ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάπτυξη Διαδικτυακής Εφαρμογής Καταχώρησης και Παρακολούθησης Ραντεβού Νοσοκομείου με Λειτουργίες που χρησιμοποιούν Νευρωνικό Δίκτυο**
Ονοματεπώνυμο Φοιτητή	**Κωνσταντίνος Ράμμος του Εμμανουήλ**
Αριθμός Μητρώου	**ΜΠΣΠ15073**
Κατεύθυνση	**Προηγμένες Τεχνολογίες Ανάπτυξης Λογισμικού (ΠΡΟΤΑΛ)**
Επιβλέπων	**Ευθύμιος Αλέπης, Επίκουρος Καθηγητής**
Βοηθός Επιβλέποντος	**Σπυρίδων Παπαδημητρίου, Υποψήφιος Διδάκτορας**

Ημερομηνία Παράδοσης: **8 Νοεμβρίου 2017**

Η σελίδα αφέθηκε σκόπιμα κενή

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	*(υπογραφή)*	*(υπογραφή)*
Γεώργιος Τσιχριντζής	Μαρία Βίρβου	Ευθύμιος Αλέπης
Καθηγητής	Καθηγήτρια	Επίκουρος Καθηγητής

Η σελίδα αφέθηκε σκόπιμα κενή

ΕΥΧΑΡΙΣΤΙΕΣ

Η εκπόνηση της παρούσας μεταπτυχιακής διατριβής δε θα ήταν δυνατή χωρίς τη συμβολή ανθρώπων, που με βοήθησαν με κάθε δυνατό τρόπο.

Πρωτίστως, θα ήθελα να ευχαριστήσω τους καθηγητές μου, τον κύριο Αλέπη Ευθύμιο, που με εμπιστεύτηκε να προχωρήσουμε μαζί στην εκπόνηση της παρούσας εργασίας, αλλά και τον κύριο Παπαδημητρίου Σπυρίδωνα, για τις πολύτιμες συμβουλές και κατευθύνσεις που μου παρείχε και το χρόνο που μου αφιέρωσε.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τη Νίκη μου για την αμέριστη αγάπη τους, την υποστήριξη και την προτροπή τους να πραγματοποιώ κάθε μου όνειρο.

Η σελίδα αφέθηκε σκόπιμα κενή

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ

`Η παρούσα μεταπτυχιακή διατριβή αφορά στη δημιουργία μίας ολοκληρωμένης, φιλικής προς το χρήστη, διαδικτυακής εφαρμογής για την αναβάθμιση του συστήματος των ραντεβού σε ένα νοσοκομείο, η οποία παρέχει διεπαφή για τον ασθενή, το γιατρό και το διαχειριστή της εφαρμογής.

Από την μία πλευρά, ο ασθενής έχει τη δυνατότητα να κλείσει εύκολα ένα ραντεβού, επιλέγοντας το ιατρείο που χρειάζεται, το γιατρό επιθυμίας του, ημερομηνία και ώρα, καθώς και να δηλώσει τα συμπτώματα που παρουσιάζει. Επίσης, δύναται να δει τα εκκρεμή και τα ολοκληρωμένα ραντεβού του, καθώς και να βαθμολογήσει κάποιο γιατρό.

Από την άλλη πλευρά, ο εκάστοτε γιατρός μπορεί να δει τα ραντεβού του για οποιαδήποτε ημέρα επιλέξει, με όλα τα στοιχεία του ασθενή και του ιατρικού του προβλήματος. Μπορεί να αλλάξει την απεικόνιση στα εν εξελίξει και στα ολοκληρωμένα ραντεβού και να καταχωρήσει τη διάρκεια αυτών. Επίσης, έχει τη δυνατότητα δήλωσης της άδειάς του, ώστε τις ημέρες απουσίας του να μην εμφανίζεται το όνομά του στο σύστημα.

Επιπροσθέτως, έχει αναπτυχθεί διεπαφή και για τον διαχειριστή της εφαρμογής, ο οποίος έχει τη δυνατότητα καταχώρησης, επεξεργασίας και διαγραφής των δεδομένων από τη βάση, αλλά και εποπτείας διάφορων στατιστικών στοιχείων, όπως προκύπτουν από τις καταχωρήσεις σε αυτή.

Τέλος, στην εργασία συμπεριλήφθηκε και ένας αλγόριθμος υλοποίησης ενός Νευρωνικού Δικτύου, μέσω του οποίου προκύπτει μία εκτίμηση της χρονικής διάρκειας του κάθε ραντεβού, βάσει της εμπειρίας που αποκτά ο αλγόριθμος από τη διάρκεια των ήδη ολοκληρωμένων.

Σημειώνεται ότι το «front-end» της εν λόγω εφαρμογής έχει υλοποιηθεί στο framework Angular2, ενώ το «back-end» είναι δομημένο σε REST API’s σε NodeJS Server. Η βάση δεδομένων που χρησιμοποιήθηκε είναι η MongoDB.

ABSTRACT

The present master's dissertation is concerned with the development of a complete, user-friendly, web application for the upgrade of the appointments system of a hospital and provides an interface for the patients, an interface for the doctors and one for the administrator.

On the one hand, the patients can easily make an appointment by selecting the doctor's office they need, the desired doctor, the date, the time and the symptom they exhibit. They can also see their pending and completed appointments and rate a doctor, too.

On the other hand, each doctor can watch their appointments on any chosen day, each one of them comprises of information concerning the patient and the medical issue. They can change the view on the current and completed appointments and additionally, register their duration. Furthermore, they can note their leave in order their name not to be appeared in the system during their days off.

Additionally, an interface for the administrator has been developed, through which, they can add, edit and delete database data, and also watch some statistics, as they are collected from the database.

In conclusion, we include in our work an algorithm for the implementation of a Neural Network, which gives us an estimation of the duration of each appointment, depending on the experience the algorithm acquires from the duration of the already completed appointments.

It is pointed out that the "front-end" of the application has been built with the framework Angular2, while the "back-end" is structured in REST API's in NodeJS Server. The used database is in MongoDB.

ΕΙΣΑΓΩΓΗ

Στις μέρες μας, το Διαδίκτυο έχει γίνει αναπόσπαστο κομμάτι της ζωής μας. Η προσφορά του στην ικανοποίηση διάφορων αναγκών μας είναι τεράστια. Μέσω αυτού, μπορούμε να βρούμε άμεσα πληροφορίες, να ενημερωθούμε, να ψυχαγωγηθούμε, να επικοινωνήσουμε με άλλους χρήστες του.

Πέραν των ανωτέρω, ακόμα μία σημαντική συνεισφορά του Διαδικτύου είναι η δυνατότητα, που μας παρέχει, να φέρουμε άμεσα εις πέρας κάποιες δουλειές μας, οι οποίες διαφορετικά θα απαιτούσαν τη φυσική μας παρουσία σε κάποιο κατάστημα ή την αναμονή μας για αρκετή ώρα στο τηλέφωνο. Χαρακτηριστικά παραδείγματα της εν λόγω συνεισφοράς είναι η αγορά εισιτηρίων για κινηματογράφο, θέατρο κ.τ.λ., με λίγα μόνο κλικ, αποφεύγοντας την ανάγκη να περιμένουμε σε ουρές με πολύ κόσμο. Ακόμα, η διαχείριση των οικονομικών μας μέσω Διαδικτύου, μας αποδεσμεύει από την ανάγκη της παρουσίας μας και αναμονής μας στην τράπεζα. Επίσης, το κλείσιμο ενός ραντεβού σε κάποιο γιατρό δύναται να γίνει πλέον υπόθεση μερικών δευτερολέπτων.

Ας επικεντρωθούμε, λοιπόν, στα ιατρικά ραντεβού. Γιατί να σπαταλάμε χρόνο περιμένοντας στο τηλέφωνο να μας απαντήσει κάποιος/α υπάλληλος; Γιατί να προσπαθούμε να συνεννοηθούμε για μία διαθέσιμη ημερομηνία και ώρα, που ταυτόχρονα βολεύει και εμάς; Πολλές δυσχέρειες μπορούν άμεσα να επιλυθούν μέσω μίας εφαρμογής online κλεισίματος ραντεβού. Από τη μία πλευρά, ο ασθενής θα μπορεί να βρει άμεσα διαθέσιμους γιατρούς, ημερομηνίες και ώρες, που τον βολεύουν, και μέσα σε λίγα δευτερόλεπτα να κλείσει το ραντεβού του. Από την άλλη πλευρά, ο γιατρός θα έχει τη δυνατότητα να παρακολουθεί τα ραντεβού του, απλά και οργανωμένα, χωρίς την ανάγκη να καταχωρεί χειρόγραφα τους ασθενείς του σε τεράστια βιβλία.

Σε αυτό το σημείο, πρέπει να τονιστεί και η σημασία των αλγορίθμων Μηχανικής Μάθησης στις εφαρμογές των Η/Υ. Οι υπόψη αλγόριθμοι έχουν την ικανότητα να «μαθαίνουν» από τα δεδομένα και να αναγνωρίζουν πρότυπα και κανονικότητες μέσα σε αυτά, που μπορεί ο ίδιος ο προγραμματιστής να μην ξέρει εκ των προτέρων. Ένα υποσύνολο των αλγορίθμων Μηχανικής Μάθησης είναι τα Νευρωνικά Δίκτυα, δηλαδή αλγόριθμοι σχεδιασμένοι να μοντελοποιούν τον τρόπο με τον οποίο ο ανθρώπινος εγκέφαλος εκτελεί μία λειτουργία.

Συνεπώς, η συμπερίληψη ενός Νευρωνικού Δικτύου αναβαθμίζει την εφαρμογή μας από μία απλή σε μία «ευφυή» εφαρμογή, προσδίδοντάς της ταυτόχρονα και τις προδιαγραφές για μελλοντικές επεκτάσεις, ώστε να γίνει ακόμα πιο ευφυής.

Συνοψίζοντας, στο πλαίσιο της μεταπτυχιακής μου διατριβής, σκοπός μου ήταν η δημιουργία ενός εύκολου, φιλικού προς το χρήστη και «ευφυούς» λογισμικού για τη διευκόλυνση της λειτουργίας του συστήματος των ραντεβού σε ένα νοσοκομείο, με προοπτικές να αντικαταστήσει άλλες παρεμφερείς, αλλά παρωχημένες, εφαρμογές.

Η παρούσα εργασία έχει δομηθεί ως κάτωθι:

• Στο κεφάλαιο 1, παρουσιάζουμε τρεις (3), παρεμφερείς με την δικιά μας, εφαρμογές, που βρίσκονται ήδη σε κυκλοφορία, και εντοπίζουμε τα θετικά και αρνητικά τους χαρακτηριστικά. Κλείνουμε το κεφάλαιο με μία παρουσίαση των χαρακτηριστικών και λειτουργιών, στα οποία υπερτερεί η προτεινόμενη στην παρούσα διατριβή εφαρμογή.

• Στο κεφάλαιο 2, παρουσιάζουμε τις λειτουργίες για κάθε μία από τις τρεις διεπαφές της εφαρμογής μας, τη διεπαφή του ασθενή, τη διεπαφή του γιατρού και τη διεπαφή του διαχειριστή, καθώς και τα ιδιαίτερα χαρακτηριστικά της. Ακόμα, αναφέρουμε τους κανόνες, που ακολουθήσαμε, ώστε να την κάνουμε φιλική προς το χρήστη και με καλή θέση στη μηχανή αναζήτησης «*Google*». Στη δεύτερη παράγραφο του κεφαλαίου, παρουσιάζουμε το εγχειρίδιο χρήσης της εφαρμογής.

• Στο κεφάλαιο 3, αρχικά παρουσιάζουμε τις γλώσσες προγραμματισμού και τα εργαλεία, που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής μας, καθώς και τη διαδικασία εγκατάστασής τους. Στη συνέχεια, παραθέτουμε τα διαγράμματα UML Περιπτώσεων Χρήσης αυτής με σκοπό την καλύτερη κατανόησή της από τον αναγνώστη. Περιγράφουμε το ρόλο κάθε αρχείου εντός του φακέλου της εφαρμογής, αλλά και αναλύουμε αξιοσημείωτα τμήματα κώδικα. Επιπλέον, κάνουμε μία πλήρη περιγραφή στο Νευρωνικό Δίκτυο, που περιλαμβάνει το θεωρητικό υπόβαθρο, τον τρόπο υλοποίησής του στην εφαρμογή μας, αλλά και τις δοκιμές που εκτελέστηκαν, ώστε να φτάσουμε στην κατάλληλη διαμόρφωσή του. Τέλος, αναφερόμαστε και στην ασφάλεια της εφαρμογής μας από γνωστές διαδικτυακές επιθέσεις.

• Στο κεφάλαιο 4, ανακεφαλαιώνουμε την εργασία μας, παραθέτουμε τα συμπεράσματά μας ύστερα από τη διεξοδική μελέτη στην οποία προβήκαμε και παρέχουμε κάποιες κατευθύνσεις σχετικά με μελλοντικές επεκτάσεις που θα μπορούσε να έχει η εφαρμογή μας.

• Στο παράρτημα, παραθέτουμε τον κώδικα του αρχείου, που υλοποιεί το Νευρωνικό Δίκτυο στην εφαρμογή μας, τον κώδικα HTML και CSS υλοποίησης ενός τρικ, που χρησιμοποιήσαμε, για ίσο ύψος στηλών με περιεχόμενο, που δεν είναι γνωστό εκ των προτέρων και τέλος παρουσιάζουμε μία καταχώρηση από κάθε collection της βάσης για καλύτερη κατανόηση της δομής της.

ΚΕΦΑΛΑΙΟ 1 -

ΑΝΑΣΚΟΠΗΣΗ ΠΕΔΙΟΥ

Η σελίδα αφέθηκε σκόπιμα κενή

ΚΕΦΑΛΑΙΟ 1 – ΑΝΑΣΚΟΠΗΣΗ ΠΕΔΙΟΥ

1.1 Εισαγωγή

Πριν περάσουμε στην ανάλυση της εφαρμογής που δημιουργήσαμε, θα αναφερθούμε σε ορισμένες παρεμφερείς εφαρμογές που κυκλοφορούν ήδη. Οπότε, σε αυτό το κεφάλαιο, παρουσιάζουμε τρεις (3) εφαρμογές, που αφορούν στο κλείσιμο ραντεβού σε γιατρό, μία (1) από το εξωτερικό, την «*DocASAP*» ([18]) και δύο (2) ελληνικές, την «*Doctoranytime*» και την «*NowDoctor*» ([19] και [20], αντίστοιχα). Οι υπόψη εφαρμογές επιλέχθηκαν λόγω της υψηλής θέσης που παρουσιάζονται, στη μηχανή αναζήτησης «*Google*». Ο σκοπός μας είναι να συγκρίνουμε θετικά και αρνητικά χαρακτηριστικά αυτών με τα αντίστοιχα της προτεινόμενης εφαρμογής της παρούσας διατριβής και έπειτα να εξάγουμε χρήσιμα συμπεράσματα για τη χρηστικότητα της εφαρμογής μας, την απλότητά της και την ευφυία της.

Σε αυτό το σημείο, πρέπει να αναφέρουμε ότι υπάρχει μία κύρια διαφορά μεταξύ της προτεινόμενης εφαρμογής και των υπολοίπων, που περιγράφονται σε αυτό το κεφάλαιο. Η εφαρμογή μας αναπτύχθηκε στο πνεύμα να βελτιώσει το σύστημα των ραντεβού ενός συγκεκριμένου νοσοκομείου. Με άλλα λόγια, ο ασθενής καλείται να επιλέξει από τους γιατρούς, που δουλεύουν στο εν λόγω νοσοκομείο, καθώς και τις διαθέσιμες σε αυτό ημερομηνίες και ώρες. Από την άλλη πλευρά, οι υπόλοιπες εφαρμογές είναι απλά μία μηχανή αναζήτησης γιατρών σε οποιαδήποτε περιοχή και όχι ειδικά για κάποιο νοσοκομείο.

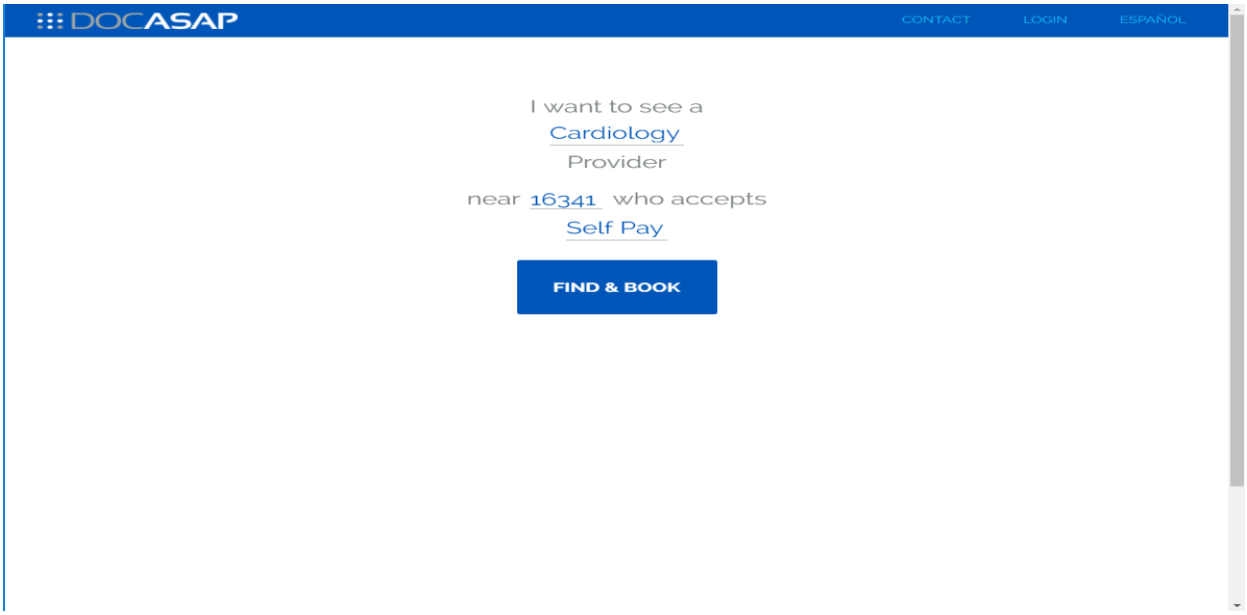
Παρά την προαναφερθείσα διαφορά, η ευρύτερη ιδέα και ο σκοπός των εφαρμογών είναι ο ίδιος, κάτι το οποίο μας δίνει τη δυνατότητα να προβούμε στη σύγκρισή τους. Στις επόμενες υποενότητες, παρουσιάζονται οι εν λόγω εφαρμογές και κατόπιν, εξάγονται χρήσιμα συμπεράσματα.

1.2 Εφαρμογή «*DocASAP*»

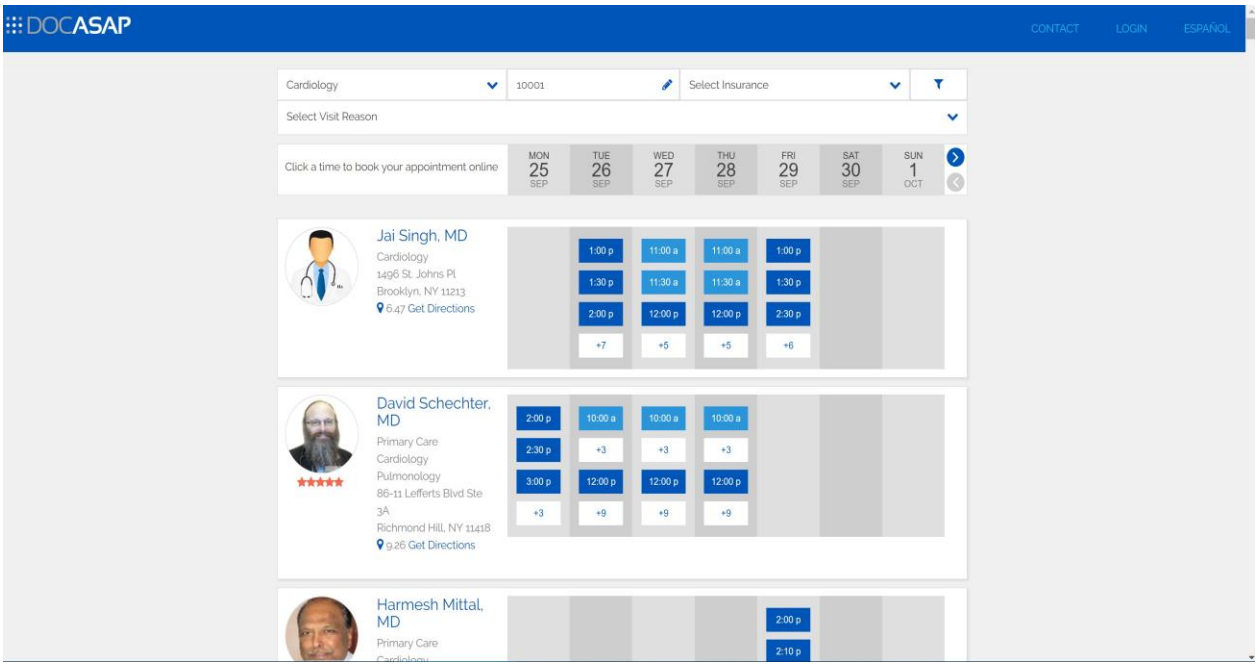
Η εφαρμογή «*DocASAP*» είναι μία αμερικάνικη εφαρμογή για εύρεση ενός γιατρού, από μία μεγάλη λίστα γιατρών όλων των ειδικοτήτων. Στην πρώτη οθόνη της εφαρμογής, ο χρήστης αρκεί να εισάγει την ειδικότητα του γιατρού που ψάχνει, τον ταχυδρομικό κώδικα, για αναζήτηση γιατρού στην αντίστοιχη περιοχή, και την ασφάλειά του. Στη δεύτερη οθόνη, εμφανίζεται μία λίστα με τους διαθέσιμους γιατρούς, ανάλογα τα κριτήρια που τέθηκαν στο πρώτο βήμα, καθώς και επιλογές για ημέρα, ώρα και λόγος της επίσκεψης. Αφού ο χρήστης επιλέξει κατάλληλα, η εφαρμογή τον οδηγεί σε μία τρίτη οθόνη επιβεβαίωσης των προηγούμενων επιλογών, ώστε τελικά να καταφέρει να κλείσει το ραντεβού του.

Συνοπτικά, η εν λόγω εφαρμογή παρουσιάζει θετικά και αρνητικά χαρακτηριστικά σε σχέση με την προτεινόμενη της παρούσας διατριβής. Από τη μία, είναι ωραία αισθητικά και η πρώτη οθόνη της είναι αρκετά φιλική προς τον χρήστη, χωρίς να τον καταιγίζει με πολλές λεπτομέρειες, πεδία επιλογών, διαφημίσεις κ.τ.λ. Επιπλέον, ο ασθενής μπορεί να δει ένα συνοπτικό βιογραφικό των γιατρών, γεγονός που θα τον βοηθήσει στην επιλογή του. Από την άλλη, για να κλείσει ένας χρήστης ραντεβού, πρέπει να

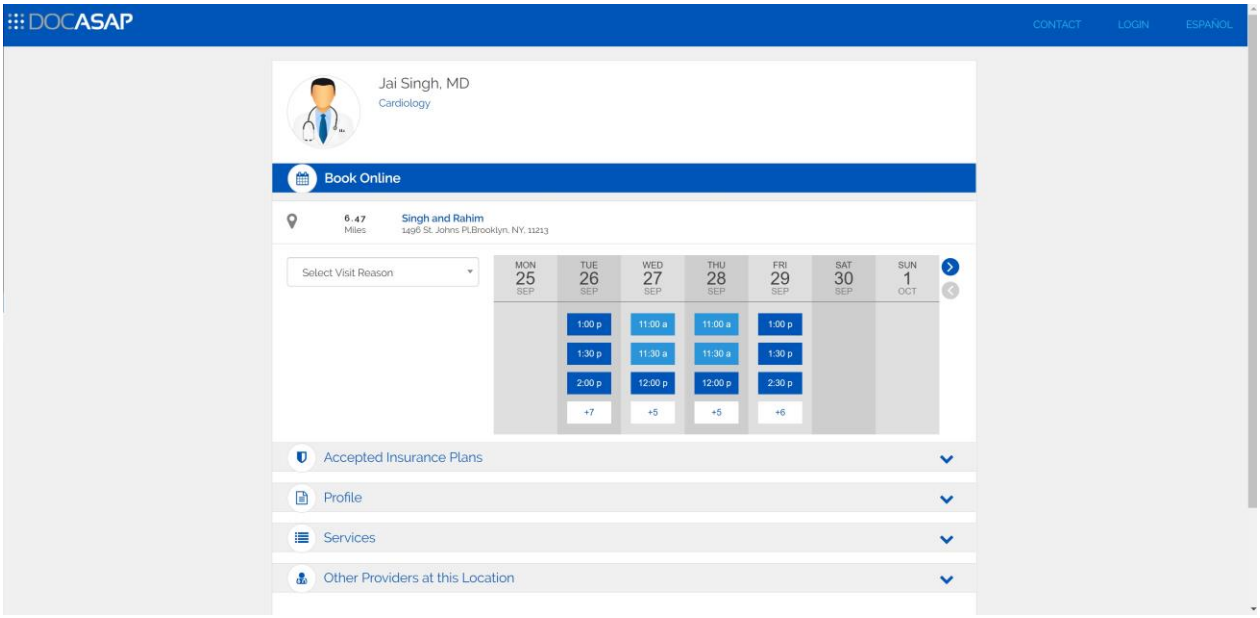
περάσει από τρεις (3) οθόνες. Τέλος, κατά την επιλογή της ημερομηνίας, είναι εμφανής μόνο μία βδομάδα, κάτι το οποίο δε θεωρείται φιλικό προς το χρήστη και τον δυσκολεύει να έχει μία καλύτερη οπτική ολόκληρου του τρέχοντος μήνα ή των επόμενων.



