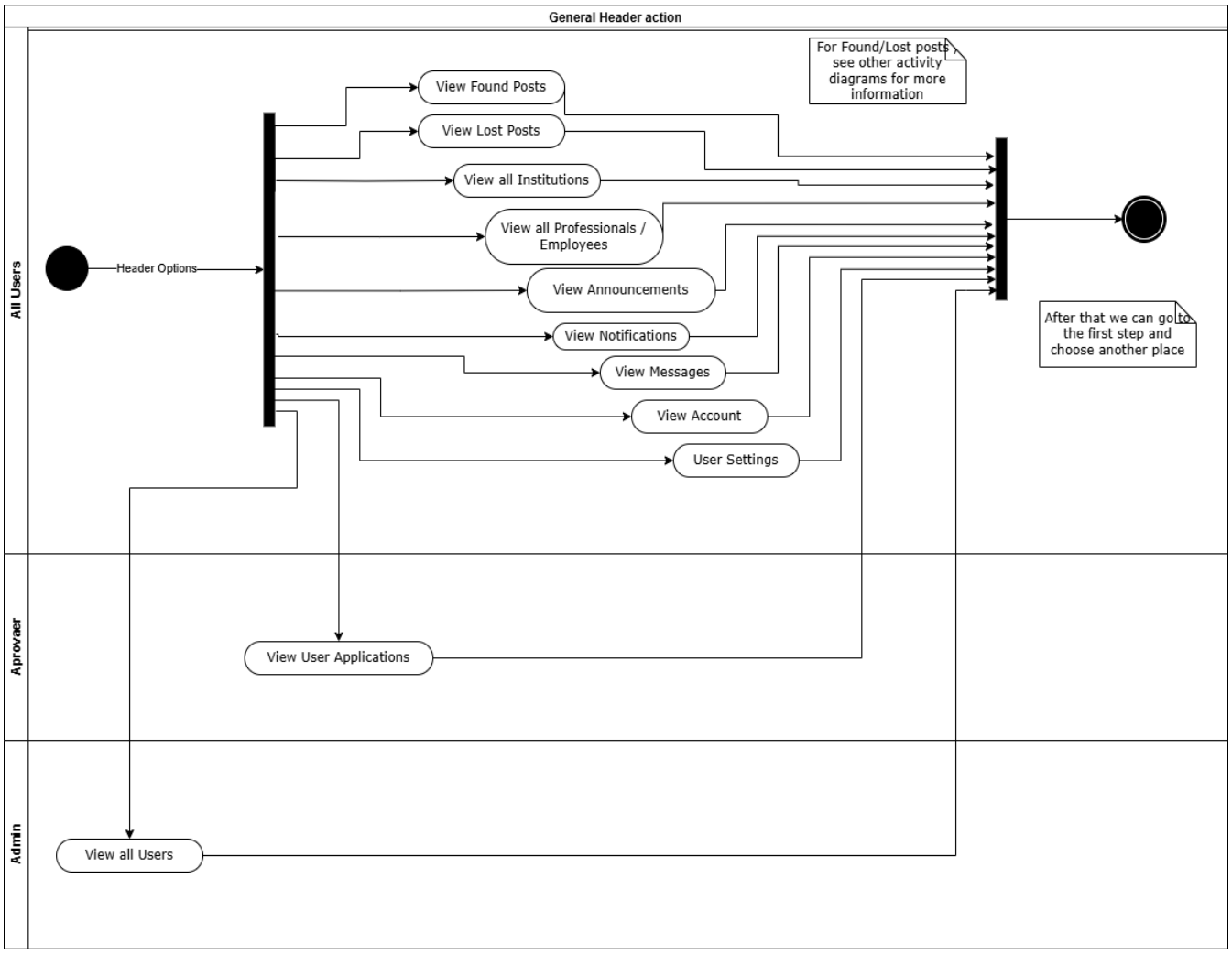
Η λογική για ένα χαμένο ζώο είναι σχεδόν παρόμοια με την εύρεση ενός (όπως βλέπουμε παραπάνω).Παρόλα αυτά μπορεί ένας άλλος χρήστης να βρει το ζώο και ενώ το ανεβάζει στο site να το δώσει σε αυτόν και έτσι να θέσει το ζώο ως επιστραμμένο στο αφεντικό του (returned). Προφανώς οι ειδοποιήσεις για το χαμένο ζώο δεν θα είναι οι ίδιες με το αν βρεθεί τραυματισμένο.



Εικόνα 2.3 Activity Diagram , Lost Post Page.

2.2.4 Header (πάνω πλαίσιο) Activity diagram.

Είναι σημαντικό να αναφέρουμε τι λειτουργείες μπορούμε να διαλέξουμε από το Header. Το header θα βρίσκεται πάντα στην κορυφή και με αυτό θα μπορούμε να πηγαίνουμε στις υπόλοιπες σελίδες. Σε ότι σελίδα πάμε θα έχει τις δικιές τις λειτουργίες που θα σχετίζουν την σελίδα αυτή. Κάποιες επιλογές δεν θα εμφανίζονται σε απλούς χρήστες , αλλά μόνο σε αυτούς που θα συνδεθούν με ένα πιο υψηλό ρόλο.

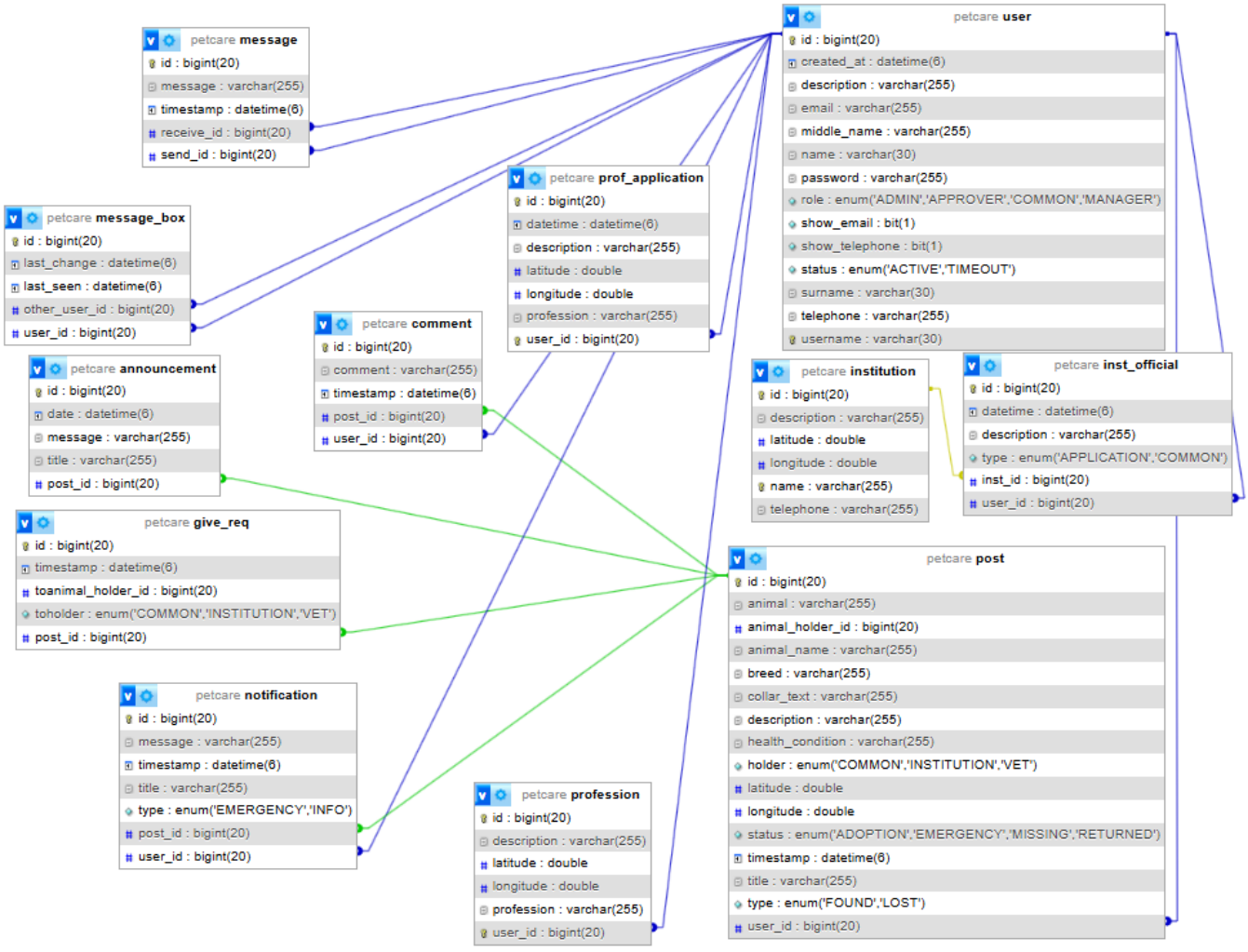


Εικόνα 2.4 Activity Diagram , Header

Όλα τα UML αρχεία θα ανέβουν στο repository του GitHub που θα είναι και η εργασία.

2.2.5 Erd διάγραμμα της βάσης

Παρακάτω , όταν προσθέσουμε τους πίνακες με τα δεδομένα στην βάση θα έχουν την δομή που δίχουμε παρακάτω.



Εικόνα 2.5 ERD της βάσης δεδομένων

Κάνοντας μία μικρή επεξήγηση του κάθε πίνακα:

- user : Έχει της πληροφορίες του κάθε χρήστη. Συνδέονται σε αυτόν πίνακες με το ότι μπορεί να δημιουργήσει (μηνύματα , κλπ.)
- post : Είναι όλες οι αναρτήσεις των χρηστών , χωρίζονται με τον τύπο (“type” column) ανάλογα αν είναι ευρισκόμενα ή χαμένα και θα πηγαίνουν στην ανάλογη σελίδα.
- porf_application : Είναι όλες οι αιτήσει επαγγέλματος γιατρού για το site.
- profession: Είναι όλοι οι επαγγελματίες γιατροί εγγεγραμμένοι και με έγκρηση στο site. Πρώτα θα πρέπει να δημιουργήσουν μία αίτηση που θα εγκριθεί από τον ανάλογο ρόλο πριν καταχωρηθούν ως επαγγελματίες.
- institution : Περιέχει όλα τα ιδρύματα , μόνο οι ADMIN μπορούν να τα δημιουργήσουν. Μπορούν να τα δουν όλοι (και οι μη συνδεδεμένοι)
- inst_official: Είναι οι αντίστοιχες εγκρίσεις για να μπορέσει κάποιος να εγκριθεί ως επαγγελματίας διαχειριστής ενός ιδρύματος , με την διαφορά ότι εδώ με το column (στοιχείο στην βάση) “type” βλέπουμε αν είναι αίτηση ή εγγεγραμμένος χρήστης και δεν διαγράφουμε την αίτηση (application).
- message_box : Είναι όλες οι επιλεγμένες επαφές μας με έναν χρήστη. Χρησιμεύει έτσι ώστε εύκολα να επιλέγουμε τους χρήστες για να επικοινωνήσουμε μαζί του. Εμφανίζει μετά όλα τα μηνύματά μας από τον πίνακα “messages” αλλά αν διαγραφεί δεν χάνονται τα μηνύματα (μπορούμε να ξανακάνουμε messagebox και να τα ξαναδούμε).
- message:Όλα τα μηνύματα που έχουν δημιουργηθεί. Όταν επιλεχθεί ένα από τα messageboxes που έχουμε δημιουργήσει (με το να επιλέξουμε να μιλήσουμε με έναν χρήση) το backend φιλτράρει τα μηνύματα και επιστρέφει μόνο αυτής της συναλληλίας.
- comment: Τα σχόλια κάθε ανάρτησης. Συνδέονται με την ανάρτησή τους.
- give_req = Οι αιτήσεις που θα γίνονται για να μεταφέρονται τα ζώα. Θα βρίσκονται στην σελίδα “notifications” και από εκεί να εγκριθούν θα δείχνει ότι μεταφέρθηκε το ζώο στο site (θα αλλάξει ο κάτοχός του)
- notification: Οι ειδοποιήσεις μας. Πέρα από τα αιτήσεις που θα εμφανίζονται στο ίδιο σημείο (με πράσινο χρώμα) θα υπάρχου και (με λευκό χρώμα) ειδοποιήσεις για κάποια συμβάντα (πχ αν ένας χρήστης έγινε “Timeout”)
- announcements : Όλες οι ανακοινώσεις της σελίδας. Θα τις βλέπει οποιασδήποτε (και οι μη συνδεδεμένοι χρήστες) και θα δημιουργούνται και θα διαγράφονται από τους ρόλους MANAGER,APPROVER,ADMIN.Το column postId είναι προαιρετικό και είναι μόνο αν μιλάμε για μία συγκεκριμένη ανάρτηση στην οποία θα υπάρχει ένα link με την ανάρτηση που θα πηγαίνει στα σχόλιά της.

3. Εργαλεία χρήσης

Πριν μπορέσουμε να δούμε την δομή της εργασίας μας θα πρέπει να αναλύσουμε τα εργαλεία που βάλαμε σε αυτήν και το πως δουλεύουν με την εγκατάστασή τους.

3.1 Γλώσσες προγραμματισμού

3.1.1 Frontend

Για την σχεδίαση του frontend χρησιμοποίησα την γλώσσα προγραμματισμού Angular. Η Angular είναι ένα είδος κατασκευής της γλώσσας TypeScript η οποία είναι βασισμένη σε JavaScript φτιαγμένη από την Google. Είναι μία γνωστή γλώσσα για πολλούς χρήστες για την σύσταση της και την απλότητα της στο να κατασκευάζει αντικείμενα σε μία σελίδα. Περιέχει πολλές λειτουργείες και θεωρείται μία πιο κατανοητή έκδοση του JavaScript. [1]

Η βασική αρχή κτήσεις αντικειμένων μέσω Angular είναι τα Components τα οποία είναι κομμάτια κώδικα που τα τοποθετούμε σε ένα σημείο στη σελίδας μας (ή σε όλη την σελίδα) και κάνουμε μία συγκεκριμένη λειτουργεία. Συνοδεύονται μαζί με ένα html και CSS αρχείο και αυτά τα εξαρτήματα (Components) τοποθετούνται στο σημείο που θέλουμε με το html αυτό.

Πέρα από τα components , η angular συνοδεύεται με services , interceptors και models που θα δούμε παρακάτω.

3.1.2 Backend και Database

Για το Backend χρησιμοποίησα το framework της Spring Boot όπου με την χρήση της γλώσσας Java αναλαμβάνει την λειτουργεία των δεδομένων και το πως θα μεταφέρονται τα δεδομένα από το frontend στο database και αντίστροφα. Μέσω της γλώσσας SQL (Structured Query Language) θα κάνουμε ερωτήματα (queries) με την βάση δεδομένων μας για την ανάρτηση δεδομένων που μετά θα αποφασίζουμε με το Spring Boot πως θα μεταφέρονται στον χρήστη. [2]

Το Spring Boot χρησιμοποιεί το MVC μοντέλο (Model – View -Controller) και επικοινωνεί με το frontend μέσω αποστολής αιτήσεων (Request Apis) σε μορφή URL (Uniform Resource Locator) και μετά με μία απάντηση του (response) επιστρέφει μία απάντηση με πιθανά δεδομένα που θα επεξεργαστούμε στο frontend. Τα δεδομένα αυτά είναι σε μορφή JSON (JavaScript Object Notation). [3]

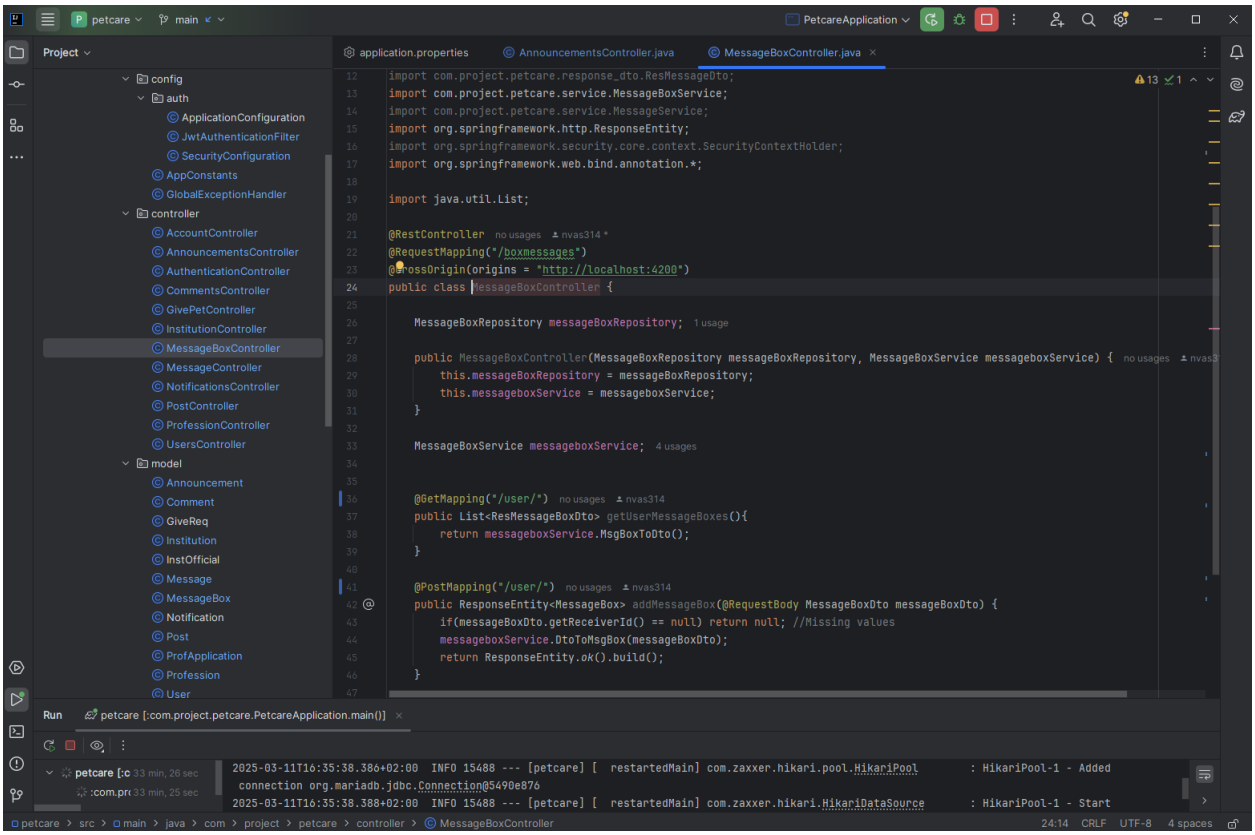
3.2 IDE και προγράμματα

Πέρα από τις γλώσσες προγραμματισμού θα πρέπει να επιλέξουμε και το περιβάλλον ανάπτυξης του λογισμικού μας και τι προγράμματα θα χρειαστούμε.

3.2.1 IntelliJ Idea

Το IntelliJ Idea (Community) είναι ένα περιβάλλον ανάπτυξης λογισμικού (IDE - Integrated Development Environment) το οποίο θα χρησιμοποιήσουμε για την ανάπτυξη του

backend μας με Spring Boot. Το IntelliJ Idea είναι ένα πρόγραμμά που εξυπηρετεί πολύ εύστοχα για κάθε ανάπτυξη λογισμικού Java και παρέχει αρκετά βοηθήματα που θα χρειαστούμε για την εγκατάσταση του backend μας , όπως το Gradle , που θα κατεβάσουμε τα πακέτα μας. [4]

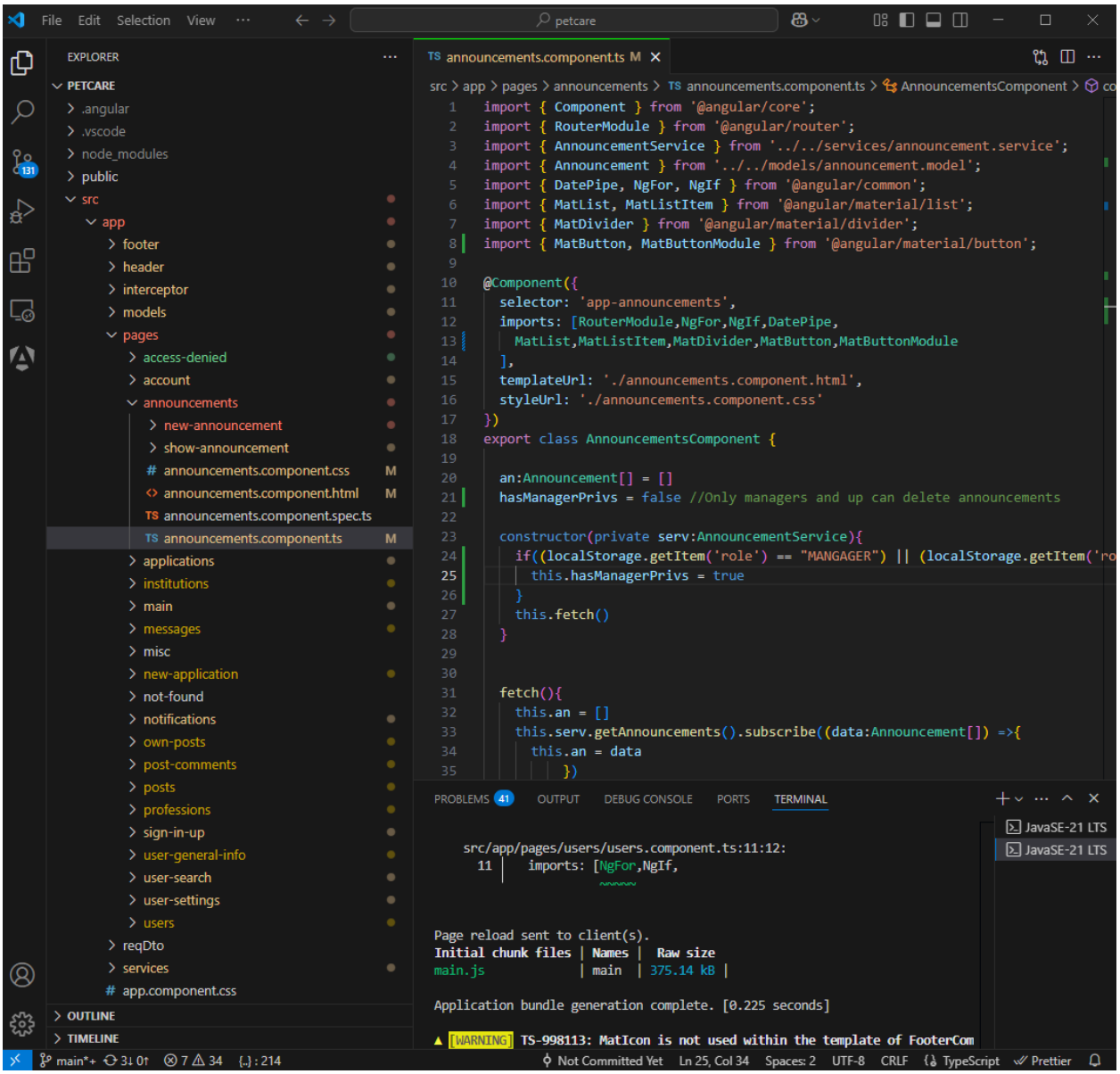


Εικόνα 3.1 - Εικόνα περιβάλλοντος του Intelij Idea με απόσπασμα από της εργασίας μας.

3.2.2 VsCode

Το VsCode είναι ένα IDE το οποίο εξυπηρετεί στην ανάπτυξη διαφόρων ειδών λογισμικού . Είναι ελαφρύ και προσαρμόζεται εύκολα στις ανάγκες του χρήστη και με αυτόν τον τρόπο μπορούμε να το χρησιμοποιήσουμε για διάφορες γλώσσες προγραμματισμού που δεν θέλουν κάποιο απαραίτητο λογισμικό για να λειτουργήσουν. Θα το χρησιμοποιήσουμε για την ανάπτυξη του frontend μας και κυρίως για HTML (Hypertext Markup Language) , CSS (Cascading Style Sheets) και Angular. Παρέχει πολλές βασικές λειτουργείες και κάνει πιο γρήγορη την διόρθωση κώδικά , αφού το VS Code υποστηρίζει πολλές γλώσσες

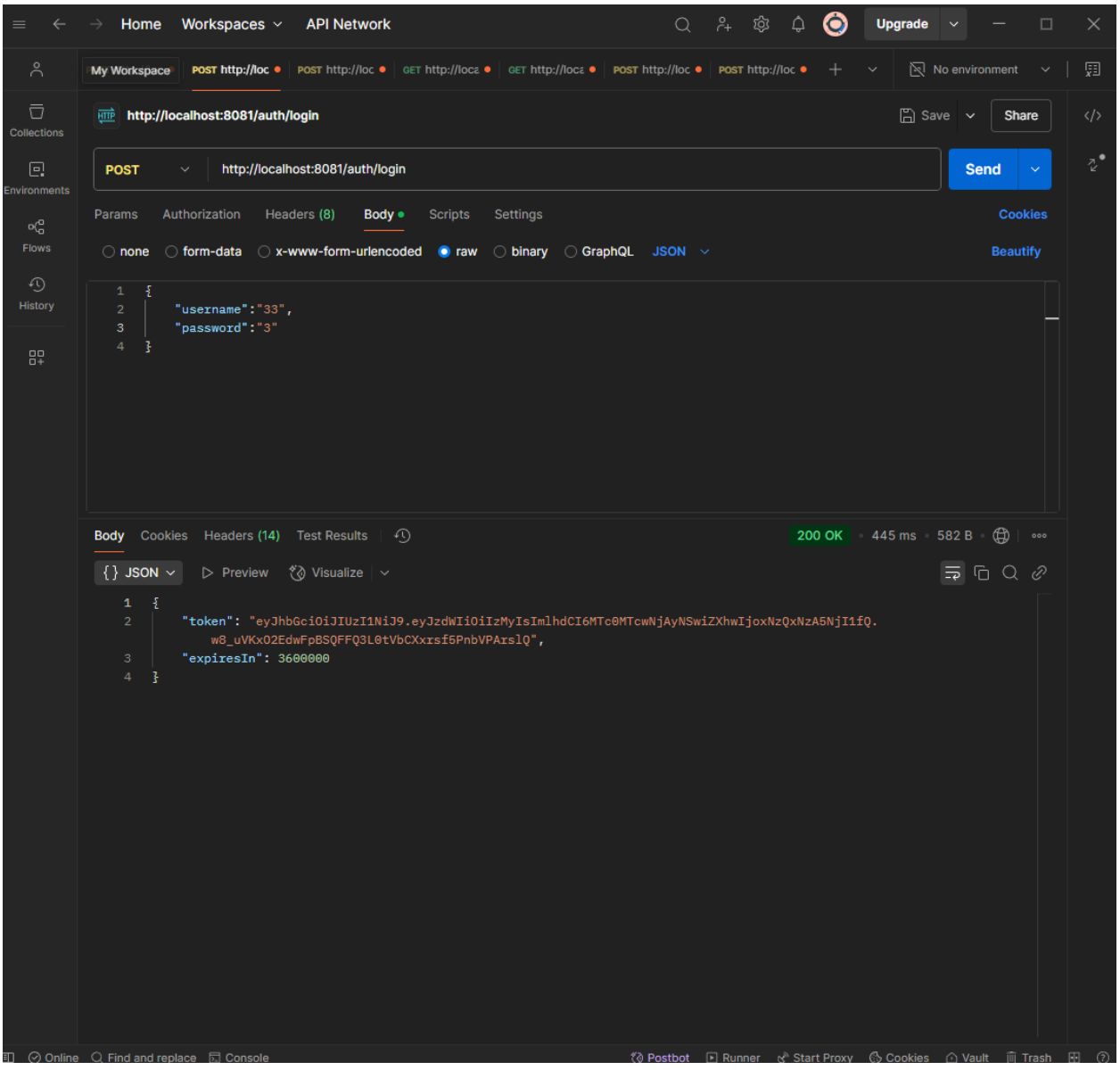
προγραμματισμού. Μπορεί να συνοδευτεί η λειτουργεία του με packages που κάνουν την λειτουργεία του ακόμα πιο εύκολη. [5]



Εικόνα 3.2 - Απόσπασμα της εργασίας μας για το περιβάλλον VsCode. Περιέχει το frontend (Angular). Από αριστερά είναι τα αρχεία μας , από κάτω δεξιά είναι το τερματικό που θα τρέχουμε τις εντολές που θέλουμε και πάνω αριστερά με αριθμό 131 είναι το εικονίδιο για το

3.2.3 Postman

Το Postman είναι ένα πρόγραμμα με το οποίο μπορούμε να στέλνουμε API (Application Programming Interface) στον server (Spring Boot - backend) με εύκολο τρόπο χωρίς να χρειαστεί η δημιουργία κώδικα. Μόνο με την επικόλληση URL μπορούμε να πάρουμε τις πληροφορίες που θέλουμε σε JSON και να ελέγχουμε (testing) αν το backend μας λειτουργεί σωστά χωρίς λάθη. [6]



Εικόνα 3.3 - Παράδειγμα χρήσης Postman.

Παράδειγμα χρήσης Postman. Αρχικά βάζουμε το URL που θέλουμε να τρέξουμε (http://localhost:8081/auth/login) Επιλέγουμε την διαδικασία δίπλα (POST) . Ακριβώς από κάτω στην καρτέλα body βάζουμε τα Json αρχεία που θέλουμε να στείλουμε (Εδώ κάνουμε login request , username και password , είναι λογαριασμός για δοκιμές). Τέλος στο πλαίσιο πιο κάτω είναι η απάντηση του backend (server response) , που είναι επίσης ένα Json με το token μας (Βλέπω JWT (JSON Web Token) token στην υλοποίηση backend).

3.2.4 Docker

Τέλος για την εγκατάσταση της βάσης δεδομένων μας θα χρησιμοποιήσουμε το Docker. Το Docker παρέχει υπηρεσίες πλατφορμών (PaaS - Platform as a Service) και .Μπορούμε να αποθηκεύσουμε και να εγκαταστήσουμε λογισμικό σε δοχεία (containers) όπου δεν θα επικοινωνούν με το υπόλοιπο λειτουργικό μας σύστημα. Το Docker θα το χρησιμοποιήσουμε για την εγκατάσταση της βάσεως δεδομένων. Δεν είναι απαραίτητο να χρησιμοποιήσουμε Docker , αλλά με αυτό μπορούμε εύκολα να εγκαταστήσουμε την βάση που θέλουμε και συνιστάται για να αλλάζουμε εύκολα την έκδοση (version) της βάσης. [7]

3.3 Εγκατάσταση

Για την εγκατάσταση του VSCode, του Intelij , του Postman και του Docker παίρνουμε τα εκτελέσημα από τις ιστοσελίδες τους . Για να μπορέσουμε να εγκαταστήουμε την Angular θα θέλουμε να έχουμε εγκατεστημένο το Node.js . Μόλις το κατεβάσουμε μπορούμε να εγκαταστήσουμε την Angular με αυτό πηγαίνοντας στο VSCode και γράφωντας στο ενσωματομένα τοερματικό του (Terminal).

Πίνακας 3.1 – Εγκατάσταση της Angular με το NodeJs.

```
npm install -g @angular/cli
```

Επιπλέον επειδή έχουμε βάλει .gitignore το project μας έχω βάλει να μην ανεβαίνουν τα Angular packages που κατεβάσαμε , γι’ αυτό για να τρέξει κάποιος το project πρέπει να εγκαταστήσει και τα packages.

Πίνακας 3.2 - Εγκατάσταση πακέτων στην Angular.