Εικόνα 1.1: *Εφαρμογή «DocASAP», πρώτη οθόνη.*



Εικόνα 1.2: *Εφαρμογή «DocASAP», δεύτερη οθόνη.*

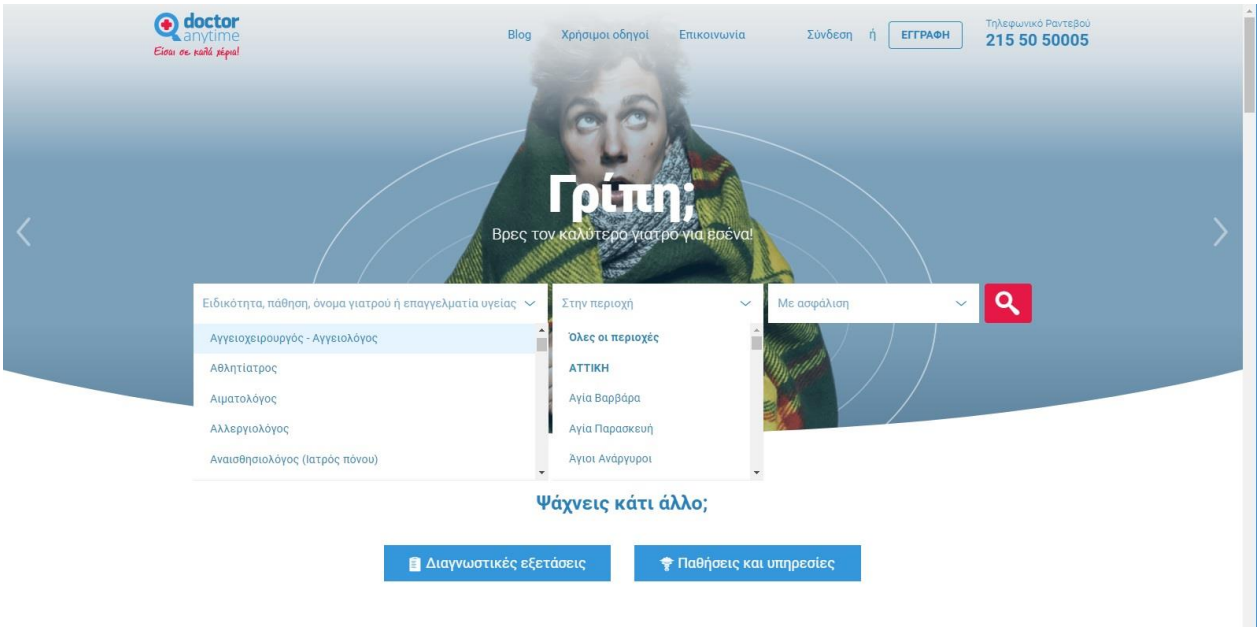


Εικόνα 1.3: *Εφαρμογή «DocASAP», τρίτη οθόνη.*

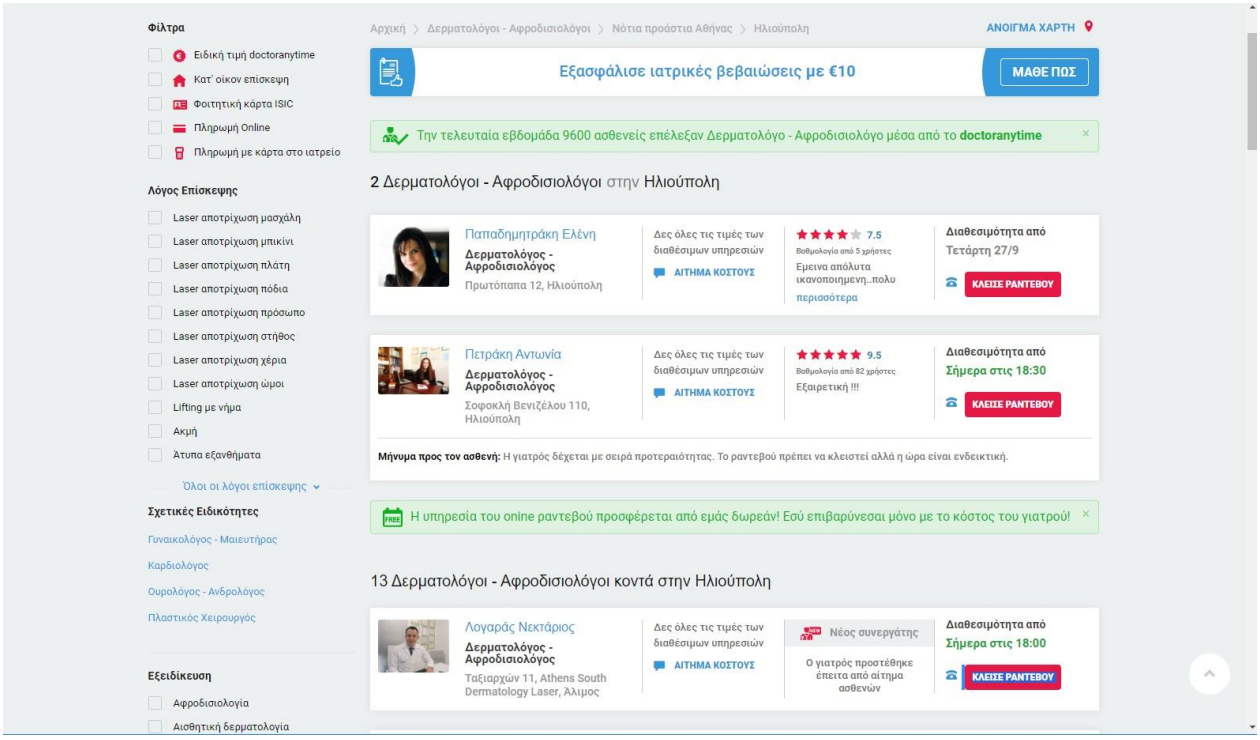
1.3 Εφαρμογή «Doctoranytime»

Η εφαρμογή «*Doctoranytime*» είναι μία ελληνική εφαρμογή για την εύρεση ενός γιατρού και εν συνεχεία για το κλείσιμο ραντεβού. Ομοίως με την ανωτέρω εφαρμογή, στην πρώτη οθόνη επιλέγεται η ειδικότητα του γιατρού και η επιθυμητή περιοχή έδρας του, όπως επίσης και η ασφάλιση. Στη δεύτερη οθόνη, εμφανίζεται η λίστα των αποτελεσμάτων. Ο χρήστης επιλέγει γιατρό και κατόπιν μεταβαίνει σε μία τρίτη οθόνη για να επιλέξει ημέρα και ώρα για το ραντεβού του. Στη συνέχεια, στην τέταρτη οθόνη, επιβεβαιώνει τις επιλογές του και κλείνει τελικά το ραντεβού.

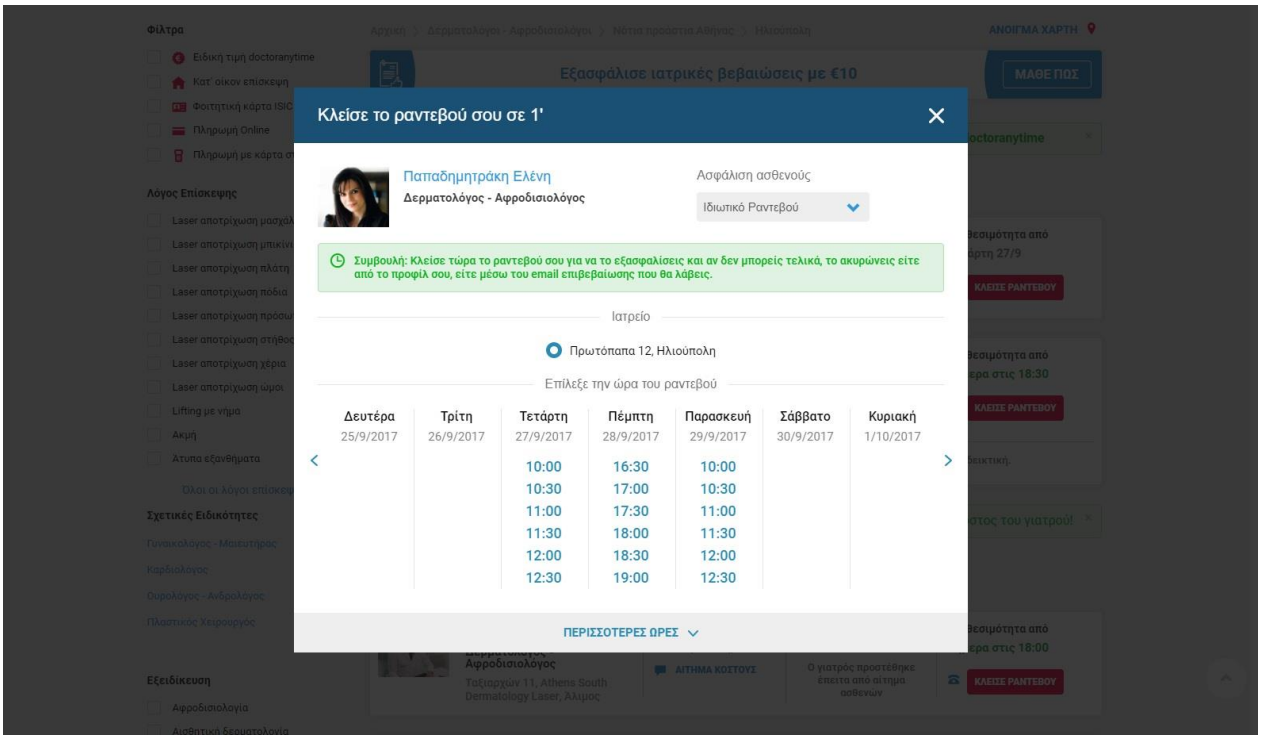
Η υπόψη εφαρμογή παρουσιάζει θετικά και αρνητικά στοιχεία. Ενδεικτικά αναφέρουμε ότι διαθέτει «*rating*» για τους γιατρούς, γεγονός που θα κατατοπίσει καλύτερα τον ασθενή στην επιλογή του, αλλά ταυτόχρονα θα δώσει και κίνητρο στο γιατρό να βελτιωθεί και να αυξήσει τη βαθμολογία του. Επίσης, υπάρχει η δυνατότητα αναζήτησης ειδικότητας, ονόματος γιατρού ή πάθησης από την πρώτη κιόλας οθόνη, βοηθώντας το χρήστη να βρει γρήγορα αυτό που ψάχνει. Τα προφίλ των γιατρών περιλαμβάνουν συνοπτικά τις σπουδές τους, διευκολύνοντας το χρήστη στην επιλογή του, την τοποθεσία του ιατρείου τους στο χάρτη, καθώς και φωτογραφίες του ιατρείου. Από την άλλη πλευρά, βέβαια, η συγκεκριμένη εφαρμογή αναγκάζει τον ασθενή να περάσει από τέσσερις (4) οθόνες μέχρι να καταφέρει να κλείσει το ραντεβού του. Ακόμα, θα μπορούσαμε να πούμε ότι η επιλογή της ημερομηνίας και ώρας του ραντεβού δεν είναι πολύ φιλική προς το χρήστη. Εμφανίζεται στην οθόνη μικρό πλήθος ημερών προς επιλογή, στερώντας από αυτόν τη δυνατότητα να έχει μία καλύτερη εικόνα ολόκληρου του μήνα, ώστε να επιλέξει κατάλληλα.



Εικόνα 1.4: *Εφαρμογή «Doctoranytime», πρώτη οθόνη.*



Εικόνα 1.5: *Εφαρμογή «Doctoranytime», δεύτερη οθόνη.*



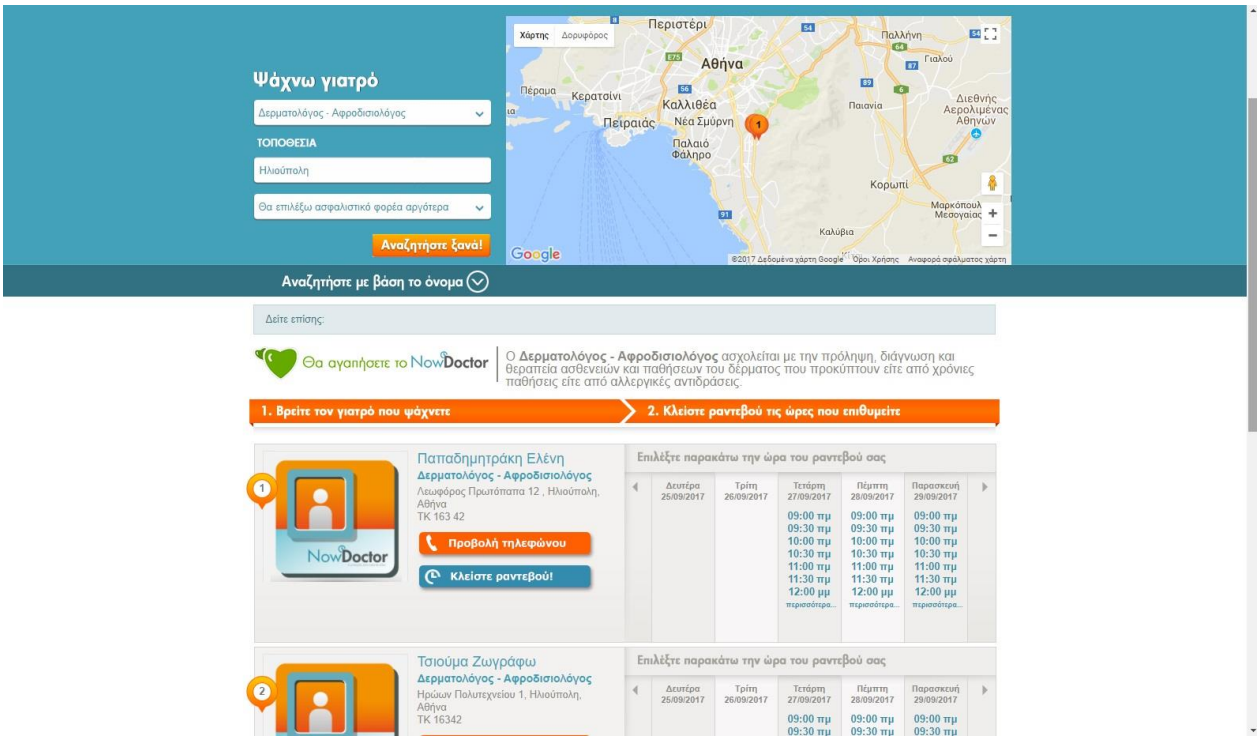
Εικόνα 1.6: *Εφαρμογή «Doctoranytime», τρίτη οθόνη.*

1.4 Εφαρμογή «*NowDoctor*»

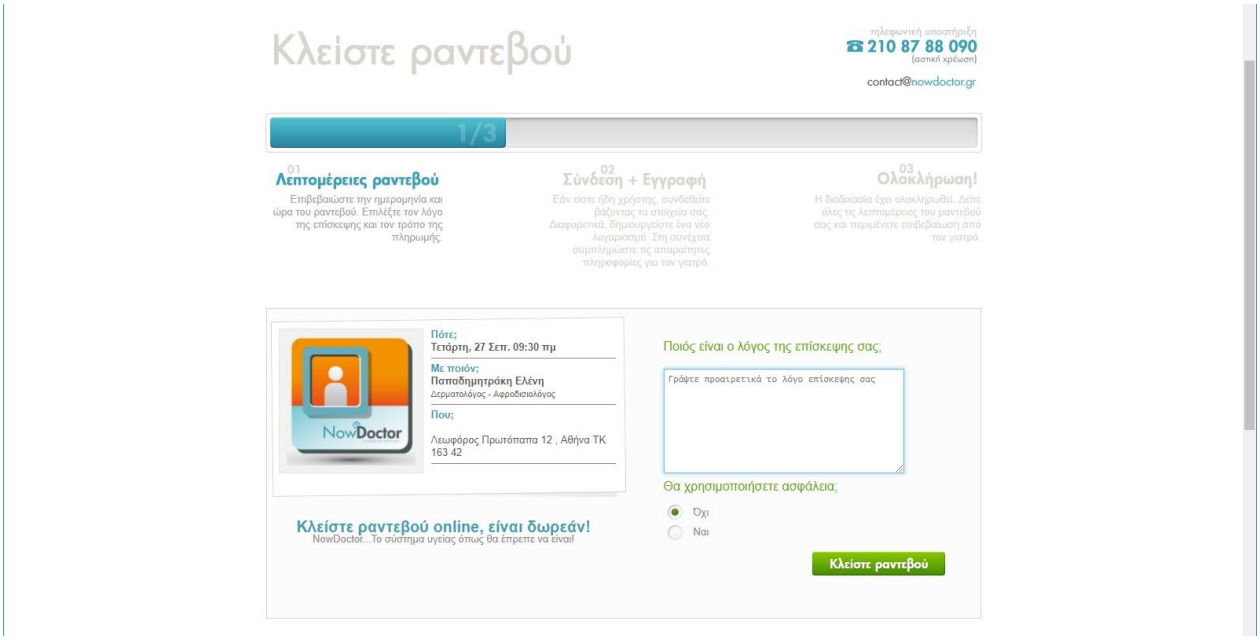
Η τρίτη εφαρμογή, που επιλέξαμε να παρουσιάσουμε στην παρούσα ενότητα, είναι η ελληνική εφαρμογή «*NowDoctor*». Η δομή της είναι παρόμοια με τις προαναφερθείσες, με μικρές διαφοροποιήσεις. Αρχικά, στην πρώτη οθόνη, ο ασθενής καλείται να επιλέξει την ειδικότητα γιατρού που επιθυμεί, την περιοχή αναζήτησης και την ασφάλισή του. Στην επόμενη οθόνη, εμφανίζονται τα αποτελέσματα αναζήτησης σε λίστα, αλλά και με απεικόνιση πάνω σε χάρτη. Κάθε αποτέλεσμα, που αντιστοιχεί σε ένα γιατρό, περιλαμβάνει τις διαθέσιμες ώρες για ραντεβού για τις πέντε (5) επόμενες ημέρες. Ο χρήστης επιλέγει, λοιπόν, γιατρό, ημέρα και ώρα και κατόπιν μεταβαίνει στην τρίτη οθόνη, όπου επιβεβαιώνει τις επιλογές του, συμπληρώνει σε ελεύθερο κείμενο το λόγο επίσκεψής του και κλείνει τελικά το ραντεβού του.

Το θετικό στοιχείο της εν λόγω εφαρμογής είναι η δυνατότητα επιλογής κάποιου γιατρού πάνω στο χάρτη. Αντιθέτως, η εφαρμογή παρουσιάζει κάποια αρνητικά στοιχεία, όπως η απουσία δυνατότητας αξιολόγησης των γιατρών και άρα η μη-διευκόλυνση του χρήστη στην επιλογή του, αλλά και το πέρασμα από τρεις (3) οθόνες, έως ότου τελικά ο χρήστης να καταφέρει να κλείσει το ραντεβού. Ακόμα, η εμφάνιση των διαθέσιμων ημερών και ωρών για όλα τα αποτελέσματα της αναζήτησης δημιουργεί ένα καταιγισμό πληροφοριών, που αποπροσανατολίζουν το χρήστη.

Εικόνα 1.7: *Εφαρμογή «NowDoctor», πρώτη οθόνη.*



Εικόνα 1.8: *Εφαρμογή «NowDoctor», δεύτερη οθόνη.*



Εικόνα 1.9: *Εφαρμογή «NowDoctor», τρίτη οθόνη.*

1.5 Συμπεράσματα

Έχοντας ολοκληρώσει την περιγραφή των παρόμοιων εφαρμογών, που επιλέξαμε να παρουσιάσουμε, που βρίσκονται ήδη σε κυκλοφορία και την εξέταση των θετικών και αρνητικών τους χαρακτηριστικών, είμαστε σε θέση να εξάγουμε κάποια συμπεράσματα σχετικά με την προτεινόμενη, στην παρούσα μεταπτυχιακή διατριβή, εφαρμογή, ως κάτωθι:

α. Είναι μία ολοκληρωμένη διαδικτυακή εφαρμογή, δηλαδή δύναται να υποστηρίξει πλήρως το σύστημα των ραντεβού σε ένα νοσοκομείο. Δεν περιλαμβάνει απλά τη διεπαφή για τους ασθενείς, μέσα από την οποία κλείνουν τα ραντεβού τους, αλλά περιλαμβάνει ακόμα και διεπαφή για τους γιατρούς, οι οποίοι μπορούν μέσα από αυτή να παρακολουθούν το καθημερινό τους πρόγραμμα ραντεβού. Επιπλέον, έχει αναπτυχθεί και διεπαφή για το διαχειριστή του συστήματος, ώστε να προσθέτει, επεξεργάζεται και διαγράφει δεδομένα από τη βάση, να παρακολουθεί στατιστικά και να διαχειρίζεται το Νευρωνικό Δίκτυο, που περιλαμβάνει η εφαρμογή.

β. Αντιθέτως με τις εφαρμογές που αναφέρθηκαν στις προηγούμενες υποενότητες, η εφαρμογή μας είναι «ευφυής», δηλαδή περιλαμβάνει έναν αλγόριθμο υλοποίησης Νευρωνικού Δικτύου, με δυνατότητα εκπαίδευσης από τα δεδομένα της βάσης και η χρήση του οδηγεί σε εξαγωγή χρήσιμων πληροφοριών.

γ. Δίνει στους χρήστες τη δυνατότητα βαθμολόγησης των γιατρών του νοσοκομείου. Παρόλα αυτά, τη βαθμολογία μπορεί να τη δει μόνο ο διαχειριστής, ώστε να παρακολουθεί την ικανότητα κάθε γιατρού και την απήχησή του. Στους ασθενείς δεν έχει δοθεί η δυνατότητα να βλέπουν βαθμολογίες, διότι

οι γιατροί με τις υψηλές βαθμολογίες θα προτιμώνται και θα γέμιζαν με ραντεβού, ενώ οι γιατροί με χαμηλές βαθμολογίες όχι.

δ. Το κλείσιμο των ραντεβού είναι πολύ απλό και φιλικό προς το χρήστη, με τα απαιτούμενα πεδία προς συμπλήρωση να εμφανίζονται ένα-ένα, και η όλη διαδικασία να ολοκληρώνεται σε μία μόνο οθόνη.

ε. Δίνεται στο χρήστη η δυνατότητα γρήγορης αναζήτησης των διαθέσιμων ιατρείων, ώστε να μην χρειάζεται να ψάχνει μέσα σε μια μεγάλη λίστα.

ΚΕΦΑΛΑΙΟ 2 –

ΠΑΡΟΥΣΙΑΣΗ ΚΑΙ ΧΡΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Η σελίδα αφέθηκε σκόπιμα κενή

ΚΕΦΑΛΑΙΟ 2 – ΠΑΡΟΥΣΙΑΣΗ ΚΑΙ ΧΡΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

2.1 Παρουσίαση της Εφαρμογής

Σε αυτή την ενότητα, θα ξεκινήσουμε την ανάλυση της δικής μας εφαρμογή, παρουσιάζοντας τις λειτουργίες της και τα ιδιαίτερα χαρακτηριστικά της. Η εφαρμογή μας έχει ως σκοπό την αναβάθμιση της του συστήματος των ραντεβού σε ένα νοσοκομείο, διότι, από τη μία, παρέχει ένα πολύ εύκολο και φιλικό προς το χρήστη διαδικτυακό περιβάλλον για να μπορεί να κλείσει άμεσα ένα ραντεβού, και από την άλλη, παρέχει ένα ωραίο και πρακτικό περιβάλλον για τους γιατρούς του νοσοκομείου για την καθημερινή παρακολούθηση των ραντεβού τους.

Αναλυτικά, η εφαρμογή μας περιλαμβάνει τρεις (3) διεπαφές, τη διεπαφή του ασθενή, τη διεπαφή του γιατρού και τη διεπαφή του διαχειριστή. Κάθε κατηγορία χρήστη για να εισέλθει στη διεπαφή του πρέπει πρώτα να συνδεθεί. Ο συνδεδεμένος, λοιπόν, ασθενής δύναται να κλείσει άμεσα ένα ραντεβού, χωρίς περιττές ενέργειες και πολλά κλικ, επιλέγοντας, αρχικά, το ιατρείο που επιθυμεί, είτε μέσα από τη λίστα ιατρείων, που παρέχει η εφαρμογή, είτε γράφοντάς το στο πεδίο αναζήτησης. Στη συνέχεια, καταχωρεί τα συμπτώματά του, είτε επιλέγοντας ένα ή περισσότερα από την εμφανιζόμενη λίστα, είτε περιγράφοντάς τα ο ίδιος σε ελεύθερο κείμενο. Έπειτα, πάνω σε ένα ημερολόγιο δικής μας κατασκευής, ο χρήστης καλείται να επιλέξει την επιθυμητή ημερομηνία για το ραντεβού του. Αφού πραγματοποιηθούν οι προαναφερθείσες επιλογές, η εφαρμογή εμφανίζει μία λίστα με τους παρόντες την επιλεχθείσα ημέρα γιατρούς. Αφού ο χρήστης επιλέξει ένα γιατρό, η εφαρμογή εμφανίζει όλες τις διαθέσιμες ώρες για ραντεβού, έχοντας κόκκινες αυτές που έχουν ήδη δεσμευτεί από κάποιον άλλον ασθενή. Κατόπιν, το ραντεβού, πατώντας ο χρήστης το αντίστοιχο πράσινο κουμπί, καταχωρείται στη βάση και εμφανίζεται ένα μήνυμα επιτυχίας της καταχώρησης με όλες τις πληροφορίες του ραντεβού, ώστε ο χρήστης να τις σημειώσει, αν το επιθυμεί. Επιπρόσθετα, ο ασθενής έχει τη δυνατότητα να δει όλα τα εκκρεμή ραντεβού του, αλλά και να διαγράψει κάποιο από αυτά. Επίσης, μπορεί να δει όλα τα παρελθοντικά ραντεβού που έχει κλείσει, έχοντας ακόμα τη δυνατότητα να βαθμολογήσει με ένα (1) έως πέντε (5) αστέρια το γιατρό που τον ανέλαβε. Τέλος, έχει τη δυνατότητα να επεξεργαστεί τις πληροφορίες του προφίλ του. Κάθε ασθενής, για να γίνει χρήστης της εφαρμογής, πρέπει να κάνει εγγραφή ο ίδιος μέσα από τη φόρμα που παρέχεται από το πεδίο «Εγγραφή» στην οθόνη του login.

Από την άλλη πλευρά, η εγγραφή των γιατρών διαχειρίζεται διαφορετικά. Ο διαχειριστής της εφαρμογής είναι αυτός που περνάει στο σύστημα τους νέους γιατρούς, παρέχοντάς τους ένα username και ένα password. Ο εκάστοτε γιατρός έχει τη δυνατότητα να αλλάξει τον κωδικό του μετά την πρώτη σύνδεση. Στην αρχική σελίδα της διεπαφής των γιατρών, καθένας από αυτούς βλέπει τα ραντεβού του για τη σημερινή ημέρα ή κάποια προηγούμενη ή επόμενη, αν το επιλέξει, απεικονιζόμενα ως κουτάκια, ταξινομημένα κατά ώρα. Σε κάθε ένα από αυτά φαίνεται το όνομα, η ημερομηνία γέννησης και στοιχεία επικοινωνίας του ασθενή, αλλά και τα συμπτώματά του. Ο γιατρός δύναται να πατήσει το κουμπί

εκκίνησης, ώστε να είναι ευδιάκριτο το τρέχον ραντεβού, αφού αλλάζει εμφάνιση στο αντίστοιχο κουτάκι, αλλά και να καταχωρηθεί η ώρα που ξεκίνησε. Όταν τελειώσει το ραντεβού, ο γιατρός μπορεί να πατήσει το κουμπί της ολοκλήρωσης, με αποτέλεσμα να υπολογιστεί και να καταχωρηθεί στη βάση η διάρκειά του, αλλά και να αλλάξει το χρώμα του, παρέχοντας ξεκάθαρη εικόνα ποια ραντεβού είναι ολοκληρωμένα και ποια όχι. Τέλος, ο γιατρός μπορεί να καταχωρήσει και την άδειά του, εύκολα και γρήγορα με μόνο τρία (3) κλικ, επιλέγοντας πάνω σε ένα ημερολόγιο την αρχική και την τελική ημέρα άδειας και καταχώρηση.

Επιπλέον, έχει κατασκευαστεί φιλική διεπαφή και για το διαχειριστή του συστήματος, ώστε και αυτός να μπορεί εύκολα και γρήγορα να εκτελεί τις ενέργειές αρμοδιότητάς του. Η κύρια καρτέλα του αφορά στην προσθήκη, επεξεργασία και διαγραφή των ιατρείων και των γιατρών της βάσης. Μέσα σε μία σελίδα, ο διαχειριστής δύναται να εκτελέσει όλες τις παραπάνω εργασίες, άμεσα και εύκολα, χωρίς να μπερδεύεται ή να ψάχνει τι πρέπει να πατήσει μετά. Ακόμα, η διεπαφή του διαχειριστή περιλαμβάνει μία καρτέλα με κουμπί για την εκκίνηση της εκπαίδευσης του Νευρωνικού Δικτύου, που εμπεριέχει η εφαρμογή μας, και επί αυτού θα γίνει πλήρης ανάλυση σε επόμενη ενότητα. Τέλος, ο διαχειριστής διαθέτει δύο ακόμα καρτέλες, στις οποίες όμως δεν μπορεί να κάνει κάποια ενέργεια, αλλά μόνο να δει τα στατιστικά εμφάνισης όλων των καταχωρημένων στη βάση συμπτωμάτων και τις βαθμολογίες όλων των γιατρών.

Σε αυτό το σημείο, πρέπει να επισημάνουμε ότι η εφαρμογή μας, σε όλη της την έκταση, είναι πλήρως «*Responsive*». Έχουμε χρησιμοποιήσει τη βιβλιοθήκη «*Bootstrap*». Κάθε οθόνη της εφαρμογής έχει προσαρμοστεί και ελεγχθεί σε όλα τα είδη οθονών, από πολύ μικρές (κινητά) έως και πολύ μεγάλες (οθόνες Η/Υ). Η λογική που χρησιμοποιήθηκε κατά την ανάπτυξη είναι η «*mobile first*», δηλαδή φτιάξαμε τις σελίδες μας προσαρμοσμένες στις οθόνες των κινητών και μετά, με τη συμπερίληψη «*media queries*», τις προσαρμόσαμε κατάλληλα και για μεγαλύτερες οθόνες.

Κατά τα στάδια σχεδίασης και ανάπτυξης της εφαρμογής μας ακολουθήθηκαν κάποιοι βασικοί κανόνες, ώστε αυτή να καταστεί φιλική προς το χρήστη και εμφανίσιμη. Ενδεικτικά, αναφέρουμε:

- Για όλες τις καρτέλες της εφαρμογής, έγινε προσπάθεια να είναι πολύ απλές και χωρίς περισπασμούς. Αρχικά, απεικονίζονται τα απολύτως απαραίτητα τμήματα της σελίδας και στη συνέχεια εμφανίζονται επιπλέον πληροφορίες ή πεδία, ανάλογα με τις ενέργειες του χρήστη.

- Οι γραμμές κύλισης δε θεωρούνται φιλικές προς το χρήστη, οπότε στην εφαρμογή μας αποφύγαμε τη χρήση οποιασδήποτε οριζόντιας γραμμής κύλισης και σχετικά με τις κάθετες, χρησιμοποιήθηκαν στα σημεία που ήταν απολύτως απαραίτητο.

- Κάθε μενού, καρτέλα, τίτλος στην εφαρμογή μας συνοδεύεται από ένα ευρέως γνωστό και κατανοητό εικονίδιο, ώστε να βελτιώνεται η εμπειρία του χρήστη και φυσικά να μην μπερδεύεται.

- Τα πολυχρησιμοποιούμενα κουμπιά είναι σε κατάλληλη θέση, ώστε να αποφεύγονται μακρινές και χρονοβόρες κινήσεις του ποντικιού.

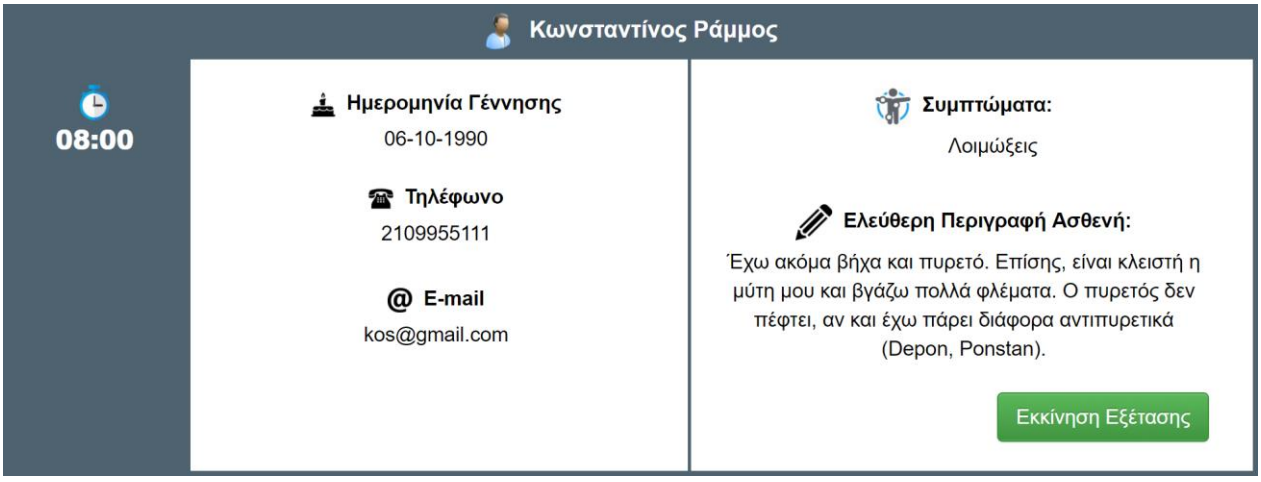
- Έχουμε συμπεριλάβει animation, βελτιώνοντας την εμπειρία του χρήστη και κάνοντας την εφαρμογή μας πιο ενδιαφέρουσα.

- Τα κουμπιά για αντίθετες ενέργειες (πχ Επιβεβαίωση και Ακύρωση) είναι πολύ εμφανή, μεγάλα, με επαρκές κενό ανάμεσά τους και φυσικά με πράσινο και κόκκινο χρώμα, ώστε να μην εκτελεστεί κατά λάθος κάποια μη επιθυμητή ενέργεια.

- Πριν την εκτέλεση κάποιας διαγραφής δεδομένων, εμφανίζεται μήνυμα επιβεβαίωσης, ώστε να αποφευχθούν ακούσιες ενέργειες.

- Έχουν συμπεριληφθεί στην εφαρμογή μας πολλά μηνύματα λάθους, εκφρασμένα σωστά και περιεκτικά, που κατατοπίζουν ακριβώς το χρήστη τι πήγε στραβά.

Αναφορικά με το UI της εφαρμογής μας, υλοποιήθηκε ένα τρικ με κώδικα HTML και CSS, ώστε να πετύχουμε τρεις (3) στήλες, των οποίων το περιεχόμενο δεν είναι γνωστό εκ των προτέρων, να έχουν το ίδιο ύψος. Στην εικόνα 2.1, βλέπουμε ένα ραντεβού με τις πληροφορίες του, όπως αυτό εμφανίζεται στη διεπαφή του γιατρού. Η πρώτη στήλη απεικονίζει την ώρα του ραντεβού, η δεύτερη τις πληροφορίες του ασθενή και η τρίτη τα συμπτώματά του. Το κουτί του ραντεβού είναι ένα εξωτερικό «*container*», που περιλαμβάνει μέσα του τρία (3) άλλα «*containers*», ένα για κάθε στήλη. Το ύψος του εξωτερικού *container* θα είναι πάντα ίσο με το ύψος της μεγαλύτερης στήλης. Η επίτευξη του ανωτέρω στόχου δεν ήταν απλή, αλλά απαιτούσε τη χρήση ενός τρικ. Ο κώδικας παρουσιάζεται στις εικόνες 2 και 3 του Παραρτήματος.



Εικόνα 2.1: *Επίτευξη ίσου ύψους τριών (3) στηλών, όταν δεν ξέρουμε εξ' αρχής το περιεχόμενό τους.*

Όπως προαναφέρθηκε, στην εφαρμογή μας χρησιμοποιείται και ένα Νευρωνικό Δίκτυο. Η εξήγηση του τι ακριβώς είναι και πώς υλοποιείται θα δοθεί σε ξεχωριστή ενότητα. Στο πλαίσιο της παρούσας ενότητας, απλά θα αναφέρουμε ότι, σε γενικές γραμμές, ένα ΝΔ είναι ένας αλγόριθμος, ο οποίος, αφού δεχτεί ένα σύνολο δεδομένων εκπαίδευσης, δηλαδή κάποιες εισόδους με τις αντιστοιχούμενες εξόδους

τους, μπορεί να προσαρμοστεί κατάλληλα μόνος του, χωρίς να έχει προγραμματιστεί εξ' αρχής ρητά, ώστε να βγάζει την αντίστοιχη έξοδο για οποιαδήποτε άλλη είσοδο.

Η χρήση ενός Νευρωνικού Δικτύου στην εφαρμογή μας την κάνει να ξεχωρίζει από τις υπόλοιπες της ίδιας κατηγορίας και επίσης της ανοίγει το δρόμο για πάρα πολλές μελλοντικές επεκτάσεις. Στην παρούσα φάση, το ΝΔ μας εκπαιδεύεται κατόπιν εντολής του διαχειριστή της εφαρμογής, έχοντας ως σύνολο δεδομένων εκπαίδευσης συγκεκριμένες πληροφορίες από τα ήδη ολοκληρωμένα ραντεβού. Πιο συγκεκριμένα, το υπό εκπαίδευση ΝΔ τροφοδοτείται με εισόδους το ιατρείο και το σύμπτωμα, που δήλωσε κάποιος ασθενής σε ένα ραντεβού, και με έξοδο το χρόνο που διήρκησε το εν λόγω ραντεβού, όπως αυτός καταγράφηκε από το γιατρό. Από την άλλη, η χρήση του ΝΔ γίνεται μέσα στη διεπαφή του ασθενή, καθώς αυτός κλείνει ένα νέο ραντεβού. Αφού επιλέξει ιατρείο και σύμπτωμα, η εφαρμογή ενεργοποιεί το ΝΔ, ώστε αυτό να υπολογίσει την εκτιμώμενη διάρκεια του ραντεβού και να την εμφανίσει στο χρήστη. Προκύπτει λοιπόν ότι η διεπαφή του ασθενή πρέπει να έχει συνεχώς πρόσβαση στο ήδη εκπαιδευμένο ΝΔ. Αυτός είναι και ο λόγος που το αποθηκεύουμε στη βάση δεδομένων.

Επιπλέον, δε μπορούμε να παραλείψουμε να αναφερθούμε στον τρόπο αυθεντικοποίησης κάθε χρήστη της εφαρμογής. Κάθε κατηγορία χρήστη έχει μία δική της οθόνη login, με links που παραπέμπουν και στις υπόλοιπες οθόνες. Όταν κάποιος χρήστης συνδεθεί επιτυχώς (ασθενής, γιατρός, διαχειριστής), ο Server δημιουργεί ένα token, μοναδικό για το συγκεκριμένο χρήστη, και του το στέλνει. Ο εν λόγω χρήστης, σε κάθε request από εκείνη τη στιγμή και μετά, πρέπει να το στείλει πίσω στον Server, ώστε αυτός να τον αυθεντικοποιήσει. Μόνο στην περίπτωση επιτυχούς αυθεντικοποίησης θα εκτελεστεί το request, διαφορετικά θα επιστραφεί ένα μήνυμα σφάλματος εξουσιοδότησης. Με αυτό τον τρόπο, αυξάνεται η ασφάλεια της εφαρμογής μας απέναντι σε επιθέσεις κακόβουλων χρηστών. Επίσης, η εφαρμογή μας έχει υλοποιηθεί έτσι, ώστε ένας συνδεδεμένος χρήστης να προσπερνάει την οθόνη του login και να μεταβαίνει στις εσωτερικές σελίδες της, ενώ ένας μη-συνδεδεμένος χρήστης, αν προσπαθήσει να μεταβεί μέσω του URL κατευθείαν σε κάποια εσωτερική σελίδα, η εφαρμογή τον ανακατευθύνει στο login.

Τέλος, τονίζουμε και την αξία του «*Google Search Engine Optimization Starter Guide*» ([17]), του οποίου τους κανόνες δε θα μπορούσαμε να μην συμπεριλάβουμε στην εφαρμογή μας. Ο εν λόγω οδηγός είναι ένα πολύ χρήσιμο εγχειρίδιο της *Google*, ώστε να μπορέσουμε να βελτιώσουμε τη θέση της ιστοσελίδας μας στα αποτελέσματα της μηχανής αναζήτησής της. Υλοποιήθηκαν στην εφαρμογή μας πολλές προτάσεις από αυτό. Αρχικά, χρησιμοποιήθηκαν περιεκτικές αλλά ταυτόχρονα περιγραφικές φράσεις στα πεδία του «*head*» της αρχικής μας σελίδας «*index.html*», όπως φαίνεται και στην κάτωθι εικόνα. Η εφαρμογή μας είναι πολύ καλά δομημένη και τα URL's της είναι άρτια ορισμένα, με αποτέλεσμα η πλοήγηση μέσα στην εφαρμογή να είναι πολύ εύκολη. Ακόμα η εφαρμογή μας περιλαμβάνει κείμενο εύκολο να διαβαστεί, τα links που χρησιμοποιούνται φαίνονται ξεκάθαρα και οι εικόνες περιλαμβάνουν την «*alt*» ιδιότητα.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Online Ραντεβού</title>
        <link rel="icon" type="image/ico" href="img/icon.ico" />