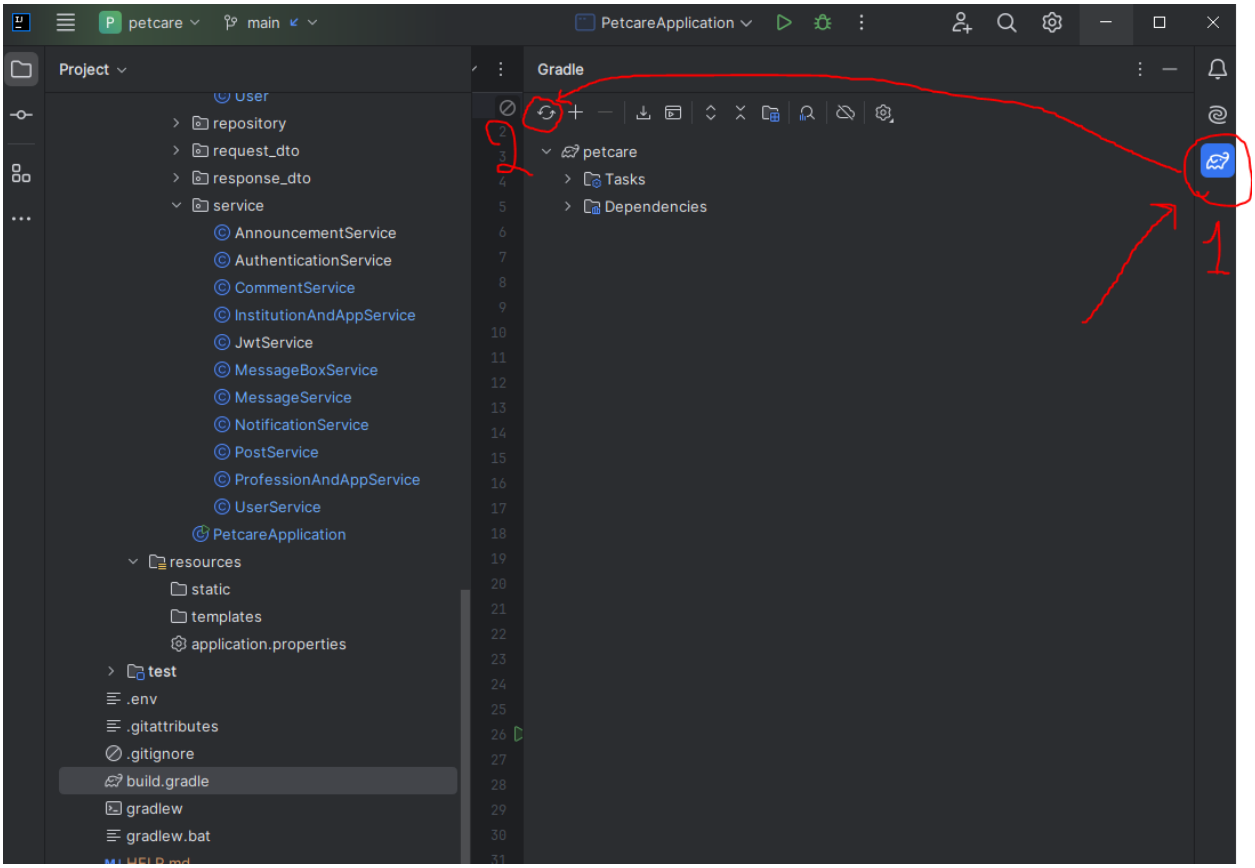
```
npm install leaflet
npm i --save-dev @types/leaflet


ng add @angular/material
```

Εγκατάσταση των πακέτων της Angular (leaflet για προβολή χάρτη και , angular material για γραφικά) στο τερματικό μας.

Η παραπάνω εντολές θα πρέπει να τρέξουν όσο είμαστε με το τερματικό μας μέσα στον αρχικό φάκελο του project μας , για να φτιαχτεί ένας φάκελος node_modules/ που θα έχει όλα τα πακέτα κώδικα που θα χρειαστούμε.

Θα χρειαστεί να εγκαταστήσουμε και τα dependencies του Gradle για το Spring Boot γιατί λόγω του .gitignore μπορεί να μην υπάρχουν στο GitHub. Το IntelliJ κάνει την πιο πολύ δουλεία μόνο του για να αναγνωρίσει το project , απλά μπορεί να χρειαστεί να κάνουμε ανανέωση από το εικονίδιο του Gradle δεξιά και μετά από το κουμπί ανανέωσης που λέει “Sync all gradle projects”



Εικόνα 3.4 - Εγκατάσταση dependencies μέσω Gradle στο IntelliJ Idea.

Για το Docker , πέρα από το πρόγραμμα θα πρέπει να εγκαταστήσουμε και την βάση μας σε αυτό. Θα επιλέξω την βάση δεδομένων MariaDB γιατί είναι αρκετά διαδεδομένη και μοιάζει αρκετά στην MySQL (μία άλλη βάση δεδομένων).Εκτός από την βάση μας θα πρέπει να εγκαταστήσουμε και το πρόγραμμα διεπαφής μας με την βάση. Σε αυτήν τη περίπτωση έχω επιλέξει το phpMyAdmin το οποίο είναι επίσης πολύ διαδεδομένο.

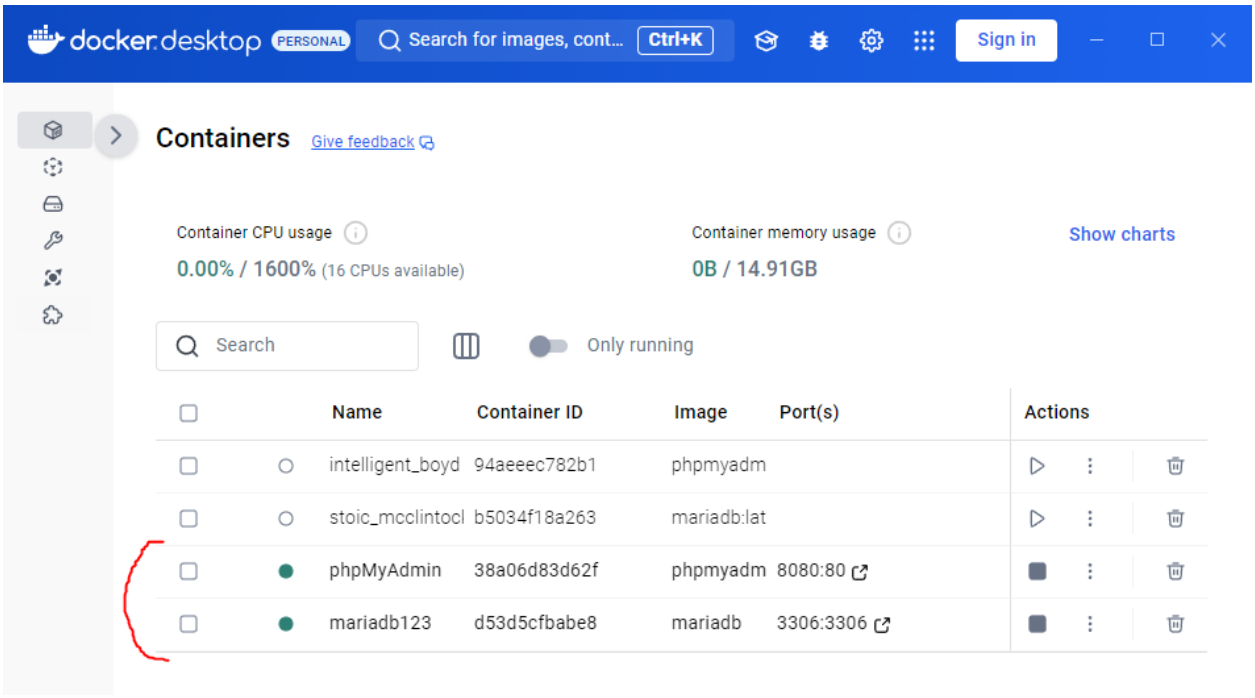
Και τα δύο αυτά τα προγράμματα θα εγκατασταθούν σε δοχεία (containers) του Docker και θα δουλεύουν σαν να είναι σε δικιά τους εικονική μηχανή (virtual machine). Για να εγκατασταθούν θα πρέπει να τρέξουμε τον κώδικα παρακάτω (Σε PowerShell).

Πίνακας 3.3 - Εγκατάσταση των container με την βάση και την διαχείριση της βάσης στο Docker.

```
docker pull phpmyadmin/phpmyadmin
docker run --name=mariadb123 -e MYSQL_ROOT_PASSWORD=123456 -e
MYSQL_DATABASE=mydp -p 3306:3306 -d mariadb
docker run --name phpMyAdmin -d --link mariadb123:db -p 8080:80
phpmyadmin/phpmyadmin
```

Παραπάνω είναι ο κώδικας εγκατάστασης της βάσης , στις γραμμές 2 και 3 εγκαθιστούμε το MariaDB στην υποδοχή (port) 3306 και στις γραμμές 1 και 4 εγκαθιστούμε το πρόγραμμα εξυπηρέτησης phpMyAdmin , το οποίο ενώνουμε με την βάση και μέσω του port 8080 θα πηγαίνουμε στον σύνδεσμο http://localhost:8080/ και θα κάνουμε τις αλλαγές μας.

Έπειτα , το Docker μας θα έχει 2 container ως εξής:



Εικόνα 3.5 – Εκκίνηση διαχείριση βάσης δεδομένων (phpMyAdmin) και βάσης (MariaDB).

Χρειάζεται επίσης να δημιουργήσουμε κάποιους φακέλους για να δουλέψει το πρόγραμμα. Επειδή έχω βάλει εικόνες οι ποιες είναι ανεβασμένες σε αρχείο σε έναν ξεχωριστό φάκελο από το project (Είναι πιο σωστό να μην είναι μέσα στο Spring όλες οι εικόνες) , θα πρέπει να δημιουργηθούν και κάποιοι φάκελοι. Οι διευθύνσεις βρίσκονται στο application.properties

Πίνακας 3.4 - Πόρισμα από application.properties για την τοποθεσία των εικόνων του Spring Boot.

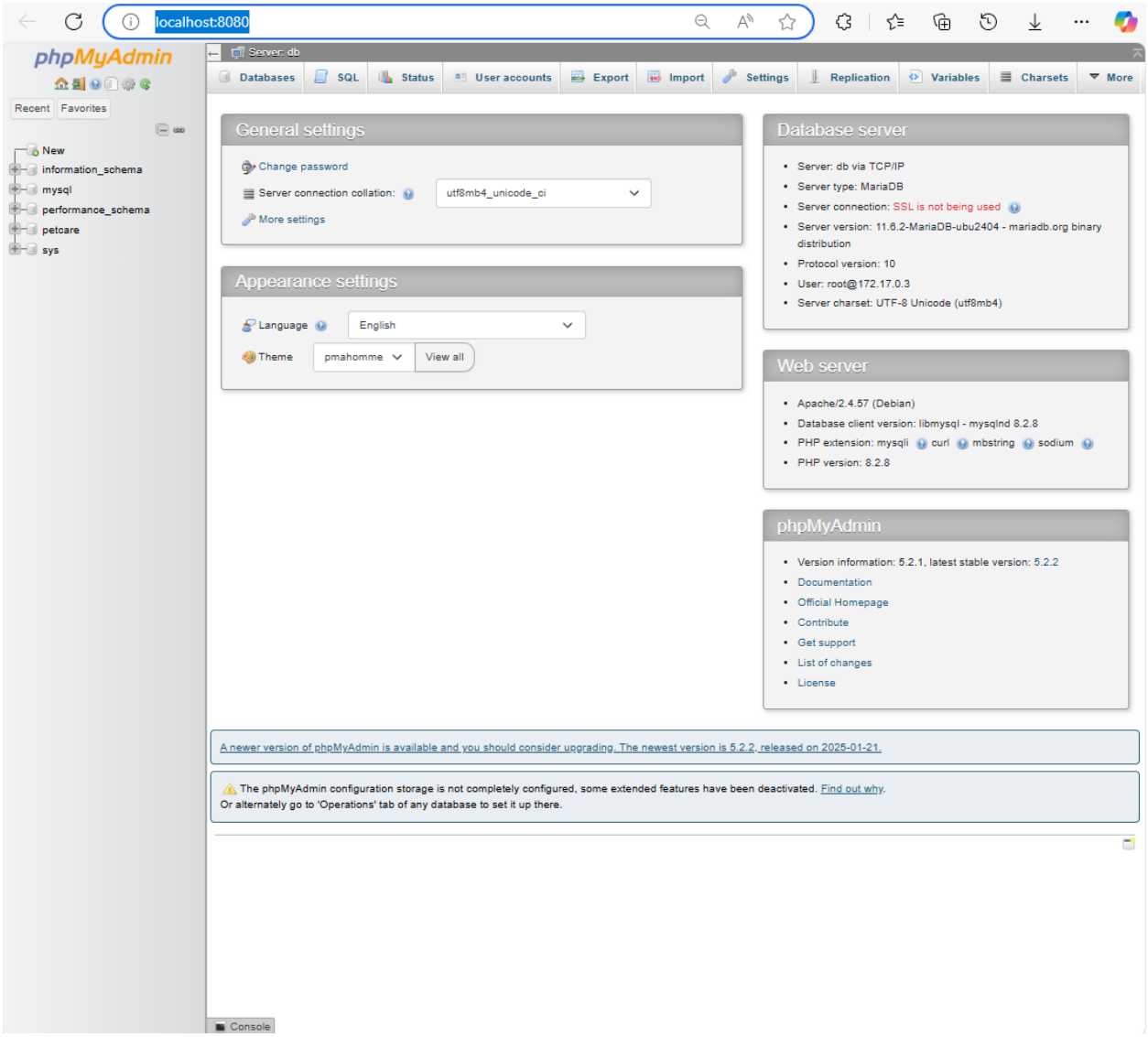
```
file.image.path.post=C:/data/post
file.image.path.account=C:/data/account
```

Θέλουμε 3 φακέλους που θα είναι στο C Directory (data/ , account/ και post/). Επέλεξα εαυτόν τον φάκελο γιατί είναι σχετικά προσβάσιμος σε όλους (Δεν χρειάζεται να δημιουργήσουμε πάρα πολλά αρχεία).

3.4 Χρήση προγραμμάτων

Αρχικά για να ανοίξουμε την ιστοσελίδα όλη με το frontend και backend θα πρέπει να ανοίξουμε την βάση. Μέσω της εικόνας παραπάνω για το Docker βλέπουμε τα 2 τελευταία αντικείμενα που είναι για την βάση μας ότι έχουν ένα πράσινο κύκλο , αυτό σημαίνει ότι είναι ανοιχτά. Για να τα ανοίξουμε θα πρέπει να πατήσουμε το εικονίδιο “Play” και περιμένουμε μερικά δεύτερα.

Τώρα μπορούμε να επισκεφτούμε την βάση άμα θέλουμε. Το post 3306 δεν κάνει κάτι στο browser αλλά θα χρειαστούμε το 8080 όπου άμα συνδεθούμε στο http://localhost:8080/ και βάλουμε τα στοιχεία μας (username root και password 123456 όπως τα θέσαμε όταν τα εγκαταστήσαμε στο docker) .

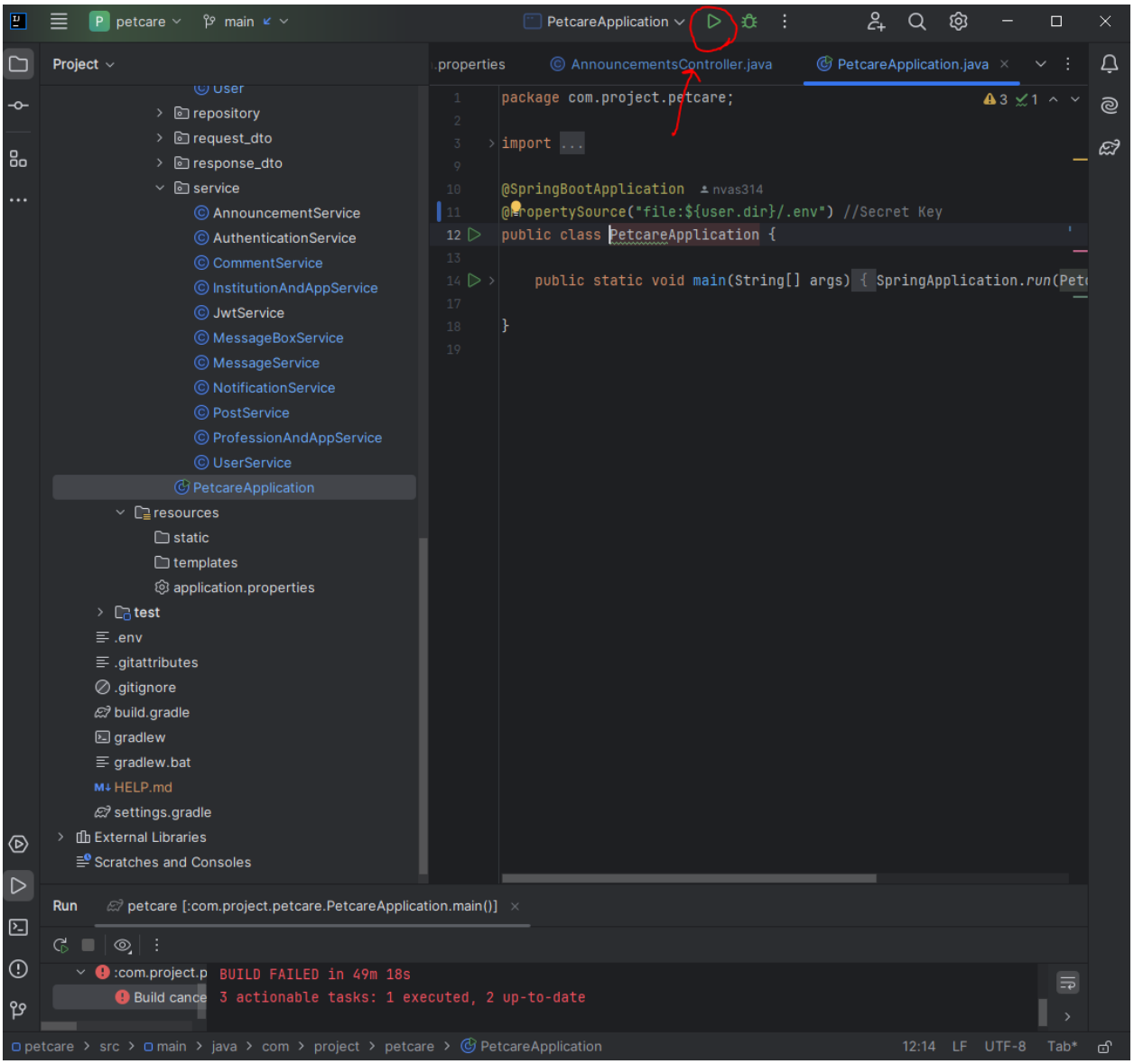


Εικόνα 3.6 - Η σελίδα του localhost:8080 (phpMyAdmin).

Παραπάνω στην σελίδα βλέπουμε από τα αριστερά τις βάσεις μας. Μπορούμε να βρούμε και την βάση petcare που θα χρησιμοποιήσουμε (την θέτουμε πιο μετά στο κείμενο , στο application.properties) , τώρα μπορεί να έχουμε μία άλλη που μπορούμε να διαγράψουμε.

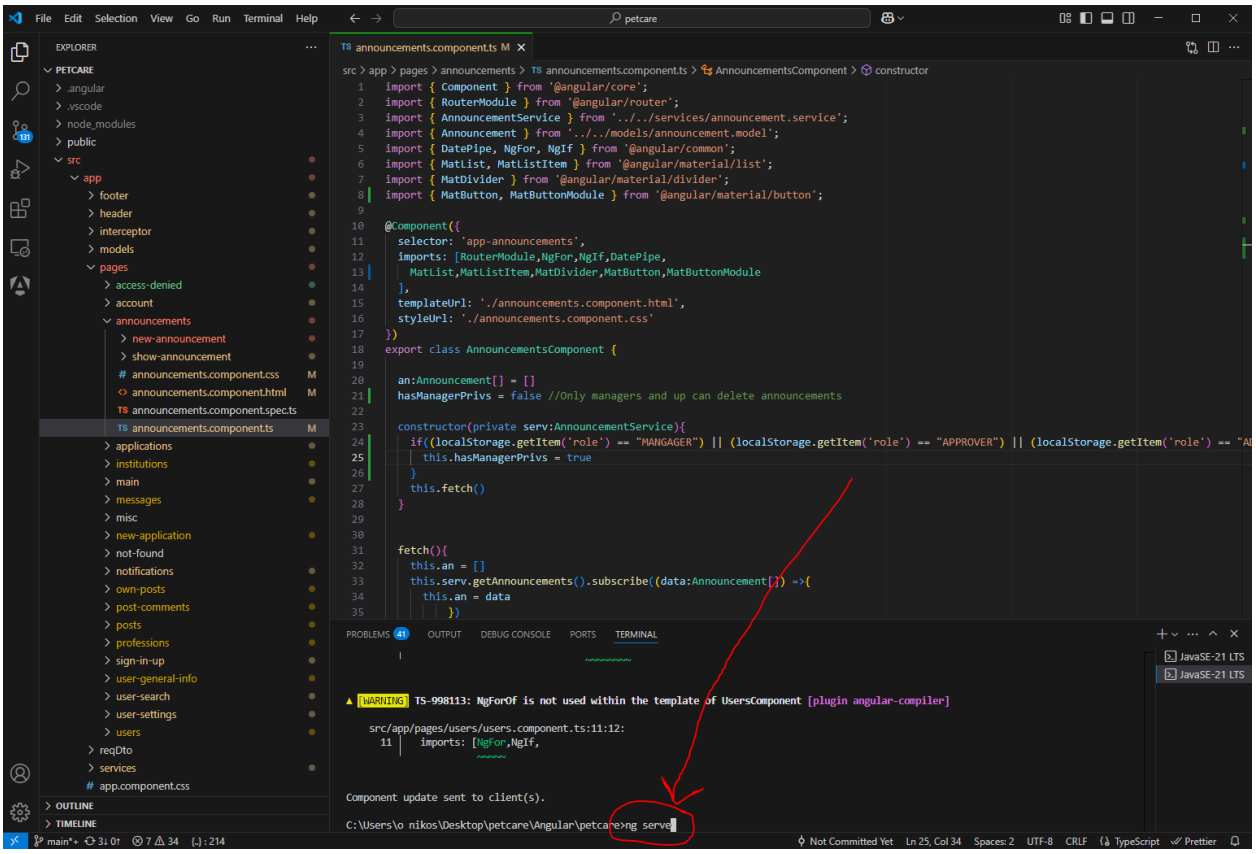
Έπειτα όταν ανοίξει η βάση μπορούμε να ανοίξουμε το Spring boot backend. Για να το κάνουμε θα πατήσουμε το κουμπί “Play” που βρίσκεται πάνω δεξιά. Άμα δεν εμφανίζεται , τότε πάμε στο αρχείο της main του project μας (PetcareApplication.java) και εκεί θα υπάρχει

ένα κουμπί που πρέπει να πατήσουμε. Περιμένουμε μερικά δευτερόλεπτα και από κάτω στην κονσόλα φαίνεται να τρέχει η διαδικασία που πρέπει. Μόλις ανοίξει θα μπορούμε να ανοίξουμε το localhost:8081 στο οποίο δεν βάλαμε να κάνει κάτι μέσα για το browser.



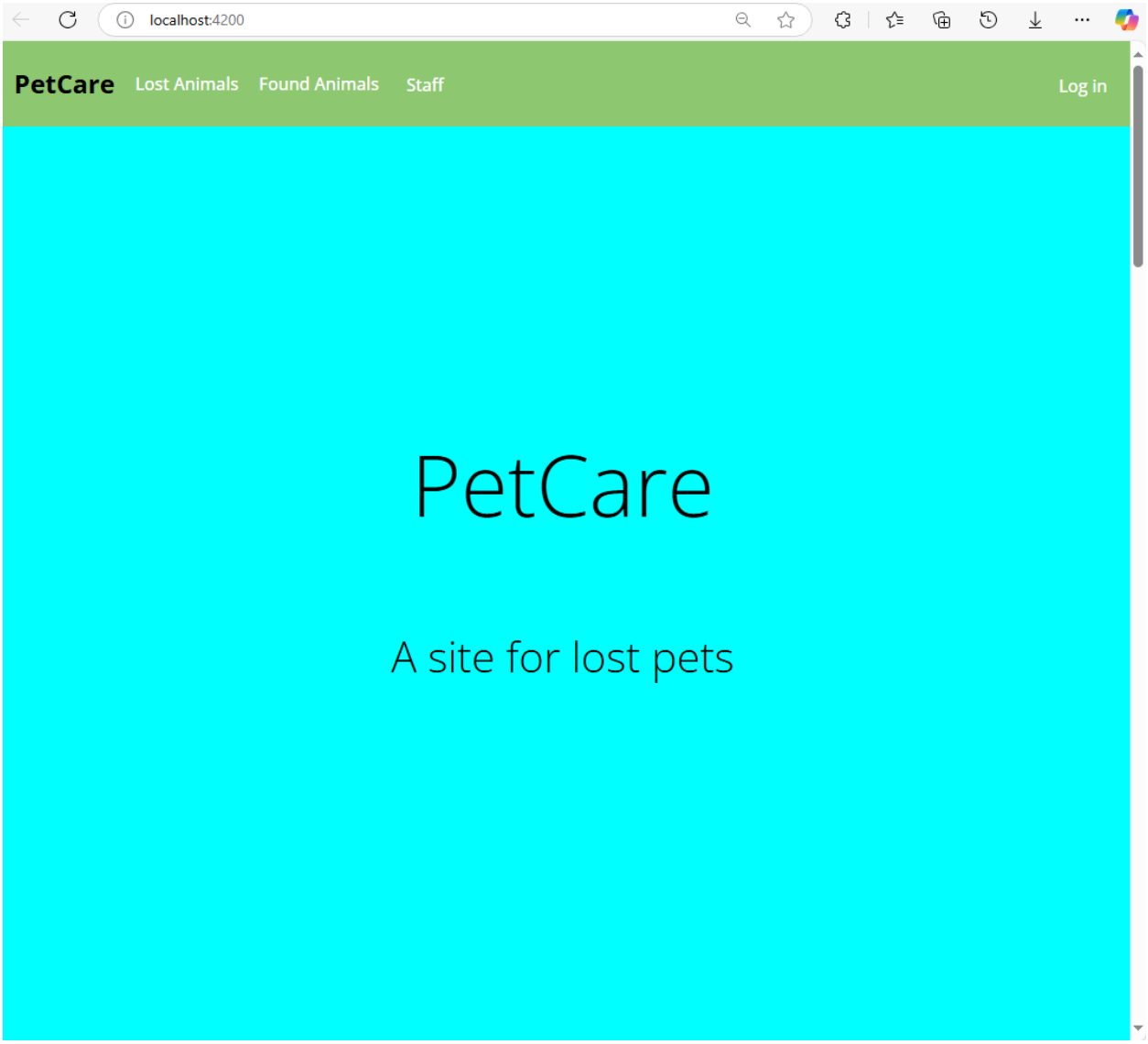
Εικόνα 3.7 - Εκκίνηση Spring Boot backend.

Τέλος θα πρέπει να ανοίξουμε το frontend μας (την Angular). Για να γίνει αυτό θα πρέπει να μπούμε στο VsCode και από το τερματικό να πάμε σε έναν φάκελο που είναι μέσα στο frontend (μέσα στον φάκελο Angular/petcare/ που έχουμε) και να γράψουμε εκεί ng serve. Μετά από μερικά δευτερόλεπτα θα ανοίξει και το frontend μας.



Εικόνα 3.8 - Εκκίνηση Angular frontend.

Μόλις ανοίξει μπορούμε να επισκεφτούμε τον ισότοπο localhost:4200 που μπορούμε πιο εύκολα να κάνουμε με το να γράψουμε αγγλικό “o” στο τερματικό και μετά να πατήσουμε Enter.



Εικόνα 3.9 - Αρχική σελίδα Petcare.

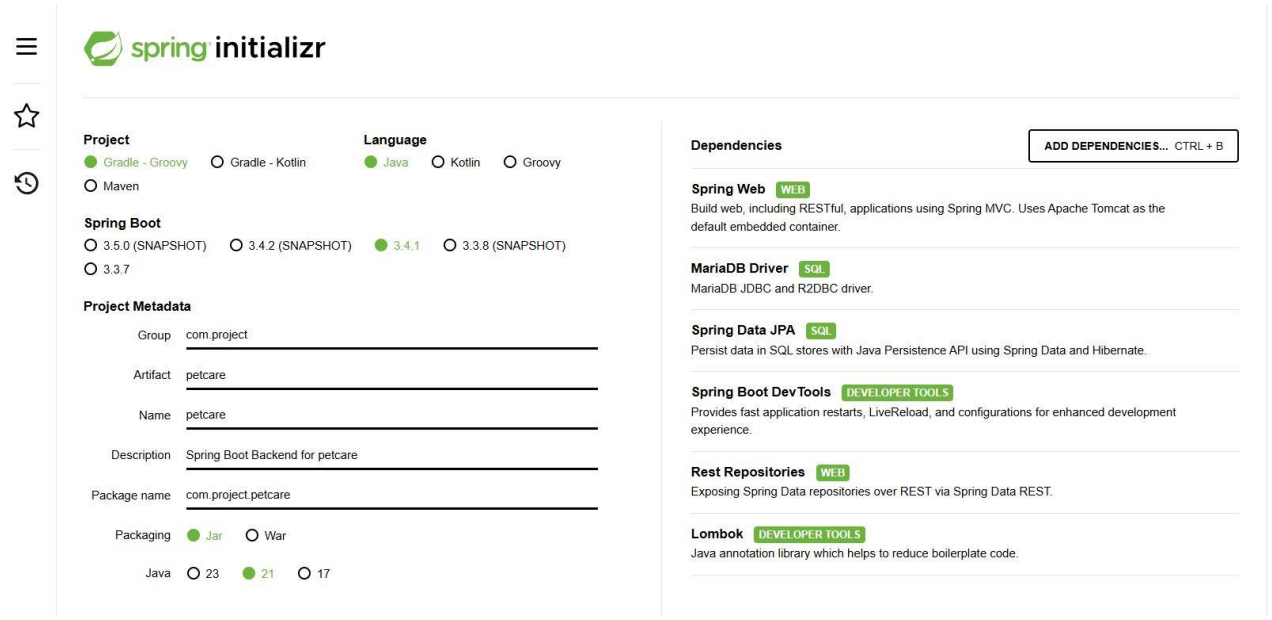
Η αρχική σελίδα του site μας.

4. Υλοποίηση

4.1 Backend

4.1.1 Δημιουργία project

Για να μπορέσουμε να αρχίσουμε να προγραμματίζουμε στο Spring boot framework θα πρέπει πρώτα να φτιάξουμε ένα project με όλα τα αρχεία του Spring που μας βοηθάνε να αρχίσουμε έναν server. Για να το κάνουμε αυτό μπορούμε μέσω της ιστοσελίδας https://start.spring.io/ να φτιάξουμε ένα αρχικό κενό project στο οποίο μπορούμε μέσα του να φτιάξουμε αρχεία που θα εξυπηρετούν στην λειτουργεία του Backend που θέλουμε.



Εικόνα 4.1 - Spring Intializr. Δημιουργία νέου project.