        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="keywords" content="Rantevou, Online, Ραντεβού, Γιατροί, Νοσοκομείο">
        <meta name="description" content="Κλείστε ραντεβού σε γιατρό εύκολα και γρήγορα.">
        <meta name="author" content="Kostas Rammos">
```

Εικόνα 2.2: *Εισαγωγή μεταδεδομένων σε κάθε σελίδα της εφαρμογής μας για μία καλύτερη θέση στο Google.*

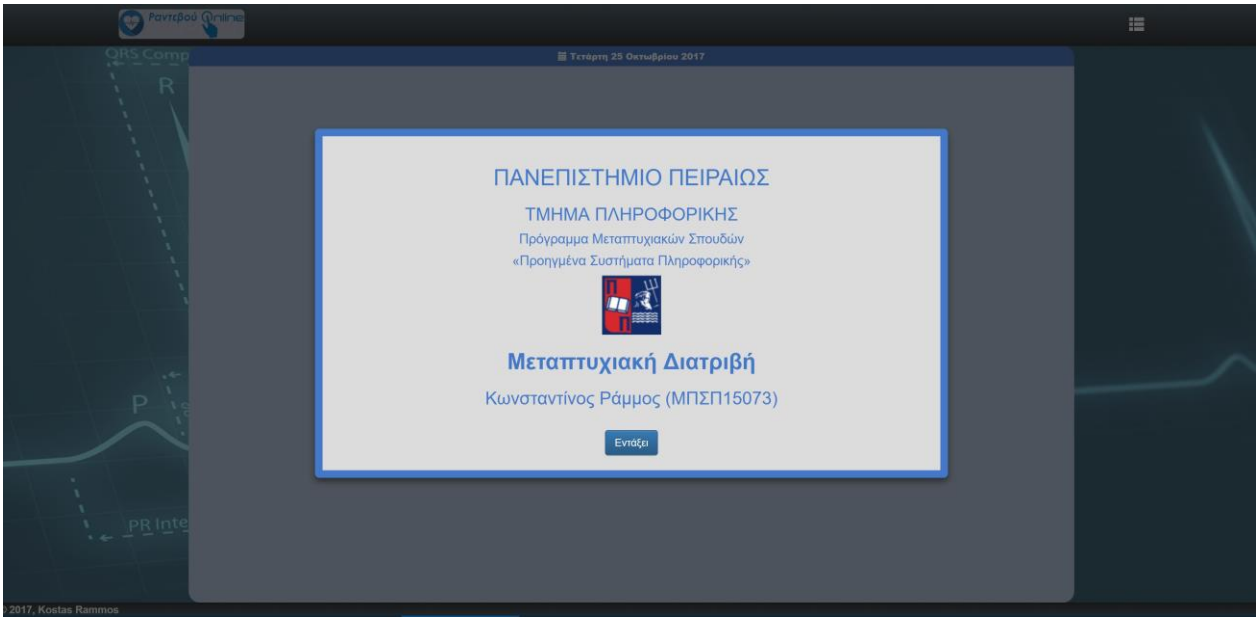
2.2 Εγχειρίδιο Χρήσης της Εφαρμογής

Στην προηγούμενη υποενότητα, είδαμε συνοπτικά τις λειτουργίες και τα ιδιαίτερα γνωρίσματα της εφαρμογής μας. Σε αυτή την υποενότητα, θα δούμε αναλυτικά πώς χρησιμοποιείται και τι ακριβώς μπορεί να κάνει. Θα ξεκινήσουμε την περιγραφή από τη διεπαφή του ασθενή, θα συνεχίσουμε με τη διεπαφή του γιατρού και θα κλείσουμε με τη διεπαφή του διαχειριστή του συστήματος.

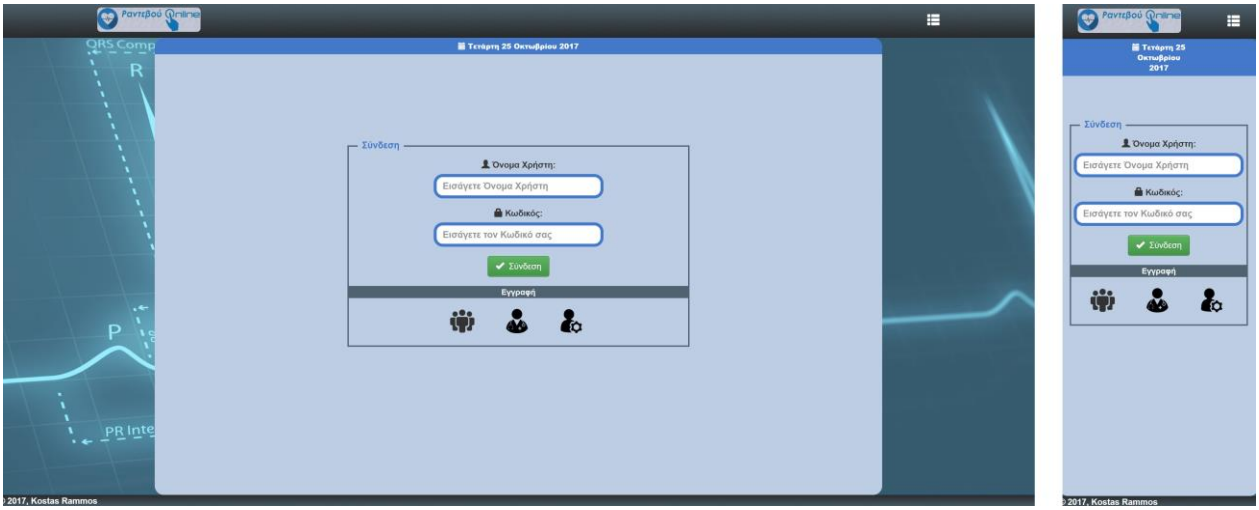
Αρχικά, πρέπει να αναφέρουμε ότι η εφαρμογή μας αποτελείται από κάποια σταθερά τμήματα και από κάποια μεταβλητά. Αναφορικά με τα σταθερά τμήματα, σε κάθε οθόνη από κάθε χρήστη, στο επάνω μέρος της σελίδας, είναι τοποθετημένη η επικεφαλίδα (Header), η οποία απεικονίζει το logo της εφαρμογής και διαθέτει επίσης ένα μενού, το οποίο, αν ο χρήστης δεν είναι συνδεδεμένος, εμφανίζει μόνο την επιλογή «*Σχετικά με την Εφαρμογή*», ενώ, αν είναι συνδεδεμένος, εμφανίζει και κάποιες επιπρόσθετες επιλογές, που θα δούμε στη συνέχεια. Η επιλογή «*Σχετικά με την Εφαρμογή*» ανοίγει την οθόνη της εικόνας 2.3. Επιπλέον, μόνιμα απεικονίζονται η εικόνα του background, μία navigation bar με την τρέχουσα ημερομηνία και στο κάτω μέρος της σελίδας, το Footer. Το κεντρικό τμήμα της οθόνης είναι το μεταβλητό και πάνω σε αυτό διαδραματίζεται η κυρίως αλληλεπίδραση της εφαρμογής με το χρήστη.

Η πρώτη οθόνη, που εμφανίζεται, όταν εκκινεί η εφαρμογή είναι η φόρμα για το login του χρήστη (εικόνα 2.4). Σε αυτή, ο εκάστοτε χρήστης εισάγει το username και το password του και, αν είναι σωστά, η εφαρμογή τον μεταφέρει στην καρτέλα κλεισίματος ραντεβού. Αν εισαχθεί λανθασμένη τιμή σε κάποιο από τα δύο πεδία, η εφαρμογή εμφανίζει μήνυμα λάθους, ώστε ο χρήστης να προσπαθήσει ξανά (εικόνα 2.5). Ακόμα, από την εν λόγω οθόνη, ο κάθε χρήστης μπορεί να μεταβεί στην οθόνη σύνδεσης, είτε του γιατρού, είτε του διαχειριστή.

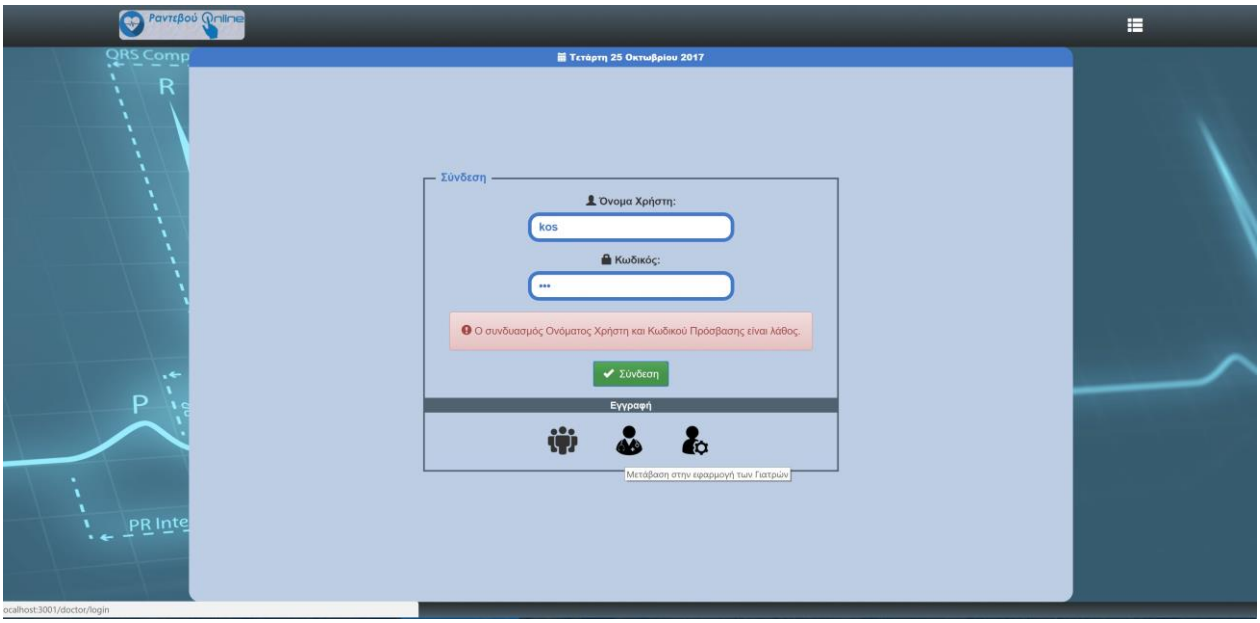
Όταν κάποιος χρήστης πρόκειται να χρησιμοποιήσει την εφαρμογή για πρώτη φορά, πρέπει να κάνει εγγραφή. Στην οθόνη του login, πρέπει να πατήσει το κουμπί «*Εγγραφή*» και εμφανίζεται μία φόρμα για να συμπληρώσει τα στοιχεία του (εικόνα 2.6). Αν συμπληρώσει σωστά όλα τα πεδία εμφανίζεται το μήνυμα «*Επιτυχής Εγγραφή*» και ο νέος χρήστης καταχωρείται στη βάση. Διαφορετικά, εμφανίζεται μήνυμα λάθους, ώστε να προσπαθήσει ξανά.



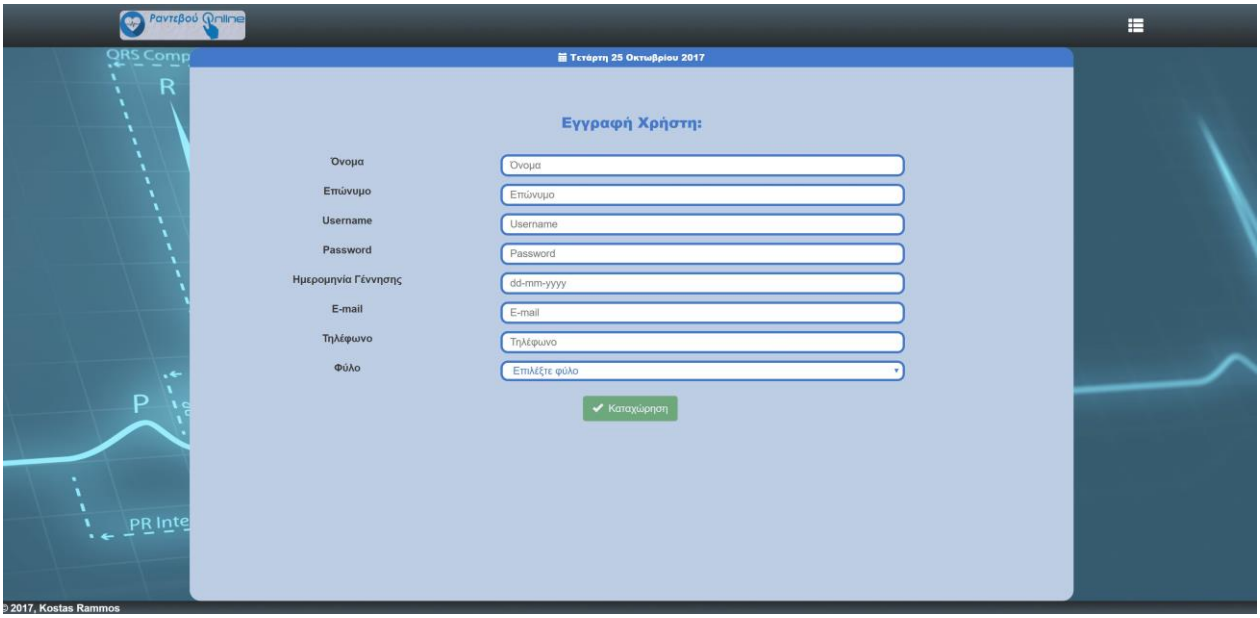
Εικόνα 2.3: *Οθόνη που εμφανίζεται επιλέγοντας «Σχετικά με την Εφαρμογή» από το μενού του χρήστη, για οποιαδήποτε διεπαφή.*



Εικόνα 2.4: *Διεπαφή ασθενή – Οθόνη σύνδεσης για μικρές και μεγάλες συσκευές.*



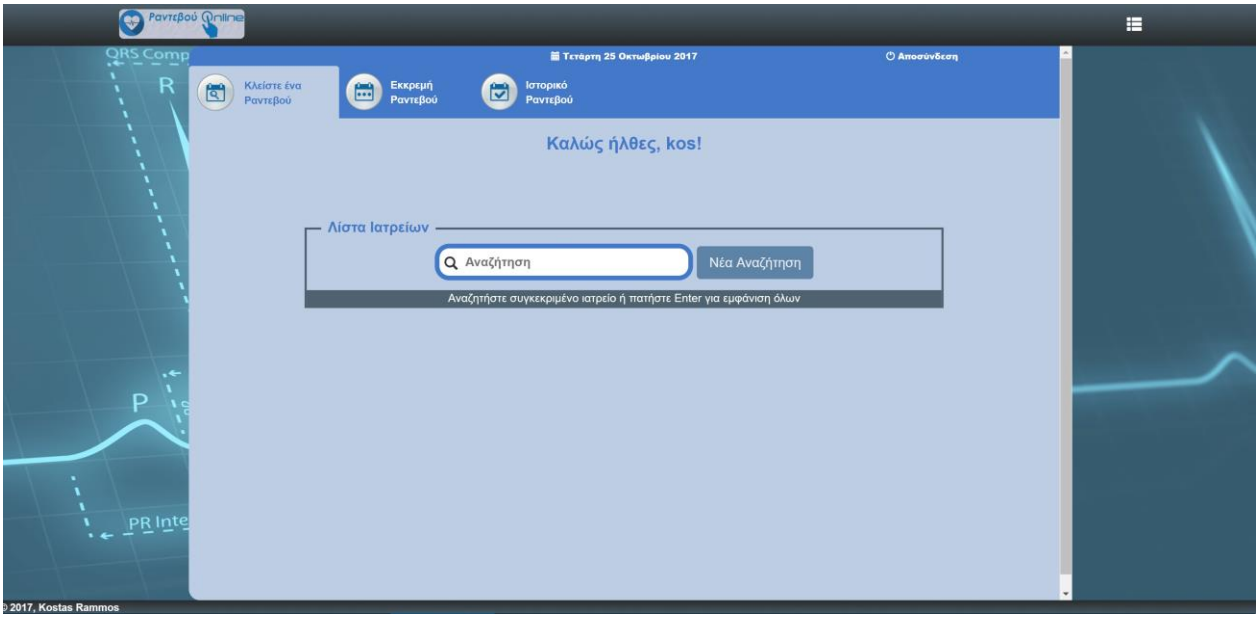
Εικόνα 2.5: *Διεπαφή ασθενή – Οθόνη σύνδεσης με μήνυμα λάθους.*



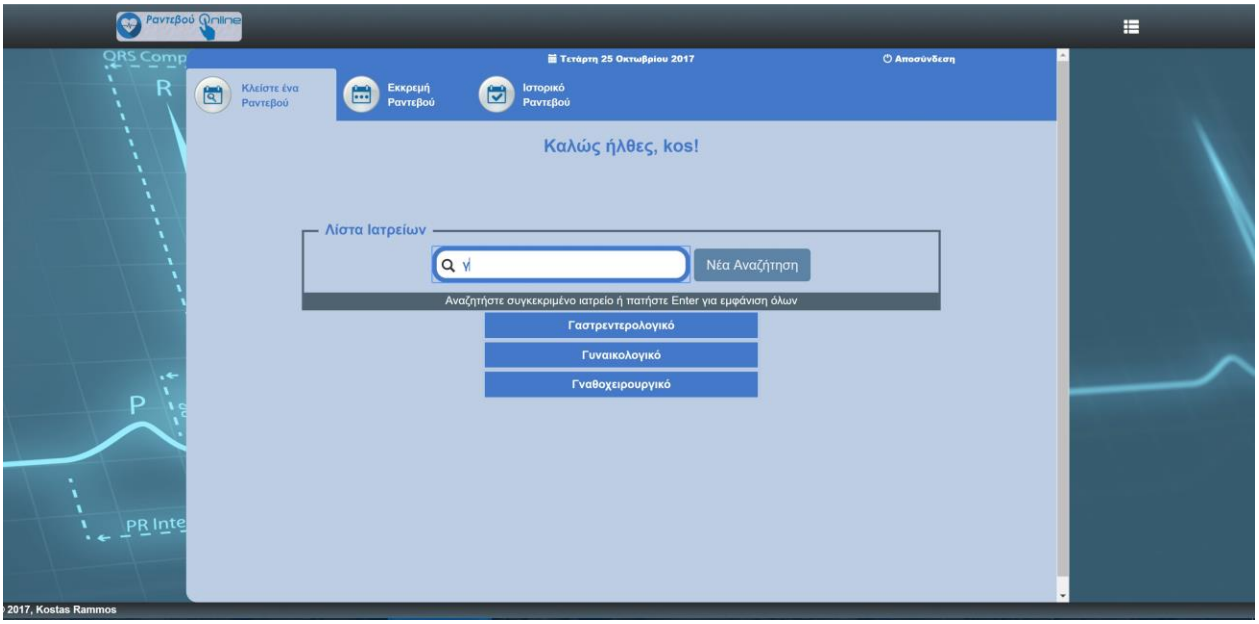
Εικόνα 2.6: *Διεπαφή ασθενή – Οθόνη εγγραφής*

Αφού ο ασθενής συνδεθεί επιτυχώς, μεταφέρεται στην οθόνη κλεισίματος ραντεβού (εικόνα 2.7). Αρχικά, καλείται να επιλέξει ένα ιατρείο, είτε γράφοντας στο πεδίο αναζήτησης, είτε πατώντας Enter για να εμφανιστούν όλα. Η αναζήτηση είναι υλοποιημένη κατάλληλα, ώστε με κάθε πλήκτρο που πατιέται, να εμφανίζονται τα αντίστοιχα αποτελέσματα (εικόνα 2.8). Το εν λόγω πεδίο της αναζήτησης συνοδεύεται

από το κουμπί «*Νέα Αναζήτηση*». Ο χρήστης μπορεί να το πατήσει οποιαδήποτε στιγμή της διαδικασίας κλεισίματος του ραντεβού, ώστε να ακυρωθεί η τρέχουσα και να ξεκινήσει νέα.

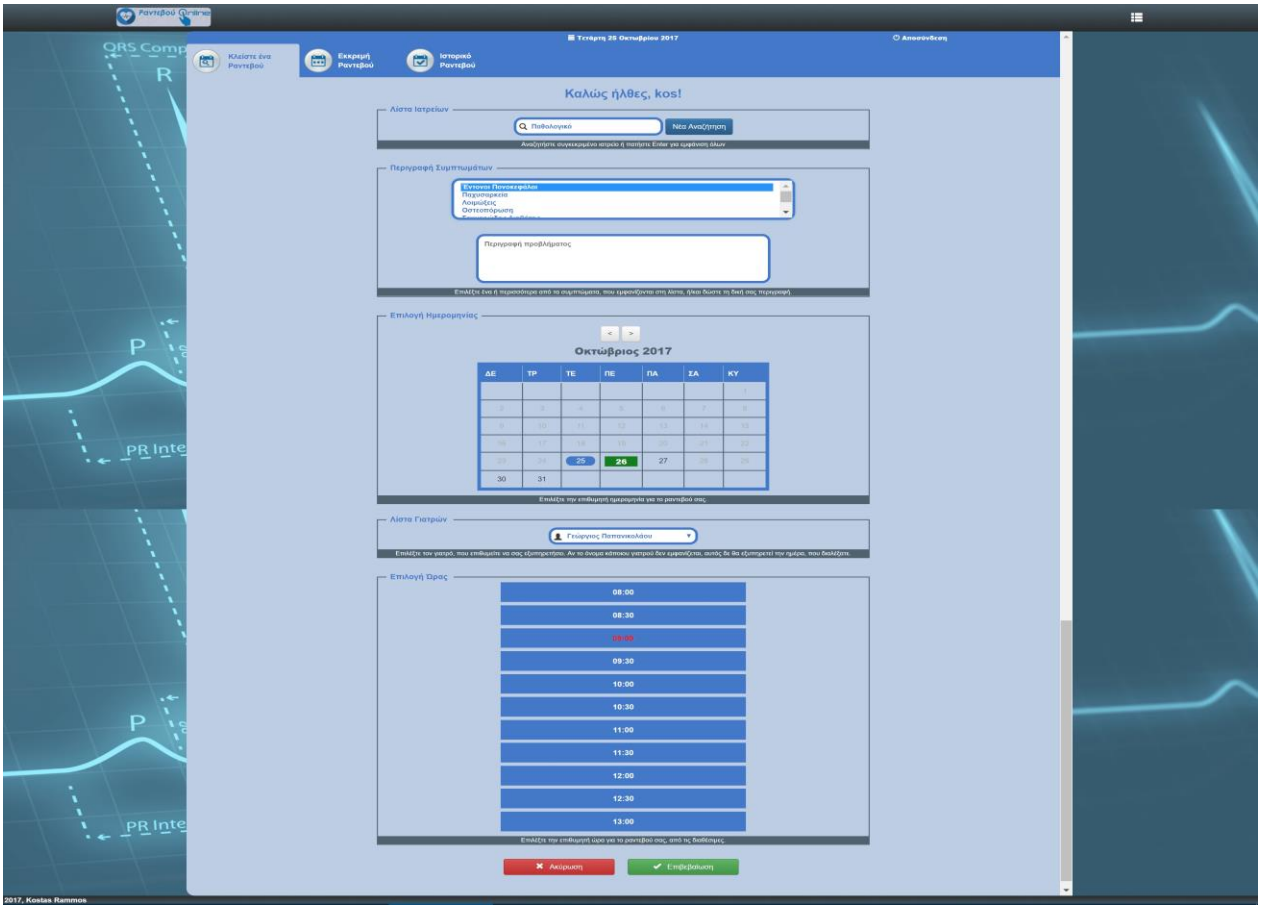


Εικόνα 2.7: *Διεπαφή ασθενή – Οθόνη κλεισίματος ραντεβού. Αρχικά εμφανίζεται μόνο το πεδίο επιλογής ιατρείου. Είναι μεγάλο και ευδιάκριτο και στη συνέχεια, με animation, μικραίνει.*

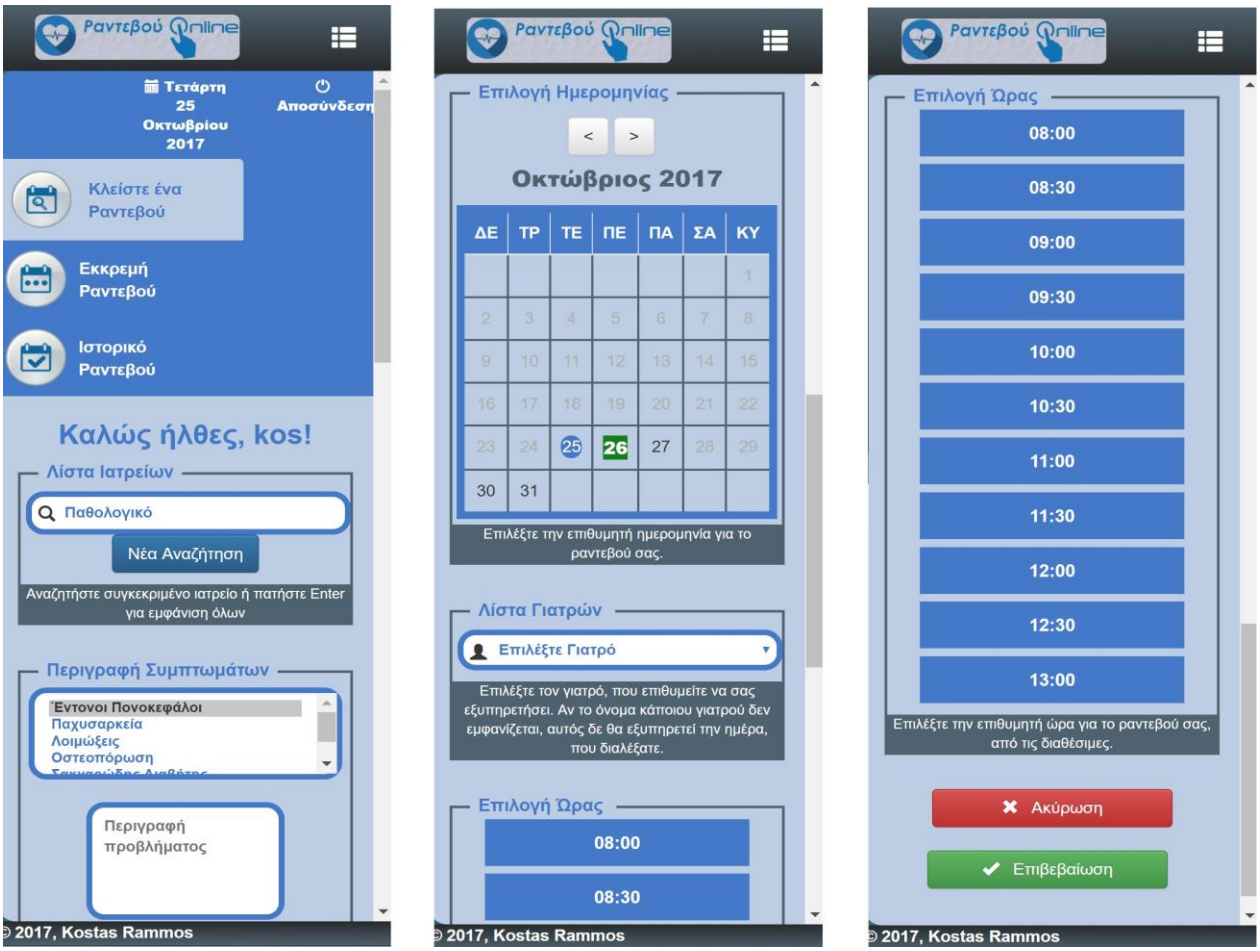


Εικόνα 2.8: *Διεπαφή ασθενή – Οθόνη κλεισίματος ραντεβού. Με το πάτημα κάθε πλήκτρου από το χρήστη, η αναζήτηση εμφανίζει και τα αντίστοιχα αποτελέσματα.*

Αφού επιλεχτεί ιατρείο, εμφανίζονται το πεδίο για την καταχώρηση των συμπτωμάτων και ένα ημερολόγιο για την επιλογή της επιθυμητής ημερομηνίας του ραντεβού. Στο πεδίο των συμπτωμάτων, ο χρήστης πρέπει είτε να επιλέξει από τη λίστα ένα ή περισσότερα από τα προκαταχωρημένα, είτε να γράψει με ελεύθερο κείμενο το πρόβλημά του. Στο ελεύθερο κείμενο επιτρέπονται μόνο γράμματα, αριθμοί, κενά και οι ειδικοί χαρακτήρες «? , . ; [ ] { } – ( ) _ = + !». Αν εισαχθεί οτιδήποτε άλλο, η εφαρμογή δε θα δύναται να προχωρήσει. Στο ημερολόγιο, ο χρήστης επιτρέπεται να επιλέξει μόνο κάποια μελλοντική και εργάσιμη ημέρα. Αφού επιλεχτούν τα συμπτώματα και η επιθυμητή ημερομηνία για το ραντεβού, τότε εμφανίζονται οι διαθέσιμοι γιατροί και οι διαθέσιμες ώρες (εικόνα 2.9). Οι εμφανιζόμενοι γιατροί είναι όσοι θα είναι παρόντες στο ιατρείο την επιλεγμένη ημερομηνία και όχι όσοι έχουν άδεια. Τα time slots των ραντεβού, αν είναι κλεισμένα, εμφανίζονται με κόκκινο χρώμα και ο χρήστης δεν μπορεί να τα επιλέξει. Να σημειωθεί ότι πρέπει να συμπληρωθούν όλα τα πεδία, αλλιώς προκύπτει το μήνυμα σφάλματος της εικόνας 2.11. Τελικά, αφού ο χρήστης συμπληρώσει σωστά όλα τα πεδία και πατήσει «*Καταχώρηση*», εμφανίζεται η οθόνη επιβεβαίωσης του ραντεβού της εικόνας 2.12, η οποία ανακεφαλαιώνει τις επιλογές του και δίνει και μία εκτίμηση για τη διάρκεια του ραντεβού, με τη χρήση κατάλληλα σχεδιασμένου Νευρωνικού Δικτύου.

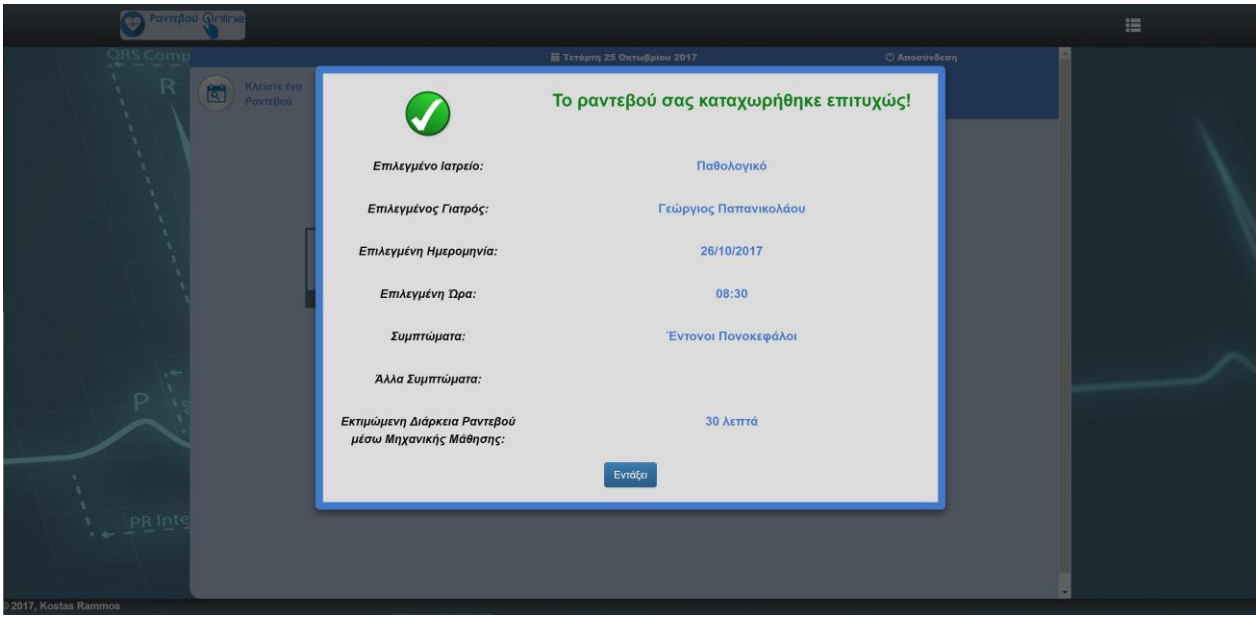


Εικόνα 2.9: *Διεπαφή ασθενή – Οθόνη κλεισίματος ραντεβού. Αφού επιλεχτεί ιατρείο, εμφανίζονται το πεδίο εισαγωγής των συμπτωμάτων και το ημερολόγιο. Κατόπιν των επιλογών του χρήστη, εμφανίζονται η λίστα επιλογής γιατρού και η λίστα για επιλογή ώρας. Αν κάποιο time slot είναι κλεισμένο, εμφανίζεται με κόκκινο και δεν μπορεί να επιλεχτεί.*



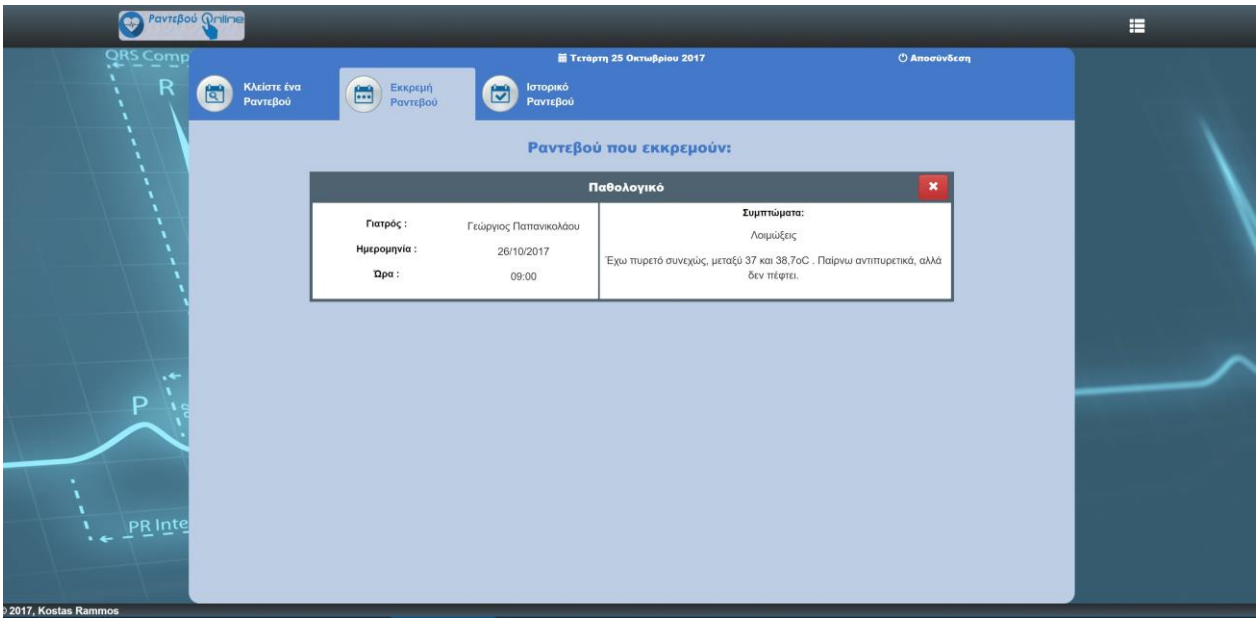
Εικόνα 2.10: *Διεπαφή ασθενή – Οθόνη κλεισίματος ραντεβού για μικρές συσκευές.*

Εικόνα 2.11: *Διεπαφή ασθενή – Οθόνη κλεισίματος ραντεβού. Αν κάποιο από τα πεδία δεν συμπληρωθεί, εμφανίζεται μήνυμα λάθους.*

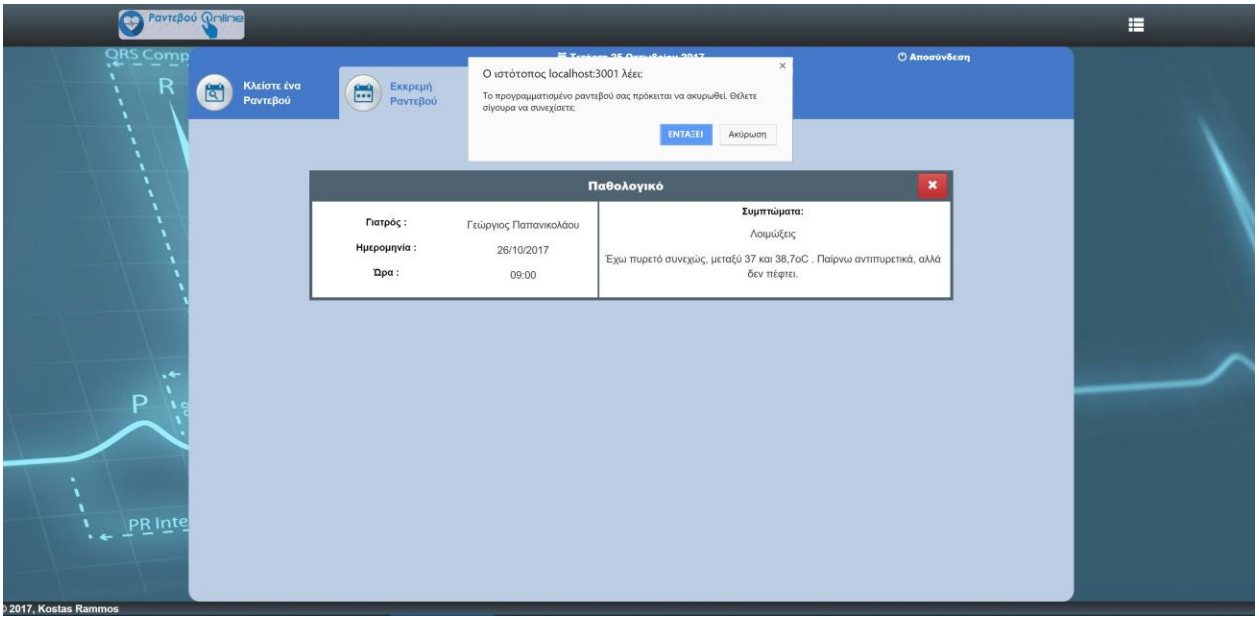


Εικόνα 2.12: *Διεπαφή ασθενή – Οθόνη επιβεβαίωσης ραντεβού.*

Αν ο χρήστης μεταβεί στην καρτέλα «*Εκκρεμή Ραντεβού*» (εικόνα 2.13), μπορεί να δει τις πληροφορίες όλων των ραντεβού του, των οποίων η προγραμματισμένη ημερομηνία δεν έχει εκπνεύσει ακόμα. Πατώντας το κόκκινο «Χ», έχει τη δυνατότητα να ακυρώσει το αντίστοιχο ραντεβού. Πριν την οριστική διαγραφή του ραντεβού, η εφαρμογή θα εμφανίσει ένα μήνυμα επιβεβαίωσης στο χρήστη (εικόνα 2.14).

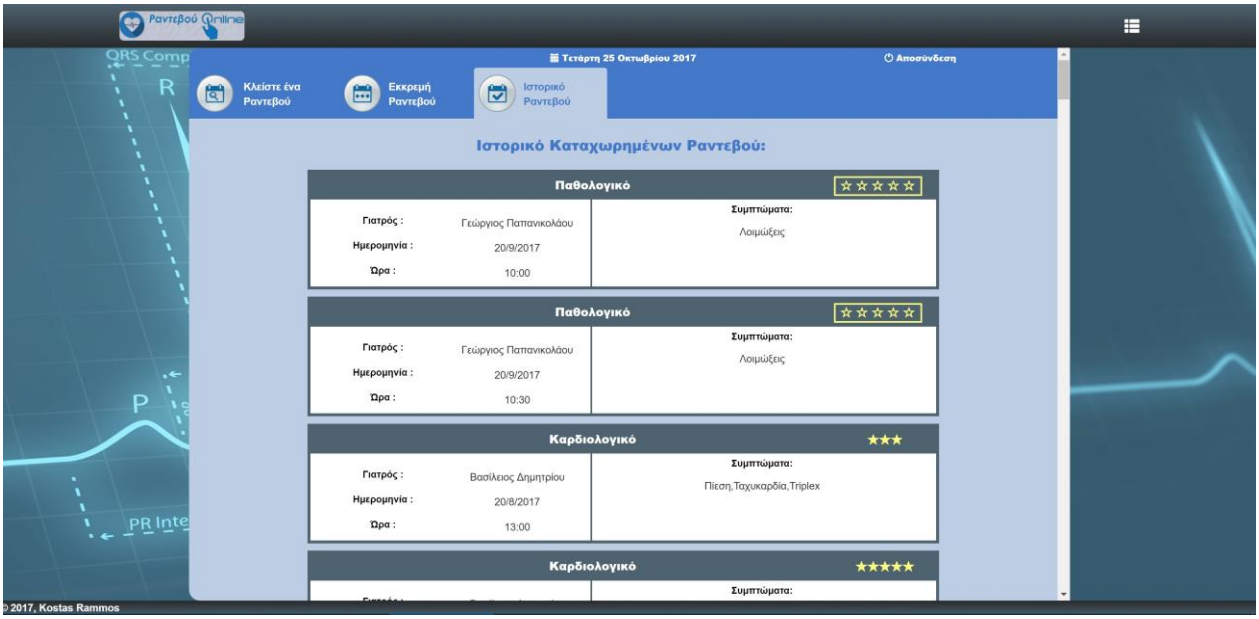


Εικόνα 2.13: *Διεπαφή ασθενή – Οθόνη εκκρεμών ραντεβού. Το κουμπί «Χ» ακυρώνει το αντίστοιχο ραντεβού.*



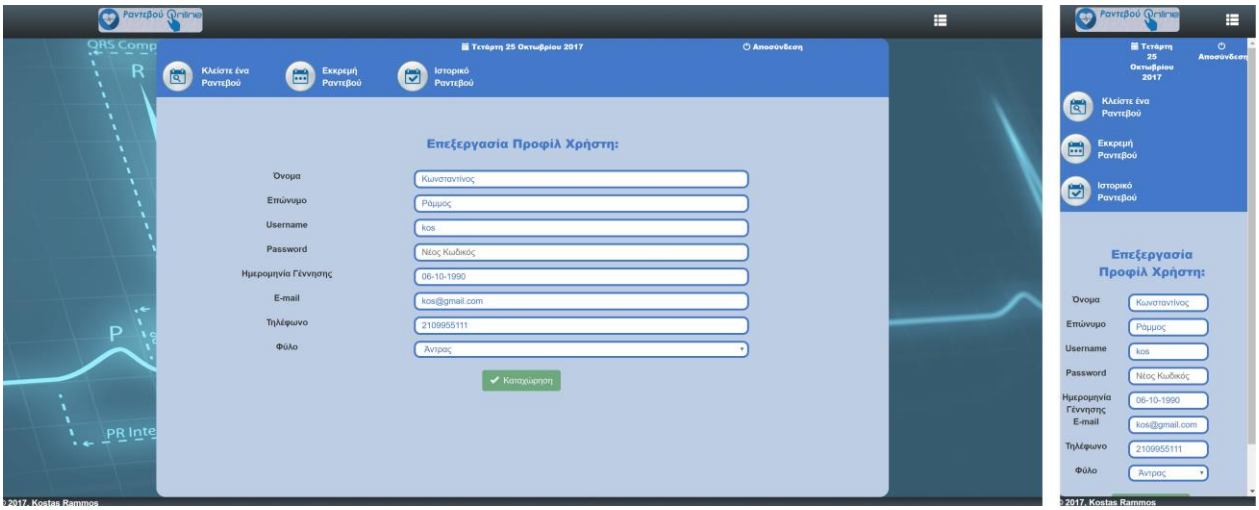
Εικόνα 2.14: *Διεπαφή ασθενή – Οθόνη εκκρεμών ραντεβού. Εμφάνιση μηνύματος επιβεβαίωσης για διαγραφή του ραντεβού.*

Αν ο χρήστης μεταβεί στην καρτέλα «*Ιστορικό Ραντεβού*» (εικόνα 2.15), μπορεί να δει τις πληροφορίες όλων των ολοκληρωμένων ραντεβού του. Έχει τη δυνατότητα να βαθμολογήσει το γιατρό, που τον εξέτασε, με ένα (1) έως πέντε (5) αστέρια και να δει τις βαθμολογίες, που έχει ήδη καταχωρήσει.



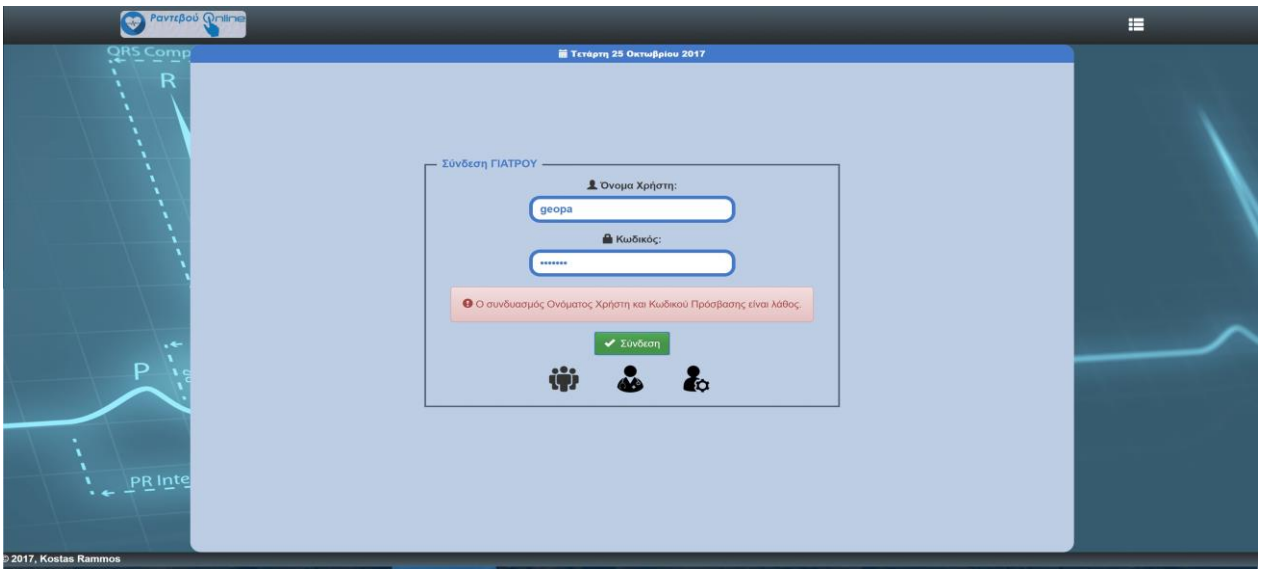
Εικόνα 2.15: *Διεπαφή ασθενή – Οθόνη ολοκληρωμένων ραντεβού. Ο χρήστης έχει τη δυνατότητα να βαθμολογήσει τον γιατρό που τον εξέτασε με ένα έως πέντε αστέρια. Όσα αστέρια είναι σε κουτάκι, σημαίνει ότι ο αντίστοιχος γιατρός δεν έχει βαθμολογηθεί ακόμα.*

Στο μενού του χρήστη, όταν είναι συνδεδεμένος, εμφανίζεται η επιλογή «*Επεξεργασία Προφίλ Χρήστη*». Στην εικόνα 2.16, φαίνεται η οθόνη επεξεργασίας του προφίλ, η οποία απεικονίζει ήδη τα καταχωρημένα στοιχεία για διευκόλυνση του χρήστη, εκτός βέβαια από τον κωδικό του. Αν κάποιο από τα πεδία παραμείνει κενό ή δεν συμπληρωθεί όπως προβλέπεται, δεν ενεργοποιείται το κουμπί «*Καταχώρηση*», ώστε να μπορέσει ο χρήστης να προχωρήσει με τη διαδικασία.



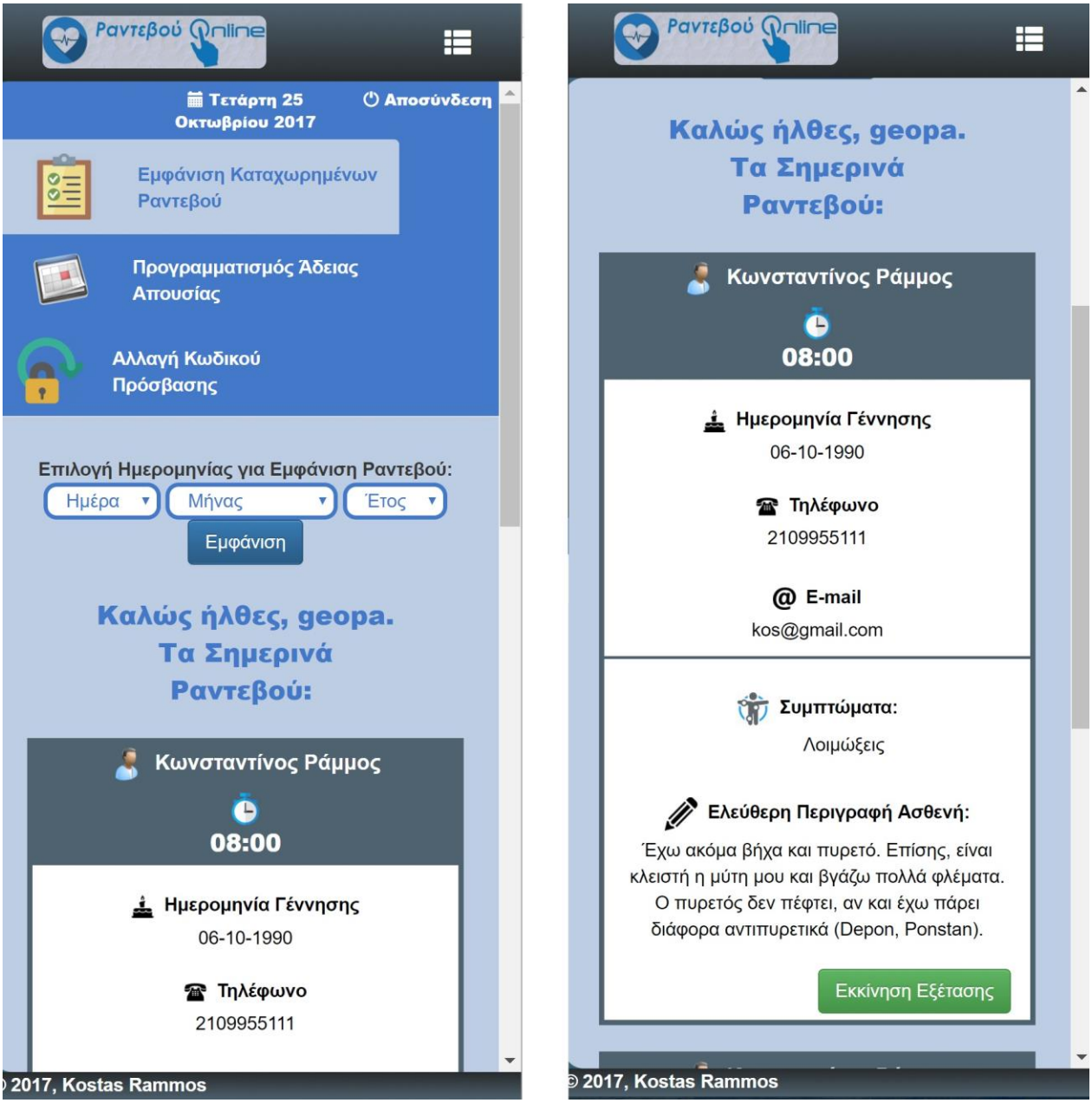
Εικόνα 2.16: *Διεπαφή ασθενή – Οθόνη επεξεργασίας προφίλ χρήστη για μικρές και μεγάλες συσκευές.*

Όσον αφορά στη διεπαφή του γιατρού, απεικονίζεται στην εικόνα 2.17, η οθόνη σύνδεσης αυτού. Αντίθετα με τον ασθενή, εδώ δεν υπάρχει η δυνατότητα εγγραφής, καθώς οι γιατροί περνιόνται στο σύστημα κατευθείαν από το διαχειριστή. Τα εικονίδια κάτω από το κουμπί «*Καταχώρηση*» είναι σύνδεσμοι για τις οθόνες σύνδεσης του ασθενή και του διαχειριστή. Ακόμα, αν το username και το password, που θα εισάγει ο γιατρός, δεν είναι σωστά, εμφανίζεται μήνυμα λάθους.

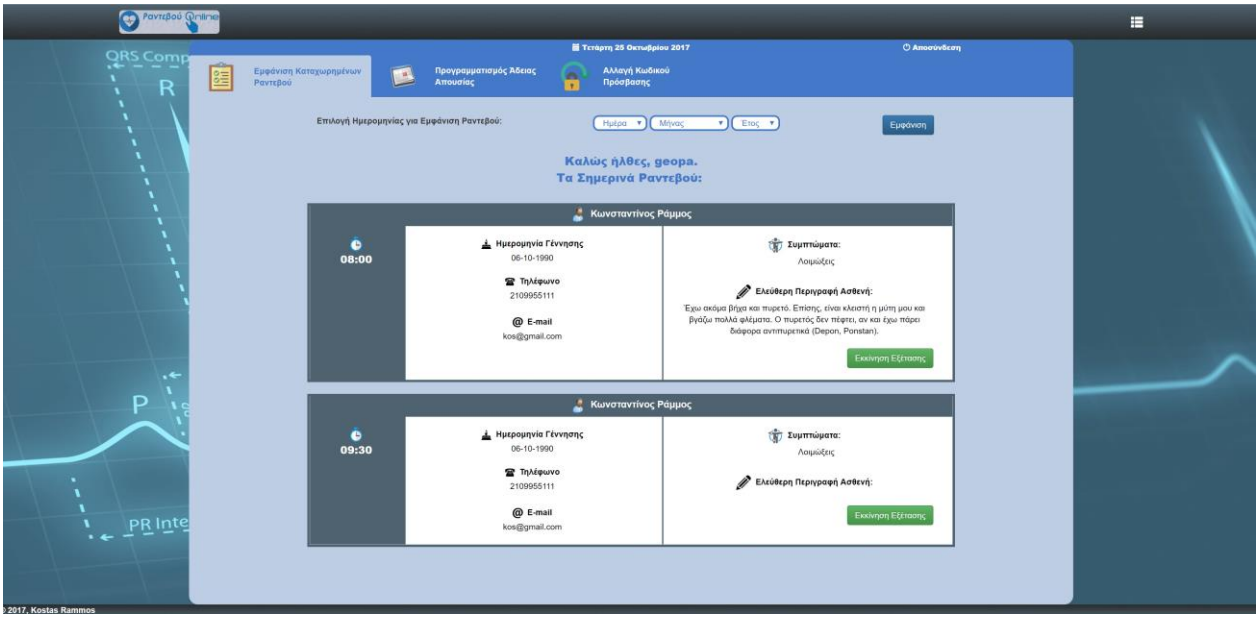


Εικόνα 2.17: *Διεπαφή γιατρού – Οθόνη σύνδεσης με μήνυμα λάθους, αν τα στοιχεία δεν είναι σωστά.*

Αφού ο εκάστοτε γιατρός συνδεθεί επιτυχώς, μεταφέρεται στην οθόνη απεικόνισης των ραντεβού του (εικόνες 2.18, 2.19). Εμφανίζονται από προεπιλογή όλα τα ραντεβού της τρέχουσας ημέρας, ταξινομημένα κατά ώρα. Σε κάθε ένα από αυτά περιλαμβάνονται η ώρα, το ονοματεπώνυμο του ασθενή, η ημερομηνία γέννησής του, στοιχεία επικοινωνίας με αυτόν και τα συμπτώματα που δήλωσε.

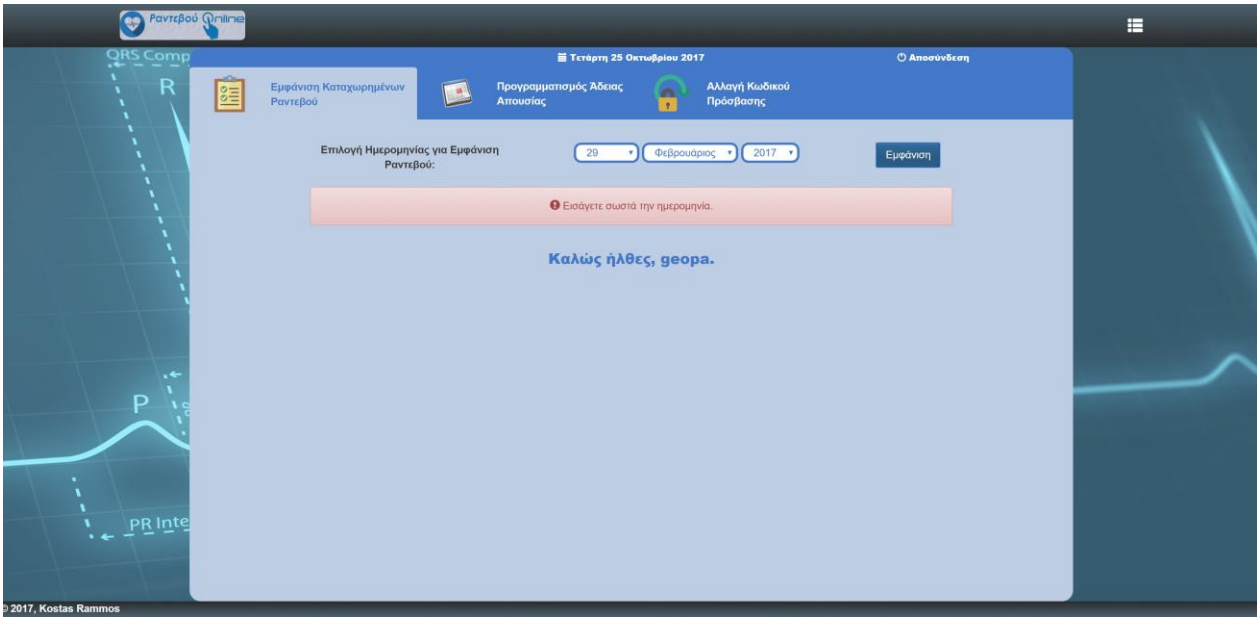


Εικόνα 2.18: *Διεπαφή γιατρού – Οθόνη προβολής ραντεβού για μικρές συσκευές.*



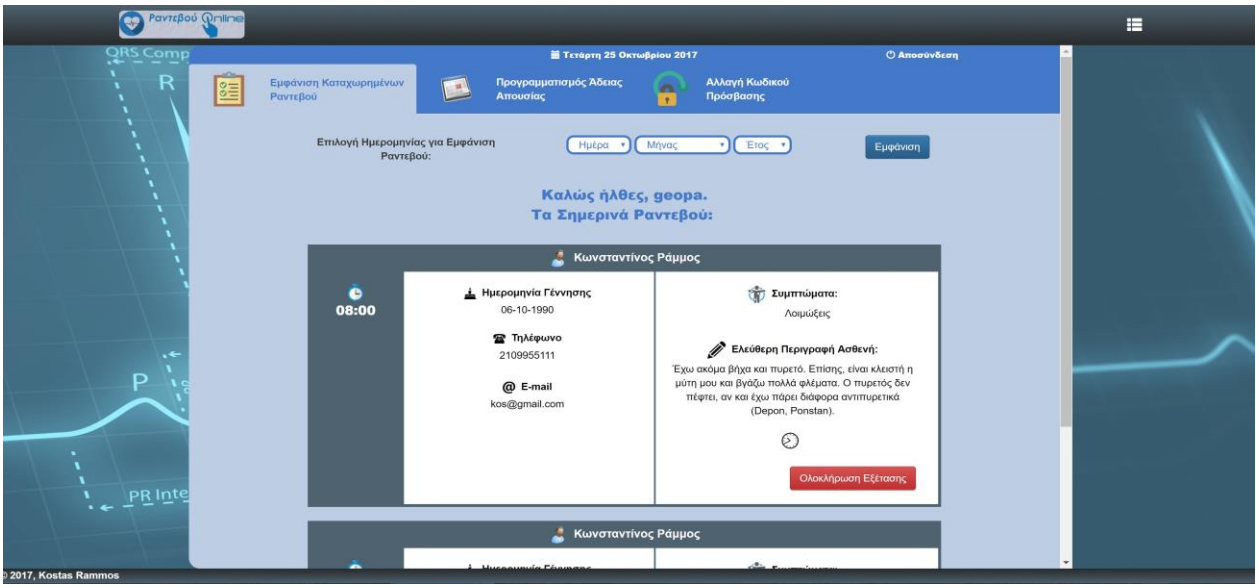
Εικόνα 2.19: *Διεπαφή γιατρού – Οθόνη προβολής ραντεβού για μεγάλες συσκευές.*

Ακόμα, ο γιατρός μπορεί να δει τα ραντεβού μίας άλλης ημέρας, επιλέγοντας κατάλληλα στα πεδία «*Επιλογή Ημερομηνίας για Εμφάνιση Ραντεβού*». Η εφαρμογή έχει υλοποιηθεί έτσι, ώστε λανθασμένες επιλογές, όπως 31 Νοεμβρίου, 29 Φεβρουαρίου σε μη-δίσεκτο έτος κ.τ.λ., να εμφανίζουν μήνυμα λάθους (εικόνα 2.20). Να επισημανθεί ότι η λίστα με τα έτη περιλαμβάνει πάντα το τρέχον έτος, τα δύο (2) προηγούμενα και τα δύο (2) επόμενα.

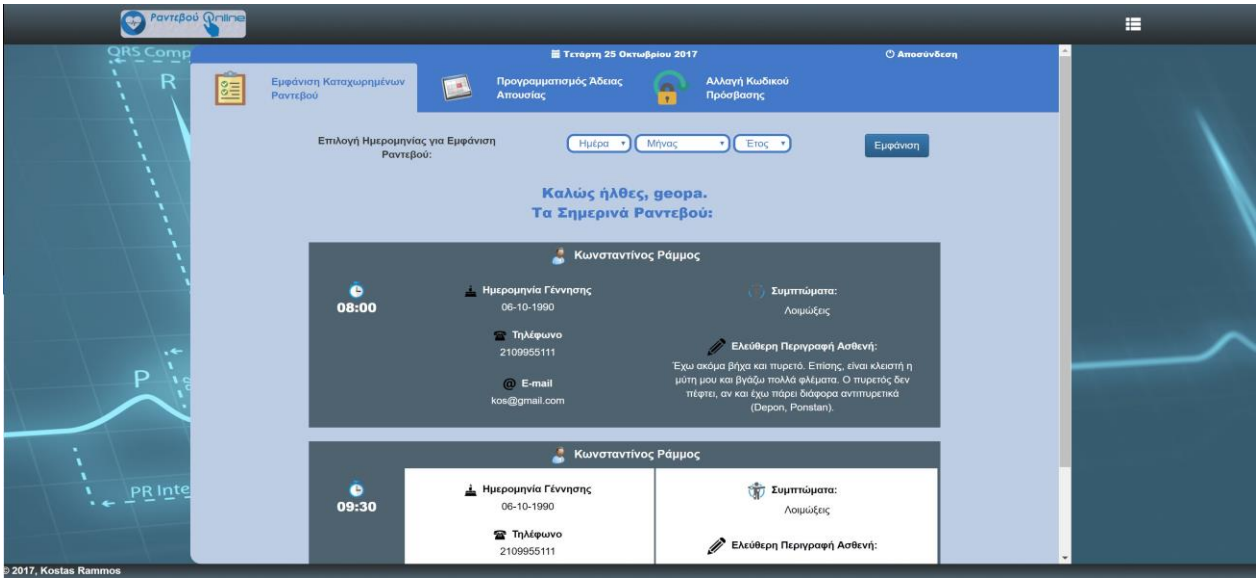


Εικόνα 2.20: *Διεπαφή γιατρού – Εμφάνιση μηνύματος λάθους, αν οι ημερομηνίες δεν είναι σωστές.*

Επίσης, κάθε ραντεβού περιλαμβάνει το κουμπί «Εκκίνηση Εξέτασης». Ο γιατρός πρέπει να το πατήσει, όταν ξεκινήσει το αντίστοιχο ραντεβού. Πρώτον, όπως φαίνεται και στην εικόνα 2.21, θα μπορεί να έχει ξεκάθαρη εικόνα για το τρέχον ραντεβού και δεύτερον, η εφαρμογή χρειάζεται την ώρα εκκίνησης, ώστε έπειτα να μπορέσει να υπολογίσει τη διάρκεια του ραντεβού. Όταν έχει εκκινήσει κάποιο ραντεβού, απενεργοποιούνται τα κουμπιά «*Εκκίνηση Ραντεβού*» όλων των υπολοίπων ραντεβού. Όταν ολοκληρωθεί το ραντεβού, ο γιατρός πρέπει να πατήσει «*Ολοκλήρωση Εξέτασης*». Έτσι, εμφανίζεται η οθόνη της εικόνας 2.22, όπου βλέπουμε ότι είναι ευδιάκριτο ποια ραντεβού έχουν ολοκληρωθεί και ποια όχι, αλλά επίσης καταχωρείται και στη βάση η διάρκεια του ραντεβού, η οποία χρησιμοποιείται για την εκπαίδευση του Νευρωνικού Δικτύου της εφαρμογής.

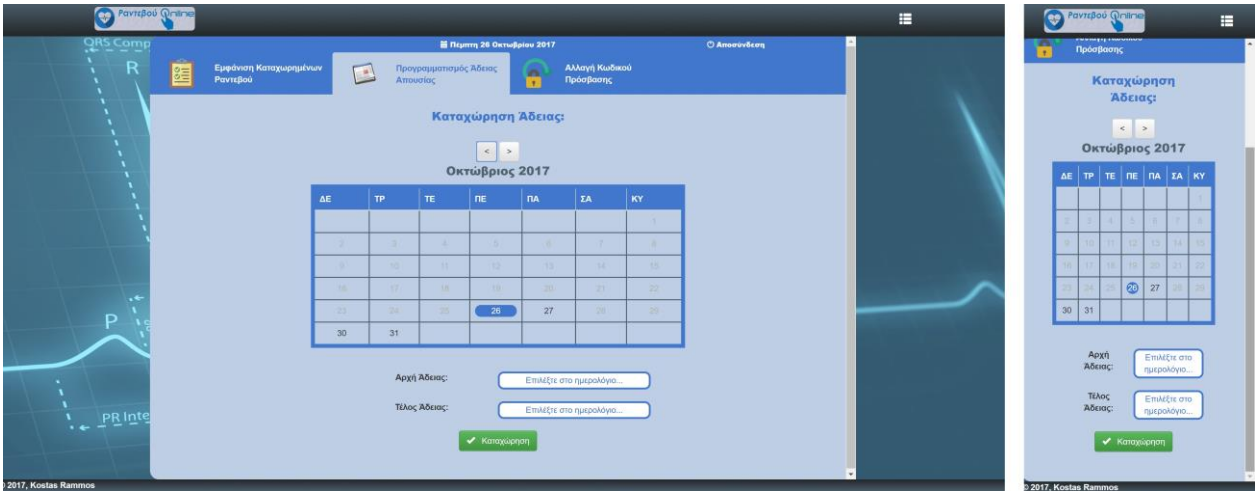


Εικόνα 2.21: *Διεπαφή γιατρού – Οθόνη κατά τη διάρκεια εξέτασης ασθενή.*

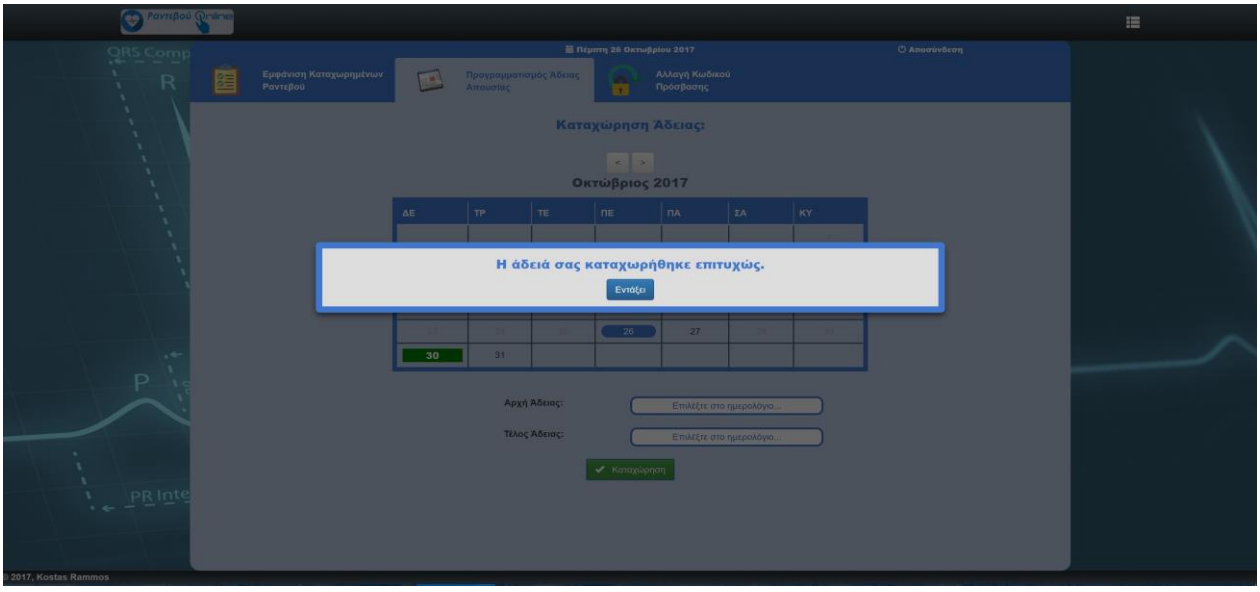


Εικόνα 2.22: *Διεπαφή γιατρού – Οθόνη ολοκληρωμένων ραντεβού.*

Επιπροσθέτως, ο γιατρός δύναται να δηλώσει τις ημέρες απουσίας του από την καρτέλα «*Προγραμματισμός Άδειας Απουσίας*». Με το που ανοίγει η καρτέλα, εμφανίζεται ένα ημερολόγιο, που απεικονίζει τον τρέχοντα μήνα (εικόνα 2.23). Ο γιατρός μπορεί να μετακινηθεί σε προηγούμενους ή επόμενους μήνες, πατώντας τα δύο κουμπιά με τα βελάκια. Οι παρελθοντικές ημερομηνίες και τα σαββατοκύριακα του ημερολογίου είναι gray out και δεν μπορούν να επιλεχτούν. Ο χρήστης αρκεί να κάνει δύο (2) μόνο κλικ πάνω στο ημερολόγιο, με το πρώτο θα επιλέξει την ημερομηνία που αρχίζει η άδειά του και με το δεύτερο την ημερομηνία που τελειώνει η άδειά του. Κάθε επιλογή απεικονίζεται στα πεδία «*Αρχή Άδειας*» και «*Τέλος Άδειας*», αντίστοιχα. Με το πάτημα του κουμπιού «*Καταχώρηση*», η άδεια καταχωρείται στο σύστημα (εικόνα 2.24) και εκείνο το διάστημα δεν εμφανίζεται το όνομά του στο πεδίο «*Επιλογή Γιατρού*» στην καρτέλα του ασθενή για το κλείσιμο ραντεβού.