Από πάνω βλέπουμε μία εικόνα από το site στο οποίο θα φτιάξουμε την εργασία μας. Μπορούμε να επιλέξουμε τις επιλογές που θέλουμε , έχω επιλέξει να χρησιμοποιήσω build.gradle ως βοηθό , γλώσσα Java με την αντίστοιχη έκδοση (21).Από δεξιά βλέπουμε όλες τα dependencies από τα οποία η εργασία θα εξαρτάται για να λειτουργήσουν μερικές προεκτάσεις που θα θέλουμε να βάλουμε . Δεν χρειάζεται εδώ να τα βάλουμε όλα όμως γιατί μπορούμε και μέσα από το project μας να βάλουμε στο αρχείο build.gradle έπειτα ότι dependencies θέλουμε.

Όταν ολοκληρώσουμε την επιλογή μας θα μπορούμε να κατεβάσουμε το πρόγραμμα και μετά να το τρέξουμε. Για να το τρέξουμε επιλέγουμε τον φάκελο με το IntelliJ και το βρίσκει από μόνο του.

Η δομή του project μας έχει αρκετά αρχεία ,αλλά εμείς θα βασιστούμε στα αρχεία που είναι στον φάκελο scr/ και σε αρχεία στον αρχικό φάκελο που project , όπως το Gradle. Συγκεκριμένα το βασικό πρόγραμμά μας θα βρίσκεται στον φάκελο src/main/java/ όπου εκεί θα έχουμε όλες τις τάξεις (classes) μας με ότι αρχείο χρειάζεται. Επιπλέον για να μπορέσει να τρέξει το πρόγραμμά μας σωστά και να ενωθεί με την βάση που έχουμε θα πρέπει να βάλουμε το εξής configuration στο αρχείο src/main/java/resounces/application.properties :

Πίνακας 4.1 - Πόρισμα από application.properties για την σύνδεση της βάσης δεδομένων με το Spring Boot backend.

```
spring.application.name=petcare
spring.datasource.url=jdbc:mariadb://localhost:3306/petcare?autoReconnect=true&creat
eDatabaseIfNotExist=true&useSSL=false&useLegacyDatetimeCode=false&serverTimezone
=UTC
spring.datasource.username=root
spring.datasource.password=123456


spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect

server.port=8081
```

Στον παραπάνω κώδικα επιλέγουμε αρχικά την βάση δεδομένων μάς με το spring.datasource.url και συνδεόμαστε με αυτήν με το username και password πιο κάτω. Τα υπόλοιπά είναι απλές ρυθμίσεις που θα πρέπει να κάνει στο Spring με την βάση μας

Τώρα μπορούμε να ανοίξουμε την βάση όπως δείξαμε στο παραπάνω κεφάλαιο και να γίνει η σύνδεση.

4.1.2 Dependencies

Αυτή είναι η τελική μορφή του αρχείου build.gradle που βρίσκεται στον αρχικό φάκελο:

Πίνακας 4.2 - Πόρισμα από το αρχείο build.gradle για τα dependencies.

```
dependencies {
   implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
   implementation 'org.springframework.boot:spring-boot-starter-data-rest'
   implementation 'org.springframework.boot:spring-boot-starter-web'
   compileOnly 'org.projectlombok:lombok'
   developmentOnly 'org.springframework.boot:spring-boot-devtools'
   runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
   annotationProcessor 'org.projectlombok:lombok'
   testImplementation 'org.springframework.boot:spring-boot-starter-test'
   testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
   //
   implementation "org.springframework.boot:spring-boot-starter-security"
   implementation 'org.springframework.security:spring-security-test'
   implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
   implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
   implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'
   implementation 'org.springframework.boot:spring-boot-starter-validation'

   implementation 'com.auth0:java-jwt:4.4.0'

   implementation 'org.springframework.boot:spring-boot-starter-oauth2-resource-server'
}
```

Παρατηρούμε ότι τα πρώτα dependencies είναι και αυτά που βάλαμε από το initializr. Γενικά η δουλειά των dependencies είναι να διαθέτουν κώδικα για κάποιες παραπάνω λειτουργείες του backend όπως να δημιουργει τους πίνακες της βάσης από τα models μας (με το JPA και το Hibernate) και λειτουργικότητα των JWT.

4.1.3 Βασική δομή των αρχείων Spring Boot

Για να δουλέψει σωστά μία εφαρμογή Spring Boot με REST API θα χρειαστεί τα παρακάτω αρχεία:

4.1.4 Controllers:

Παρέχει το API (Application Programming Interface) στο πρόγραμμα που το χρησιμοποιεί (εδώ Angular και Postman). Δημιουργεί μία διεπαφή μεταξύ frontend και backend υλοποιώντας endpoints που χρησιμοποιούνται ως REST API (Representational State Transfer API) τα οποία είναι URLs που μέσω αυτών επικοινωνεί και επιστρέφει την ανάλογη πληροφορία.

Κάθε controller μπορεί να περιέχει πολλά endpoints τα οποία όταν τα καλεί ο χρήστης με το frontend να κάνουν κάτι διαφορετικό που να συμβάλουν όμως στην ίδια κατηγορία πληροφοριών (πχ ένα controller να ασχολείται με τα σχόλια).Έχουν το Annotation (το οποίο δείχνει πως θα χρησιμοποιηθεί μία τάξη ή μία συνάρτηση στο πρόγραμμα, αρχίζουν πάντα με @) @RestController

για τον λόγο που αναφέραμε πιο πάνω.

Για να μπορέσουμε να επικοινωνήσουμε με το Backend μέσω του frontend χρησιμοποιούμε από το frontend το REST API τα οποία είναι links που ορίζονται από τα controllers και αρχίζουν μία συγκεκριμένη συνάρτηση. Το REST API που χρησιμοποιούμε είναι στην ουσία URLS τα οποία όταν καλούνται από πάνε στο backend και αυτό διαχειρίζεται το URL με το τρόπο του χρειάζεται. Τα URLs αυτά αρχίζουν με την τοποθεσία του backend που θέλουμε να επικοινωνήσουμε (Εδώ είναι το http://localhost:8081/ με το port 8081 που θέσαμε στο application.properties και βάζοντας μετά ένα συγκεκριμένο path που θα επικοινωνήσει με ένα controller)

Σε κάθε controller θα δούμε ότι υπάρχουν συναρτήσεις που υποστηρίζουν ένα URL (Μέσω ενός Mapping που υπάρχει σε κάθε Annotation του controller) και επιστρέφουν μία απάντηση μετά σε αυτόν που το κάλεσε , που μπορεί να είναι ένα HTTP Response (Ένας τριψήφιος αριθμός που εγκρίνει αν ένα αίτημα στον server ήταν αποτελεσματικό ή όχι) μαζί με ένα JSON αρχείο που μπορεί να είναι δεδομένα που ζητήσαμε (DTOs από τον server δηλαδή που θα αναφέρουμε παρακάτω) .

Το κάθε controller που θα διαχειρίζεται το API θα κάνει ανασκόπηση μέσω του αρχείου service για να επιστρέψει αυτό δεδομένα που μετά θα τα επιστρέψει το controller πίσω σε εμάς. Μπορούμε επίσης να εμποδίσουμε σε κάποιους χρήστες την επαφή τους με κάποια controller μέσω κάποιου ονόματος στο path (τοποθεσία) του URL που καλούμε (αν πχ έχει

path /admin/ να μην μπορεί να το εκτελέσει ένας απλός χρήστης το API και να του επιστραφεί ERROR response από τον server)

Τα URLS χωρίζονται σε 4 βασικές κατηγορίες ανάλογα την δουλειά που θα θέλουμε να κάνουμε οι οποίες είναι POST,GET,DELETE,PUT.Εκτός από το URL θα πρέπει να υπάρχει και αυτήν την επισήμανση για το τι θα πρέπει να κάνει το controller στην περίπτωση αυτή. Το POST είναι για την αποστολή δεδομένων , το GET είναι για να πάρουμε δεδομένα , το DELETE για να διαγράψουμε και το PUT για να αλλάξουμε κάποια δεδομένα. Εκτός από αυτά σε κάθε controller θα πρέπει να επισημάνουμε το CORS Configuration (Cross-origin Resource Sharing) στο οποίο θα διευκρινίζεται από ποια μέρη θα ακούει το κάθε controller μας για να κάνει τις λειτουργείες του. Συγκεκριμένα εδώ χρειαζόμαστε μόνο την τοποθεσία του Angular frontend μας που θα κάνει τα API CALLS (http://localhost:4200).

Τα controllers έχουν Annotation @RestController.Επίσης έχουν και mappings όπως το @RequestMapping("/take") που είναι στην αρχή του path μετά το localhost και μας λέει πιο controller θα χρησιμοποιήσουμε . Επίσης χρησιμοποιούμε συγκεκριμένα mapping όπως @GetMapping("/user/institution/{post_id}/{i_id}") που μιλάμε για συγκεκριμένες συνάρτησης και όπως εδώ μπορούμε να βάλουμε παραμέτρους εκεί που είναι οι αγκύλες.

4.1.5 Models:

Τα models είναι όλες οι κατηγορίες δεδομένων που αναφέραμε παραπάνω στο κείμενο. Κάθε model σχηματίζει το πως κάθε κατηγορία δεδομένου θα συνυπάρχει στην βάση και συμβολίζει έναν πίνακα σε αυτήν. Κάθε model είναι μία τάξη όπου οι μεταβλητές της μπορούν να συμβολίζουν μία στήλη στον πίνακα της βάσης μας ή ένα κλειδί που θα ενώνει έναν άλλον πίνακα (foreign key). Κάθε model θα ορίζει έναν τύπο αρχείου που θέλουμε να αποθηκεύσουμε (Posts , Comments ,κλπ.).

Κάθε μεταβλητή στον model μπορεί να οροθετηθεί με κάποιους περιορισμούς οι οποίοι θα φιλτράρουν τι είδους αντικείμενα μπορούν να δημιουργηθούν πριν αποθηκευτούν στην βάση. Εδώ μπορούμε να οριοθετήσουμε βασικούς κανόνες που θα συμβάλουν στην ορθή λειτουργεία των δεδομένων με το υπόλοιπο πρόγραμμα, όπως το να μην είναι μία μεταβλητή κενή (@NotNull Annotation) ή στην μοναδικότητα ενός String (@Column(unique = true)) που το χρειαζόμαστε στα usernames.Τα μοντέλα έχουν το Annotation @Entity που λέει ότι η τάξη μπορεί να γραφθεί και ως πίνακας στην βάση.

Με τα models μπορούμε να σχηματίσουμε το πως θα είναι σχηματισμένα οι πίνακες μας στην βάση. Μπορούμε να κάνουμε συνδέσεις πινάκων (onetoone , onetomany , κπλ) με το να συνδέουμε τα models μεταξύ τους. Αν ένα model έχει πολλά από ένα άλλο θα εμφανίζεται το άλλος σε λίστα , αλλιώς σαν ένα μόνο αντικείμενό του.

Κάθε αρχείο model στο backend σχηματίζει έναν πίνακα στην βάση δεδομένων , κάθε μεταβλητή σχηματίζει μία σειρά (column) σε αυτή ή ένα κλειδί (foreign key).Μέσω του διαγράμματος ERD που αναφέραμε στο κεφάλαιο 2 μπορούμε να δούμε ότι ο κάθε ένας

πίνακας σχηματίστηκε από ένα αρχείο model στο Spring Boot με το όνομα αυτό (που είναι στον φάκελο models/)

4.1.6 Repositories

Τα repositories είναι interfaces που διαχειρίζονται το πως θα επεξεργάζονται τα δεδομένα σε μία βάση. Μέσω των models που δημιουργήσαμε μπορούμε για κάθε αντίστοιχο model (και μόνο ένα την φορά) να δημιουργήσουμε έναrepository που θα διαχειρίζεται τα αρχεία. Ένα repository μπορεί να διαγράψει , να εισάγει ή να πάρει αρχεία από μία βάση , ακριβώς ότι μπορούμε να κάνουμε και με εντολές SQL. Αυτό ονομάζεται dao (Data Access Object) γιατί μεταφέρει αρχεία SQL και τα δημιουργεί με εύκολο τρόπο σε τάξης Java [8]. Έχουν το Annotation @Repository

Τα repositories θα έχουν την ευθύνη να τραβάνε πληροφορίες από την βάση. Μέσω κάποιων ερωτημάτων (queries) που θα ορίσουμε στην τάξη αυτή θα μπορούμε μετά να επιστρέφουμε τα δεδομένα που ανακαλέσαμε. Το τι θα τραβάμε θα ορίζεται από συναρτήσεις που θα βάλουμε στο repository class τα οποία είναι έτοιμα από το Spring με μία ονομασία ή θα πρέπει να δημιουργήσουμε συγκεκριμένο query σε Annotation σε γλώσσα SQL.

Όταν αυτές οι συναρτήσεις επιστρέψουν κάποιες πληροφορίες αυτές θα έχουν την δομή αντικειμένων σε μία λίστα ή ενός αντικειμένου , που μετά μπορούμε τα το επεξεργασθούμε για να στείλουμε το τελικό αποτελέσματα.

Παράδειγμα repository CommentRepository.java. Με τις συναρτήσεις που έχουμε εκεί μπορούμε να τις χρησιμοποιήσουμε με το service και να πάρουμε τα δεδομένα που λέει στο όνομα της συνάρτησης.

Πίνακας 4.3 - Παράδειγμα αρχείου repository στο Spring.

```
@Repository
public interface CommentRepository extends CrudRepository<Comment,Long> {//Crud??
   List<Comment> findAll();
   List<Comment> findByPostId(Long postId);
}
```

4.1.7 Services:

Στα services υλοποιείται η βασική λειτουργικότητα του συστήματος όπως οι χρήστες να αποκτήσουν από την βάση τα αρχεία που χρειάζονται και να κάνει τις ενέργειες που του έχουν δοθεί. Μέσω του controller που καλούμε από το frontend μπορούμε να καλούμε

συναρτήσεις που είναι μέσα σε ένα service και να μας επιστρέφει τα δεδομένα μας πίσω στο controller που θα παραδώσουμε στον χρήστη που τα κάλεσε. Τα αρχεία service έχουν το Annotation @Service

4.1.8 Dtos (Data Transfer Objects):

Τα Dtos (ή αλλιώς Data Transfer Objects) είναι τάξεις με τις οποίες μπορούμε να μεταφέρουμε και να στείλουμε δεδομένα στο frontend. Στην ουσία είναι ένα σχεδιάγραμμα για το τί θέλουμε να μεταφέρουμε. Τα Dtos χωρίζονται σε 2 κατηγορίες αυτά που παίρνουμε και αυτά που δίνουμε στο frontend (request_dto και response_dto φάκελοι στο backend αντίστοιχα). Ο τρόπος που μεταφέρονται τα αρχεία σε frontend και backend είναι με την μορφή JSON. Αυτό το JSON το spring το μετατρέπει αυτόματα μέσω του DTO σε αντικείμενό του αν οι μεταβλητές και των δύων τους έχουν ακριβώς το ίδιο όνομα. Μπορούμε επίσης να στείλουμε και dto τα οποία μετατρέπονται σε JSON πριν σταλούν στο frontend.

Το repository δεν δημιουργεί αυτόματα τις πληροφορίες μας σε dto , αλλά επιστρέφει όλο το μοντέλο που έχουμε . Είναι καλύτερο να μετατρέψουμε το μοντέλο μας σε dto πριν το μεταφέρουμε για τους εξής λόγους. Αρχικά είναι πιο ασφαλές γιατί δεν μεταφέρονται κρίσιμες πληροφορίες που μπορεί να έχει το μοντέλο. Για παράδειγμα μπορεί άμα μεταφέρουμε τις πληροφορίες ενός χρήστη να μεταφέρουμε όλο το μοντέλο σε ένα χρήστη που θα έχει κρίσιμες πληροφορίες. Επίσης άμα ένα μοντέλο είναι συνδεδεμένο με τα άλλα μπορεί να μεταφέρει και τα μοντέλα στα οποία είναι συνδεδεμένα , παράδειγμα άμα μεταφέρουμε έναν χρήστη να μεταφέρουμε όλες τις αναρτήσεις του και τα σχόλιά του , το οποί μπορεί να κάνει πολύ μεγάλο το JSON και να βγάλει error το Spring (Αυτό γίνεται επειδή οι συνδέσει μου είναι τύπου fetch = FetchType.*EAGER* το οποίο ψάχνει όλα τα δεδομένα αναδρομικά , το fetch = FetchType.*LAZY* όμως βγάζει άλλα προβλήματα όπως ότι δεν μπορεί να είναι κενό άμα δεν υπάρχουν ενώσεις.

Για να μεταφέρουμε τα αρχεία μας σε Dto θα πάρουμε τα δεδομένα μας από το repository και έπειτα παίρνουμε τα δεδομένα (αν είναι σε λίστα τα κάνουμε με stream().map()) και φτιάχνουμε νέα αντικείμενα dtos με τα δεδομένα που θα θέλουμε να μεταφέρουμε.

Παράδειγμα δημιουργίας λίστας από το repository (professionals) που την μετατρέπουμε σε dto (στο stream().map()). Το ResUserDto είναι η τάξη Dto που έχουμε για να στέλνουμε στο frontend. Το frontend έχει δικά του dtos (τα ονομάζει models) τα οποία παίρνουν τις πληροφορίες αυτόματα αν έχουν ι μεταβλητές τα ίδια ονόματα με του backend.Η λίστα εδώ δημιουργείται σε ένα service και επιστρέφεται στο controller που την καλεί , που μετά την επιστρέφει με απάντηση στο frontend.

Πίνακας 4.4 - Παράδειγμα μεταφοράς δεδομένων από ένα repository σε DTO.

```
public List<ResUserDto> ProfessionsDtos(){
  List<User> professionals = userRepository.findAll();
  professionals.removeIf(p -> p.getProfession() == null);
```

```
  return professionals.stream().map(p->
      new ResUserDto(
          p.getId(),
          p.getName(),
          p.getSurname(),
          p.getMiddleName(),
          p.getProfession().getProfession(),
          p.getProfession().getId(),
          p.getProfession().getLongitude(),
          p.getProfession().getLatitude()
      )
      ).collect(Collectors.toList());
}
```

Κάθε αρχείο κάνει την δουλειά του για να μπορέσει να ανακαλέσει τα αρχεία που θέλουμε. Παρατηρούμε ότι τα αρχεία που αναφέραμε παραπάνω δουλεύουν αλυσιδωτά μεταξύ τους και συνεργάζονται για να μεταφέρουν μία πληροφορία πίσω στον χρήστη. Καλό θα είναι , για λόγους συνέπειας του κώδικά μας για να κάνουμε κάθε λειτουργεία που θέλουμε να έχουμε ένα συγκεκριμένο σετ αρχείων (controller, services ,κλπ.) που θα ασχολούνται με μία συγκεκριμένη κατηγορία δεδομένων (πχ για τις αναρτήσεις).Γι' αυτό παρατηρούμε ότι ο κώδικας έχει χωριστεί έτσι ώστε να υπάρχουν πολλά αρχεία που ασχολούνται μόνο με μία συγκεκριμένη κατηγορία δεδομένων (πχ PostController, PostRepository , κλπ).

Κάθε ένα από τα δεδομένα μας θα ανακτάται με ένα URL που θα αρχίζει με τον τύπο των δεδομένων που θέλουμε να πάρουμε και μετά με τον ρόλο του χρήστη που μπορεί να κάνει αυτήν την λειτουργεία (πχ τον προορισμό /profs/approve/apps/all ασχολούμαστε με τα δεδομένα Profession και ProfApplication ,λόγω του "/profs/" που μπορούν να δουν αυτοί που έχουν ρόλο τουλάχιστον approver , λόγω του "/approve/"). Θα μας διευκολύνει πολύ στο να θέσουμε περιορισμούς για το που θα μπαίνει ο κάθε ρόλος χρήστη που θα αναφέρουμε παρακάτω.

4.1.9 Δομή αρχείων στο project

Μέσα στον φάκελο /src/main/java έχουμε το πακέτο com.project.petcare . Μέσα σε αυτό υπάρχουν οι εξής φάκελοι.

- config/
- controller/
- repository/
- request_dto/
- response_dto/
- service/

Έχουμε επίσης ένα αρχείο PetcareApplication το οποίο έχει την αρχική του προγράμματος (main) και κάνει την εκκίνηση του backend.

Σε κάθε ένας από τους φακέλους παραπάνω χρησιμοποιούνται για κάθε σχεδόν μοντέλο που έχουμε δημιουργήσει. Γι' αυτό έχω δημιουργήσει ίδιου τύπου αρχεία γεια κάθε μοντέλο. Για παράδειγμα το μοντέλο Post που είναι μέσα στο models/ έχει PostRepository , PostController , PostDto (το παίρνουμε από το frontend) , ResPostDto (Res = Response το δίνουμε στο frontend) και PostService στους ανάλογους φακέλους.

Για τα πιο πολλά έχουμε μιλήσει. Ο φάκελος config / περιέχει σταθερές τύπου enum που θα τις χρησιμοποιήσουμε στην δημιουργία ων μοντέλων. Επίσης έχει όλα τα απαραίτητα αρχεία για να υπάρχει ένα σωστό session management που θα αναφέρουμε παρακάτω , και συμβάλει στην αυθεντικότητα του χρήστη μέσα στο site.

Άλλα αρχεία που χρησιμοποιήσαμε στον αρχικό φάκελο είναι τα:

- .env Περιέχει το κλειδί με το οποίο θα αποκρυπτογραφούμε τα JWT tokens (Θα μιλήσουμε παρακάτω).
- .gitignore Τί θα ανεβαίνει στο git. Χρησιμοποιείται αν έχουμε πολλά μεγάλα αρχεία που κατεβάσαμε, όπως πακέτα , και μπορεί ο κάθε χρήστης της εφαρμογής να κατεβάσει μόνος αυτόματα του από άλλα repositories. Το χρησιμοποιούμε για να μην πιάνει ανώφελα πολύ χώρο το GitHub repository μας.
- build.gradle Το gradle configuration με τα dependencies που θα χρησιμοποιήσουμε.

Επίσης χρησιμοποιήσαμε το αρχείο παραπάνω

- /src/main/resources Που κάνει επιπλέον configuration.

4.1.10 Αυθεντικοποίηση χρήστη (Session Management)

Για να μπορέσει ένας χρήστης να κάνει κάποιες ενέργειες στο site θα πρέπει να είναι συνδεδεμένος με λογαριασμό. Έτσι θα συμβάλλεται μία ασφάλεια δεδομένων στην ιστοσελίδα όπου ο κάθε χρήστης θα μπορεί να διαβάζει δεδομένα που ανήκουν σε αυτόν και να μην τα βλέπουν οι άλλοι (πχ προσωπικά μηνύματα). Δεν είναι όμως παραγωγικό κάθε φορά που

κάποιος χρήστης θα θέλει να τραβήξει ή να αναρτήσει κάποια δεδομένα να χρειάζεται πάνα επανασύνδεση. Γι' αυτό θα πρέπει να βρούμε έναν τρόπο όταν συνδέεται ένας χρήστης από ένα σημείο να καταλαβαίνει το Spring ότι ο χρήστης έχει αυθεντικοποιηθεί και δεν χρειάζεται πάντα να συνδέεται εκείνο το χρονικό διάστημα.

Το Spring boot μας διαθέτει μία έτοιμη μεθοδολογία με την οποία μπορούμε να αποθηκεύουμε την σύνδεση ενός χρήστη , τα JWT tokens. Τα JWT (JSON Web Token) αποθηκεύουν την κάθε σύνδεση που γίνεται σε ένα token το οποίο είναι ένα κρυπτογραφημένο μήνυμα που μπορεί να αποκρυπτογραφήσει μόνο το backend με ένα κρυφό κλειδί που διαθέτει. Κάθε φορά που συνδεόμαστε , θα αποθηκεύουμε αυτό το token στον υπολογιστή μας (μέσα στον browse , όχι σε cookie) και θα το στέλνουμε στο backend κάθε φορά που θέλουμε να κάνουμε μία ενέργεια που θέλει σύνδεση.

Αρχικά για να γίνει αυτή η αυθεντικοποίηση και εφόσον έχουμε κάνει λογαριασμό στο site θα πρέπει να συνδεθούμε με τα στοιχεία μας. Μόλις βάλουμε στο frontend το username και το password αυτό καλεί μία συνάρτηση από το controller AuthenticationController με προορισμό το /auth/login (Το /auth είναι στο RequestMapping του controller)

Πίνακας 4.5 - Η συνάρτηση που εκτελείται στο AuthenticationController.java για να αποκτήσουμε το JWT.

```
@PostMapping("/login")
public ResponseEntity<LoginResponse> authenticate(@RequestBody LoginUserDto
loginUserDto) {

   User authenticatedUser = authenticationService.authenticate(loginUserDto);//(1)

   String jwtToken = jwtService.generateToken(authenticatedUser);//(2)

   LoginResponse loginResponse = new LoginResponse();
   loginResponse.setToken(jwtToken);
   loginResponse.setExpiresIn(jwtService.getExpirationTime());

   return ResponseEntity.ok(loginResponse);
}
```

Παραπάνω παρατηρούμε την συνάρτηση που θα τρέξει το controller μας. Αρχικά θα πρέπει να βρούμε αν υπάρχει ο χρήστης και είναι σωστά τα στοιχεία του. Για να γίνει αυτό καλούμε την συνάρτηση στην γραμμή (1) που σημειώσαμε με το Dto που έχει τα στοιχεία που βάλαμε όπου αν είναι όλα σωστά επιστρέφει τον χρήστη. Έπειτα στην σειρά (2) δημιουργείται

το token που θέλουμε για τον χρήστη αυτόν και θα δουλεύει μόνο για εκείνον. Τέλος στις υπόλοιπες γραμμές προσπαθούμε να δωρίσουμε το token με κάποια επιπλέον στοιχεία , όπου μετά θα αποθηκεύσει το frontend για να το χρησιμοποιεί στα API Calls του.

Για να λειτουργήσουν σωστά αυτές οι συναντήσεις παραπάνω όμως θα πρέπει να χτίσουμε την λειτουργεία με την οποία θα γίνεται η σύνδεση. Αρχικά για την γραμμή (1) το authenticate μας πάει στο αρχείο AuthenticationService.java

Πίνακας 4.6 - Πόρισμα από το αρχείο AuthenticationService.java για σύνδεση.

```
public User authenticate(LoginUserDto input) {
  authenticationManager.authenticate(
      new UsernamePasswordAuthenticationToken(
          input.getUsername(),
          input.getPassword()
      )
  );

  return userRepository.findByUsername(input.getUsername())
      .orElseThrow();
}
```

Πρέπει να τρέξουμε το authenticationManager που είναι αντικείμενο του AuthenticationManager ,μια έτοιμη διεπαφή (interface) του Spring Boot.Τώρα για να λειτουργήσει όμως αυτήν η συνάρτηση πρέπει να την ρυθμίσουμε (configure) σε ένα άλλο αρχείο με το όνομα ApplicationConfiguration.java . Με το Annotation @Configuration , η τάξη ApplicationConfiguration μας διαθέτει τις ρυθμίσεις που θα θέλαμε να κάνουμε για το πως θα αυθεντικοποιείται ένας χρήστης.

Πίνακας 4.7 - Αρχείο ApplicationConfiguration.java .

```
@Configuration
public class ApplicationConfiguration {
  private final UserRepository userRepository;

  public ApplicationConfiguration(UserRepository userRepository) {
    this.userRepository = userRepository;
  }
```

```
  @Bean
  UserDetailsService userDetailsService() {
    return username -> userRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User not found"));
  }

  @Bean
  BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Bean
  public AuthenticationManager authenticationManager(AuthenticationConfiguration
config) throws Exception {
    return config.getAuthenticationManager();
  }

  @Bean
  AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

    authProvider.setUserDetailsService(userDetailsService());
    authProvider.setPasswordEncoder(passwordEncoder());

    return authProvider;
  }
}
```

Για να ελεγχθούν τα username και password του χρήστη και να αυθεντικοποιηθεί ο χρήστης μέσω authenticationManager.authenticate θα πρέπει να ρυθμίσουμε την συνάρτηση στο από πάνω αρχείο authenticationManager με ανάλογο τρόπο . Έπειτα ελέγχεται το authenticationProvider το οποίο ψάχνει τον χρήστη με το username που έβαλε στην βάση και βρίσκει αν ο κωδικός είναι σωστός (με το να κάνει hash αυτόν που βάλαμε και να τους συγκρίνει αν είναι ίδιοι).

Πίσω στο αρχείο AuthenticationController.java στην γραμμή (2) χρειαζόμαστε ένα service που θα χρησιμοποιήσουμε εμείς. Στο JwtService.java έχουμε όλη την λειτουργία που κάνουμε για τα JWTs

Πίνακας 4.8 - Αρχείο JwtService.java .

```
@Service
public class JwtService {
  @Value("${security.jwt.secret-key}")
  private String secretKey;

  @Value("${security.jwt.expiration-time}")
  private long jwtExpiration;

  public String extractUsername(String token) {
    return extractClaim(token, Claims::getSubject);
  }

  public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = extractAllClaims(token);
    return claimsResolver.apply(claims);
  }

  public String generateToken(UserDetails userDetails) {
    return generateToken(new HashMap<>(), userDetails);
  }

  public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
    return buildToken(extraClaims, userDetails, jwtExpiration);
  }

  public long getExpirationTime() {
    return jwtExpiration;
  }

  private String buildToken(
      Map<String, Object> extraClaims,
      UserDetails userDetails,
      long expiration
  ) {
    return Jwts
        .builder()
        .setClaims(extraClaims)
        .setSubject(userDetails.getUsername())
```

```
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expiration))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
    }

    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

Περιληπτικά αυτό που κάνει η παραπάνω τάξη είναι να δημιουργεί tokens (με την συνάρτηση buildToken) και να τα επιστρέφει , να διαβάζει τα στοιχεία των token που του δίνουμε (Όπως το Username με την συνάρτηση extractUsername ή όλα τα δεδομένα με την συνάρτηση extractAllClaims) και για να δει αν είναι έγκυρα (συναρτήσεις isTokenExpired και isTokenValid).Δουλεύει ως υπηρεσία (με το Annotation @Service) και λειτουργεί ως μεσολαβητής για την μεταφορά των πληροφοριών για κάθε token.

Τέλος η LoginResponse στο τέλος της συνάρτησης authenticate στο AuthenticationController απλά αποθηκεύει το token και το expiration time που θα πάρουμε εμείς μετά μέσω frontend.

Δεν αρκεί όμως η σύνδεσή μας και η αυθεντικοποίηση θα πρέπει επίσης να δείξουμε στο Spring πως θα διαχειρίζεται τα tokens και επίσης ποια μέση θα επιτρέπονται να μπει (και με ποιόν ρόλο).Θα πρέπει να δούμε πως θα ελέγχεται η αυθεντικοποίηση των token στο server και τα μέρη που μπορούμε να πάμε.

Πίνακας 4.9 - Αρχείο SecurityConfiguration.java .

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
  private final AuthenticationProvider authenticationProvider;
  private final JwtAuthenticationFilter jwtAuthenticationFilter;

  public SecurityConfiguration(
      JwtAuthenticationFilter jwtAuthenticationFilter,
      AuthenticationProvider authenticationProvider
  ) {
    this.authenticationProvider = authenticationProvider;
    this.jwtAuthenticationFilter = jwtAuthenticationFilter;
  }

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.csrf()
        .disable()
        .cors().and()//
        .authorizeHttpRequests()
        .requestMatchers("/auth/**").permitAll()
        .requestMatchers("/*/manage/**").hasRole("MANAGER")
        .requestMatchers("/*/approve/**").hasRole("APPROVER")
        .requestMatchers("/*/admin/**").hasRole("ADMIN")
        .requestMatchers("/*/user/**").hasRole("USER")
        .requestMatchers("/**")
          .permitAll()
```

```
            .anyRequest()
            .authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();

        configuration.setAllowedOriginPatterns(List.of("http://localhost:4200"));
        configuration.setAllowedMethods(List.of("GET","POST", "PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

        source.registerCorsConfiguration("/**",configuration);

        return source;
    }
}
```

Αρχικά θα μελετήσουμε την συνάρτηση SecurityFilterChain . Η Συνάρτηση SecurityFilterChain ελέγχει το πώς θα διαχειριστούμε τα tokens των χρηστών ανάλογα με τις προδιαγραφές τους . Με το csrf().disable() ακυρώνει το Cross-site Request Forgery το οποίο είναι ένας τρόπος επίθεσής που αναγκάζει τον χρήστη να κάνει ενέργειες στο site μέσω ενός άλλου χρήστη [9].Όμως με τα JWT tokens δεν χρειαζόμαστε αυτήν την ενέργεια , γιατί είναι πιο ασφαλή από το csrf και είναι κρυπτογραφημένων μέσω ενός κρυφού κλειδιού [10].