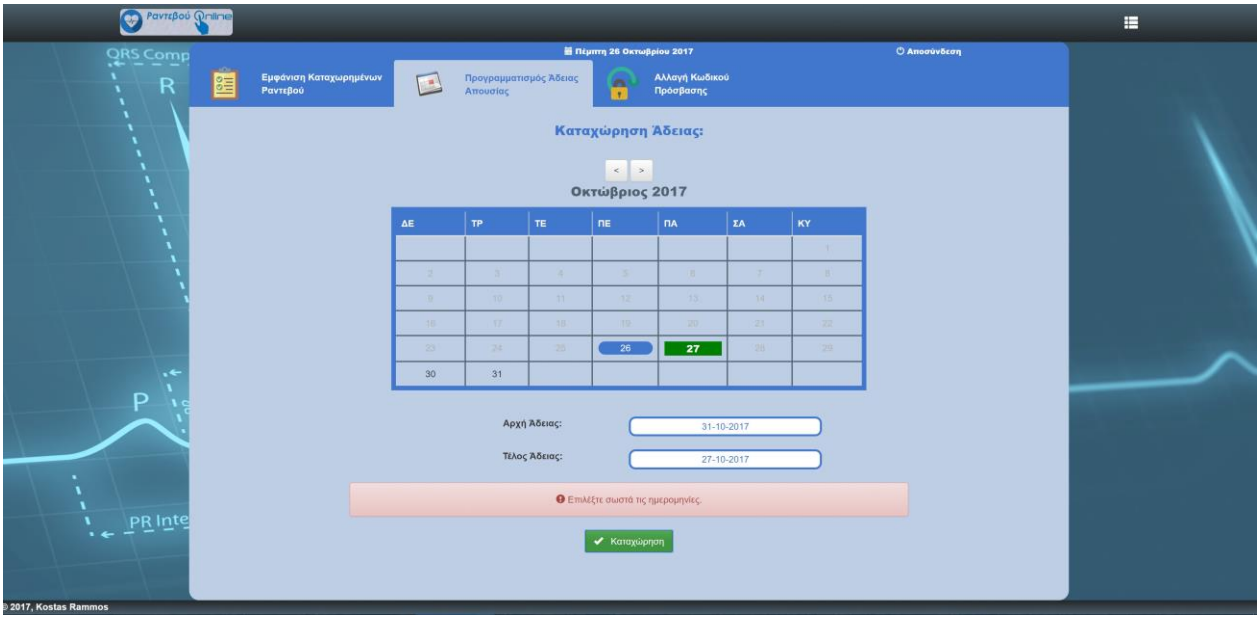


Εικόνα 2.23: *Διεπαφή γιατρού – Οθόνη προγραμματισμού άδειας για μικρές και μεγάλες συσκευές.*



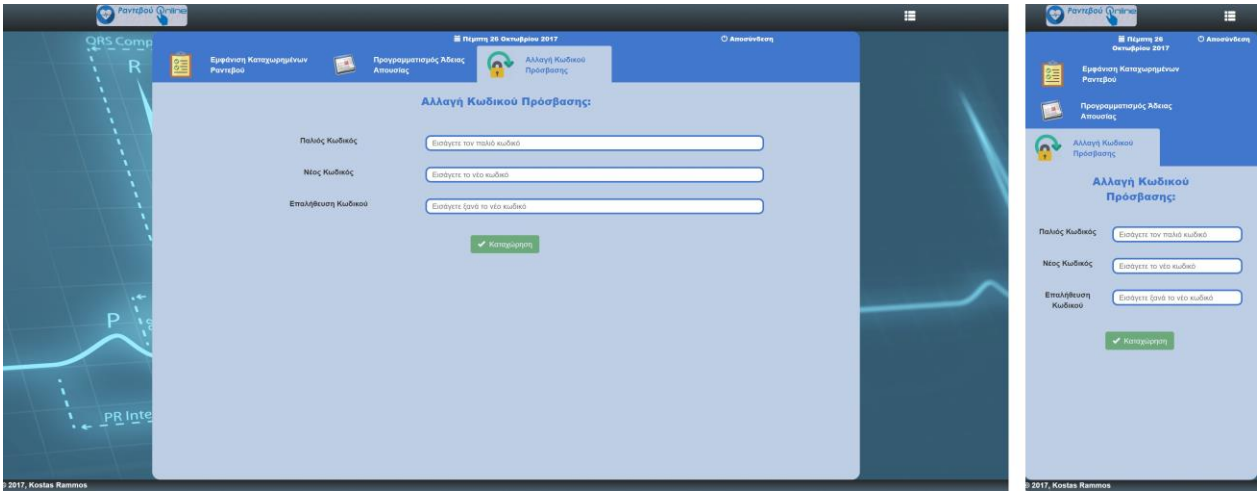
Εικόνα 2.24: *Διεπαφή γιατρού – Επιτυχής καταχώρηση άδειας.*

Αν οι ημερομηνίες δεν έχουν εισαχθεί σωστά, για παράδειγμα, η αρχή της άδειας είναι μία ημέρα μεταγενέστερη από την ημέρα τέλους της άδειας, η εφαρμογή δεν συνεχίζει και εμφανίζεται το μήνυμα λάθους της εικόνας 2.25.



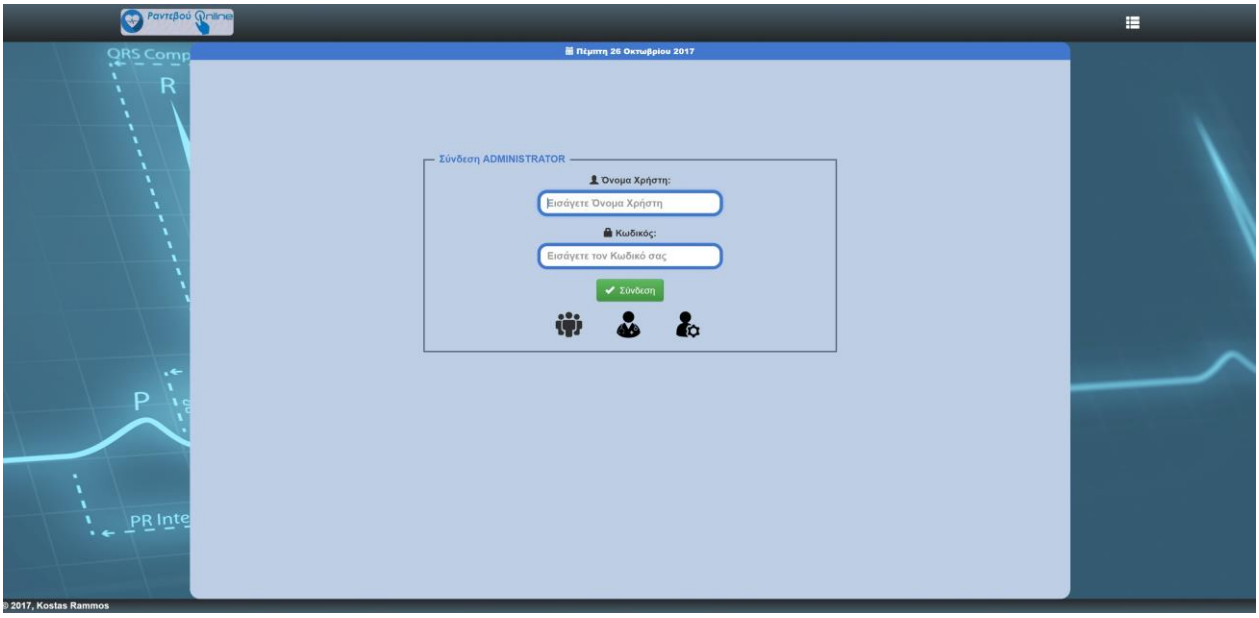
Εικόνα 2.25: *Διεπαφή γιατρού – Εμφάνιση μηνύματος λάθους αν οι ημερομηνίες δεν έχουν εισαχθεί σωστά.*

Ακόμα, ο γιατρός έχει τη δυνατότητα να αλλάξει τον κωδικό πρόσβασής του στο σύστημα μέσω της καρτέλας «*Αλλαγή Κωδικού Πρόσβασης*». Πρέπει να εισάγει αρχικά τον παλιό κωδικό του για επιβεβαίωση της ταυτότητάς του και στη συνέχεια το νέο κωδικό. Αν η αλλαγή ολοκληρωθεί επιτυχώς ή όχι, εμφανίζεται και το αντίστοιχο μήνυμα.



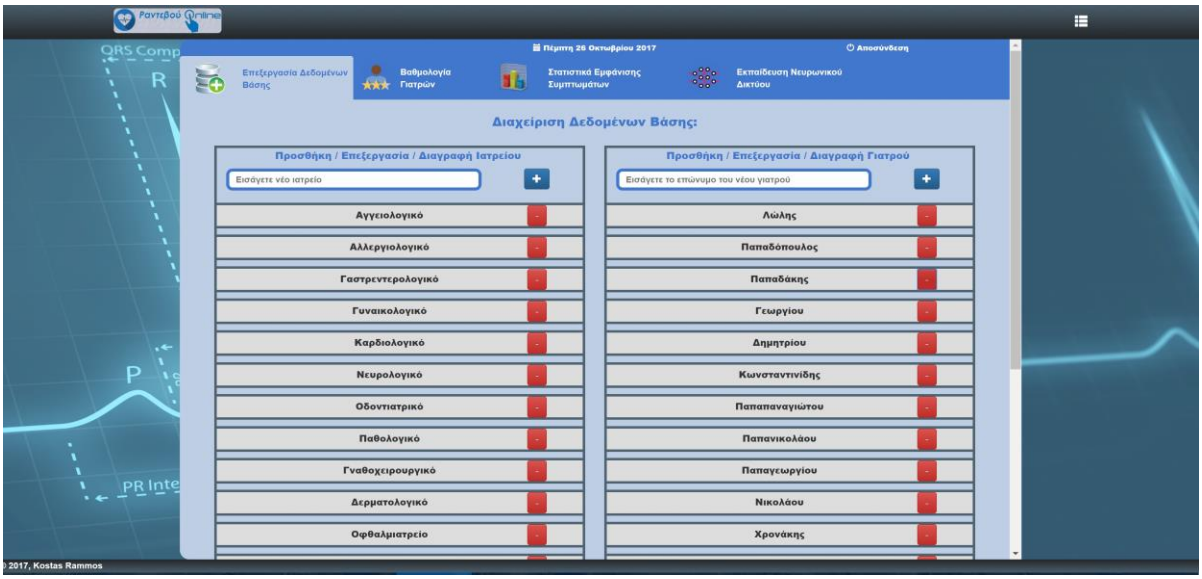
Εικόνα 2.26: *Διεπαφή γιατρού – Οθόνη αλλαγής κωδικού χρήστη για μικρές και μεγάλες συσκευές.*

Τέλος, θα παρουσιάσουμε τη διεπαφή του διαχειριστή του συστήματος. Αρχικά, θα πρέπει να συνδεθεί στο σύστημα. Η οθόνη σύνδεσης φαίνεται στην εικόνα 2.27. Αν το όνομα χρήστη ή ο κωδικός είναι λάθος, εμφανίζεται μήνυμα λάθους. Η οθόνη σύνδεσης δε διαθέτει κουμπί εγγραφής, καθώς δεν απαιτείται για το διαχειριστή, αλλά διαθέτει συνδέσμους για τις οθόνες σύνδεσης των ασθενών και των γιατρών ακριβώς κάτω από το κουμπί «*Καταχώρηση*».

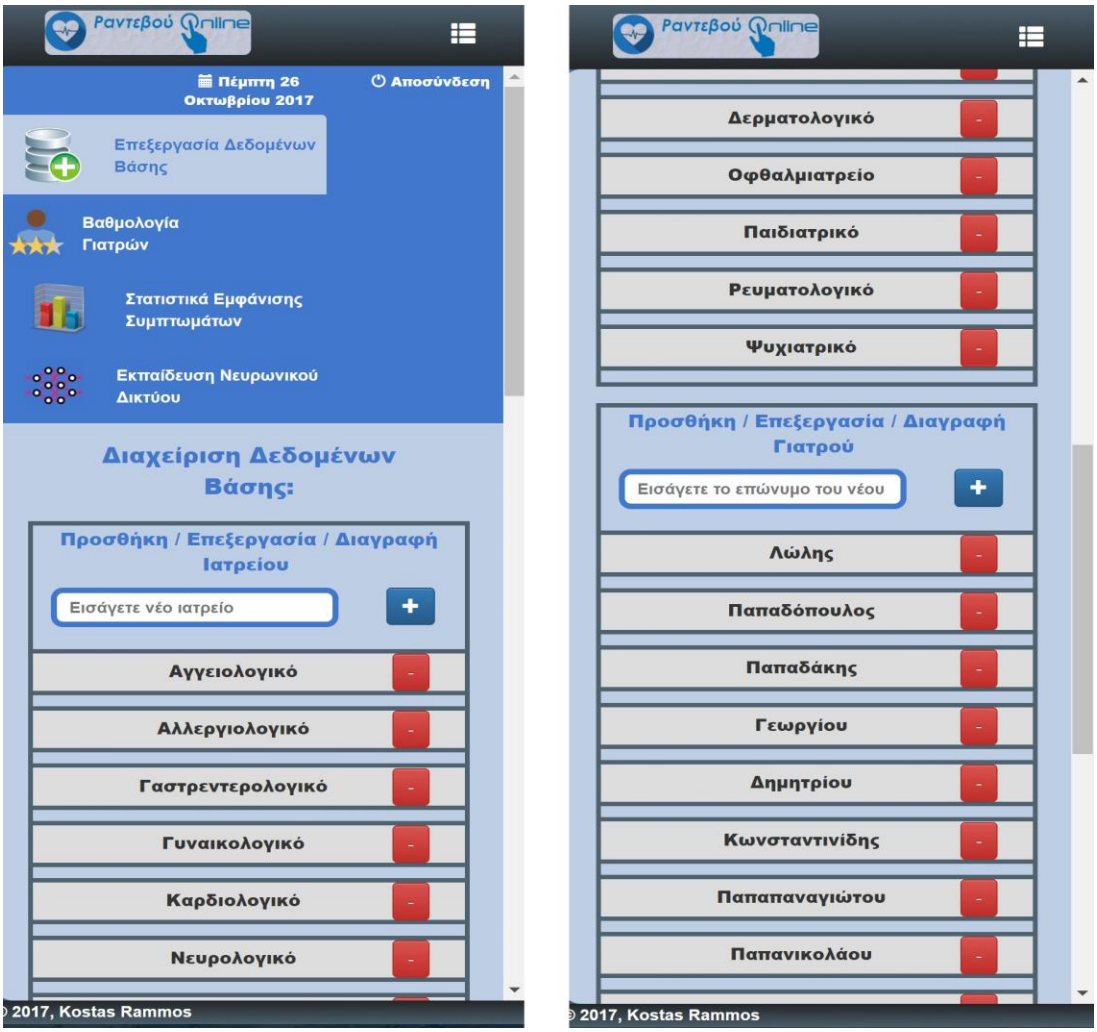


Εικόνα 2.27: *Διεπαφή διαχειριστή – Οθόνη σύνδεσης.*

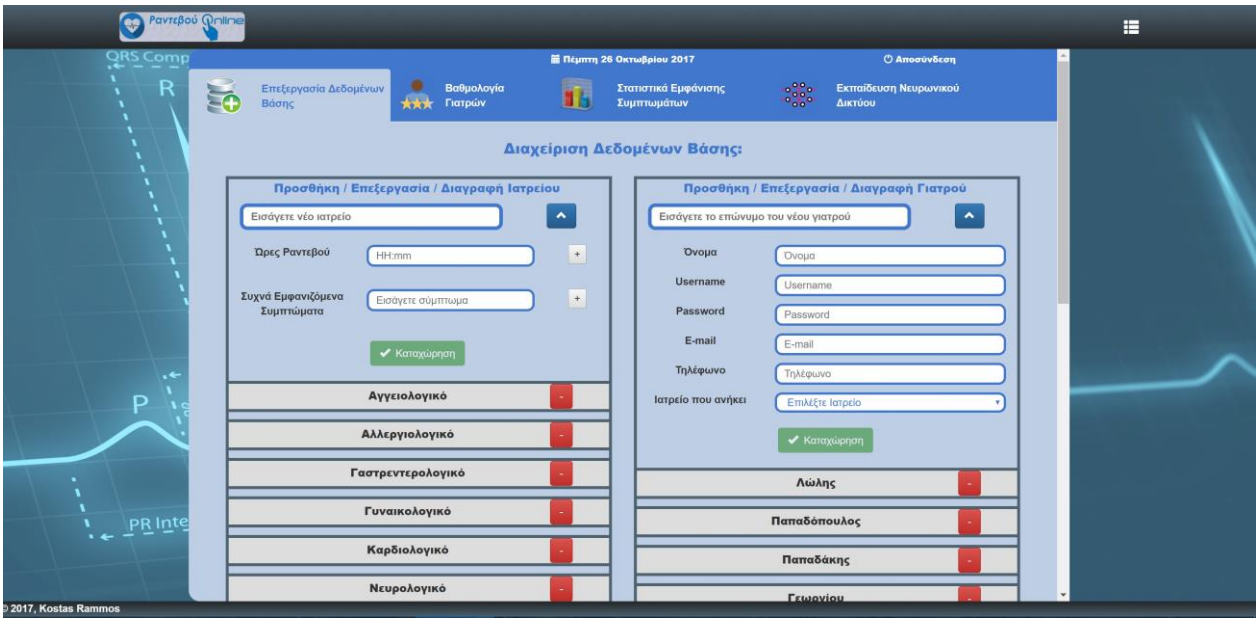
΄Υστερα από την επιτυχή σύνδεση του διαχειριστή, ανοίγει η καρτέλα επεξεργασίας δεδομένων της βάσης (εικόνες 2.28, 2.29). Εκεί ο διαχειριστής δύναται να προσθέσει, επεξεργαστεί και διαγράψει δεδομένα από τη βάση. Η οθόνη χωρίζεται σε δύο τμήματα. Το ένα αφορά στη διαχείριση των ιατρείων και το δεύτερο στη διαχείριση των γιατρών. Και τα δύο είναι δομημένα με τον ίδιο τρόπο. Στο επάνω μέρος τους, έχει τοποθετηθεί ένα κενό πεδίο, στο οποίο ο διαχειριστής μπορεί να γράψει το όνομα ενός ιατρείου ή γιατρού, που θέλει να προσθέσει. Πατώντας το μπλε κουμπί «+», ανοίγουν οι επιπλέον πληροφορίες που πρέπει να καταχωρηθούν (εικόνα 2.30). Αν κάποιο από τα πεδία δεν συμπληρωθεί ή συμπληρωθεί λανθασμένα, το κουμπί «*Καταχώρηση*» δεν ενεργοποιείται και το περίγραμμα του λανθασμένου πεδίου γίνεται κόκκινο, για να ξέρει ο χρήστης που είναι το λάθος. Τα γκρι κουμπιά με το «+» πρέπει να πατηθούν για να προστεθεί η αντίστοιχη τιμή.



Εικόνα 2.28: *Διεπαφή διαχειριστή – Οθόνη επεξεργασίας δεδομένων βάσης για μεγάλες συσκευές.*

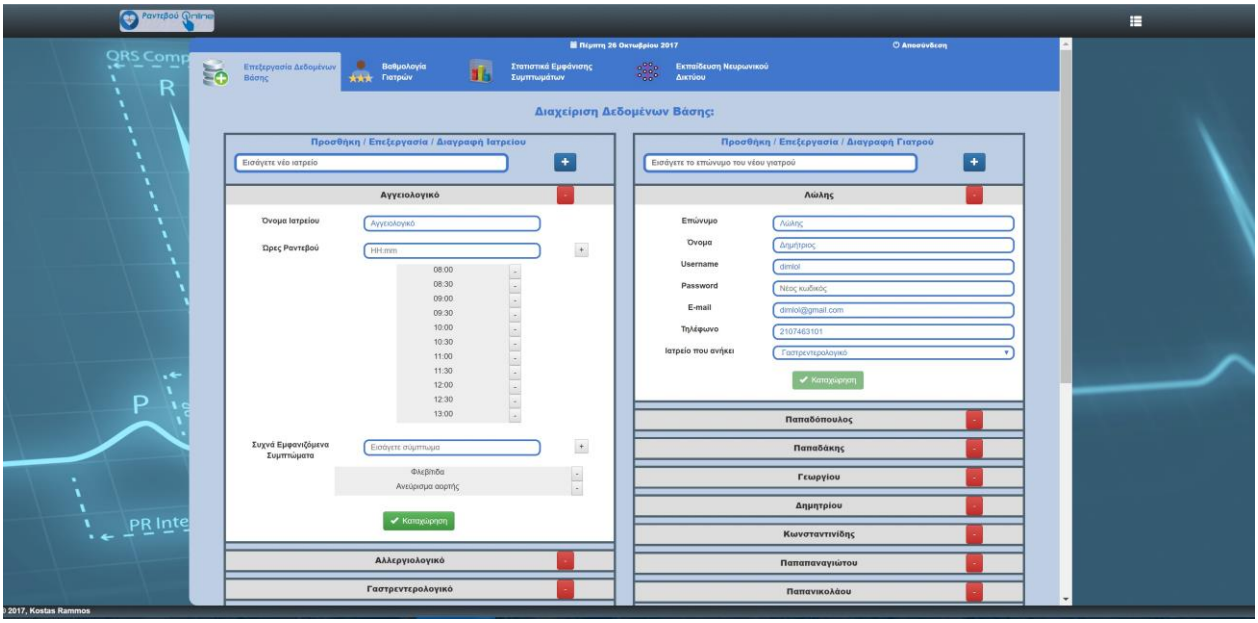


Εικόνα 2.29: *Διεπαφή διαχειριστή – Οθόνη επεξεργασίας δεδομένων βάσης για μικρές συσκευές.*



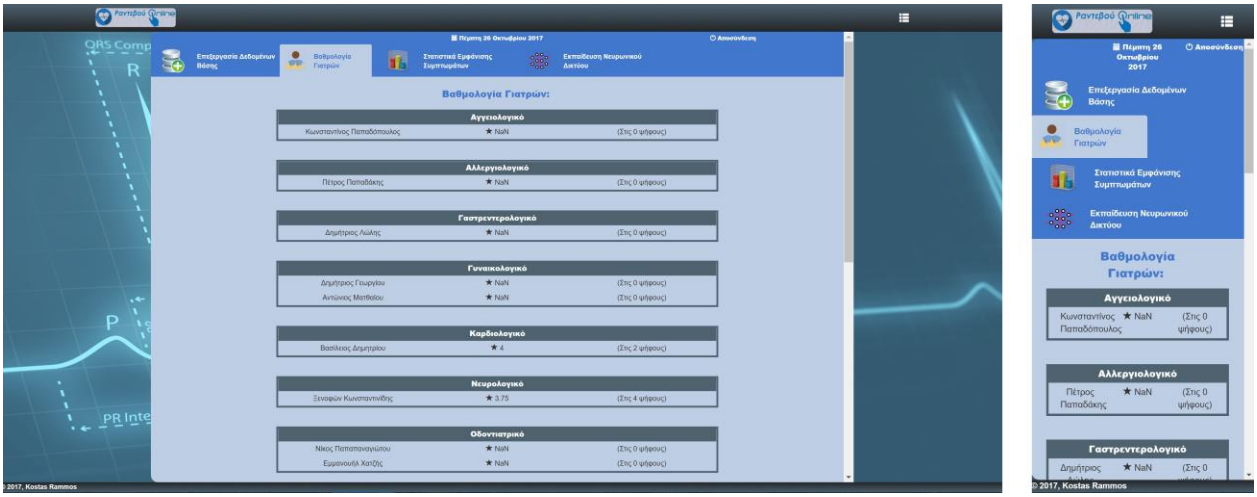
Εικόνα 2.30: *Διεπαφή διαχειριστή – Εισαγωγή νέων δεδομένων βάσης.*

Κάτω από τα πεδία εισαγωγής νέων δεδομένων, εμφανίζονται όλα τα ιατρεία και οι γιατροί της βάσης. Όταν πατηθεί κάποιος γιατρός ή ιατρείο, ανοίγουν οι επιπλέον πληροφορίες που περιλαμβάνει. Ομοίως με την εισαγωγή δεδομένων, τα πεδία αυτά πρέπει να μη μείνουν κενά και να συμπληρωθούν ορθά. Αν αυτό δε συμβεί, το κουμπί «*Καταχώρηση*» δεν ενεργοποιείται και η εφαρμογή δεν προχωράει. Επίσης, με το κόκκινο «-» πάνω σε κάθε καταχώρηση, ο διαχειριστής δύναται να τη διαγράψει από τη βάση. Θα ζητηθεί επιβεβαίωση, ώστε να αποφευχθεί η ακούσια διαγραφή της.



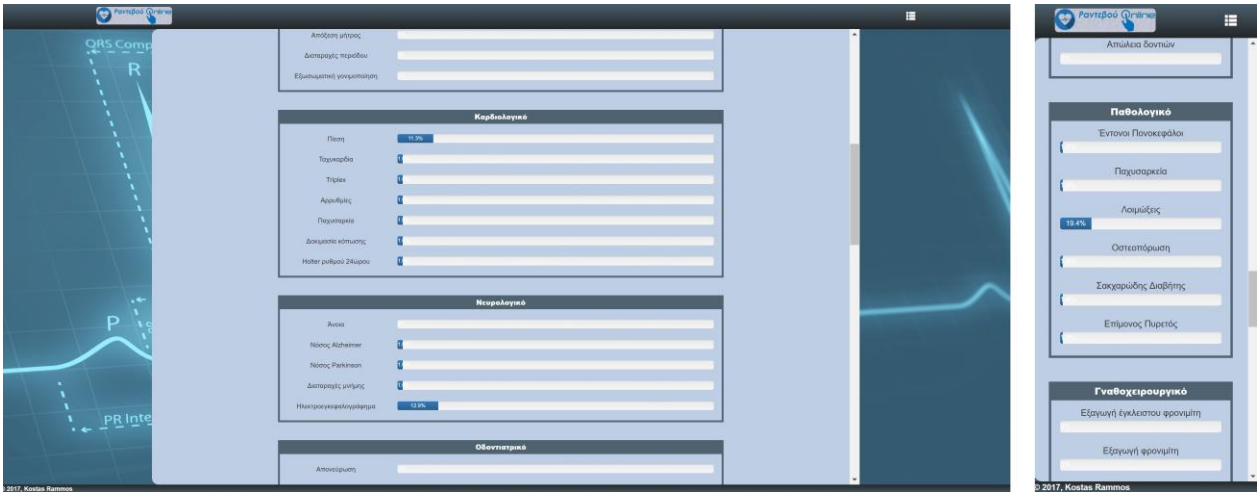
Εικόνα 2.31: *Διεπαφή διαχειριστή – Επεξεργασία νέων δεδομένων.*

Στην καρτέλα «*Βαθμολογία Γιατρών*», ο διαχειριστής μπορεί να δει τις βαθμολογίες που έχει συγκεντρώσει κάθε γιατρός, κάθε ιατρείου, από την αξιολόγηση των ασθενών (εικόνα 2.32). Ο σκοπός ύπαρξης της εν λόγω καρτέλας είναι να αποκτηθεί μία γενική εικόνα αναφορικά με την αποδοχή που αποκτά κάθε γιατρός από τους ασθενείς του και να βρεθούν αυτοί, που δεν κάνουν καλά τη δουλειά τους.



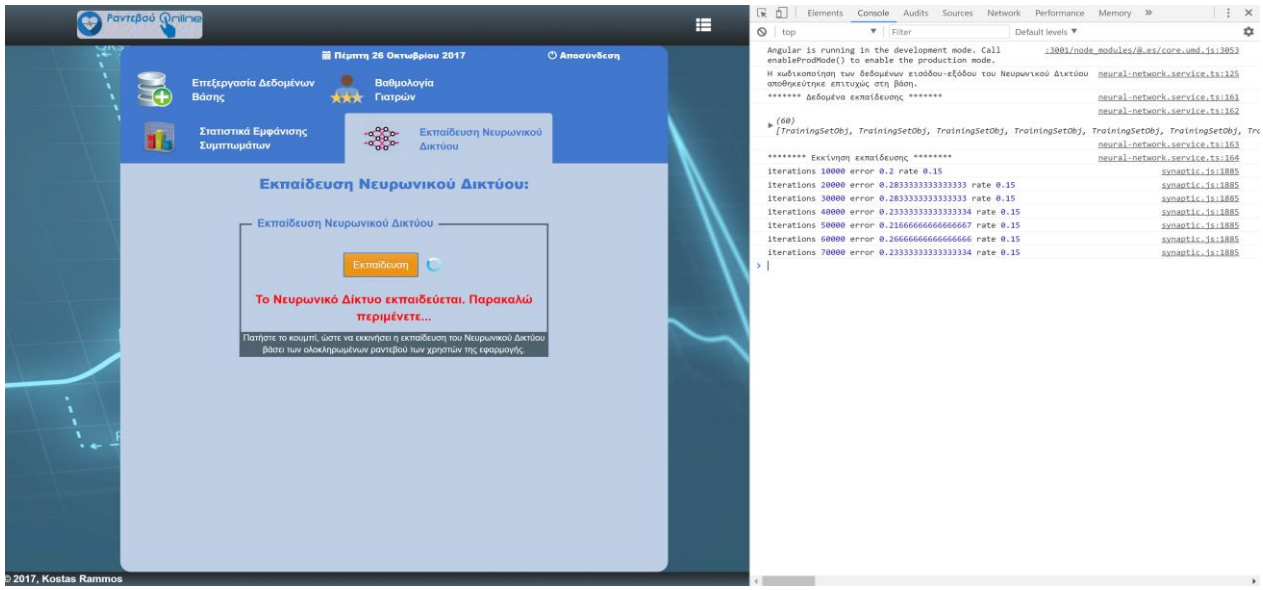
Εικόνα 2.32: *Διεπαφή διαχειριστή – Οθόνη προβολής βαθμολογίας γιατρών για μικρές και μεγάλες συσκευές.*

Στην καρτέλα «*Στατιστικά Εμφάνισης Συμπτωμάτων*» εμφανίζονται όλα τα συμπτώματα που είναι δηλωμένα σε κάθε ιατρείο (εικόνα 2.33). Για κάθε ένα από αυτά, υπολογίζεται το ποσοστό επί τοις εκατό των φορών εμφάνισης εκάστου συμπτώματος προς το συνολικό αριθμό των καταχωρημένων ραντεβού. Έτσι αποκτάται μία εικόνα σχετικά με ποια ιατρεία έχουν την περισσότερη δουλειά και ποια συμπτώματα εμφανίζονται πιο συχνά.

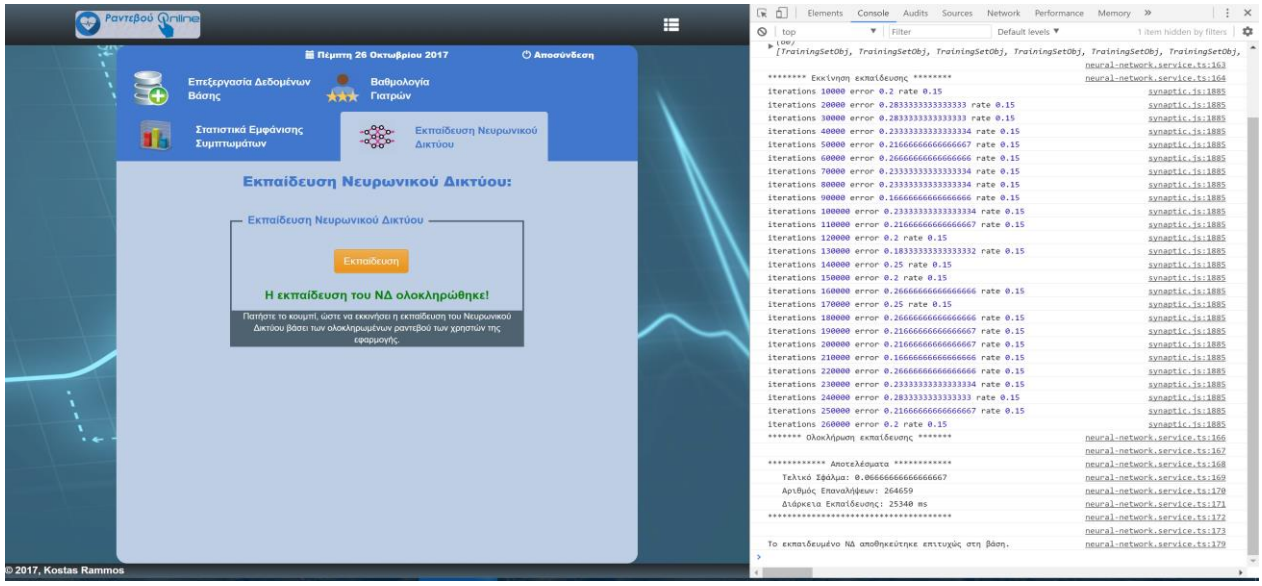


Εικόνα 2.33: *Διεπαφή διαχειριστή – Οθόνη στατιστικών στοιχείων συμπτωμάτων για μικρές και μεγάλες συσκευές.*

Τέλος, ο διαχειριστής διαθέτει μία καρτέλα για την εκπαίδευση του Νευρωνικού Δικτύου της εφαρμογής (εικόνες 2.34, 2.35). Η καρτέλα αυτή περιλαμβάνει το κουμπί, που πρέπει να πατήσει, για να εκκινήσει η εκπαίδευση και σε αυτή εμφανίζονται τα διάφορα μηνύματα, όπως «Το Νευρωνικό Δίκτυο εκπαιδεύεται» ή «Η Εκπαίδευση του ΝΔ ολοκληρώθηκε». Να επισημανθεί σε αυτό το σημείο ότι ο διαχειριστής έχει τη δυνατότητα να παρακολουθήσει την πορεία της εκπαίδευσης και τα τελικά αποτελέσματα αυτής (τελικό σφάλμα, διάρκεια εκπαίδευσης, αριθμός επαναλήψεων) μέσα από την κονσόλα του browser (Ctrl + Shift + I).



Εικόνα 2.34: *Διεπαφή διαχειριστή – Οθόνη κατά τη διάρκεια εκπαίδευσης του Νευρωνικού Δικτύου. Με Ctrl+Shift+I ανοίγει η κονσόλα του browser, μέσα από την οποία ο διαχειριστής μπορεί να παρακολουθήσει την πορεία της εκπαίδευσης.*



Εικόνα 2.35: *Διεπαφή διαχειριστή – Οθόνη κατά την ολοκλήρωση της εκπαίδευσης του Νευρωνικού Δικτύου. Με Ctrl+Shift+I ανοίγει η κονσόλα του browser, μέσα από την οποία ο διαχειριστής μπορεί να παρακολουθήσει τα τελικά αποτελέσματα της εκπαίδευσης.*

Η σελίδα αφέθηκε σκόπιμα κενή

ΚΕΦΑΛΑΙΟ 3 –

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

Η σελίδα αφέθηκε σκόπιμα κενή

ΚΕΦΑΛΑΙΟ 3 – ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