Το authorizeHttpRequests() με όλα τα requestMatchers μας λέει τι URLs μπορεί να εκτελέσει ένας χρήστης ανάλογα του ρόλου του. Με τον τρόπο που τα δημιουργήσαμε λέμε ότι πρώτα θα υπάρχει ένα directory στο οποίο θα πηγαίνει ένας χρήστης στο συγκεκριμένο controller (για παράδειγμα το “/profs/”) , έπειτα θα είναι ο ρόλος του χρήστη (“/admin/”,

“/manage/”,κλπ. όπως βλέπουμε πιο πάνω για τον κάθε ρόλο) και μετά ότι θέλουμε. Κάθε ρόλος θα μπορεί να περάσει τα URL που έχουν να κάνουν με τον ρόλο του και έχουν ρυθμιστεί στο Spring Boot .

Τα requestMatchers είναι πολύ σημαντικά για την ασφάλεια της ιστοσελίδας γιατί εκεί επιλέγουμε ποιοι χρήστες μπορούν να κάνουν κάποιες ενέργειες. Ακόμη και αν στο frontend περιορίσουμε αυτή την λειτουργεί (με το να μην εμφανίζουμε κάποια κουμπιά σε συγκεκριμένους ρόλους χρηστών) δεν είμαστε απόλυτα ασφαλείς γιατί μπορεί κάποιος να τρέξει τα δικά του calls (ή μέσω ενός σφάλματος του frontend να κάνει κάτι που δεν πρέπει) και να καλέσει εντολές που κάνουν ευάλωτα ευαίσθητες πληροφορίες του site.

Η αναγνώριση των ρόλων για το Spring Boot, είναι μέσα στο μοντέλο τάξης User που χρησιμοποιεί (implements) την διεπαφή (interface) UserDetails του SpringBoot για αναγνώριση χρηστών. Μέσω μίας μεταβλητής του μοντέλου αποκτώνται όλοι οι ρόλοι που μπορεί να έχει ο χρήστης που του επιτρέπουν ανάλογα τι μπορεί να τρέξει στο backend (μέσω των controller).

Πίνακας 4.10 - Απόσπασμας από το user.model.java για την αναγνώρηση ρόλων από το Spring Boot.

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
  switch (this.role){
    case COMMON -> {
      return List.of(new SimpleGrantedAuthority("ROLE_USER"));
    }
    case MANAGER -> {
      return List.of(new SimpleGrantedAuthority("ROLE_MANAGER"),
          new SimpleGrantedAuthority("ROLE_USER"));
    }
    case APPROVER -> {
      return List.of(new SimpleGrantedAuthority("ROLE_APPROVER"),
          new SimpleGrantedAuthority("ROLE_MANAGER"),
          new SimpleGrantedAuthority("ROLE_USER"));
    }
    case ADMIN -> {
      return List.of(new SimpleGrantedAuthority("ROLE_ADMIN"),
          new SimpleGrantedAuthority("ROLE_APPROVER"),
          new SimpleGrantedAuthority("ROLE_MANAGER"),
          new SimpleGrantedAuthority("ROLE_USER"));
    }
  }
  return List.of(new SimpleGrantedAuthority("ROLE_USER"));
}
```

Το sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*) είναι επειδή δεν χρειαζόμαστε session γιατί το αναλαμβάνουμε αυτό με τα JWT και με το πότε λήγουν.Το .authenticationProvider(authenticationProvider) ελέγχει αν έχει γίνει αυθεντικοποίηση την αρχή με username και password (όπως είδαμε στην αρχή για το authenticationProvider στο login)

Τέλος το .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class μας λέει ότι πρώτα θα ελέγξουμε αν υπάρχει jwt και μετά αυθεντικοποίηση με username και password. Το jwtAuthenticationFilter είναι ένα αντικείμενο της τάξης JwtAuthenticationFilter του αρχείου JwtAuthenticationFilter.java. Μέσω της OncePerRequestFilter που κάνει extend μπορούμε να φιλτράρουμε τα tokens με το να βλέπουμε τι υλικό μεταφέρουν και αν εγκρίνεται για το έργο που θέλει να κάνει ο χρήστης στον server.

Πίνακας 4.11 - Αρχείο JwtAuthenticationFilter.java .

```
package com.project.petcare.config.auth;

import com.project.petcare.service.JwtService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.lang.NonNull;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.web.servlet.HandlerExceptionResolver;

import java.io.IOException;
```

```
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {
  private final HandlerExceptionResolver handlerExceptionResolver;

  private final JwtService jwtService;
  private final UserDetailsService userDetailsService;

  public JwtAuthenticationFilter(
      JwtService jwtService,
      UserDetailsService userDetailsService,
      HandlerExceptionResolver handlerExceptionResolver
  ) {
    this.jwtService = jwtService;
    this.userDetailsService = userDetailsService;
    this.handlerExceptionResolver = handlerExceptionResolver;
  }

  @Override
  protected void doFilterInternal(
      @NonNull HttpServletRequest request,
      @NonNull HttpServletResponse response,
      @NonNull FilterChain filterChain
  ) throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
      filterChain.doFilter(request, response);
      return;
    }

    try {
      final String jwt = authHeader.substring(7);
      final String userName = jwtService.extractUsername(jwt);

      Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

      if (userName != null && authentication == null) {
        UserDetails userDetails =
this.userDetailsService.loadUserByUsername(userName);

        if (jwtService.isTokenValid(jwt, userDetails)) {
```

```
            UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities()
            );

            authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);
          }
        }

        filterChain.doFilter(request, response);
      } catch (Exception exception) {
        handlerExceptionResolver.resolveException(request, response, null, exception);
      }
    }
}
```

Στο παραπάνω αρχείο ,μέσω της συνάρτησης doFilter , παίρνουμε το token και διαβάζοντας τα αρχεία του (αφού έχουμε το κρυφό κλειδί για να το κάνουμε αυτό) , αν έχουμε σωστό username τότε αποθηκεύουμε τα στοιχεία του ως έγκυρα στο Spring και προσπαθούμε να κάνουμε την ενέργεια που θέλουμε (άμα μπορούμε).

Τέλος έχουμε και error handling (που βρίσκει τα σφάλματα) για κάθε διαδικασία παραπάνω.

Πίνακας 4.12 - Πόρισμα από αρχείο GlobalExceptionHandler.java στον φάκελο config για την αναγνμώρηση σφαλμάτων.

```
@ExceptionHandler(Exception.class)
public ProblemDetail handleSecurityException(Exception exception) {
  ProblemDetail errorDetail = null;

  // TODO send this stack trace to an observability tool
  exception.printStackTrace();
```

```
  if (exception instanceof BadCredentialsException) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(401),
exception.getMessage());
    errorDetail.setProperty("description", "The username or password is incorrect");

    return errorDetail;
  }

  if (exception instanceof AccountStatusException) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(401),
exception.getMessage());
    errorDetail.setProperty("description", "The account is locked");
  }

  if (exception instanceof AccessDeniedException) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(403),
exception.getMessage());
    errorDetail.setProperty("description", "You are not authorized to access this resource");
  }

  if (exception instanceof SignatureException) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(401),
exception.getMessage());
    errorDetail.setProperty("description", "The JWT signature is invalid");
  }

  if (exception instanceof ExpiredJwtException) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(401),
exception.getMessage());
    errorDetail.setProperty("description", "The JWT token has expired");
  }

  if (errorDetail == null) {
    errorDetail = ProblemDetail.forStatusAndDetail(HttpStatusCode.valueOf(500),
exception.getMessage());
    errorDetail.setProperty("description", "Unknown internal server error.");
  }

  return errorDetail;
}
```

Όταν υπάρχει ένα σφάλμα στο Spring και συγκεκριμένα στο παραπάνω παράδειγμα αν υπάρχει σφάλμα στην αυθεντικοποίηση του username και password και του token , θα εκτελέσει το ανάλογο exceptionhamdling για το exception που έγινε . Το αποτέλεσμα θα μας το εμφανίζει στην κονσόλα (F12) του browser μας και μαζί με το HttpsStatusCode που μεταφέρει (τριψήφιος αριθμός) μπορούμε να τον διαχειριστούμε να κάνει κάποια πράγματα στο frontend μέσω ενός interceptor (δηλαδή άμα για παράδειγμα βρει σφάλμα τύπου 401 να κάνει μία συγκεκριμένη λειτουργεία).

4.1.11 Μεταφορά εικόνων.

Για την σωστή λειτουργεία του backend μας θα πρέπει να αποθηκεύονται και κάποιες εικόνες , για τις εικόνες έχουμε δύο σταθερές με την θέσει στον υπολοίπου θα πηγαίνουν και είναι στο application.properties (τα οποία είναι αναγκαία να δημιουργήσουμε μόνοι μας τους φακέλους αυτούς για να μπορεί να πάει). Για να κατεβάσουμε τις εικόνες από το server χρησιμοποιούμε ένα URL σε ένα service του frontend.Μετά μπορούμε με αυτό το link να φέρνουμε κάθε φορά την εικόνα. Οι εικόνες κατεβαίνουμε με base64 που είναι η εικόνα σε μορφή string .Για τον ανέβασμα μία εικόνας πρέπει να το μετατρέψουμε σε formdata από το frontend και μετά θα μεταφερθεί σαν MultipartFile στο Spring boot που είναι για την μεταφορά αρχείων. Για να μεταφέρουμε και ένα DTO μαζί το βάζουμε μέσα στο formdata της Angular.

4.2 Frontend

4.2.1 Δομή της Angular

Η Angular στηρίζεται την δημιουργία Components (εξαρτημάτων , κομματιών) για την εμφάνιση περιεχομένου. Ένα component είναι ένα κείμενο html μαζί με ένα css αρχείο και ένα αρχείο angular.Τα components μπορούν να χρησιμοποιηθούν ως κομμάτι μίας σελίδας (που μπορούμε να βάλουμε με ένα html template που δημιουργείται) ή σαν μία ολόκληρη σελίδα που θα τρέχουμε το component από το αρχείο app.routes.ts με ένα συγκεκριμένο URL. Η εφαρμογή ξεκινάει με το app component που βρίσκεται στον φάκελο src/app/ και μέσω της δημιουργίας του Router outlet μέσα στο html του app component μπορούμε να εμφανίσουμε κάθε σελίδα που θέλουμε με αυτό , αν έχουμε σωστό link.

Μέσα στον φάκελο app θα είναι σχεδόν όλη μας η εφαρμογή , πέρα από το app component υπάρχουν και οι εξής φάκελοι που κάνουν τα εξής:

- header = είναι το βασικό component που θα υπάρχει σε όλες τις σελίδες. Το βάζουμε μαζί με το router-outlet από πάνω του για να δείξουμε ότι πρέπει να εμφανίζεται σε

κάθε σελίδα στην κορυφή. Με το header όπως δείξαμε στο uml πιο πάνω θα πατάμε τα links που υπάρχουν και θα πηγαίνουμε σε μία άλλη σελίδα του site.

- footer = θα είναι μόνο στην σελίδα main κάτω κάτω ως το footer της σελίδας μας.
- interceptor (αναχαιτιστής) = Λειτουργεί ως μεσολαβητής για κάθε API Call που κάνουμε , κάθε φορά που θέλουμε να κάνουμε κάτι στο backend βλέπει τα δεδομένα μας (request και response). Εδώ χρησιμοποιείται για την αυθεντικοποίηση του χρήστη με tokes όπου μεταφέρει το token που έχουμε με κάθε call. // όλοι οι χρήστες μπορούν να μπουν στο auth
- models = Είναι τάξης της angular και χρησιμοποιούνται για να αποθηκεύουν τα dtos που στέλνουμε από το backend.Μετά αυτά εμφανίζουν τα δεδομένα με βάση αυτά τα δεδομένα. Μπορούμε να στείλουμε και δεδομένα με μερικά models
- pages = Είναι όλες οι σελίδες που ανοίγουν από το app.routes.ts. Περιέχει όλα τα components που χρειάζονται για να ανοίξουν αυτές και μέσα στους φακέλους αυτούς υπάρχουν άλλα βοηθητικά components για τις σελίδες αυτές.
- reqDto = Είναι μονάχα για αποστολή dto στο backend για πράγματα που δεν θέλουμε να εμφανίζουμε. Για παράδειγμα αλλαγή ρυθμίσεων του χρήστη.
- services= Είναι το μέρος που παίρνουμε και επεξεργαζόμαστε τα δεδομένα που πήραμε από backend. Μπορούμε επίσης να στείλουμε δεδομένα μέσω αυτού. Κάνει τα API CALLS.
- app.config.ts = Έχει configuration όπως providers για κάποια εργαλεία που θα χρησιμοποιήσουμε (όπως το interceptor)
- app.routes.ts = Έχει όλα τα urls και το τι component θα τρέχει όταν πάνε στο καθένα.
- Άλλα αρχεία /φάκελοι που χρησιμοποιούμε είναι :

Στον φάκελο src:

- constants.ts= Σταθερές μεταβλητές
- assets/ = έχει αρχεία όπως εικόνες που θέλει το site να τοποθετήσει.
- index.html,style.css,main.ts = είναι τα main αρχεία της ιστοσελίδας μας , μπορούμε ότι αλλαγές κάνουμε εδώ να επηρεάσουν όλο το site (όπως το CSS)

4.2.2 Επικοινωνία με το backend

Όπως αναφέραμε παραπάνω τα services της Angular έχουν την ευθύνη να επικοινωνούν με την βάση. Τα services αυτά καλούν το backend και υστέρα γίνεται η διαδικασία του controller με το backend που χρειάζεται.

notifications.compnent.ts

Πίνακας 4.13 - Πόρισμα απότο αρχείο notifications.compnent.ts

```
fetch(){
  this.serv.getNotifications().subscribe((data:UserNotification[]) =>{
   this.notif = data
  })
```

Πίνακας 4.14 - Πόρισμα απότο αρχείο notification.service.ts

```
 getNotifications(){
  return this.http.get<Notification[]>(BACKEND_URL+this.PAGE_MAPPING+"/user/get")
 }
```

Πίνακας 4.15 - Πόρισμα απότο αρχείο notification.compnent.ts

```
<div *ngFor="let n of notif">
 <mat-card>
  <mat-card-content>
   <h2 style="margin: 10px;">{{n.title}}</h2> <div style="margin: 10px;">{{n.message}}</div>
<div class="date">{{n.timestamp| date:'dd/MM/YYYY'}}</div>
   <button mat-icon-button class="button" aria-label="Close">
    <mat-icon (click)="DeleteNotif(n.id!)">close</mat-icon>
   </button>
  <div *ngIf="n.postId != null">
   <a style="margin: 10px;" [routerLink]="['/comments/',n.postId]">Post</a>
  </div>
  </mat-card-content>
 </mat-card>
</div>
```

Παραπάνω είναι μία διαδικασία εμφάνισης ειδοποιήσεων στο site.Αρχικά όταν ανοίγουμε την σελίδα εκτελείτε η συνάρτηση fetch() , η οποία πάει στην συνάρτηση getNotifications() του notification.service.ts. Μετά από το notification.service.ts καλούμε από το backend τα δεδομένα που θέλουμε. Τέλος ξανά στο αρχείο notifications.component.ts

κάνουμε .subscribe στη συνάρτηση που καλέσαμε το backend που μέσα σε αυτήν παίρνουμε τα δεδομένα από την βάση και τα αποθηκεύουμε στην σελίδα στην μεταβλητή notif. Τέλος στο notifications.component.html παίρνει την μεταβλητή notif και για κάθε αντικείμενό της εμφανίζει ένα κουμπί. Αυτή είναι γενικά η λογική για να κάνουμε μία ενέργεια με την βάση , λειτουργεί ακριβώς με τον ίδιο τρόπο για POST , GET και DELETE calls με την μόνη διαφορά ότι στο PUT και POST θέλουμε στο service πριν καλέσουμε το API να βάλουμε και άλλη μία παράμετρο που θα στέλνουμε τα δεδομένα στο backend με την μορφή τάξης (που θα σταλθεί στο dto που έχουμε για αυτήν στο backend).

Πίνακας 4.16 – Αρχείο service comment.service.ts .

```
import { HttpBackend, HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { PostComment } from '../models/post-comment.model';
import { Post } from '../models/post.model';
import { BACKEND_URL } from '../../constants';

@Injectable({
 providedIn: 'root'
})
export class CommentService {


 constructor(private http:HttpClient) { }

 getComments(post_id:string){
  return this.http.get<PostComment[]>(BACKEND_URL+"/comments/" + post_id);
 }

 addComments(post_id:string , comment:PostComment){
  return this.http.post<PostComment>(BACKEND_URL+"/comments/user/" +
post_id,comment);
 }

 deleteComment(comment_id:string){
  return
this.http.delete<PostComment>(BACKEND_URL+"/comments/manage/"+comment_id)
 }
}
```

Παράδειγμα service αρχείου comment.service.ts . Με τα service αρχεία στέλνουμε requsets που μηλάνε μέσω του API στο Spring και επιστρέφονται ανάλογα τα δεδομένα που πρέπει (DTOs) και άν πρέπει.Βλέπουμε 3 ειδών requests , ένα GET , ένα POST και ένα DELETE.Με το POST και DELETE αλλάζουμε τα δεδομένα στην βάση (στέλνουμε και ένα DTO για το POST).Με το GET παίρνουμε DTOs από το backend που μπορεί να είναι ένα ήσε μία λίστα.

Τα DTOs για να τα μεταφέρουμε τα βάζουμε συνήθως σε ένα model της Angular που έχουμε φτιάξει και μοιάζει με τις τάξεις των DTO που υπάρχουν στο Spring Boot , εφόσων εκεί τα μεταφέρουμε.Αν κάνουμε GET request τότε μπορούμε να πειστρέψουμε τα στοιχεία στο site μέσω δυναμικού κώσικα σε HTML. Αν είναι σε λίστα κάνουμε ένα for loop (“*ngFor”) στο HTML και τα παράγει όλα (κυρίως για να εμφανήσουμε για παράδειγμα όλα τα ιδρύματα της βάσεις μας).

4.2.3 Interceptor

Τα interceptors είναι ένας άλλος τύπος αρχείου της Angular με τον οποίο μπορούμε να διαβάσουμε όλα της αιτήσεις (request) και της απαντήσεις (response) των πακέτων Http που μεταφέρονται ανάμεσα στο frontend και στο backend.

Τα interceptors θα χρησιμοποιηθούν για να επεμβαίνουν στο API του backend και στα URL που στέλνονται. Όταν μεταφέρεται ένα Http πακέτο ως response , μπορούμε να δούμε αρκετές πληροφορίες που μας επιστρέφονται με κάθε API Call , μία από αυτές είναι τα σφάλματα του καλέσματος. Όπως είπαμε παραπάνω στο backend μπορούμε άμα υπάρχει ένα σφάλμα να το διαχειριστούμε (ErrorHandling) όπου θα επιστρέφει ένα συγκεκριμένο Http error response. Τα σφάλματα που γυρνάει ένα Http πακέτο είναι τριψήφιου αριθμοί με έναν συγκεκριμένο κωδικό. Παραδείγματα κωδικού είναι το 401 για μη αυθεντικοποιημένους χρήστε , 404 όταν δεν βρεθεί προορισμός και 403 όταν δεν μας επιτρέπετε να πάρουμε κάποια στοιχεία. Μέσω του backend μπορέσαμε να διαχειριστούμε το σφάλματα σωστά και τώρα χρειάζεται επεξεργασία του frontend για το πως θα αλληλοεπιδράει με αυτά.

Πίνακας 4.17 - Αρχείο authorize.interceptor.ts .

```
import { HttpErrorResponse, HttpInterceptorFn } from '@angular/common/http';

import { Injectable } from '@angular/core';
import {HttpEvent,HttpInterceptor,HttpHandler,HttpRequest} from
'@angular/common/http';
import { catchError, Observable, throwError } from 'rxjs';
import { AuthService } from '../services/auth.service';
import { Router } from '@angular/router';
```

```
@Injectable()
export class AuthorizeInterceptor implements HttpInterceptor {
 constructor(private authService: AuthService,
  private router:Router
 ) {}

 intercept(
  req: HttpRequest<any>,
  next: HttpHandler
 ): Observable<HttpEvent<any>> {
  const authToken = this.authService.GiveToken();

  let request = req;

  if (authToken) {

   request = req.clone({
    setHeaders: {
     Authorization: `Bearer ${authToken}`
    }
   });
  }

  return next.handle(request).pipe(
   catchError((error: HttpErrorResponse) => {
    if (error.status === 401){
     console.log("Session Expired , please login again.")
     this.authService.Disconect()
     this.router.navigate(['/login'])
    }

    if (error.status === 403){
     this.router.navigate(['/denied'])
    }

    if (error.status === 404){
     this.router.navigate(['/404'])
    }
    return throwError(() => new Error(error.message))
   })
  )
 }
```

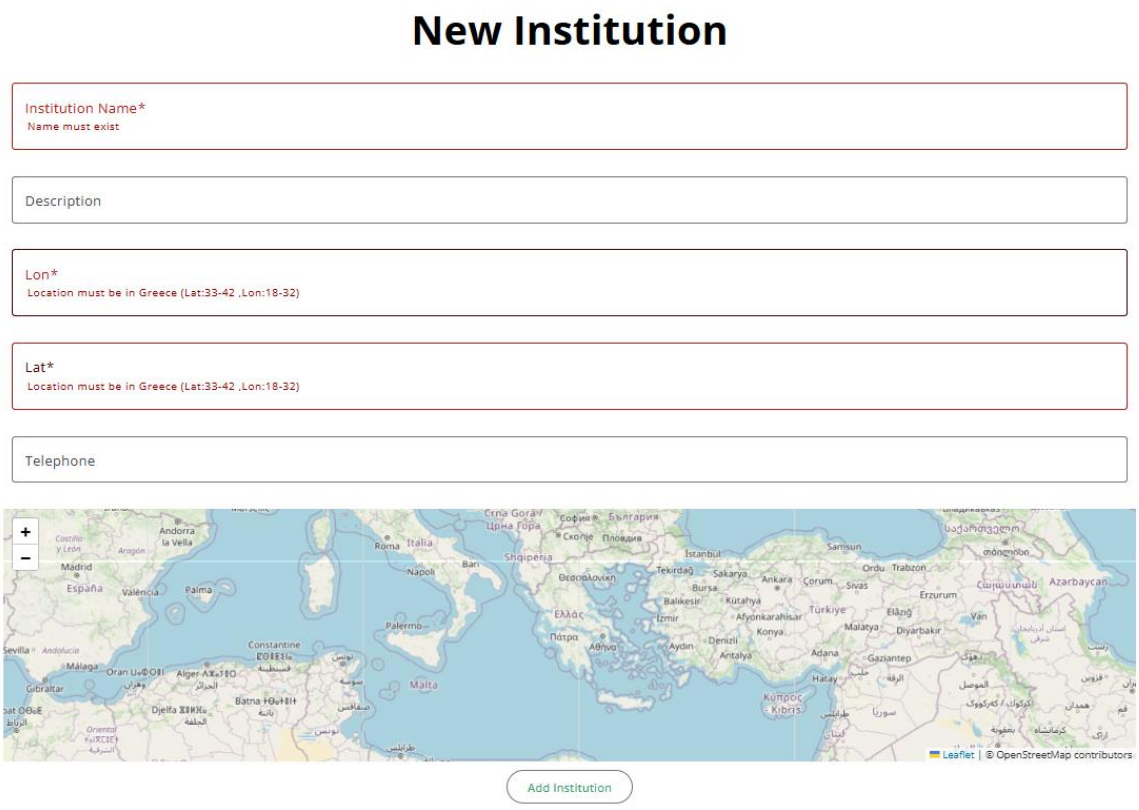
```
}
```

Παραπάνω βλέπουμε στο interceptor της εφαρμογής μας. Παρατηρούμε ότι προς το τέλος ανάλογα της τιμής error.status κάνουμε μία διαφορετική λειτουργεία. Οι λειτουργείες είναι , για 404 σφάλμα να δείχνει ότι δεν υπάρχει προορισμός , για 403 να μα πηγαίνει στην σελίδα που δείχνει το μήνυμα “Access Denied” και για 401 να μας πηγαίνει να ξανασυνδεθούμε και μας αποσυνδέει από τον χρήστη (Είναι κυρίως άμα λήξει το JWT).

Εκτός αυτού παρατηρούμε στο αρχείο ένα σημείο που λέει “ Authorization: `Bearer ${authToken}` ” εδώ άμα ο χρήστης είναι συνδεδεμένος και έχει token (“if (authToken)”) του προσθέτει μπροστά ένα Header (κεφαλίδα) με ένα μήνυμα μπροστά “Bearer “ (το χρειαζόμαστε για την σύνδεση) και μετά το JWT.Με αυτόν τον τρόπο θα πρέπει να στέλνεται το token του κάθε χρήστη για να το διαβάζει το backend.Τέλος στέλνει το request (“return next.handle(request)”) και άμα δεν υπάρξει σφάλμα (“pipe(catchError ” αμέσως μετά)) το στέλνει στο backend και κάνει αυτό την δουλειά του.

4.2.4 Περιορισμοί Δεδομένων

Μέσα σε κάποια κενά στις φόρμες έχω βάλει να μην εγκρίνει (κάνει submit) την φόρμα αν τα αρχεία δεν είναι σωστής δομής. Έτσι θα είναι τα αρχεία με σωστή δομή και δεν θα προκύπτουν σφάλματα (πχ χρήστης χωρίς username).



Εικόνα 4.2 - Φόρμα για καινούργιο ίδρυμα στην ιστοσελίδα , άμα πατήσουμε το "Add Institution" κάτω κάτω ή άμα πατήσουμε σε κάποια από τα κενά και δεν βάλουμε σωστά στοιχεία μας δίχνει το σφάλμα που έχει το κάθε ένα.

```
institution_form = new FormGroup({
  name : new FormControl("",[Validators.required]),
  description : new FormControl(),
  longitude : new FormControl("",[Validators.required,Validators.min(18),Validators.max(32)]),
  latitude : new FormControl("",[Validators.required,Validators.min(33),Validators.max(42)]),
  telephone : new FormControl(),
})
```

Εικόνα 4.3 - Πόρισμα από το αρχείο new-institution.component.ts. Εδώ γίνεται η έγκληση των στοιχείων του χρήστη.

Δεν αρκεί όμως αυτό μόνο γιατί μπορεί κάποιος να προσπεράσει αυτούς τους περιορισμούς αν είναι μόνο στο frontend.Οι περιορισμού στο frontend είναι μόνο για

εξυπηρέτηση. Θα πρέπει να βάλουμε και περιορισμούς στο backend για να έχουμε ασφάλεια. Για να γίνει αυτό πάμε στο Spring Boot project και βάζουμε στα model το Annotation που θέλουμε.

```java
@Entity 13 usages  nvas314
@Data
@Accessors(chain = true)
public class Institution {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false,unique = true)
    @NotBlank
    private String name;
    private String description;
    private String telephone;

    @Min(value = 18, message = "Coordinates of greece are 18-32 lon and 33-42 lat.")
    @Max(value = 32, message = "Coordinates of greece are 18-32 lon and 33-42 lat.")
    private double longitude;

    @Min(value = 33, message = "Coordinates of greece are 18-32 lon and 33-42 lat")
    @Max(value = 42, message = "Coordinates of greece are 18-32 lon and 33-42 lat.")
    private double latitude;

    @OneToMany(fetch = FetchType.EAGER,mappedBy = "institution",cascade = CascadeType.ALL)
    private List<InstOfficial> officials;
}
```

Εικόνα 4.4 - Πόρισμα από το αρχείο τύπου model Institution.java , παρατηρούμε τους περιορισμούς που υπάρχουν στην δημιουργεία νέου ιδρύματος.

4.2.5 Leaflet

Το Leaflet είναι ένα πακέτο του NodeJS που μας παρέχει την βασική λειτουργεία χαρτών με την εμφάνιση του παγκόσμιου χάρτη και ακριβή λεπτομερειών. Μπορούμε μέσω αυτού να επιλέγουμε ένα σημείο με τις συντεταγμένες του (Latitude και Longitude) και να καρφιτσώνει το σημείο εκείνο. Μπορούμε με αυτό το προϊών να δείξουμε στον χρήστη με ευκολία ένα σημείο στον χάρτη και να τον κάνουμε να επιλέξει ένα σημείο που θέλει πατώντας τον χάρτη.

4.2.6 Angular Material

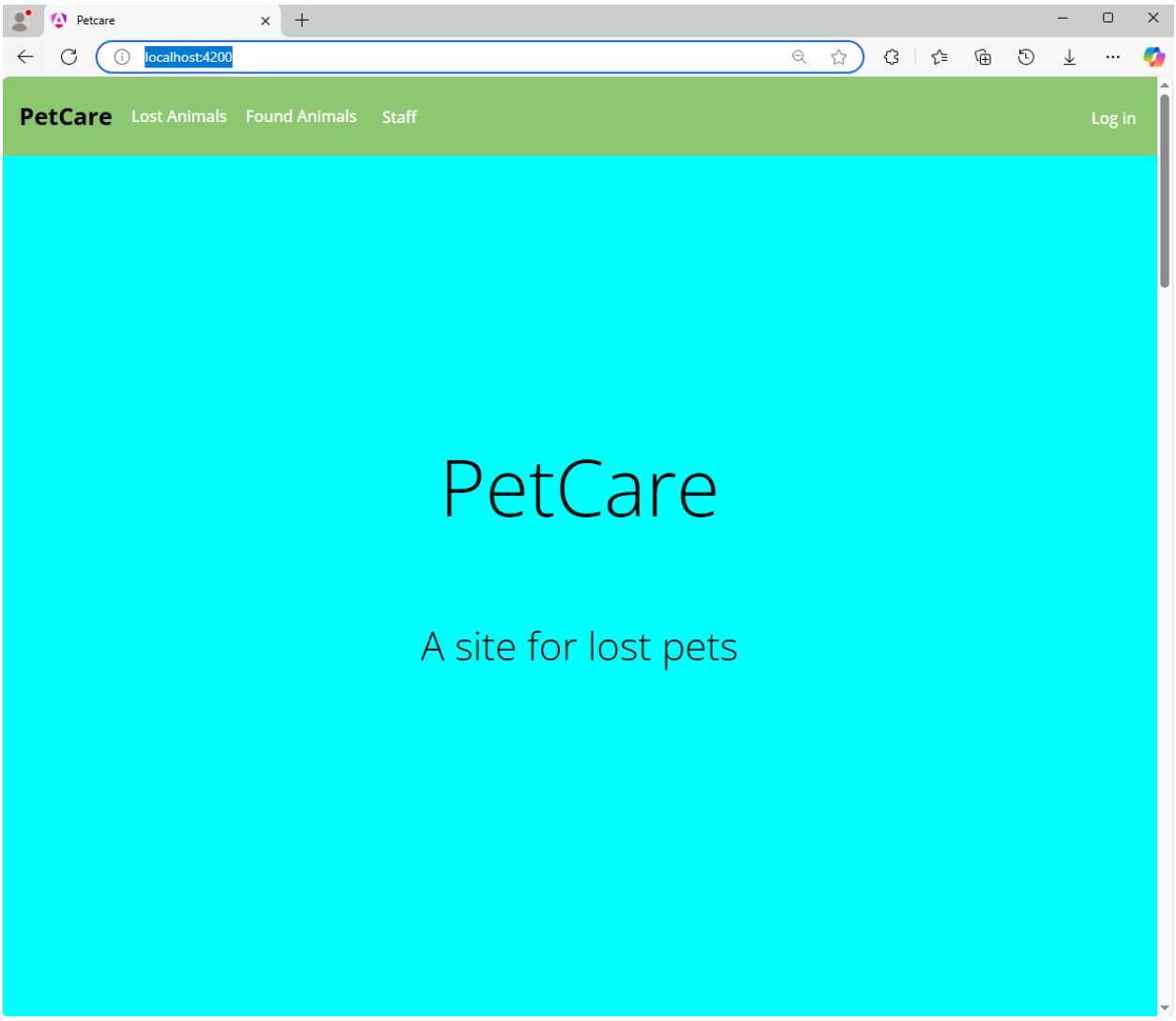
Η Angular Material είναι μία προέκταση της Angular και μας βοηθάει να δημιουργήσουμε εμφανισιακά site. Περιέχει Components τα οποία είναι προσχεδιασμένα αντικείμενα που μπορούμε να βάλουμε στο site για να φαίνεται πιο ελκυστικό. Περιέχει αρκετά αντικείμενα όπως εισαγωγές κειμένου , κουμπιά , κλπ.

5. Παρουσίαση

5.1 Πλοήγηση Ιστοσελίδας

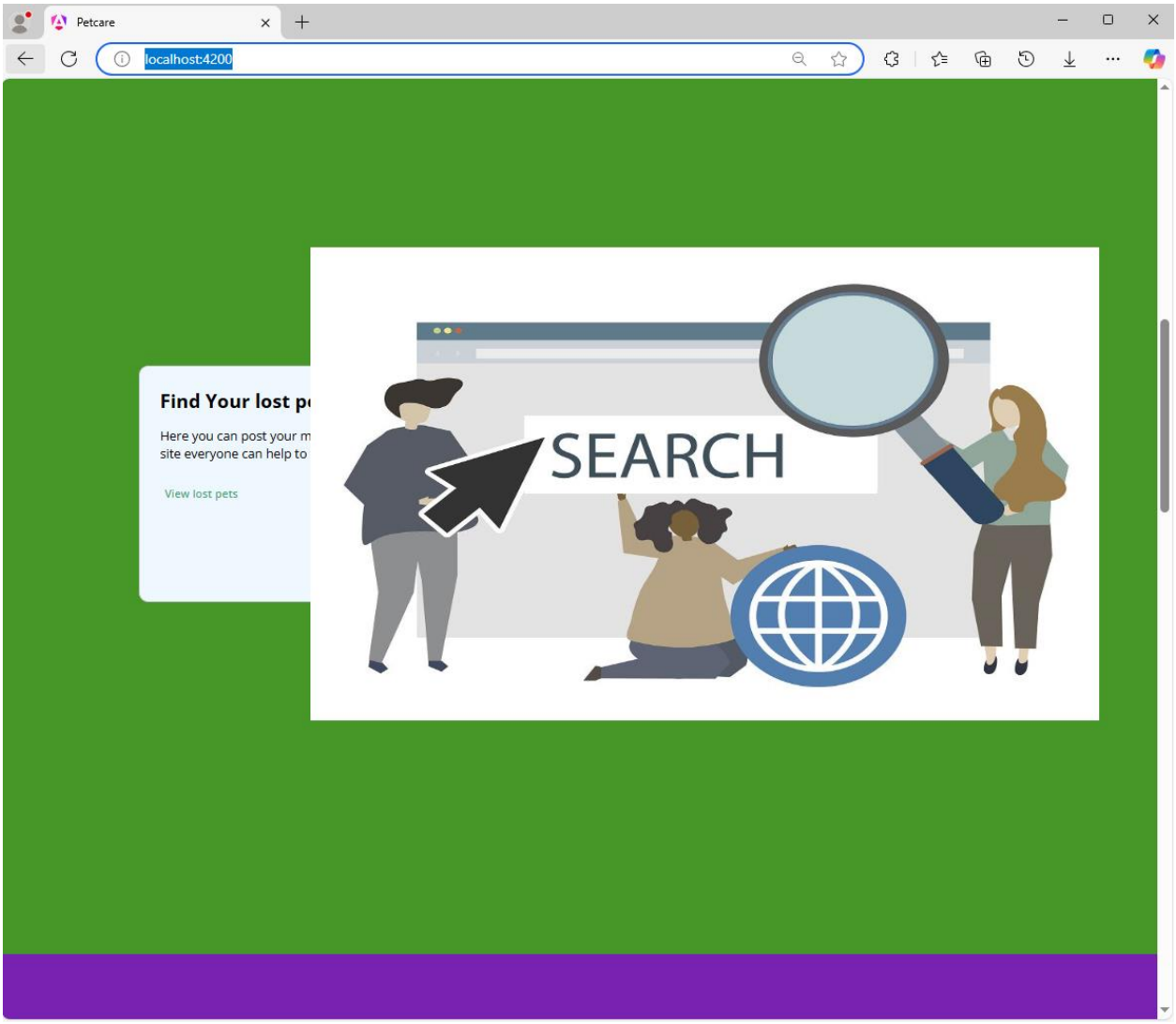
Τώρα θα γίνει η περιήγηση της ιστοσελίδα. Κάποιες ενέργειες δεν μπορούν να της κάνουν κάποιοι ρόλοι χρηστών που θα αναφέρονται όπου χρειάζεται. Γι' αυτό θα γίνου οι πιο πολλές ενέργειές ως χρήστης ADMIN.

Ανοίγοντας τα προγράμματα όπως περιγράψαμε πιο πάνω (Εγκατάσταση 3.3) και πηγαίνοντας στο http://localhost:4200/ , Μπορεί να χρειαστεί να πατηθεί και το logo πάνω αριστερά.



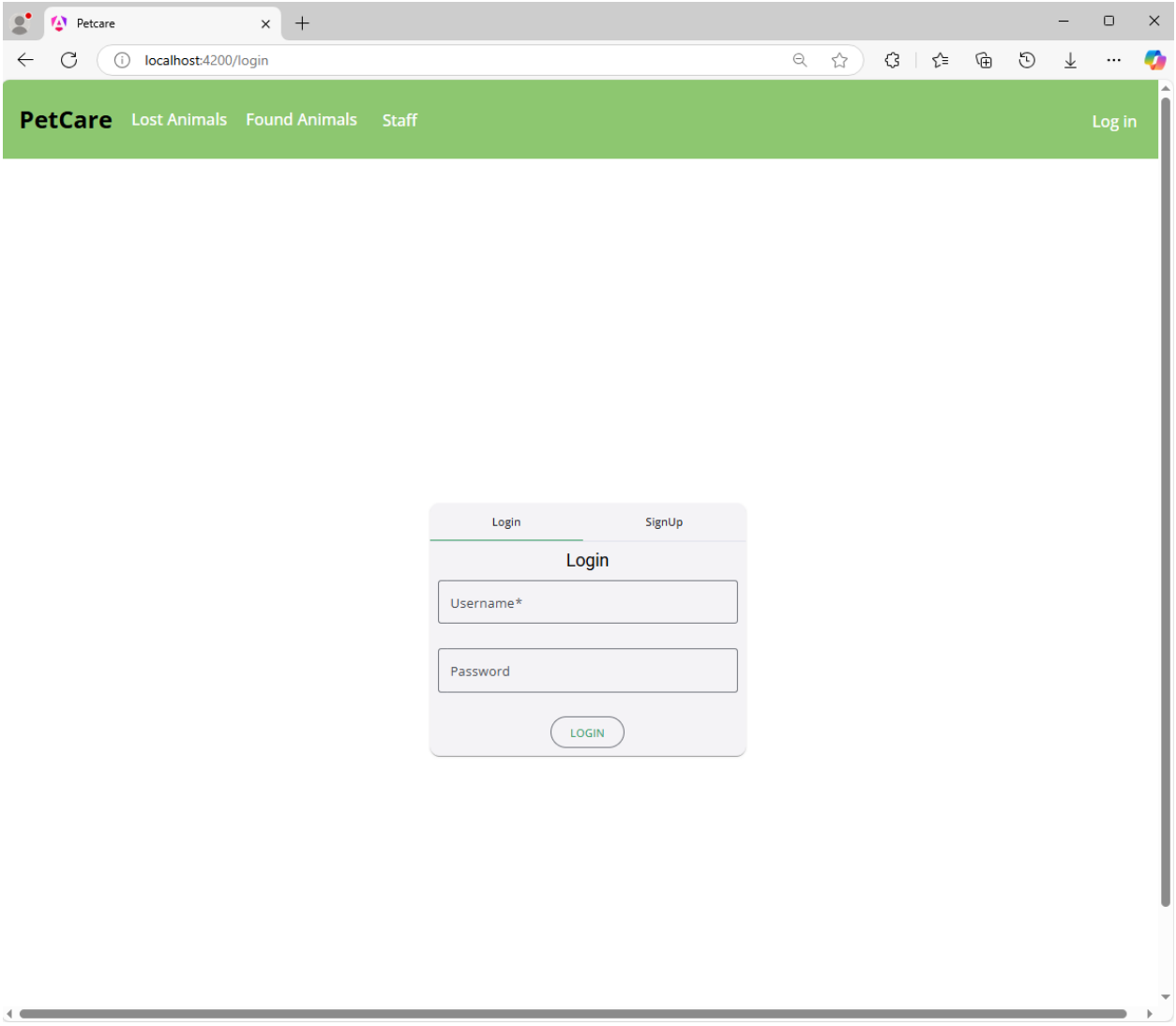
Εικόνα 5.1 - Αρχική σελίδα.

Με την ροδέλα προς τα κάτω φένεται την κεντρική σελίδα



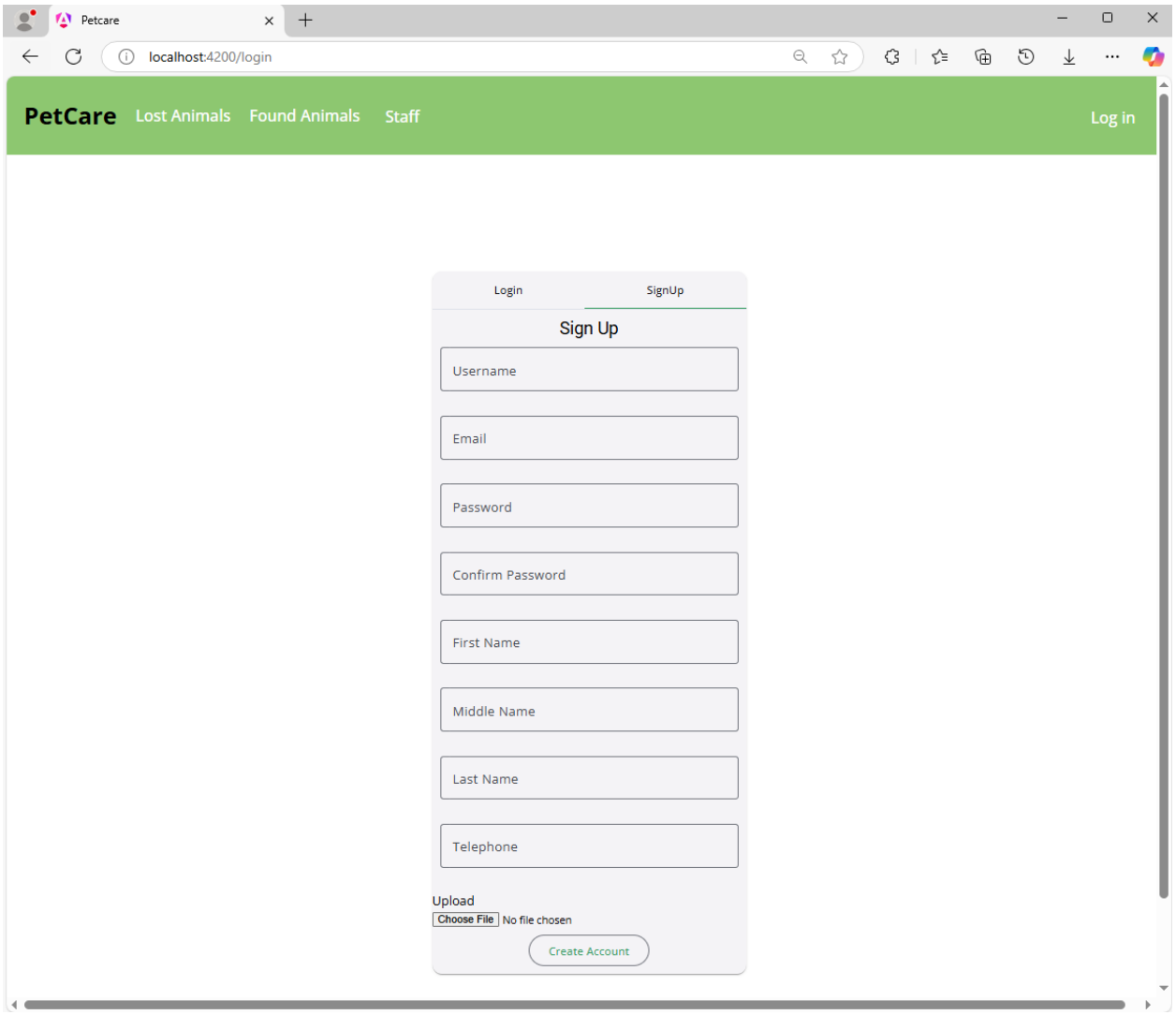
Εικόνα 5.2 - Αρχική σελίδα στο κέντρο της.

Αρχικά παγαίνοντας πάνω στο πλαίσιο στην κορυφή (header) με όλα τα links και θα πάμε στο πάνω δεξιά (Login).



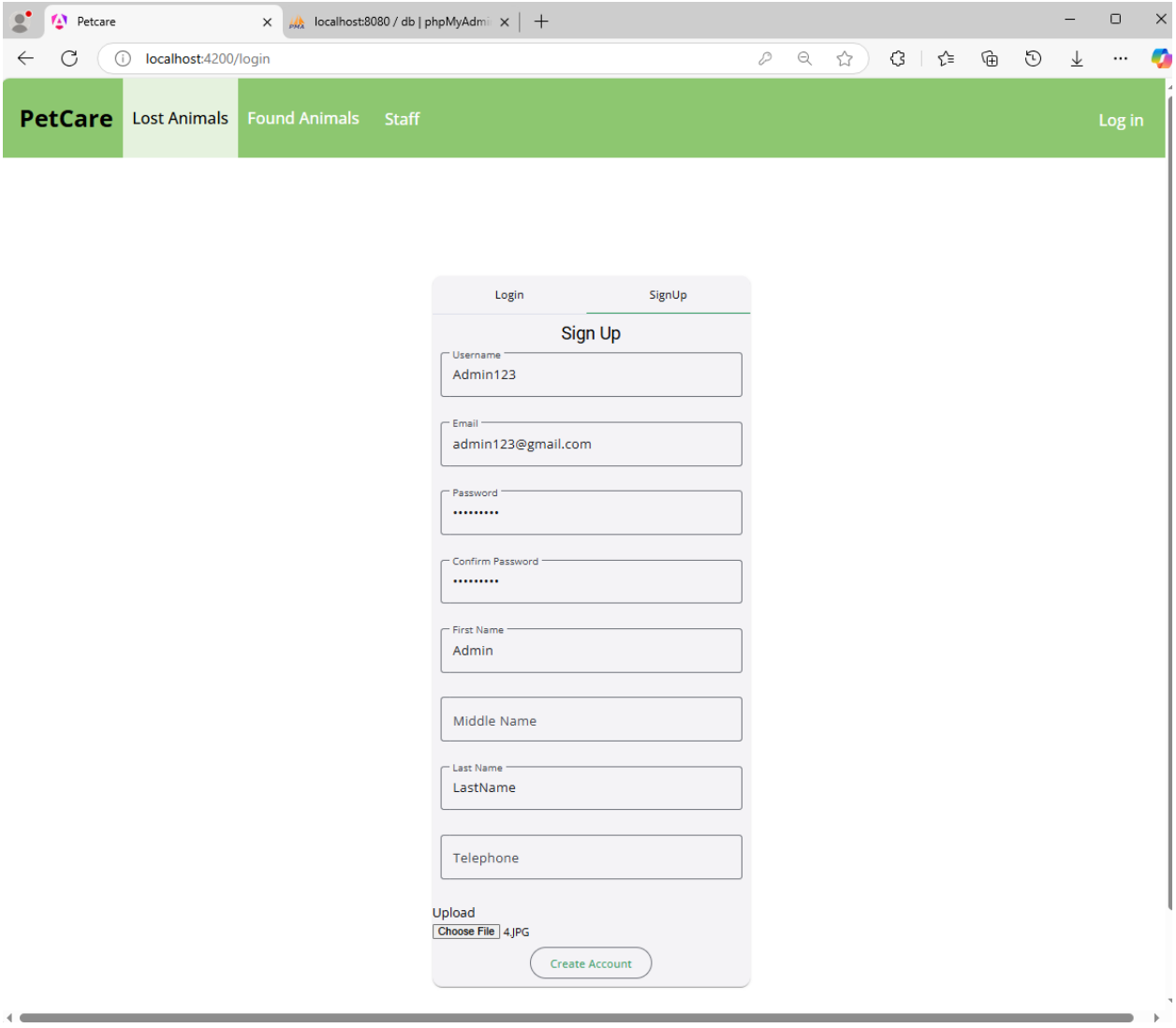
Εικόνα 5.3 - Σελίδα σύνδεσης.

Πηγαίνοντας στην καρτέλα SignUp:

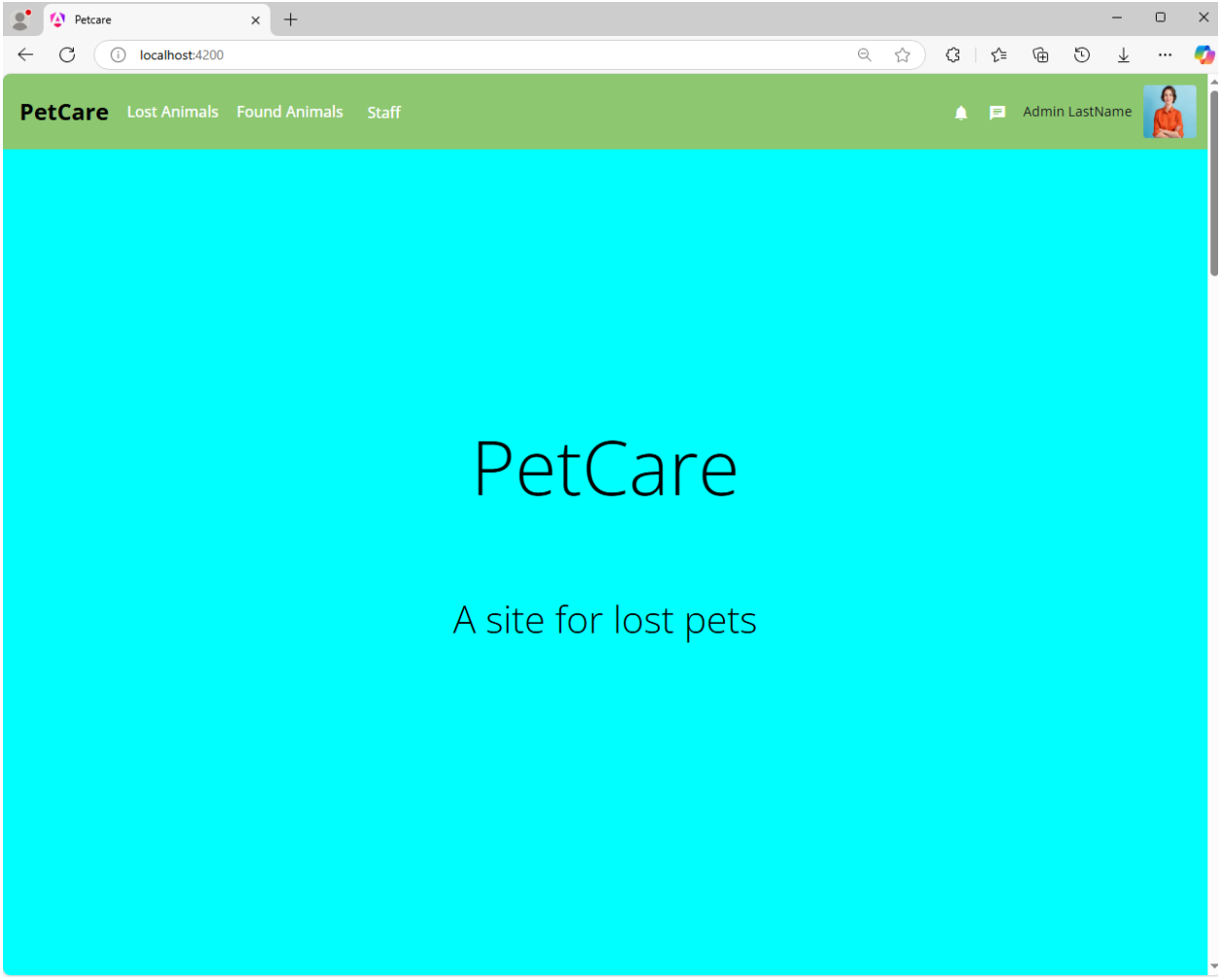


Εικόνα 5.4 - Σελίδα δημιουργίας λογαριασμού.

Τώρα θα πρέπει να μπουν τα στοιχεία του χρήστη, Υποχρεωτικά είναι το Username, Email, Password , Confirm Password , Firstname και Last Name. Τέρμα κάτω μπορούμε να επιλέξουμε και μία εικόνα από τον υπολογιστή μας. Μόλις γίνει ο χρήστης θα πρέπει να ξαναπατηθεί το login για να συνδεθούμε κανονικά.



Εικόνα 5.5 - Παράδειγμα συμπλήρωσης φόρμας για δημιουργία λογαριασμού με συμπληρωμένα τα υποχρεωτικά κενά.

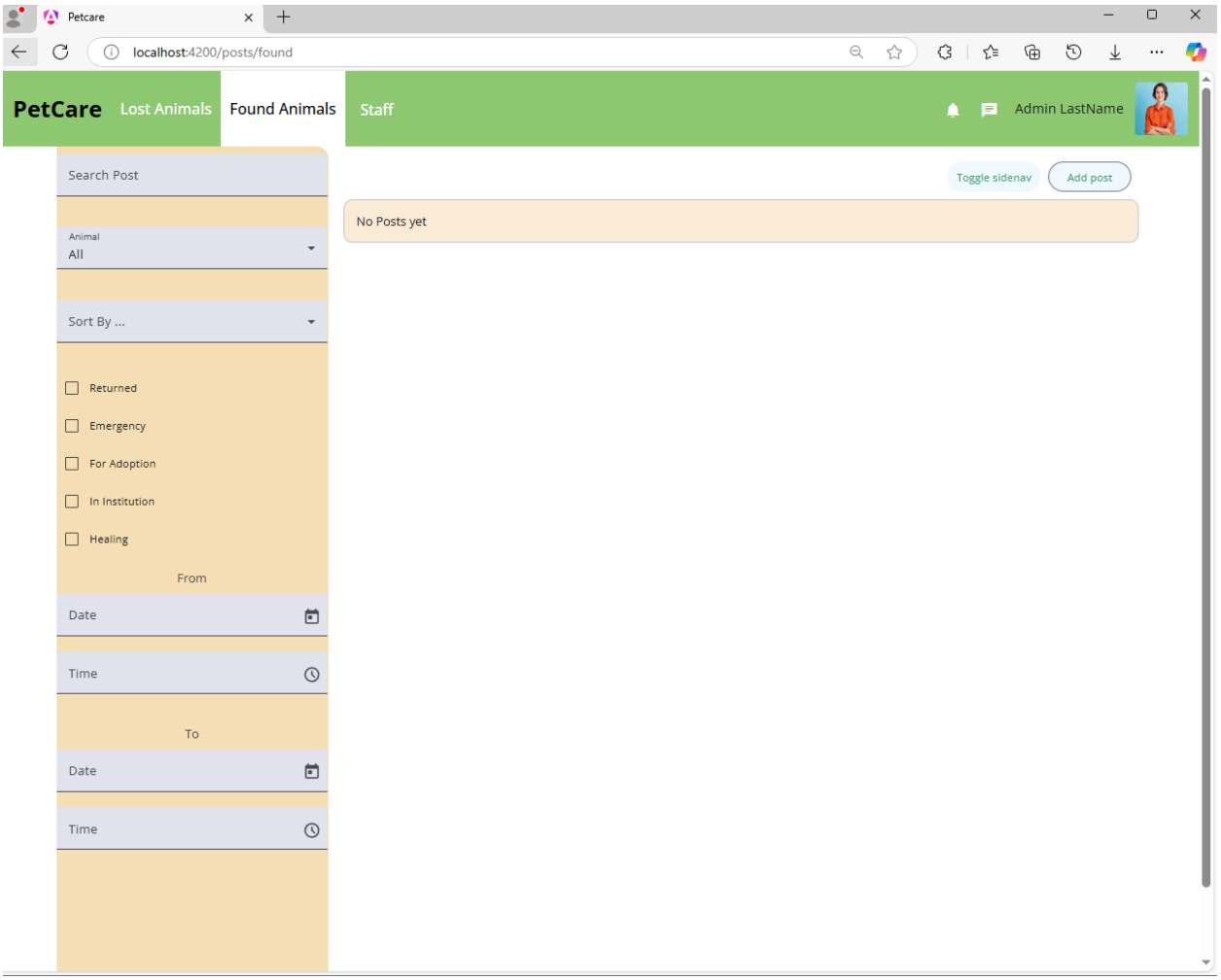


Εικόνα 5.6 - Αρχική σελίδα με συνδεδεμένο λογαριασμό με το προφίλ του χρήστη πάνω δεξιά.

Μόλις γίνει η σύνδεση πάνω δεξιά είναι η εικόνα του χρήστη. Άμα την πατήσουμε έχει επιλογές για τον λογαριασμό του.

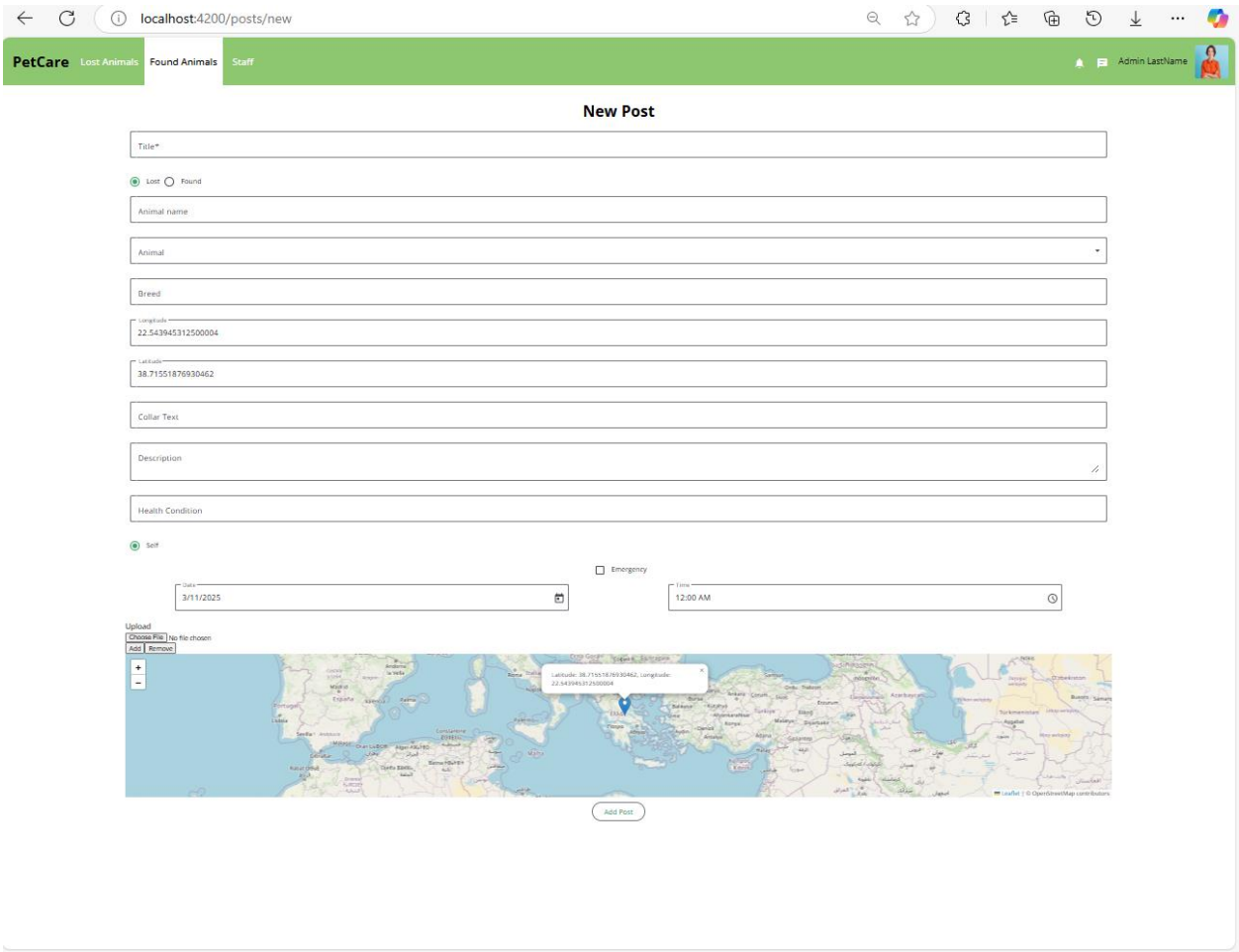
Το site αυτό έχει φτιαχτεί με τις προδιαγραφές μία οθόνης υπολογιστή. Ανάλογα με τον υπολογιστή μπορεί να είναι καλύτερα να γίνει ένα μικρό zoom out (στο 80%) για να φαίνονται τα αντικείμενα πιο καλά.

Πηγαίνοντας στο “Found Animals”:



Εικόνα 5.7 - Σελίδα με αναρτήσεις ευρισκόμενων ζώων κενή.

Αρχικά δεν έχουμε κανένα post, γι' αυτό πατώντας το κουμπί "Add post" πάνω δεξιά:

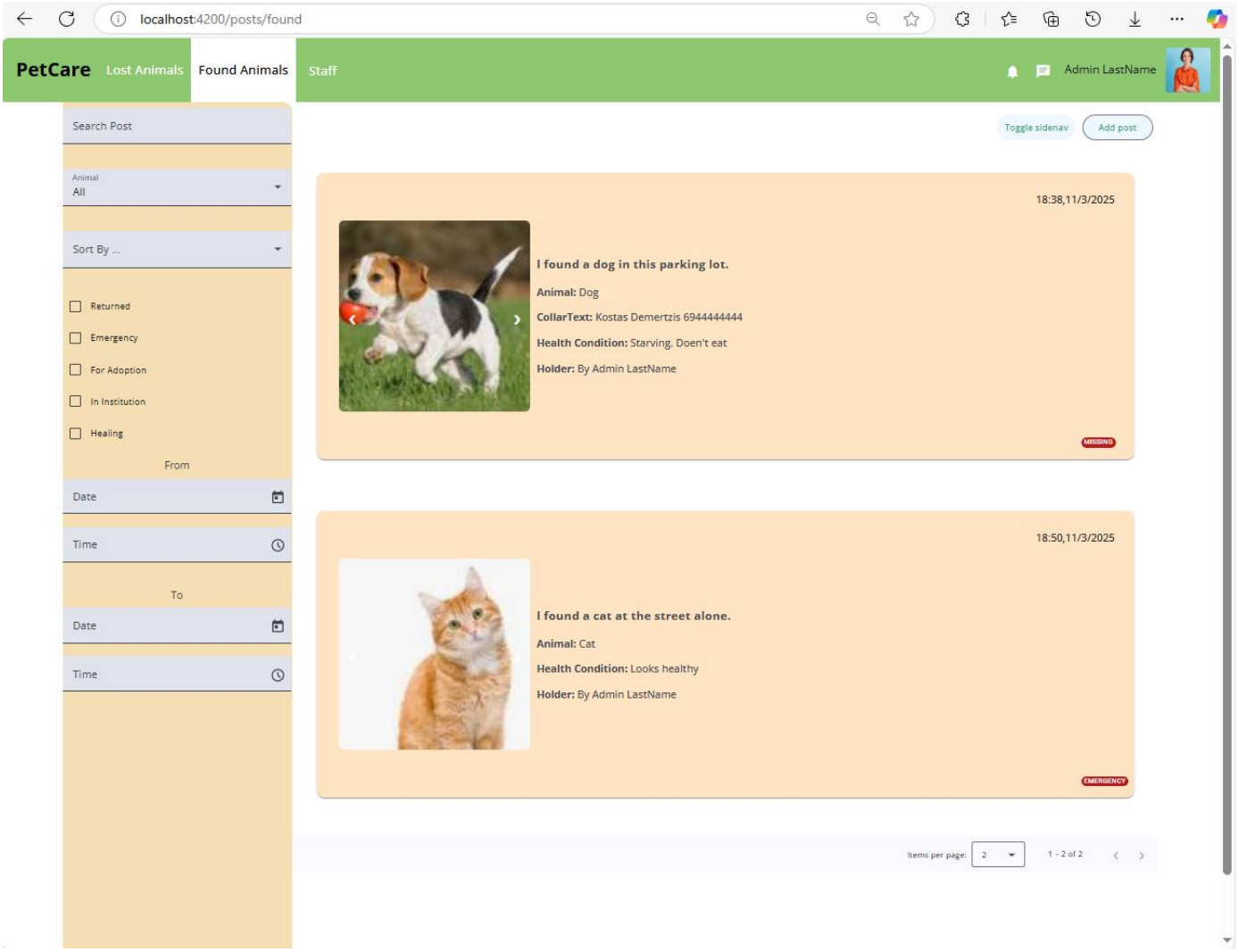


Εικόνα 5.8 - Φόρμα συμπλήρωσης για νέα ανάρτηση "New Post".

Είναι μεγάλο το μενού γι' αυτό είναι σε κάποιο zoom out (δεν χρειάζεται κανονικά).Αρχικά φαίνεται μία τοποθεσία στον χάρτη , πατώντας στον χάρτη που βρίσκεται παρακάτω, τα στοιχεία Latitude και Longitude συμπληρώνονται από μόνα τους. Γίνεται και αντίστροφα αν αλλαχθούν οι αριθμοί στα κενά αυτά να μεταφερθεί στο ανάλογο σημείο ο δείχτης στον χάρτη. Αυτός ο τρόπος εμφάνισης χάρτη χρησιμοποιήθηκε όπου χρειάστηκε λειτουργεία εισαγωγής τοποθεσίας.

Το Lost/Found είναι αν το ζώο χάθηκε ή βρέθηκε. Πιο κάτω στην Self επιλογή αν ο χρήστης είναι σε ένα ίδρυμα και δουλεύει ή κτηνίατρος να κάνει και αυτός ανάρτηση ως το ίδρυμα ή το επάγγελμα. Τώρα υπάρχει μόνο η επιλογή "Self" γιατί δεν είναι ο χρήστης σε κανένα από αυτά μέλος. Υποχρεωτικά κενά είναι η τοποθεσία , ο τίτλος , το τί ζώο είναι και μία τουλάχιστον εικόνα , μπορούν να μπουν πιο πολλές με το κουμπί "Add" (μέχρι 5 μπορούμε).

Υστέρα μετά την δημιουργία 2 αναρτήσεων “Found”:

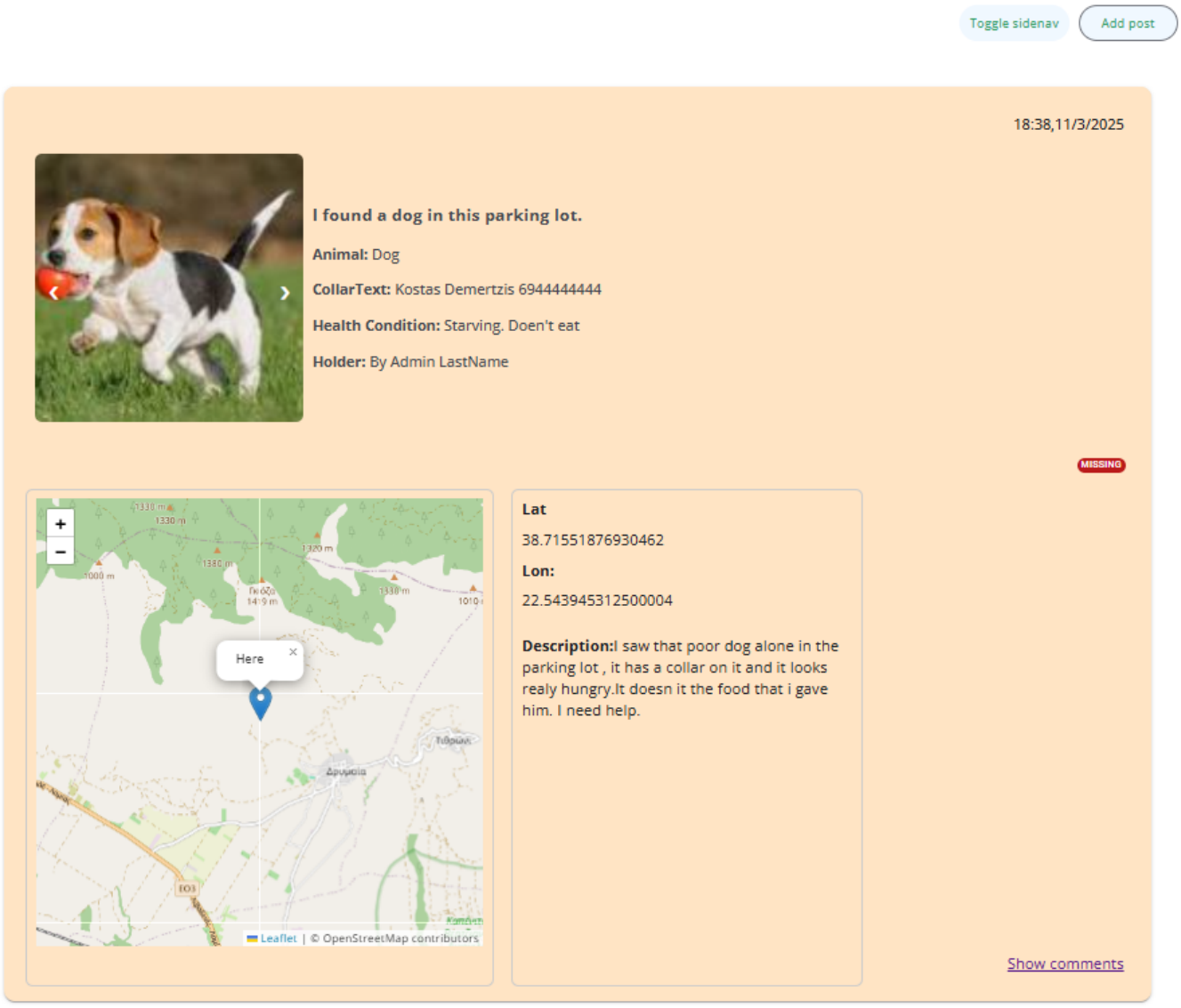


Εικόνα 5.9 - Παράδειγμα σελίδας ευρισκόμενων ζώων με αναρτήσεις.

Η σελίδα εμφανίζει όλες τις αναρτήσεις που έχουν γίνει. Αριστερά είναι το sidebar που θα φιλτράρονται οι αναρτήσεις με τους τρόπους που έχει. Από κάτω είναι ένα Paginator που αλλάζει σελίδες. Μέσα στις αναρτήσεις υπάρχουν τα στοιχεία που βάλαμε στην φόρμα. Στην εικόνα , άμα υπάρχουν πάνω από μία στην ανάρτηση, θα εμφανίζονται κάτι βελάκια που θα αλλάζουν την εικόνα. Κάτω δεξιά σε κάθε ανάρτηση είναι ένας ενημερωτικό με κόκκινο χρώμα που γράφει μέσα την κατάσταση του ζώου , η μία από τις δύο αναρτήσεις (η πρώτη) είναι κατάστασης EMERGENCY. Όταν ένας χρήστης κάνει μία ανάρτηση στο site τύπου

EMERGENCY , στέλνεται ένα μήνυμα σε όλους όσου είναι εγγεγραμμένοι γιατροί στο site , για να πάνε να βοηθήσουν το τυχόν τραυματισμένο ζώο.

Άμα πατηθούν οι αναρτήσεις πάνω επεκτείνονται και δείχνουν τις υπόλοιπες πληροφορίες για την συγκεκριμένη ανάρτηση.



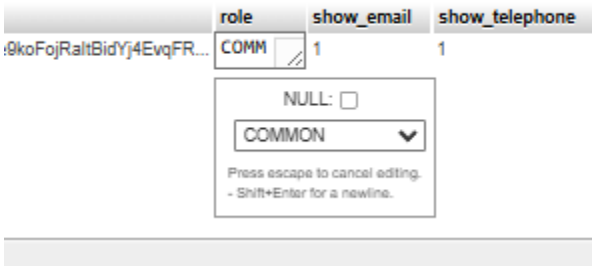
Εικόνα 5.10 - Εκτεταμένη εικόνα ανάρτησης όταν πατηθεί από τον χρήστη.

Άμα πατηθεί το “Show comments” εμφανίζονται τα σχόλια , όπου εκεί μπορεί ο κάθε χρήστης (που δεν είναι timed out) να κάνει το δικό του σχόλιο , όπου αν πατήσουμε το όνομά του να πάμε στις πληροφορίες του.

Εικόνα 5.11 - Σελίδα με τα σχόλια της ανάρτησης ("Show Comments").

Η σελίδα σχολείων στην οποία ήδη κάναμε ένα. Βλέπουμε πάλι την ανάρτηση που επιλέξαμε και μπορούμε κα ξαναπατήσουμε και στο πράσινο μέρος είναι μία φόρμα που γίνονται τα σχόλια. Πατώντας το όνομα του συνδεδεμένου χρήστη πηγαίνει στον λογαριασμό του (Όπως και με την καρτέλα “Account” παρακάτω).