3.1 Εισαγωγή

Έχοντας ολοκληρώσει την παρουσίαση της εφαρμογής μας και παρέξει τις απαιτούμενες οδηγίες ως προς τη χρήση της, μπορούμε να περάσουμε στην ανάλυση των ιδιαίτερων χαρακτηριστικών της και του κώδικα, που υλοποιήθηκε. Σε αυτό το σημείο να αναφέρουμε ότι ολόκληρος ο κώδικας είναι custom και δεν έχουν χρησιμοποιηθεί πουθενά έτοιμα τμήματα κώδικα, είτε για κάποια λειτουργία, είτε για το User Interface (UI). Μοναδική εξαίρεση αποτελεί η υλοποίηση των υπολογισμών του Νευρωνικού Δικτύου, για την οποία χρησιμοποιήσαμε μία έτοιμη βιβλιοθήκη, που θα περιγράφουμε παρακάτω. Επιπλέον, να τονίσουμε ότι η εφαρμογή μας είναι πλήρως λειτουργική σε όλη της την έκταση χωρίς να υπάρχουν λειτουργίες που δεν δουλεύουν σωστά.

Το παρόν κεφάλαιο έχει δομηθεί ως εξής: στην υποενότητα 3.2, περιγράφουμε τα εργαλεία και τις γλώσσες προγραμματισμού, που χρησιμοποιήθηκαν για τη δημιουργία της εφαρμογής μας, όπως επίσης και τη διαδικασία εγκατάστασής τους. Στην υποενότητα 3.3, παρουσιάζουμε τα διαγράμματα UML Περιπτώσεων Χρήσης της εφαρμογής με σκοπό την καλύτερη κατανόησή της από τον αναγνώστη. Στην υποενότητα 3.4, αναφέρουμε το ρόλο κάθε αρχείου του φακέλου της εφαρμογής μας και παρουσιάζουμε ένα πίνακα με τον τύπο και τη λειτουργία κάθε API, που χρησιμοποιήθηκε στο Server. Στην υποενότητα 3.5, αναλύουμε αξιοσημείωτα τμήματα κώδικα της εφαρμογής μας. Την επόμενη υποενότητα 3.6, την αφιερώνουμε στη θεωρητική περιγραφή, υλοποίηση, διαμόρφωση και χρήση του Νευρωνικού Δικτύου. Τέλος, στην υποενότητα 3.7 κάνουμε μία σύντομη ανάλυση της εφαρμογής από την σκοπιά της Ασφάλειας Λογισμικού.

3.2 Γλώσσες και Εργαλεία για την Ανάπτυξη της Εφαρμογής

3.2.1 Γενικά

Η εφαρμογή μας, όντας διαδικτυακή, απαιτεί τον προγραμματισμό του User Interface για την αλληλεπίδραση με το χρήστη, τη χρήση μίας βάσης δεδομένων, για την αποθήκευση και την ανάκτηση πληροφοριών, και ενός Server, ο οποίος θα εκτελεί τις απαιτούμενες λειτουργίες και την επικοινωνία με τη βάση. Ο πιο διαδεδομένος και χρησιμοποιούμενος συνδυασμός των ανωτέρω, σήμερα, όπως φαίνεται και από τα στατιστικά που παρατίθενται από την εταιρεία «*Tiobe*» ([8]), είναι η server-side γλώσσα *PHP*, για front-end η χρήση *HTML*, *Javascript* και *CSS* και τέλος η χρήση της *MySQL* ως βάση δεδομένων.

Παρόλα αυτά, ο σκοπός της παρούσας μεταπτυχιακής διατριβής είναι η πραγμάτευση με λιγότερο διαδεδομένα και καταγεγραμμένα αντικείμενα, με απώτερο στόχο την επέκταση των γνώσεων του συγγραφέα και τη συνεισφορά στην αύξηση της βιβλιογραφίας των εν λόγω αντικειμένων. Στο πνεύμα αυτής της φιλοσοφίας, χρησιμοποιήσαμε τις κάτωθι γλώσσες/εργαλεία:

α. Για το front-end της εφαρμογής μας, πέραν των απαραίτητων *HTML* και *CSS*, χρησιμοποιήθηκε η πολύ διαδεδομένη γλώσσα *Javascript*. Η καινοτομία σε αυτό το σημείο είναι ότι ο κώδικάς μας δε γράφτηκε απευθείας σε *Javascript*, αλλά σε *Typescript* ([10]), ένα υπερσύνολο της *Javascript*. Μέσω του επίσης ραγδαία αναπτυσσόμενου framework της Google, *Angular2* ([9]), ο κώδικας σε *Typescript* γινόταν compile σε *Javascript* σε πραγματικό χρόνο.

β. Για το back-end της εφαρμογής μας χρησιμοποιήθηκε επίσης η γλώσσα *Javascript,* μέσω του *NodeJS* Server ([11]) και πάνω από αυτόν, το middleware framework *Express.js* ([12]). Όλες οι λειτουργίες που επιτελεί ο Server, υλοποιήθηκαν μέσω *REST API's.*

γ. Ως βάση δεδομένων, χρησιμοποιήθηκε η NoSQL βάση, *MongoDB* ([13]).

3.2.2 NodeJS Server – Περιγραφή και Εγκατάσταση

Η χρήση της γλώσσας *Javascript* στο back-end μιας εφαρμογής μέσω του *NodeJS* Server, ίσως δεν είναι ιδιαίτερα διαδεδομένη ακόμα, αλλά σίγουρα ολοένα και αυξάνεται. Η *Javascript* είναι μία γλώσσα με τεράστιες δυνατότητες και η οποία, την παρούσα στιγμή, κατέχει την έκτη ($6^{η}$) θέση ανάμεσα στις πιο διαδεδομένες γλώσσες προγραμματισμού ([8]).

Όσον αφορά το NodeJS Server, αυτό που τον κάνει να ξεχωρίζει από τους άλλους Servers είναι ότι λειτουργεί ασύγχρονα. Ενώ οι περισσότεροι Servers θα είναι σε ετοιμότητα να δεχτούν το επόμενο request μόνο όταν θα έχουν ολοκληρώσει και απαντήσει στο προηγούμενο, ο NodeJS Server είναι πάντα σε ετοιμότητα να δεχτεί νέα requests, όσο ταυτόχρονα επεξεργάζεται τα ήδη εισερχόμενα. Όταν η απάντηση από κάποιο από αυτά είναι έτοιμη, επιστρέφεται αμέσως στον Client. Συνεπώς, ο NodeJS Server είναι αρκετά γρήγορος, δεν δημιουργεί καθυστερήσεις στις εφαρμογές που τρέχουν σε αυτόν («bottleneck») και γενικά ενδείκνυται για εφαρμογές με μεγάλη ροή δεδομένων. Ενδεικτικά αναφέρουμε ότι τον χρησιμοποιούν τα *PayPal*, *LinkedIn*, *Dow Jones*, κ.τ.λ.

Συμπερασματικά, η χρήση του NodeJS Server και της Javascript στο back-end της εφαρμογής μας παρέχει σε αυτή μία υπεροχή σε σχέση με άλλες παρεμφερείς εφαρμογές, που χρησιμοποιούν τις κλασικές server-side γλώσσες.

Σε αυτό το σημείο, πρέπει να τονίσουμε και την σημασία της χρήσης των *API's (Application Programming Interface)* στο Server. Αυτά έχουν τη δυνατότητα να ανεξαρτητοποιούν τις λειτουργίες του Server από αυτές του Client. Ο Server δέχεται μία κλήση σε ένα συγκεκριμένο *URI (Uniform Resource Identifier)*, το οποίο αντιστοιχεί σε κάποια λειτουργία, επιτελεί την εν λόγω λειτουργία και επιστρέφει μία απάντηση σε μορφή JSON. Δεν ξέρει από τι συσκευή δέχεται αυτή την κλήση και φυσικά η λειτουργία του παραμένει αμετάβλητη, ανεξαρτήτως του είδους της συσκευής. Ο Client, από την άλλη, μπορεί να είναι ένας Η/Υ, ένα κινητό, κ.τ.λ. οπότε η απάντηση του Server πρέπει να διαχειριστεί κατάλληλα.

Συνεπώς, στο πλαίσιο της εφαρμογής μας, όλες οι λειτουργίες που επιτελούνται από το Server, υλοποιήθηκαν σε *REST API's (Representational State Transfer API's).* Το χαρακτηριστικό τους είναι ότι είναι «*stateless*», δηλαδή δεν αποθηκεύεται στο Server καμία πληροφορία του Client και έτσι κάθε request

πρέπει να περιλαμβάνει όλες τις απαραίτητες πληροφορίες για τη λειτουργία που πρόκειται να εκτελεστεί και την αυθεντικοποίηση του χρήστη