Συνεχίζοντας στην ιστοσελίδα καλύτερα θα είναι να έχουμε χρήστη ρόλου Admin. Εφόσον δε έχουμε άλλους η καλύτερη λύση είναι να γίνει αυτός μέσω της βάσης (να αλλάξουμε τον ρόλο του) , για λόγους ευκολίας.



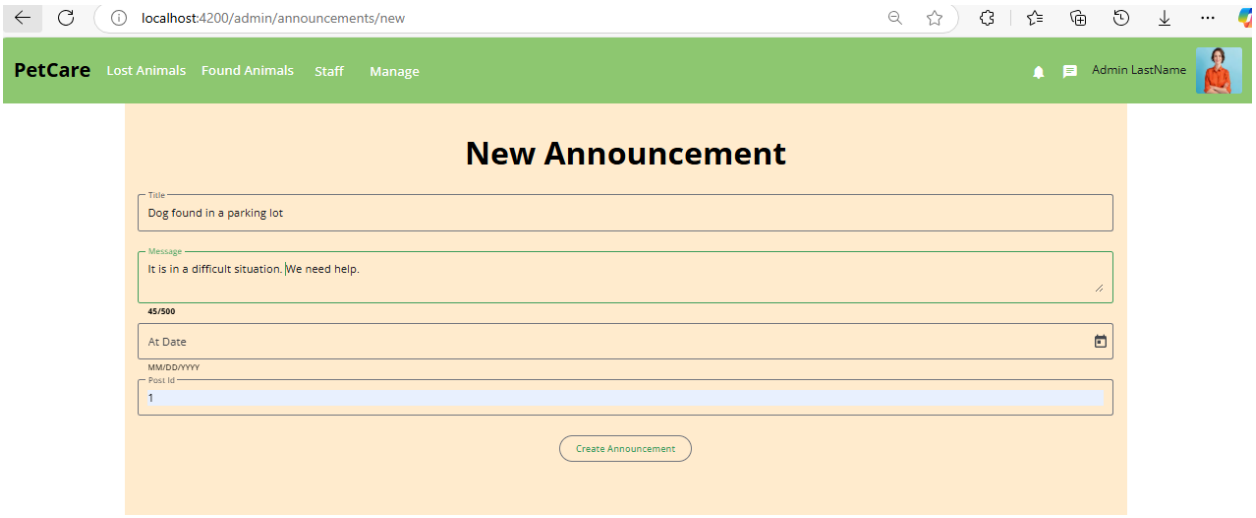
Εικόνα 5.12 - Αλλαγή δεδομένων στο phpMyAdmin (http://localhost:8080/) για τον ρόλο του χρήστη (Πίνακας "user").

Πηγαίνοντας στο localhost:8080 που είναι το phpMyAdmin μπορεί ο ρόλος του χρήστη να αλλάξει πρέπει να γίνει επανασύνδεση του χρήστη εκείνου (αν είναι ήδη συνδεδεμένος). Παρατηρείτε στο site τώρα ότι υπάρχει άλλη μία καρτέλα στο header , η “Manage” , που είναι για διαχειριστές και έγκριτές επαγγελμάτων. Επίσης με τον ρόλο του Admin υπάρχουν οι ιδιότητες του ρόλου Manager που μπορεί να διαγράψει , σχόλια και αναρτήσεις.

Πηγαίνοντας το “Announcements”:

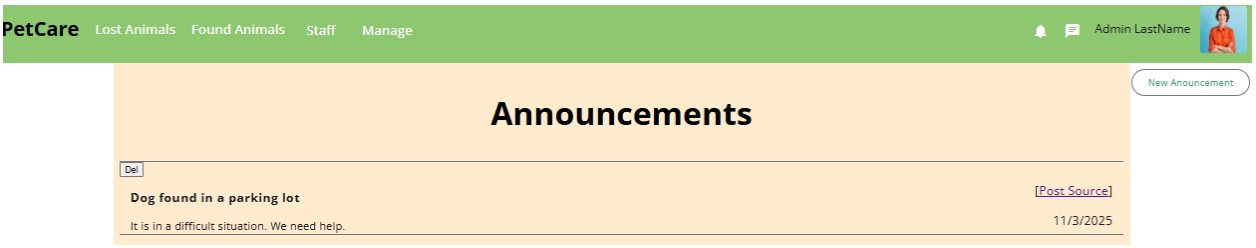
Εικόνα 5.13 - Σελίδα με όλες τις ανακοινώσεις (announcements) . Πάνω δεξιά βρίσκεται ένα κουμπί για την δημιουργεί ανακοινώσεων ("New Accouncement" , μόνο για συγκεκριμένους ρόλους).

Πηγαίνοντας στο “New Announcement” πάνω δεξιά μπορούν να γίνουν οι ανακοινώσεις. Στο PostId μπαίνει ο αριθμός μίας ανάρτησης. Ο αριθμός αυτός υπάρχει στο path του site όταν πάμε στα σχόλια της ανάρτησης αυτής. Το PostId είναι προαιρετικό αλλά άμα μπει θα υπάρχει ένα link που πάει στα σχόλια της ανάρτησης. Οι ανακοινώσει που δεν το χρησιμοποιούν είναι κυρίως περιεχομένου που έχουν να κάνουμε με ολόκληρη την σελίδα (άμα θα κλείσει για λίγο , κλπ.).



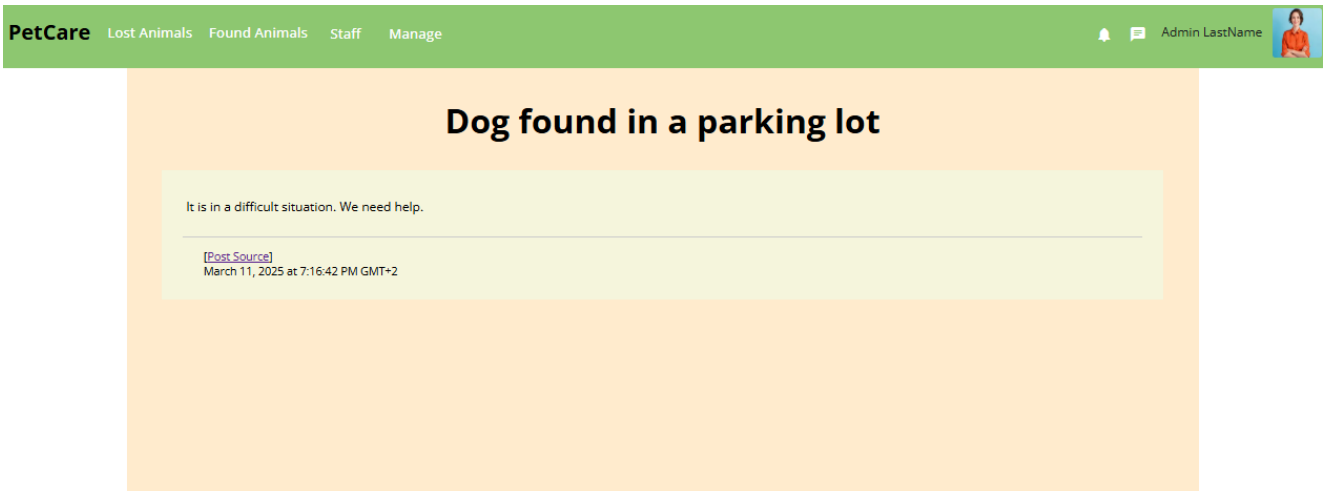
Εικόνα 5.14 - Φόρμα δημιουργίας ανακοινώσεων (Ρόλοι MANAGER , APPROVER και ADMIN).

Δημιουργώντας πάει τον χρήστη πίσω στις ανακοινώσεις:



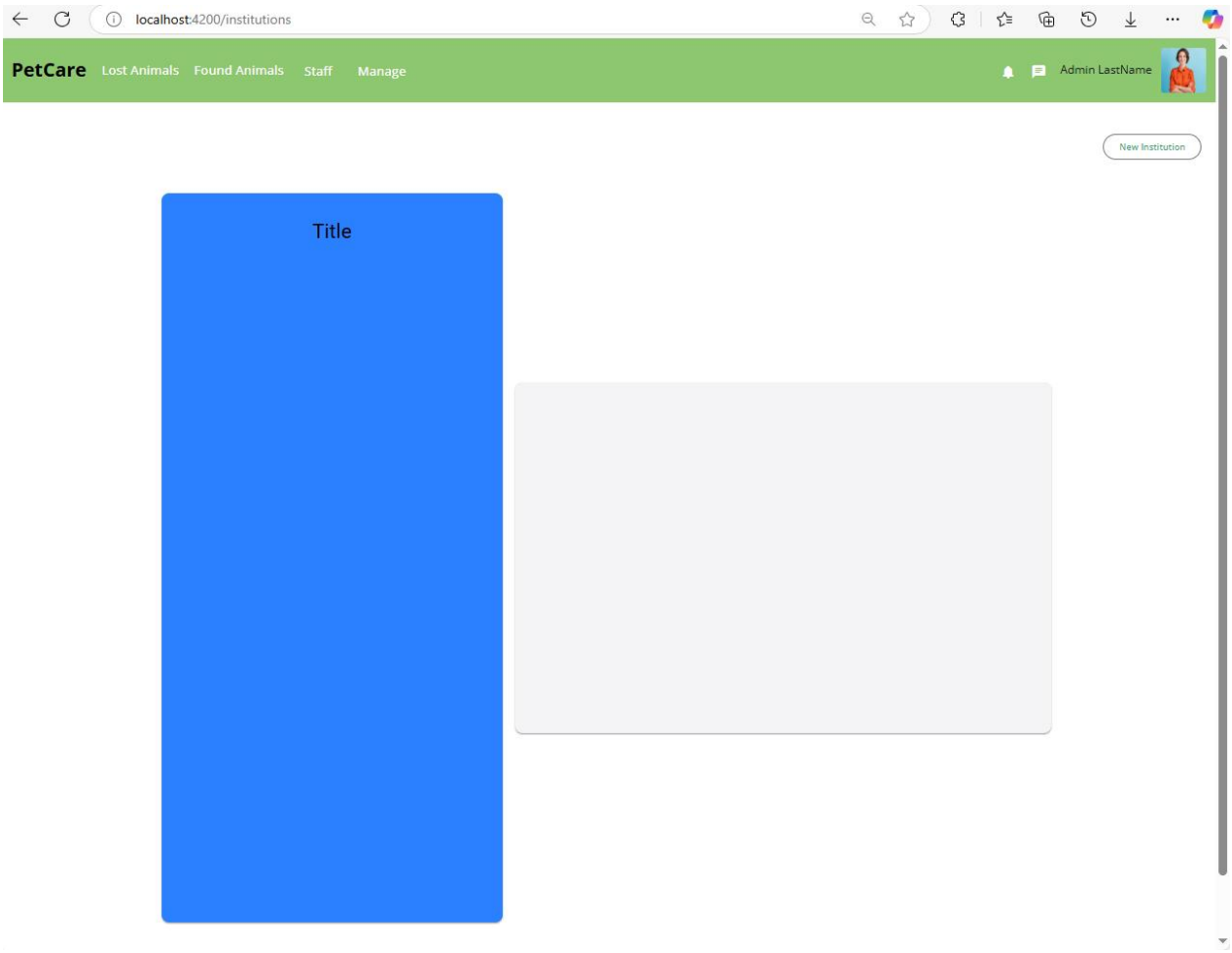
Εικόνα 5.15 - Εμφάνιση ανακοίνωσης. Περιέχει το κουμπί "del" για διαγραφή της (μόνο για MANAGER , APPROVER και ADMIN).

Πατώντας την ανακοίνωση (όχι το κουμπί) πηφαίνει στην ανακοίνωση αυτή.



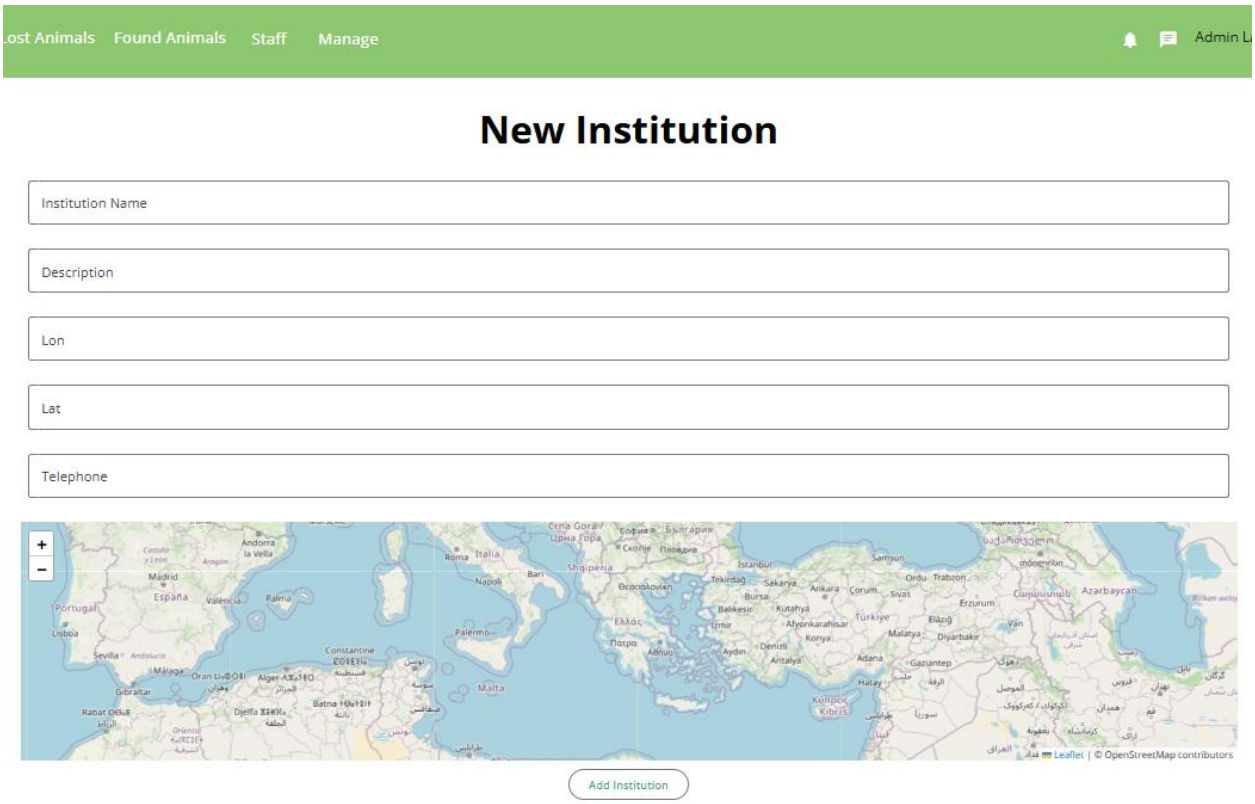
Εικόνα 5.16 - Ανοιγμα ανακοίνωσης.

Πηγαίνοντας στα ιδρύματα (institutions):



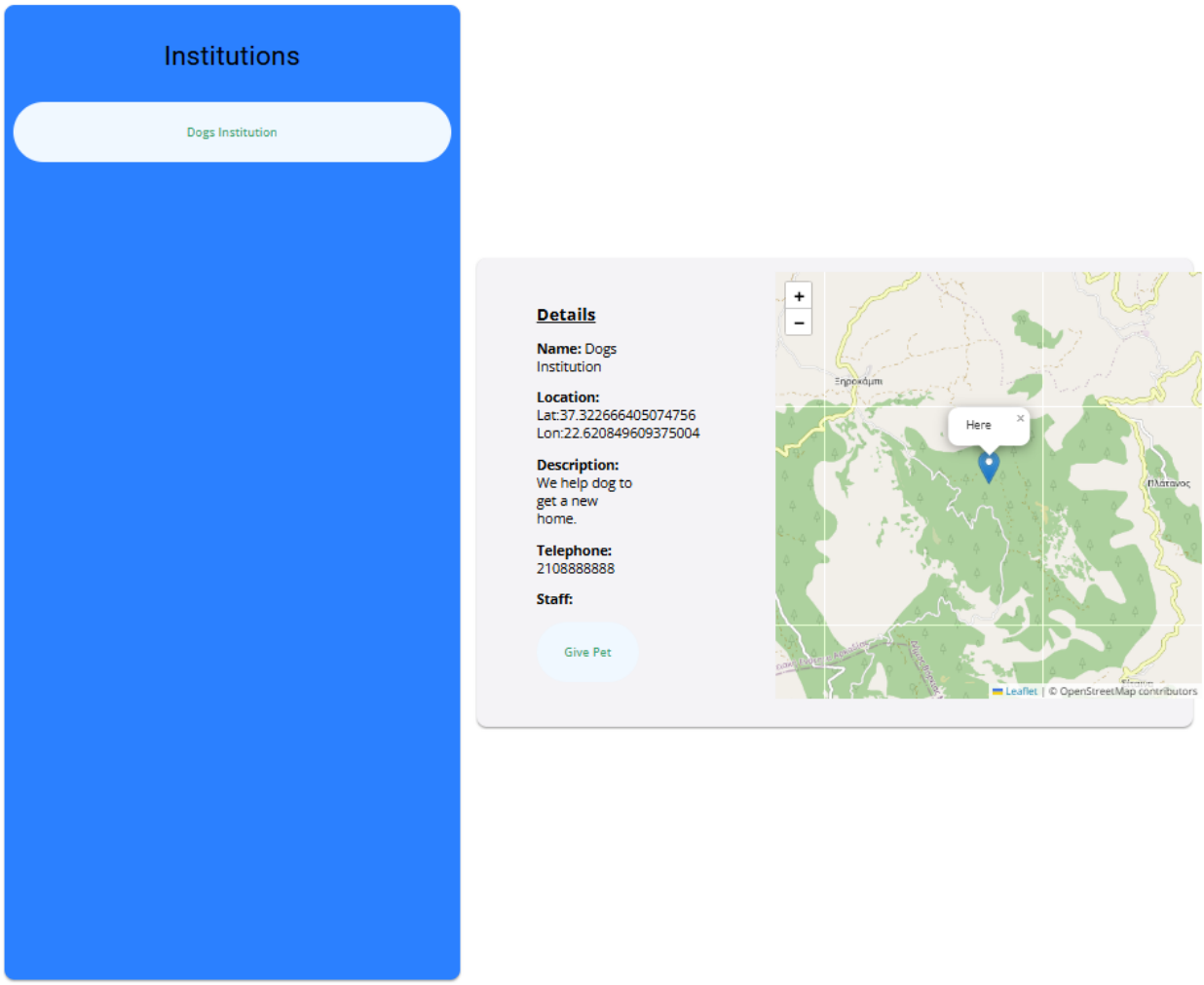
Εικόνα 5.17 - Εμφάνιση λίστας ιδρυμάτων (institutions) (κενή) (και για μη συνδεδεμένους χρήστες).

Εδώ μπορεί ο οποιοσδήποτε να δει τα ιδρύματα , αλλά μόνο οι ADMIN μπορούν να βάλουν ένα καινούργιο. Πατώντας πάνω δεξιά το “New Institution” με χρήστη ADMIN:



Εικόνα 5.18 - Φόρμα δημιουργίας ιδρύματος (μόνο χρήστες ADMIN).

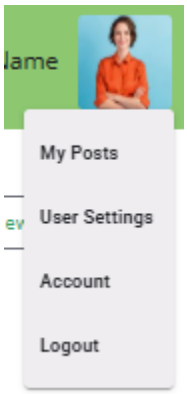
Προσθέτει το institution στον πίνακα:



Εικόνα 5.19 - Πίνακας με όλα τα ιδρύματα.

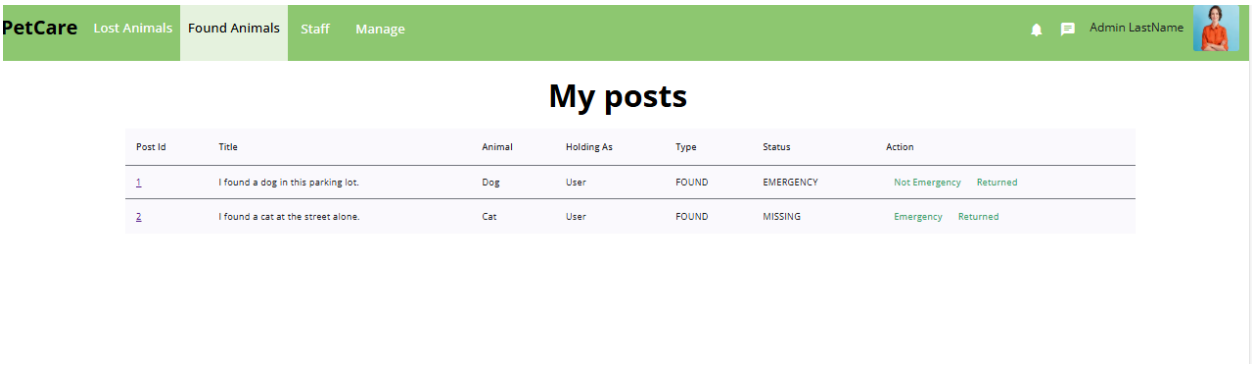
Όταν πατιέται το κουμπί του ιδρύματος φαίνεται το σημείο που επιλέξαμε στον χάρτη για το πού είναι όταν το δημιουργήσαμε. Το κουμπί “Give Pet” θα εμφανίζεται όταν ο χρήστης έχει αναρτήσεις για ευρισκόμενα ζώα (δηλαδή που έχει στην κατοχή του). Πατώντας το μπορεί να δοθεί το ζώο στο ίδρυμα αυτό που θα έρθει ένας να το πάρει. Θα πρέπει πρώτα όμως να δοθεί το ζώο αυτό και μετά να το δώσουμε μέσω του site , διότι έτσι φαίνεται ότι το ζώο παραδόθηκε και έχει νέο κάτοχο.

Τώρα πατώντας στη εικόνα του συνδεδεμένου χρήστη πάνω δεξιά μας βγαίνει αυτό το μενού:



Εικόνα 5.20 - Σελίδες χρήστη (μόνο αν πατηθεί η εικόνα του λογαριασμού πάνω δεξιά).

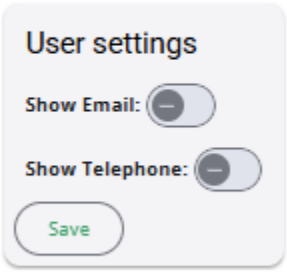
Πηγαίνοντας στο “My Posts”:



Εικόνα 5.21 - Σελίδα "My Posts".

Εδώ φαίνονται όλες τις αναρτήσεις του συγκεκριμένου χρήστη. Δίνει την δυνατότητα στον χρήστη να θέτει τις αναρτήσεις του σε έκτεκτη κατάσταση ή όχι. Όταν είναι γίνει μία νέα αίτηση δωρεάς ζώου θε εμφανίζεται ένα κουμπί δίπλα “Give” σε κάθε ανάρτηση ευρισκόμενου ζώου.

Πηγαίνοντας στο ίδιο μενού , στο “User Settings”:



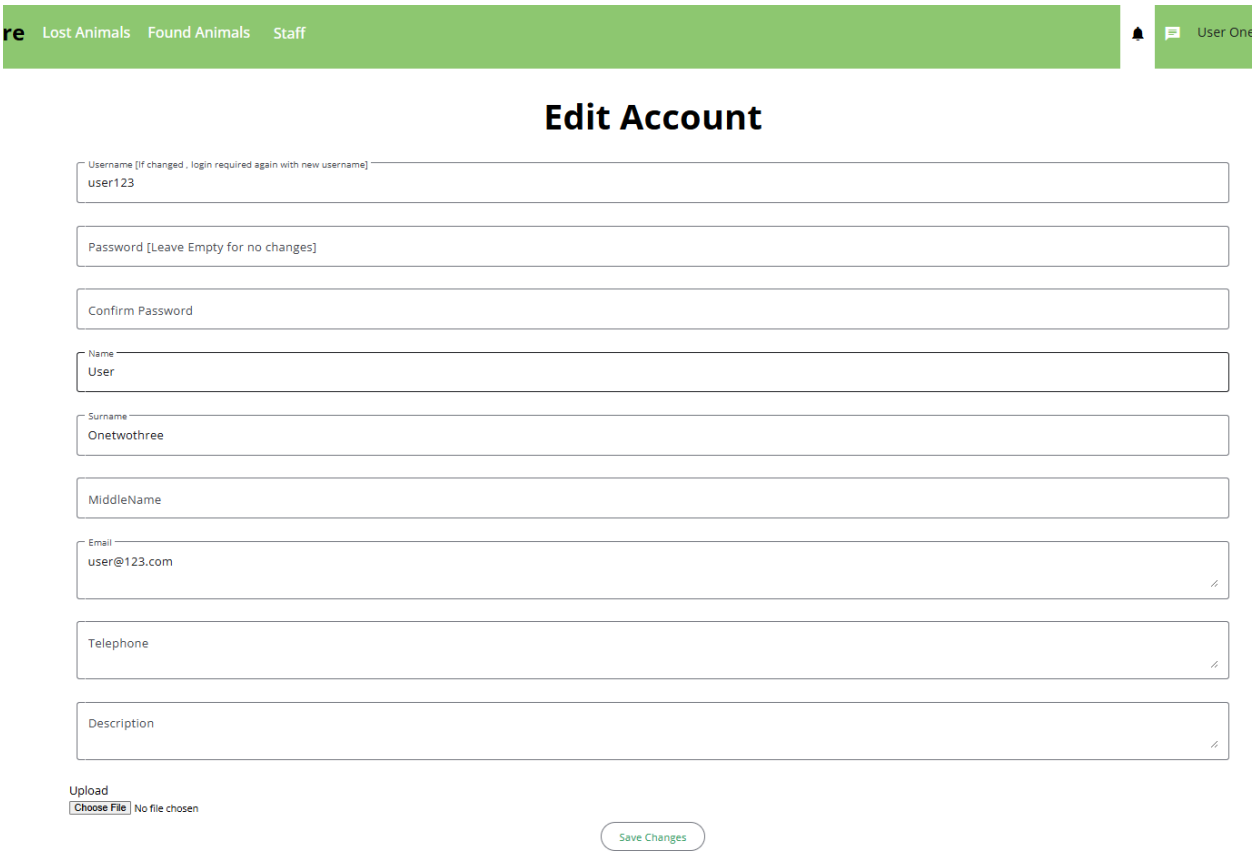
Εικόνα 5.22 - Περιεχόμενο σελίδας "User Settings".

Εδώ μπορεί να ρυθμήστεί αν θέλει ο χρήστης να εμφανίζονται στα στοιχεία του στους υπόλοιπους. Αμα τα απενεργοποιηθεί δεν θα δίχονται αυτά τα στοιχεία όταν θέλει ένας χρήστης να ψάξει τον λογαριασμό του.

Τέλος στο ίδιο μενού στο “Account”:

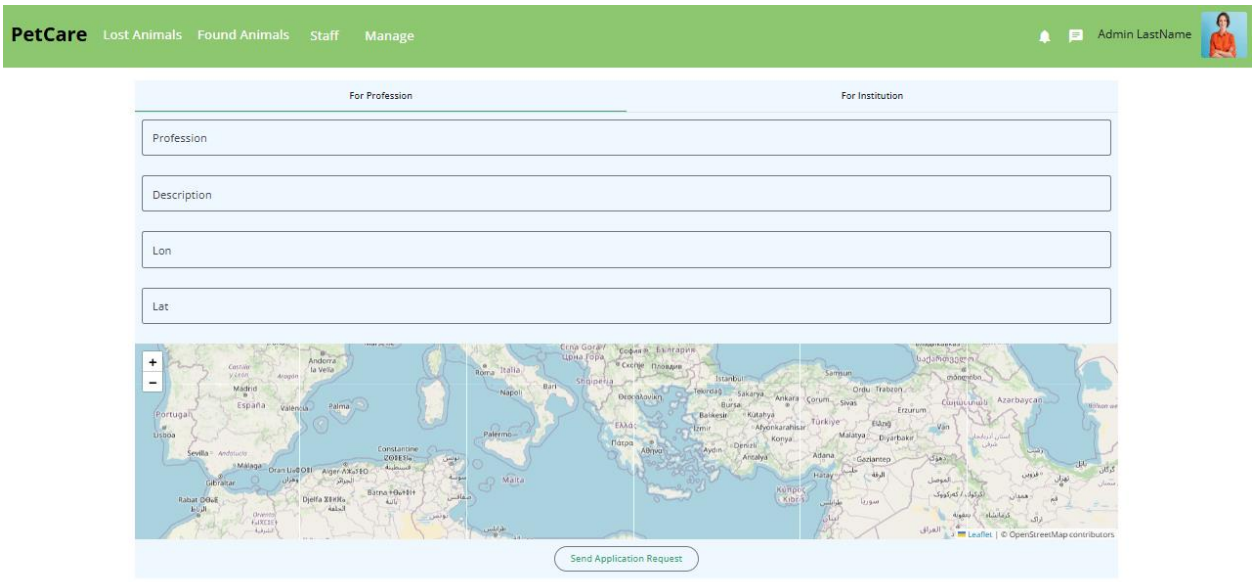
Εικόνα 5.23 - Περιεχόμενο σελίδας "My Account".

Εδώ είναι ένας πίνακας με τα στοιχεία του χρήστη. Στο “Change Details” μπορεί να αλλάξει τα στοιχεία του (username, password, name κλπ.) και να βάλει μία περιγραφή στον λογαριασμού του (που θα φαίνεται στους άλλους χρήστες) .



Εικόνα 5.24 - Φόρμα αλλαγής στοιχείων χρήστη “Change Details”.

Το “Make an Application” από κάτω στον λογαριασμό του τον πάει να κάνει μια νέα αίτηση για εγγραφή ενός επαγγέλματός του στο site που μπορεί να βοηθήσει τα χαμένα ζώα.



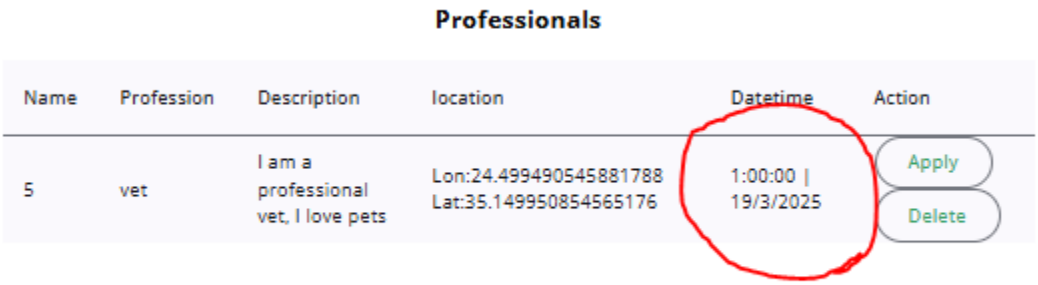
Εικόνα 5.25 - Φόρμα αποστολής αίτησης για επάγγελμα. Επιλεγμένη η πρώτη καρτέλα (tab) για του επαγγελματίες γιατρούς.

Μπορεί ο χρήστης να κάνει αίτηση για επάγγελμα γιατρού (πρώτο tab) ή μπορεί να κάνει αίτηση για διαχειριστή σε ίδρυμα (δεύτερο tab , πρέπει να υπάρχει πρώτα το ίδρυμα για να ο κάνουμε αυτό).

Όταν ολοκληρώνεται η αίτησή του θα εμφανιστεί στο “User Applications” του header.

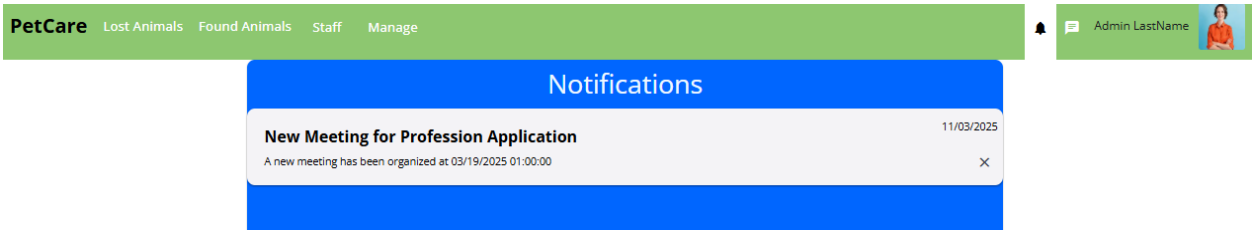
Εικόνα 5.26 - Σελίδα με όλες τις αίτησης επαγγέλματος "User Applications" (μόνο για APPROVER και ADMIN).

Εδώ μόνο οι χρήστες με τον ρόλο APPROVER και ADMIN μπορούν να δουν την σελίδα , όπως και να εγκρίνουν ή να διαγράψουν μία αίτηση. Μπορούν επίσης να διοργανώσουν μία ημερομηνία όπου θα υπάρχει κάποιο ραντεβού με τον χρήστη αυτό (αν χρειαστεί φυσική παρουσία) μέσω του “Select Date”.



Εικόνα 5.27 - Αίτηση για επάγγελμα με σχεδιασμένη ημερομηνία και ώρα για ραντεβού με τον χρήστη.

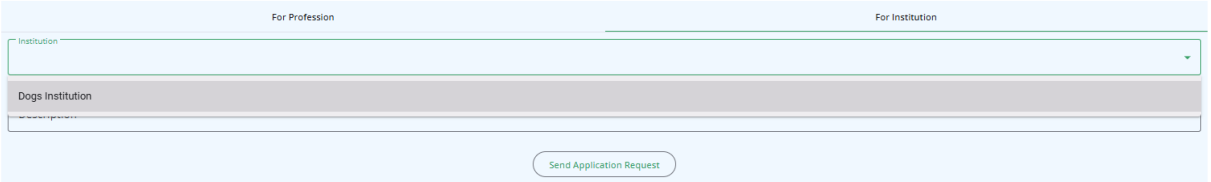
Βάζοντας μία ημερομηνία θα εμφανιστεί στην αίτηση.



Εικόνα 5.28 - Ειδοποίηση για το ραντεβού αίτησης επαγγέλματος για τον χρήστη εκείνον.

Επίσης ενημερώνει και τον χρήστη που έκανε την αίτηση από τις ειδοποιήσεις του.

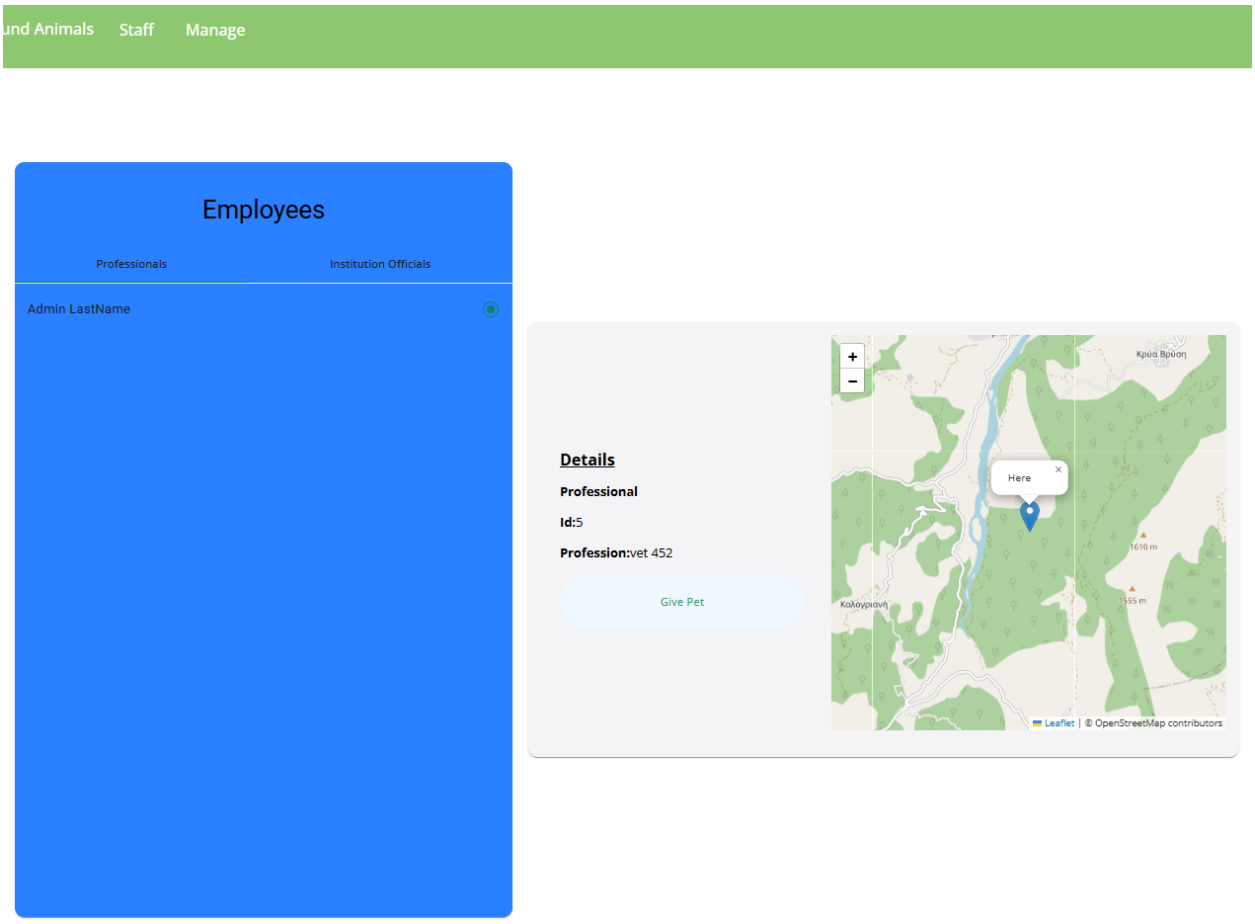
Για τους διαχειριστές σε ίδρυμα (institution officials) θα πάμε στην 2η καρτέλα και θα επιλέξουμε το ίδρυμά μας.



Εικόνα 5.29 - Φόρμα για αίτηση διαχειριστή ιδρύματος ζώων (δεύτερη καρτέλα).

Με τον ίδιο τρόπο μπορούν οι υπεύθυνοι αν εγκρίνουν ή όχι την αίτησή του.

Άμα εγκριθεί θα εμφανίζεται στην σελίδα “Professionals” που θα υπάρχει το όνομα του χρήση.



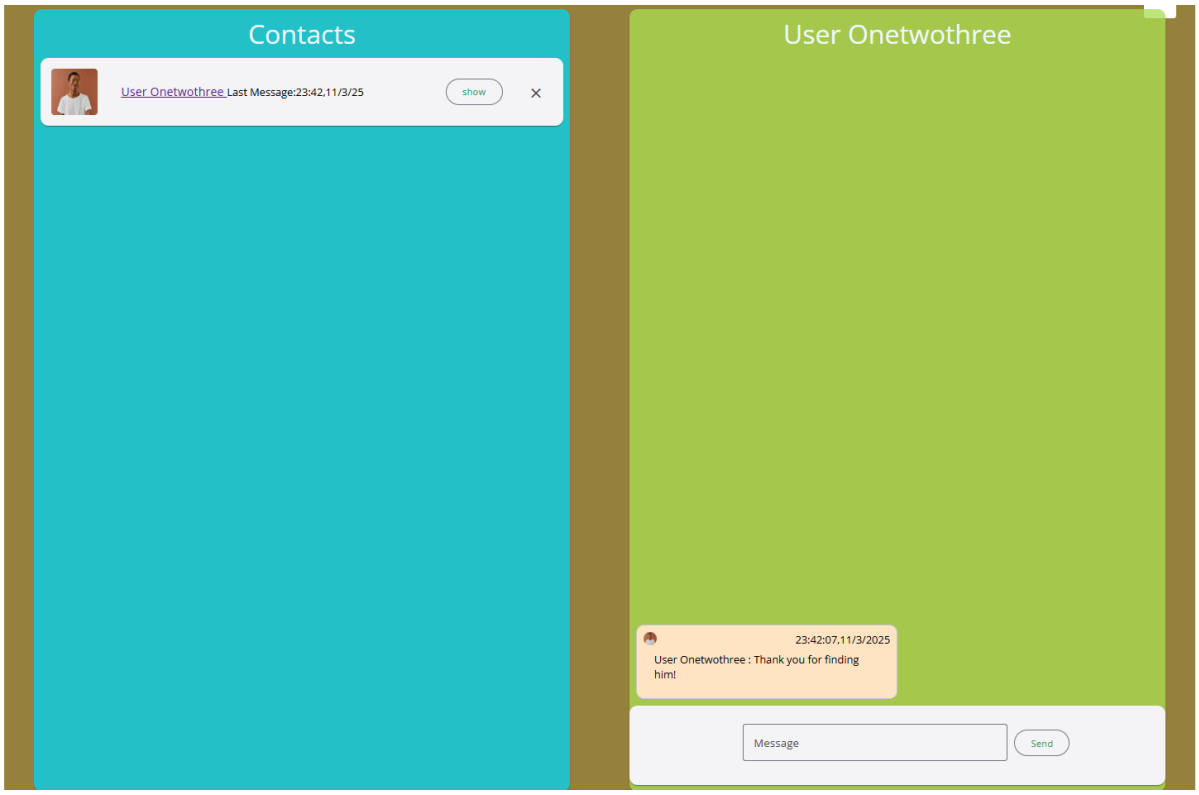
Εικόνα 5.30 - Σελίδα με τους επαγγελματίες "Professionals" (και για μη συνδεδεμένους χρήστες).

Έχει και εδώ 2 καρτέλες με επαγγελματίες ή διαχειριστές ιδρυμάτων

Τώρα πάμε στα μηνύματά μας από το εικονίδιο πάνω δεξιά:

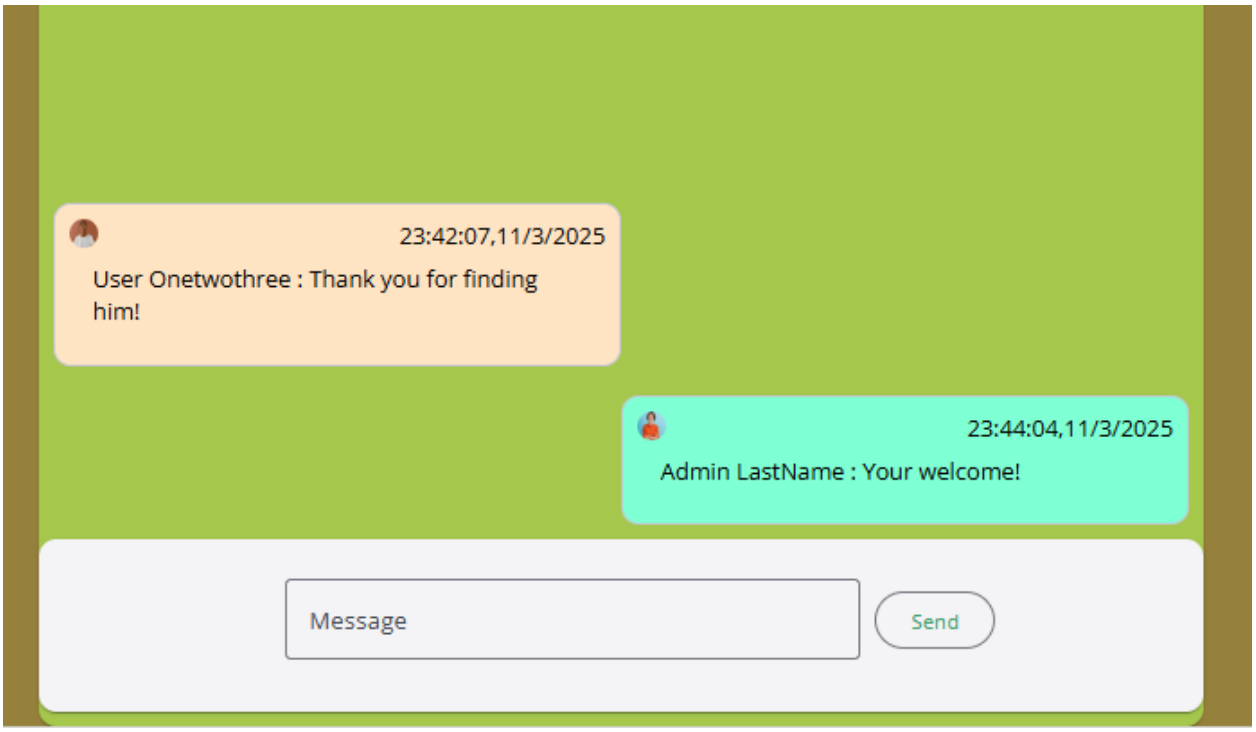
Εικόνα 5.31 - Σελίδα με τα μηνύματα του χρήση. Αριστερά με πράσινο εμφανίζονται οι χρήστες που έχουν στείλει μη διαβασμένο μήνυμα.

Υπάρχει μία επαφή (message box) με πράσινο χρώμα , που σημαίνει ότι ένας χρήστης του έστειλε μήνυμα . Πατώντας το “Show” αλλάζει το χρώμα σε άσπρο (ως αναγνωσμένο) και δεξιά φαίνονται τα μηνύματά ανάμεσα στους δύο χρήστες.



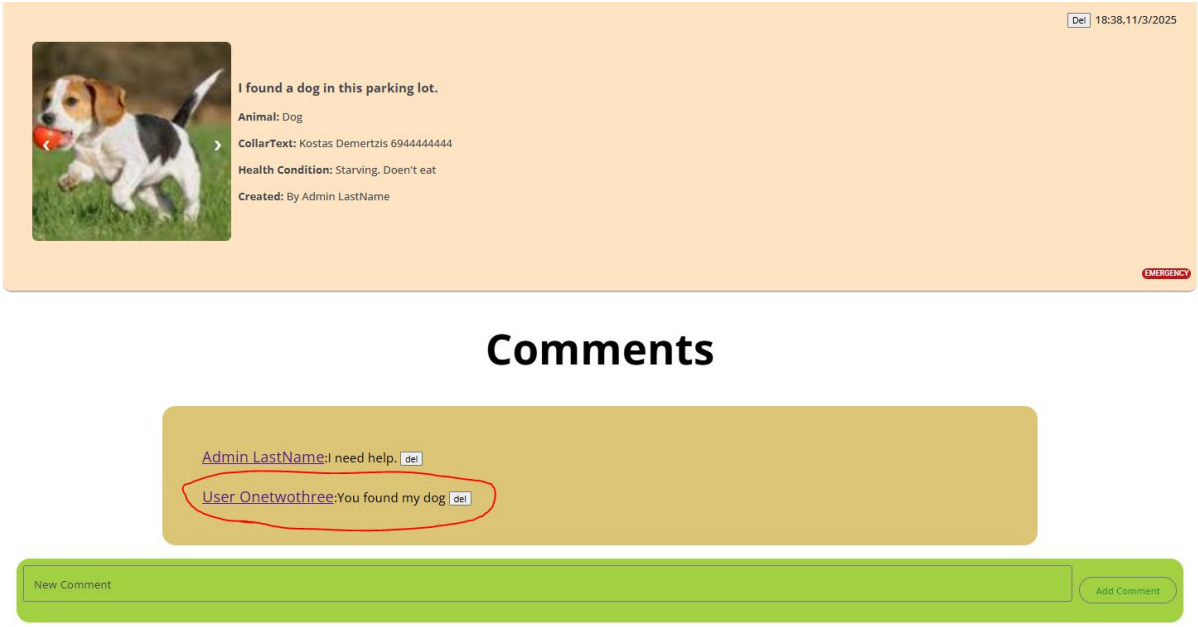
Εικόνα 5.32 - Σελίδα μηνυμάτων χρήση με εμφανισμένα τα μηνύματα του χρήστη (στα δεξιά πατώντας το κουμπί "Show" του συγκεκριμένου χρήστη).

Εκεί μπορεί να γίνει συνομιλία μεταξύ τους.



Εικόνα 5.33 - Συζήτηση δύο χρηστών με μηνύματα.

Μπορούν επίσης να γράψουμε στα σχόλια μιας αναρτήσεις αν έχουν επιπλέον πληροφορίες για αυτό.



Εικόνα 5.34 - Επικοινωνία χρήστη μέσω σχολείων σε μία ανάρτηση.

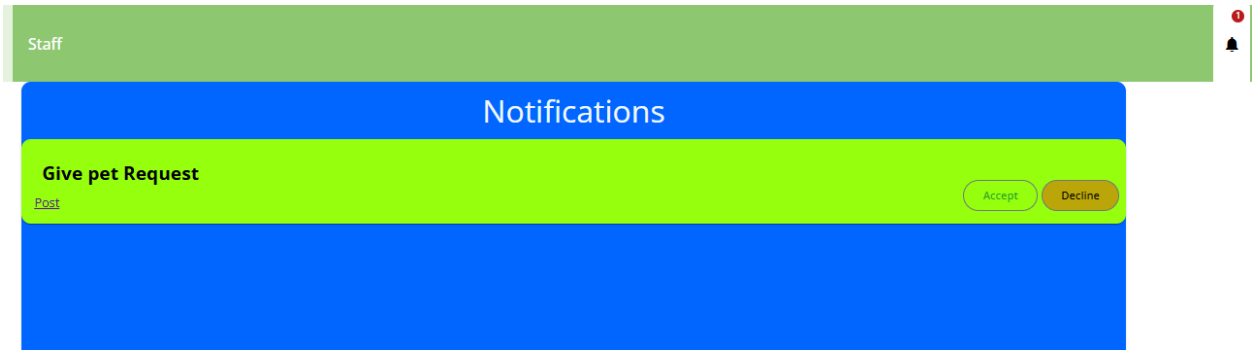
Άμα ο ιδιοκτήτης ενός ζώου βρέθηκε μπορεί ο χρήστης που έχει την ανάρτηση να πατήσει το link που είναι στο όνομά του στα σχόλια και να πάει στον λογαριασμό του , οπού εκεί μπορεί να του στείλει μήνυμα ή ένα ζώο (στην περίπτωση αυτή το σκύλο).

Εικόνα 5.35 - Εμφάνιση στοιχείων χρήστη (όχι του συνδεδεμένου) .

Πατώντας το “Give Pet”:

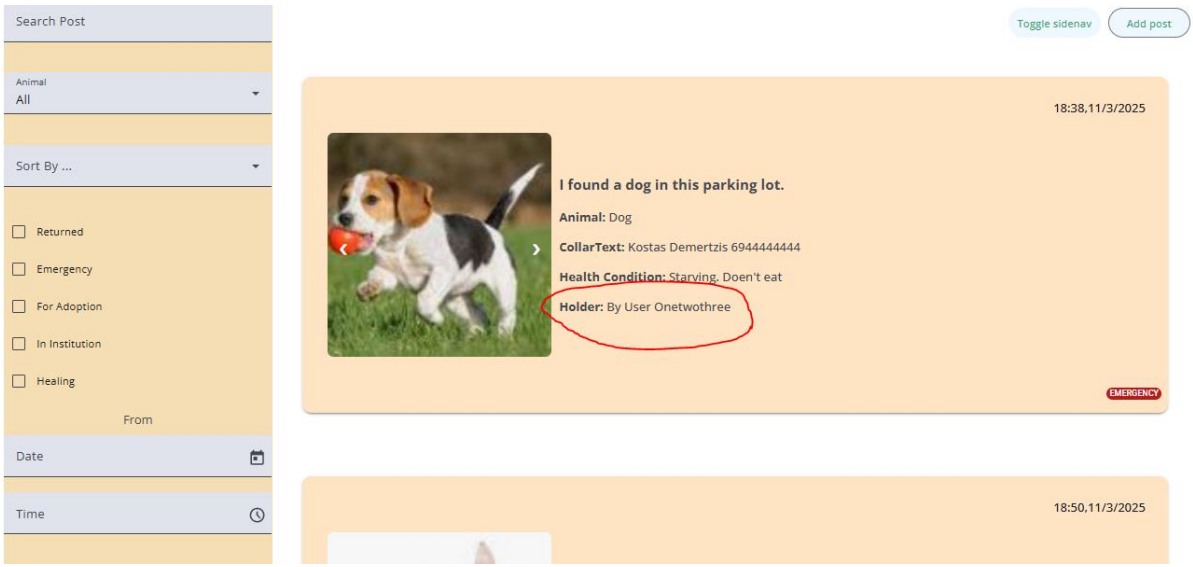
Εικόνα 5.36 - Επιλογή προσφοράς ζώου σε έναν άλλον χρήση μέσω της σελίδας "My Posts" (με το κουμπί "Give" για ευρισκόμενα ζώα μόνο).

Πατάει στην ανάρτηση που θέλει να στείλει το κουμπί “Give” και στέλνεται μία αίτηση στον χρήστη εκείνον που θέλουμε να το στείλουμε. Άμα την αποδεχτεί στην ιστοσελίδα θα δείχνει ότι ο κάτοχος του ζώου της ανάρτησης είναι εκείνος.

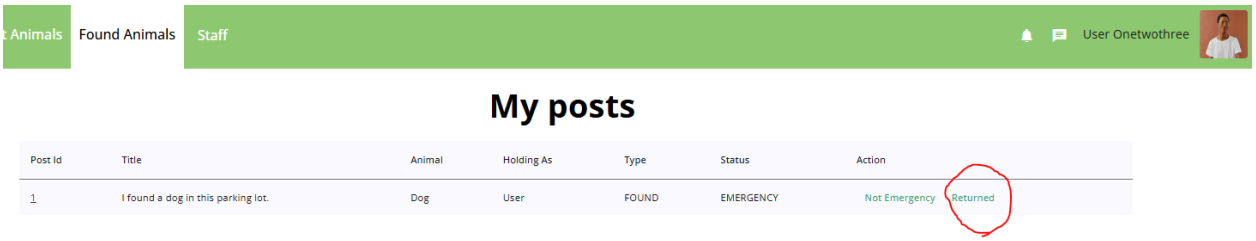


Εικόνα 5.37 - Αίτηση μεταφοράς ζώου (Give Request) από τον σελίδα ειδοποιήσεων (notifications).

Τώρα άμα στην ανάρτηση αυτή θα υπάρχει ο νέος ιδιοκτήτης (κάτοχος) του ζώου.

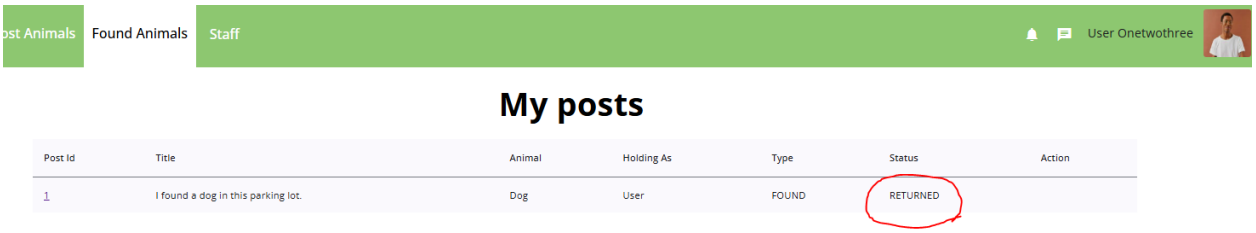


Εικόνα 5.38 - Αλλαγή κατόχου ζώου μέσω αποδοχής της αίτησης ανταλλαγής του.



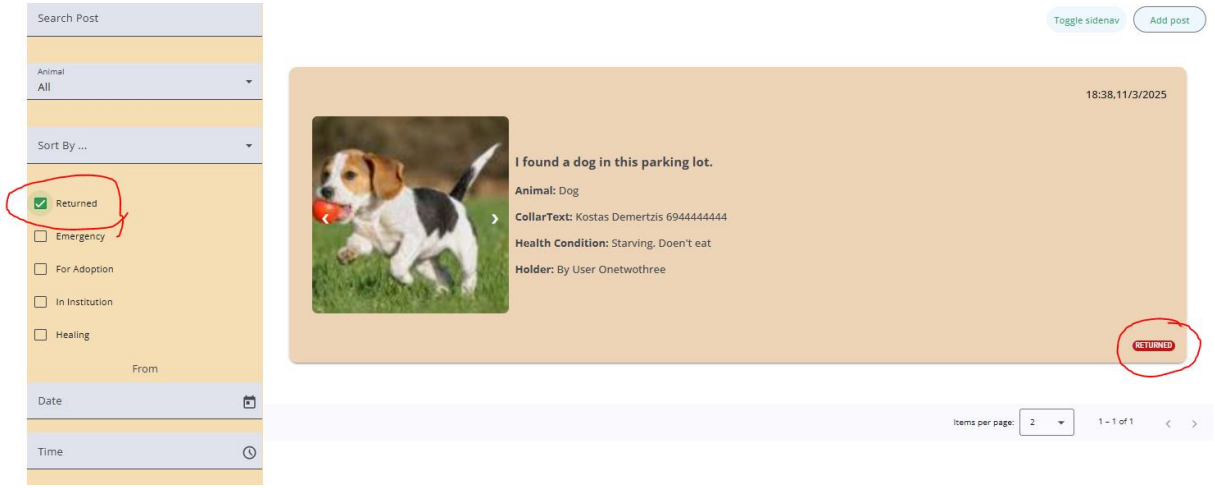
Εικόνα 5.39 - Σελίδα με τις αναρτήσεις μας πριν πατήσουμε το κουμπί "Returned".

Άμα το ζώο τώρα είναι δικό του μπορεί να το θέσει ως επιστραμμένο στον ιδιοκτήτη του (RETURNED)



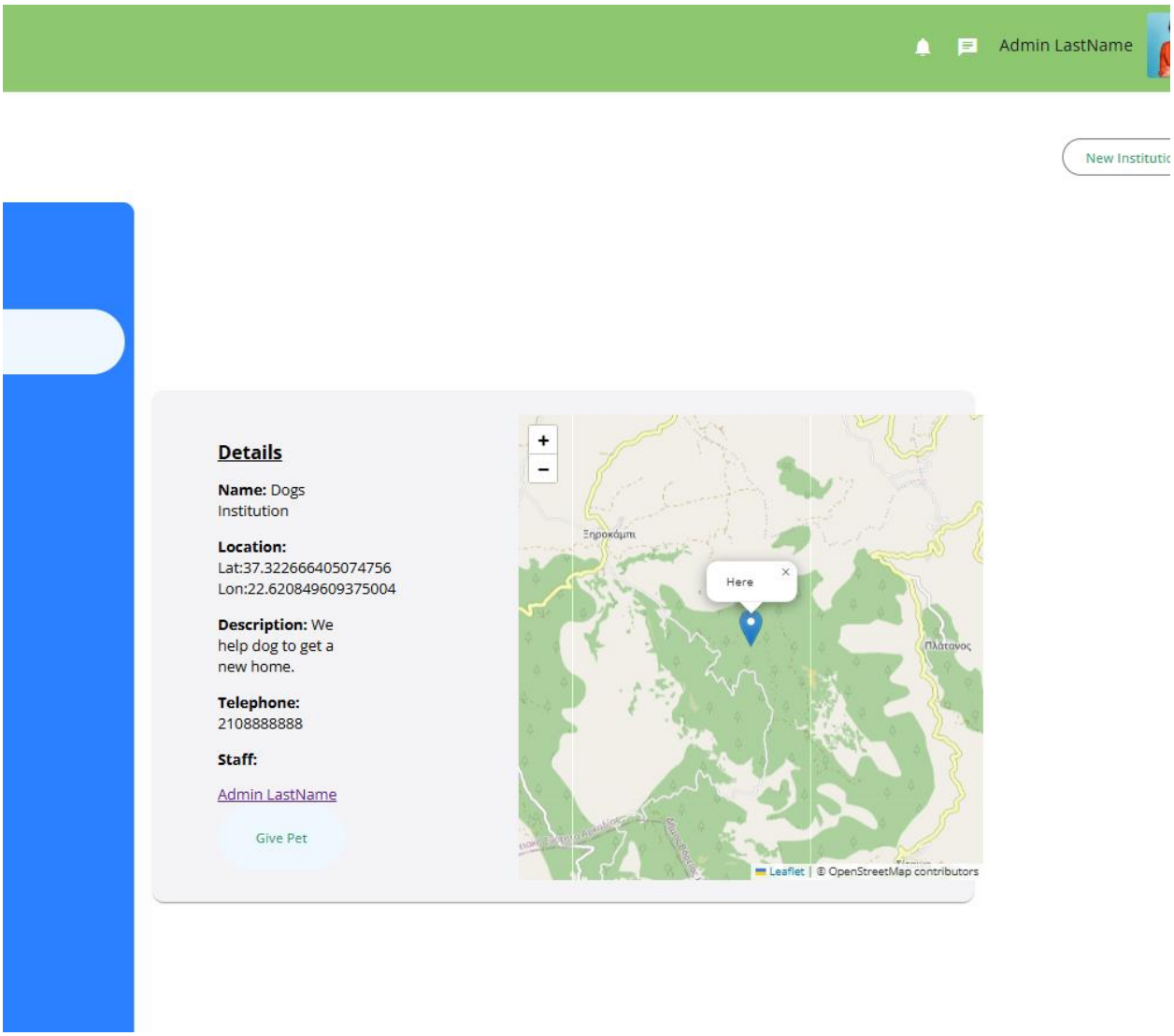
Εικόνα 5.40 - Σελίδα με τις αναρτήσεις μας μετά που θα πατήσουμε το κουμπί "Returned".

Τώρα δεν θα δείχνει το ζώο στης κανονικές αναρρήσεις. Για να βρεθεί θα πρέπει να μπει το φίλτρο από αριστερά “RETURNED” που θα δείχνει μόνο τα επιστραμμένα ζώα.



Εικόνα 5.41 - Αναρτήσεις με επιστραμμένα ζώα (πρέπει να πατήσουμε την επιλογή "Returned" για να εμφανιστεί)

Μπορούμε επίσης να δωθεί ένα ζώο σε ένα ίδρυμα ή γιατρό. Και στις δύο περιπτώσεις θα πρέπει ένας ειδικός για το ίδρυμα (ένας που είναι διαχειριστής εκεί) να αποδεχτεί το αίτημα παράδοσης ζώου.

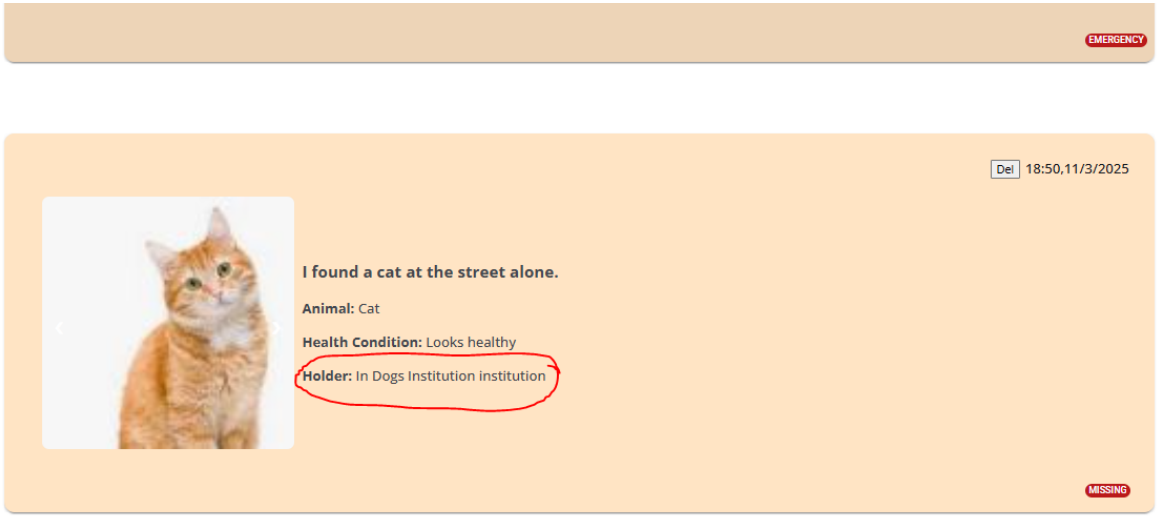


Εικόνα 5.42 - Επιλογή ενός ιδρύματος από την σελίδα "Institutions" με το κουμπί "Give Pet" για όσους χρήστες έχουν ευρισκόμενα ζώα στο site.

Πατώντας το “Give Pet” και στέλνοντας την ανάρτηση:

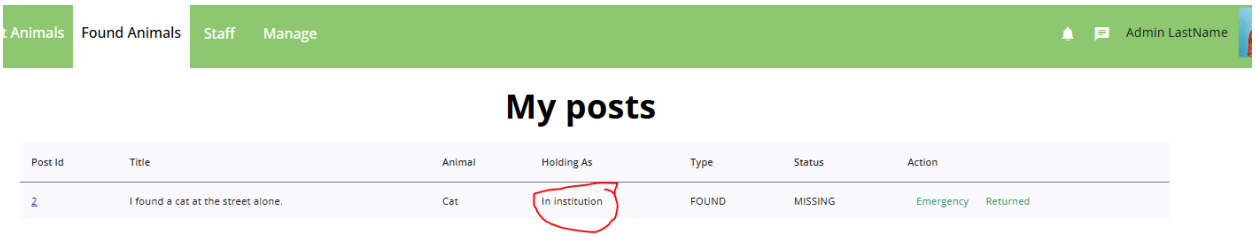
Εικόνα 5.43 - Αίτηση μεταφοράς ζώου σε ένα ίδρυμα. Όλοι οι διαχειριστές του ιδρύματος μπορούν να αποδεχτούν την αίτηση.

Με αποδοχή της αίτησης η ανάρτηση αλλάζει κάτοχο.



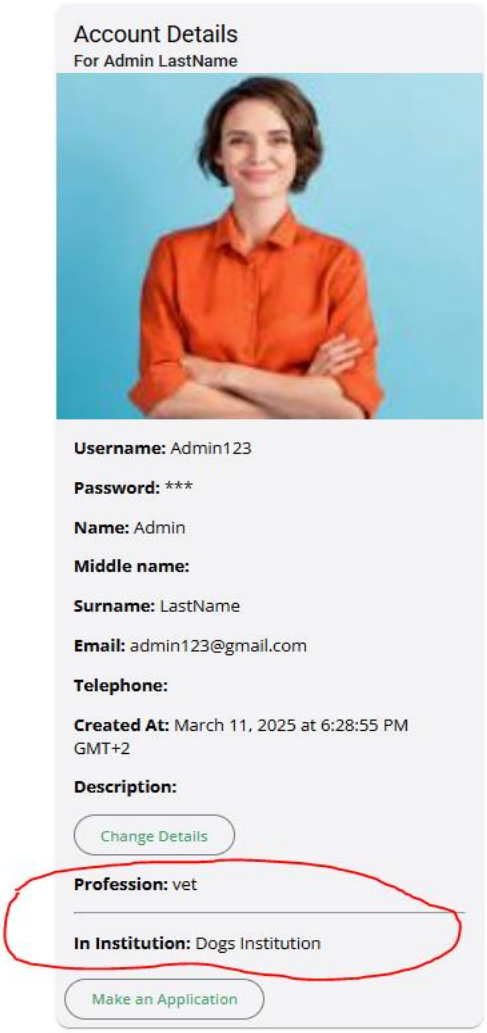
Εικόνα 5.44 - Κατοχή ανάρτησης από ίδρυμα.

Μπορεί επίσης κάθε διαχειριστής ιδρύματος να δει στις αναρρήσεις του κάθε ζώο που είναι στο ίδρυμά του.



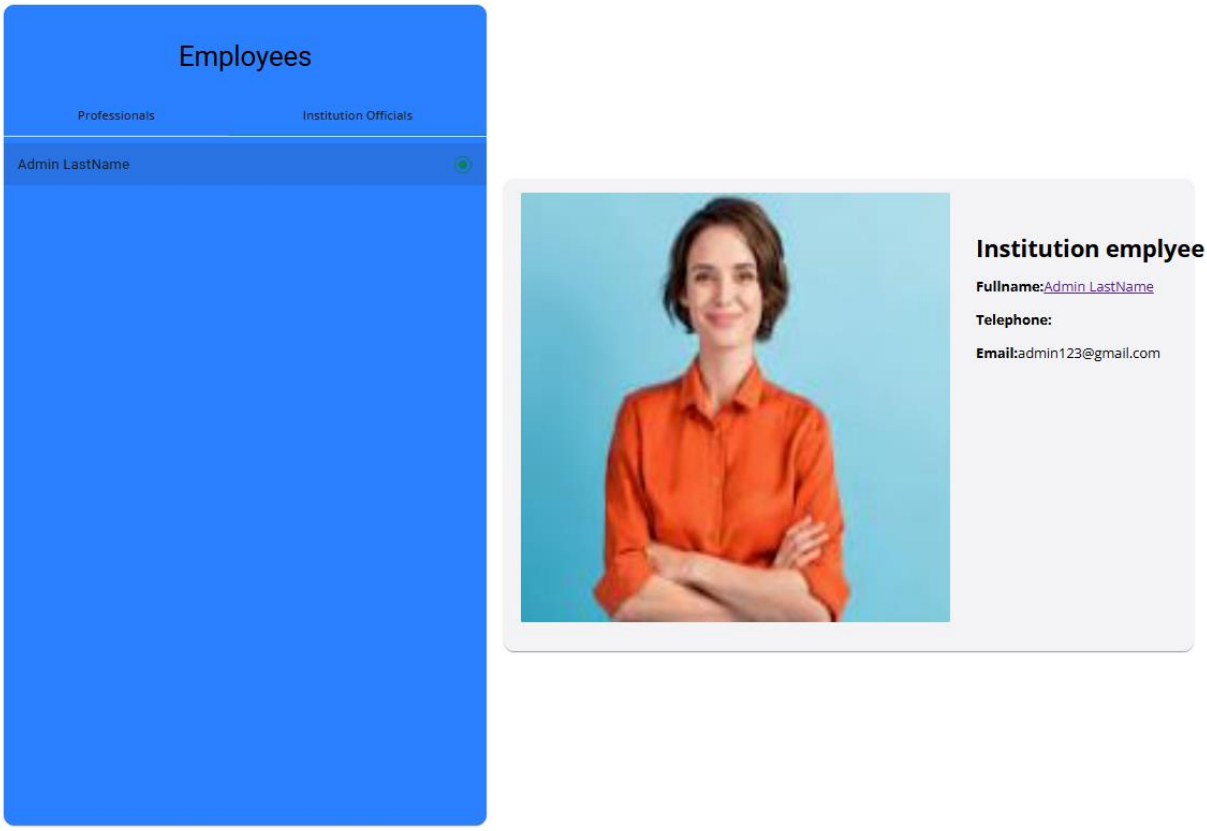
Εικόνα 5.45 - Κάθε διαχειριστής ενός ιδρύματος θα έχει στην σελίδα του "My Posts" όλα τα ζώα του ιδρύματος αυτού.

Επίσης πηγαίνοντας στον λογαριασμό κάποιου χρήστη μπορούμε να δούμε τα επαγγέλματά του (άμα τα έχει βάλει στο site).



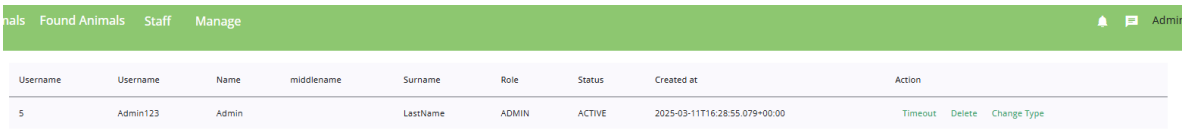
Εικόνα 5.46 - Επιλογή λογαριασμού μας ("Account" επιλογή από το header) με όλα τα επαγγέλματά μας στο site σημειωμένα σε κόκκινο κύκλο.

Επίσης οι επαγγελματίες χρήστες φαίνονται δημόσια αν υπάρχουν στη σελίδα “Professionals”:



Εικόνα 5.47 - Σελίδα "Professionals" στην δεύτερη καρτέλα "Institution Officials" με επιλεγμένο έναν διαχειριστή ιδρύματος.

Επιπρόσθετα , αν ένας χρήστης είναι του ρόλου ADMIN θα του εμφανίζεται στο header μέσα στο “Manage” η επιλογή “Users” όπου εκεί μπορεί να κόψει την επικοινωνία ενός χρήστη μέσω αναρτήσεων ή σχολείων ή να ξεκινήσει (“Timeout”/”Untimeout” λειτουργεία) που είναι αν ο χρήστης κάνει παραβιάσεις στο site (Ψεύτικες αναρτήσεις ή Spam).Επίσης μπορεί να αλλάξει τον ρόλο ενός χρήστη εκτός από ADMIN.



Εικόνα 5.48 - Πίνακας μα όλους τους χρήστες και τις ενέργειες που μπορούν να γίνουν. Επιτρέπεται η πρόσβαση μόνο σε χρήστες ADMIN.

Τέλος άμα πάει ένας χρήστης σε σελίδα που δεν επιτρέπετε (δεν έχει τον κατάλληλο ρόλο) θα του επιστραφεί error 403 από το Spring Boot και μετά με το interceptor της Angular θα τον πάει στην “Access Denied” σελίδα.

Εικόνα 5.49 - Σελίδα στην οποία θα πηγαίνουν όλοι οι χρήστες που δεν επιτρέπεται να μπουν σε μία σελίδα (λόγο του ρόλου τους , error 403)

Άμα πετάξει error 401 , σημαίνει ότι το JWT που είχε ο χρήστης έληξε. Έτσι το interceptor της Angular θα τον πάει στην σελίδα “Login”.

Συμπεράσματα

Σε αυτήν την εργασία υλοποιήθηκε ένας ισότοπος μέσα από τον οποίο μπορούν να έρθουν σε επικοινωνία πολλοί χρήστες μεταξύ τους για τον κοινό σκοπό της εύρεσης χαμένων ζώων. Έτσι πλέον το site μας δίνει την δυνατότητα να φτιάχνουμε αναρτήσεις για χαμένα ζώα , να στέλνουμε μηνύματα και ειδοποιήσεις σε χρήστες , να φέρνουμε σε επαφή επαγγελματίες που θα βοηθήσουν στην εύρεση και θεράπευση των χαμένων και ευρισκόμενων ζώων φέρνοντας έτσι νέες προκλήσεις στο μέσω χρήστη για την υπεράσπιση των ζώων. Αυτό έχει ως αποτέλεσμα να δώσει ευκαιρία σε αρκετό κόσμο να μπορέσει να αγωνιστεί και να προστατεύσει τα χαμένα και ανυπεράσπιστα ζώα , μέσω μίας ιστοσελίδας που τους περιλαμβάνει τις βασικές ανάγκες που χρειάζονται και ασφάλεια των δεδομένων τους. Επίσης πλέον αυξάνεται η πιθανότητα εύρεσης ενός χαμένου ζώου ενώ βελτιώνεται ο συντονισμός μεταξύ ανθρώπων και υπηρεσιών που συμβάλουν σε αυτήν την διαδικασία.

Περιορισμοί

Αυτή τη στιγμή το site τρέχει καλύτερα στην οθόνη ενός υπολογιστή και όχι ενός κινητού. Επίσης αυτή την στιγμή υποστηρίζεται συγκεκριμένος αριθμός χρηστών και όγκος δεδομένων που μπορεί να εξυπηρετηθεί που αν και ικανοποιητικά μεγάλος για μία τυπική εφαρμογή θα πρέπει να επανεξεταστεί σε περίπτωση που ο αριθμός χρηστών μεγαλώσει πολύ.

Μελλοντική εργασία

Σε μελλοντική έκδοση θα μπορούσε να γίνει πρόσθετη εργασία στο site έτσι ώστε να υποστηρίζει μεγαλύτερο αριθμό οθονών όπως κινητών και tablet ή ακόμα να υλοποιηθεί και ένα ξεχωριστό app (εφαρμογή κινητού).Επίσης σε περίπτωση που μπορεί αυτό κριθεί αναγκαίο θα μπορούσε να ελεγχθεί κατά πόσο θα μπορούσε να αυξηθεί ο συνολικός αριθμός των χρηστών που θα μπορούσε να εξυπηρετηθεί (scaling). Εδώ θα μπορούσε να γίνει έλεγχος πέρα του προγραμματιστικού κομματιού και στον τύπο και αριθμό των server που θα εγκατασταθεί η εφαρμογή καθώς και στην ποσότητα των δεδομένων που η υποδομή μπορεί να υποστηρίξει. Τέλος θα μπορούσε να γίνει και μία ξεχωριστή μελέτη για την ασφάλεια όπως κρυπτογράφηση δεδομένων και δημιουργία αντιγράφων ασφαλείας backups.

Πίνακας Ορολογίας

Ξενόγλωσσος Όρος	Ελληνικός Όρος
backend	Προγραμματιστικό επίπεδο διαχείρισης πληροφοριών κρυφό από την χρήστη.
frontend	Προγραμματιστικό επίπεδο παρουσιαστικό που αλληλοεπιδράει άμεσα με τους χρήστες.
database	Βάση δεδομένων , σύστημα αποθήκευσης δεδομένων.
framework	Δομή κατασκευής λογισμικού
Java	Αντικειμενοστραφής Γλώσσα Προγραμματισμού.
NodeJs	Περιβάλλον ανάπτυξης λογισμικού σε JavaScript.
phpMyAdmin	Εργαλείο διαχείρισης βάσεων δεδομένων.
MySQL	Σύστημα Διαχείρισης Βάσεων Δεδομένων
queries	Ερωτήματα , τα οποία γίνονται σε μία βάση δεδομένων για να επιστραφούν από αυτή κάποια δεδομένα.
Gradle	Βοηθητική εφαρμογή για την ανάπτυξη ενός λογισμικού (Java).Περιέχει πολλές λειτουργείες.
token	Ένδειξη
site	Ιστοσελίδα
server	Εξυπηρετητή
session	Συνεδρίαση
column	Στήλη πίνακα βάσης. Περιέχει ένα χαρακτηριστικό αποθηκευτικού χώρου ενός πίνακα.
container (Docker)	Δοχείο, έχει ένα ολοκληρωμένο περιβάλλον εκτέλεσης ενός πακέτου λογισμικού.
Annotation (Spring Boot + Angular)	Σχόλιο , δοχεία πληροφοριών και κώδικα που συμπεριλαμβάνονται στην λειτουργεία μίας εφαρμογής.
API CALLS	Κλήσεις API
foreign key	Ξένο κλειδί , σε μία βάση δεδομένων συνδέει έναν πίνακα με την στήλη ενός άλλου.
onetoone	Ένα προς ένα , σύνεση πινάκων βάσης δεδομένων.

onetomany	Ένα προς πολλά , σύνδεση πινάκων βάσης δεδομένων.
Interface (Spring Boot + Java)	Διεπαφή , περιέχει την δομή μίας συγκεκριμένης λειτουργείας.
Request	Αίτηση , για HTTP πακέτα
Response	Απάντηση , για HTTP πακέτα
Hash	Αλγόριθμος που κωδικοποιεί τον κωδικό πριν τον βάλουμε στην βάση.
interceptor	Αναχαιτιστής , για Angular
Component	Εξάρτημα , για Angular κυρίως

Πίνακας συντμήσεων – αρκτικόλεξων – ακρωνύμιων

API	Application Programming Interface
DTO	Data Transfer Object
DAO	Data Access Object
URL	Uniform Resource Locator
IDE	Integrated Development Environment
MVC	Model–View–Controller
JSON	JavaScript Object Notation
JWT	JSON Web Token
SQL	Structured Query Language
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
PaaS	Platform as a Service
HTML	HyperText Markup Language
CSS	Cascading Style Sheets

Πηγές

- [1] [https://en.wikipedia.org/wiki/Angular_(web_framework)]
- [2] [https://en.wikipedia.org/wiki/Spring_Boot]
- [3] [https://www.cloudflare.com/learning/security/api/what-is-api-call/]
- [4] [https://en.wikipedia.org/wiki/IntelliJ_IDEA]
- [5] [https://en.wikipedia.org/wiki/Visual_Studio_Code]
- [6] [https://en.wikipedia.org/wiki/Postman_(software)]
- [7] [https://en.wikipedia.org/wiki/Docker_(software)]
- [8] [https://stackoverflow.com/questions/10025028/what-is-dao-and-service-layer-exactly-in-spring-framework]
- [9] [https://en.wikipedia.org/wiki/Cross-site_request_forgery#Example]
- [10] [https://www.linkedin.com/pulse/securing-your-api-stack-benefits-replacing-csrf-tokens-mcgowan]

Παραρτήματα

1) Ολόκληρος ο κώδικας (source code) μαζί με τα UML αρχεία: https://github.com/nvas314/petcare
2) Spring Boot : https://spring.io/projects/spring-boot
3) Spring Boot Initializr: https://start.spring.io/
4) Angular : https://angular.dev/
5) Angular Leaflet : https://leafletjs.com/
6) Angular Material : https://material.angular.io/
7) Angular instlation : https://angular.dev/installation
8) Docker : https://www.docker.com/
9) Docker installation for MariaDB: https://mariadb.com/kb/en/installing-and-using-mariadb-via-docker/
10) Postman : https://www.postman.com/
11) Postman installation : https://www.postman.com/downloads/

Βιβλιογραφία

1) Maria Virvou, George A. Tsihrintzis, Efthymios Alepis, Ioanna-Ourania Stathopoulou:
Designing a multi-modal affective knowledge-based user interface: combining empirical studies. JCKBSE 2008: 250-259
2) Efthymios Alepis, Maria Virvou:
Emotional Intelligence in Multimodal Object Oriented User Interfaces. KES IIMSS 2009: 349-359
3) Efthymios Alepis, Maria Virvou:
Object oriented architecture for affective multimodal e-learning interfaces. Intell. Decis. Technol. 4(3): 171-180 (2010)
4) Maria Virvou, Katerina Kabassi, Efthimios Alepis, Achilles Kameas, Christos Pierrakeas, Aspasia Theodosiou:
Empirical study towards the creation of educational user profiles for the students of an open university. IISA 2015: 1-5
5) Efthimios Alepis, Constantinos Patsakis:
The All Seeing Eye: Web to App Intercommunication for Session Fingerprinting in Android. SpaCCS 2017: 93-107
6) Aristea Kontogianni, Efthimios Alepis:
HotSpotCrowd: A crowd sourced based application for sharing data between mobile devices. IISA 2017: 1-6
7) Katerina Kabassi, Maria Virvou, Efthymios Alepis:
Reasoning about users actions in a mobile environment using a combination of HPR with MAUT. IISA 2017: 1-6
8) Efthimios Alepis, Constantinos Patsakis:
Session Fingerprinting in Android via Web-to-App Intercommunication. Secur. Commun. Networks 2018: 7352030:1-7352030:13 (2018)
9) Katerina Kabassi, Efthimios Alepis:
Learning Analytics in Distance and Mobile Learning for Designing Personalised Software. Machine Learning Paradigms 2020: 185-203
10) Christos Troussas, Akrivi Krouska, Efthimios Alepis, Maria Virvou:
Intelligent and adaptive tutoring through a social network for higher education. New Rev. Hypermedia Multim. 26(3-4): 138-167 (2020)
11) Eugenia A. Politou, Efthimios Alepis, Maria Virvou, Constantinos Patsakis:
Privacy and Data Protection Challenges in the Distributed Era. Springer 2022, ISBN 978-3-030-85442-3, pp. 1-185
12) Alexios Nikas, Efthimios Alepis, Constantinos Patsakis:
I know what you streamed last night: On the security and privacy of streaming. Digit. Investig. 25: 78-89 (2018)

13) Maria Virvou, Efthymios Alepis, Christos Troussas:
Error diagnosis in computer-supported collaborative multiple language learning using user classification. JCKBSE 2012: 266-275
14) Panagiotis Pavlakis, Efthymios Alepis, Maria Virvou:
Intelligent Mobile Multimedia Application for the Support of the Elderly. IIH-MSP 2012: 297-300
15) Christos Lambrinidis, Efthymios Alepis:
Computer-aided diagnosis and access on peoples' health status using web technology and mobile devices. JCKBSE 2012: 247-256