Ένα επίπεδο πάνω από το περιβάλλον ανάπτυξης του NodeJS Server, χρησιμοποιήθηκε το middleware *Express* ([12]), το οποίο ουσιαστικά παρεμβάλλεται ανάμεσα στον Server και τον Client και ελέγχει τη ροή των πληροφοριών. Ο κύριος ρόλος του είναι ότι κάνει τη χρήση των λειτουργιών και των δυνατοτήτων του NodeJS Server πολύ εύκολη για τον προγραμματιστή.

Ας δούμε τώρα πώς εγκαταστήσαμε και εκκινήσαμε τον Server μας. Αρχικά κατεβάσαμε από την επίσημη ιστοσελίδα του NodeJS ([11]) το πρόγραμμα και το κάναμε install. Από εκεί και πέρα ακολουθήσαμε την κάτωθι διαδικασία:

α. Δημιουργήσαμε ένα φάκελο για το Server μας με το όνομα «*online-rantevou-server*».

β. Μέσω του CMD των Windows, μεταφερθήκαμε στον εν λόγω φάκελο με την εντολή «*cd C:\xampp\htdocs\online-rantevou-server*» και εκτελέσαμε την εντολή «*npm init*». Αυτή δημιούργησε το αρχείο «*package.json*» με τις απαραίτητες πληροφορίες αναγνώρισης του project μας και διαχείρισης των εξαρτωμένων βιβλιοθηκών του.

γ. Κατόπιν, εγκαταστήσαμε στο Server όλες τις απαραίτητες βιβλιοθήκες για την υλοποίηση της εφαρμογής μας:

(1) «*npm install express –save»* (Το middleware Express, όπως περιγράψαμε ανωτέρω)

(2) «*npm install body-parser –save»* (Middleware που διαμορφώνει κατάλληλα την κυρίως πληροφορία (το σώμα – body) του εισερχόμενου request)

(3) «npm install cors –save» (Βιβλιοθήκη που χειρίζεται το πρόβλημα ότι ο Server και ο Client βρίσκονται στο ίδιο domain)

(4) «*npm install mongojs –save»* (Για την επικοινωνία με τη βάση δεδομένων)

(5) «*npm install jsonwebtoken –save»* (Για την διαμόρφωση του κατάλληλου token σύμφωνα με τη μέθοδο JSON Web Token ([14]))

(6) «*npm install bcrypt-nodejs –save»* (Για τη μετατροπή κωδικών σε hash μορφή και τον έλεγχό τους κατά την αυθεντικοποίηση)

(7) «*npm install express-jwt –save»* (Για την αυθεντικοποίηση των HTTP requests μέσω του JSON Web Token, που περιλαμβάνουν)

δ. Έπειτα, δημιουργήσαμε το javascript αρχείο μέσω του οποίου εκκινεί ο Server μας. Αυτό φαίνεται στην εικόνα 3.1.

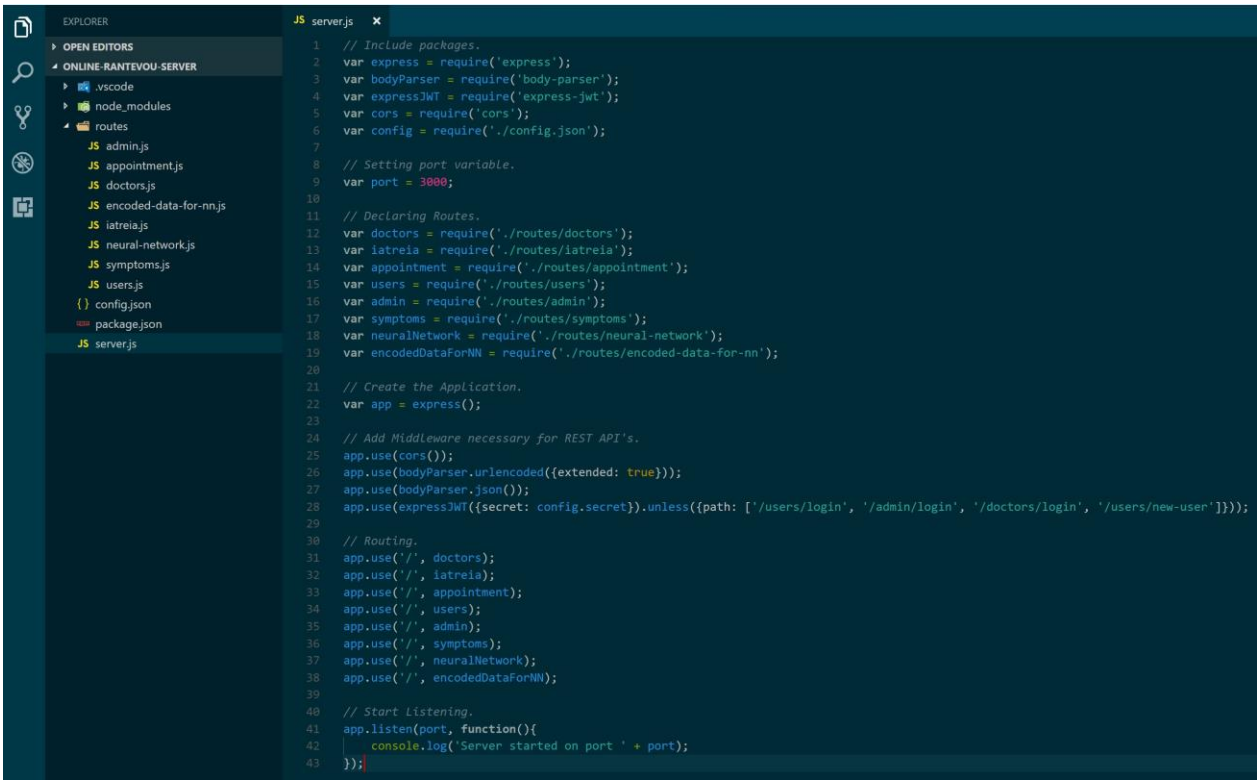
Όλα τα REST API's που δημιουργήσαμε, ώστε να υλοποιήσουμε τις απαιτούμενες λειτουργίες της εφαρμογής μας, είναι χωρισμένα σε javascript αρχεία, τα οποία βρίσκονται μέσα στο φάκελο «*routes*» του

project μας, όπως φαίνεται και στην εικόνα 3.1. Το πλήθος αυτών των αρχείων είναι ίσο με το πλήθος των Collections της βάσης, διότι κάθε ένα αρχείο συνδέεται με ένα Collection. Το απαιτούμενο κάθε φορά αρχείο εκκινεί μέσω της εντολής «app.use()» του αρχείου «server.js».

Κλείνοντας, για να εκκινήσουμε το Server μας, εκτελούμε στο CMD των Windows τις κάτωθι εντολές:

α. «*cd C:\xampp\htdocs\online-rantevou-server*»

β. «*node server.js*»



Εικόνα 3.1: *Το αρχείο «server.js» μέσω του οποίου εκκινεί ο Server μας.*

3.2.3 Angular2 – Περιγραφή και Εγκατάσταση

Για την ανάπτυξη του front-end της εφαρμογής μας, πέραν από την προφανή χρήση των *HTML* και *CSS*, χρησιμοποιήθηκε το framework της Google, *Angular2*. Αυτό είναι η μετεξέλιξη της AngularJS. Την παρούσα στιγμή, έχει κυκλοφορήσει και η έκδοση 4.4, στην οποία και έχουμε κάνει αναβάθμιση την εφαρμογή μας. Το υπόψη framework είναι ραγδαία αναπτυσσόμενο, με τεράστιες δυνατότητες, αλλά ταυτόχρονα και πολύ φιλικό προς τον προγραμματιστή. Η πιο διάσημη εφαρμογή που το χρησιμοποιεί είναι το Gmail.

Πρακτικά, η γλώσσα που υλοποιεί το front-end της εφαρμογής μας, δίνοντάς της τη δυνατότητα να τρέχει σε οποιοδήποτε Client και σε οποιοδήποτε browser, είναι η Javascript. Παρόλα αυτά, ο

προγραμματιστής γράφει κώδικα στη γλώσσα *Typescript,* ένα υπερσύνολο της Javascript ([10]) και ο οποίος μετατρέπεται, μέσω της Angular2, σε Javascript σε πραγματικό χρόνο. Η γλώσσα Typescript, η οποία είναι αντικειμενοστραφής, είναι πολύ απλή, αλλά ταυτόχρονα οι δυνατότητές της απεριόριστες. Γι' αυτό το λόγο και γνωρίζει ραγδαία ανάπτυξη.

Ας δούμε σε αυτό το σημείο πώς λειτουργεί μία εφαρμογή σε Angular2. Όπως είναι γνωστό, ο Server ψάχνει το αρχείο *index.html* για να εκκινήσει την εφαρμογή. Μέσα σε αυτό, δεν έχουμε δημιουργήσει κάποια πολύπλοκη σελίδα, αλλά μια απλή σελίδα, που δεν επιτελεί καμία λειτουργία. Πάνω σε αυτή τη σελίδα, εκκινεί η κυρίως εφαρμογή μας χάρη στα tags *<my-app></my-app>*. Ολόκληρη η εφαρμογή μας κρύβεται πίσω από αυτά τα tags. Επειδή, λοιπόν, αυτή τρέχει ουσιαστικά σε μόνο μία σελίδα, την *index.html*, ανήκει στην κατηγορία των «*single-page apps*».

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Online Ραντεβού</title>
        <link rel="icon" type="image/ico" href="img/icon.ico" />

        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="keywords" content="Rantevou, Online">
        <meta name="description" content="Online Rantevou">
        <meta name="author" content="Kostas Rammos">

        <base href="/">

        <!-- Bootstrap -->
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <link href="css/bootstrap-theme.min.css" rel="stylesheet">
        <link href="css/main-css.css" rel="stylesheet">

        <!-- Polyfill(s) for older browsers -->
        <script src="node_modules/core-js/client/shim.min.js"></script>
        <script src="node_modules/zone.js/dist/zone.js"></script>
        <script src="node_modules/systemjs/dist/system.src.js"></script>
        <script src="systemjs.config.js"></script>

        <script>
            System.import('main.js').catch(function(err){ console.error(err); });
        </script>
    </head>

    <body>
        <my-app>Loading ...</my-app>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
        <script src="js/bootstrap.min.js"></script>
    </body>
</html>
```

Εικόνα 3.2: *Το αρχείο «index.html» της εφαρμογής μας. Αυτή φορτώνεται μέσα από το <my-app></my-app>.*

Συνεχίζοντας, το εκάστοτε tag της μορφής *<my-app></my-app>* ορίζεται μέσα σε ένα *Component*, όπως ονομάζεται στην Angular2. Ένα Component αποτελείται από δύο (2) τμήματα. Το πρώτο είναι τα μεταδεδομένα (γραμμές κώδικα 6-11 στην εικόνα 3.3), τα οποία περιλαμβάνουν τον κώδικα HTML που θα

εκτελεστεί και τον αντίστοιχο CSS κώδικα (ή εναλλακτικά τα αρχεία που εμπεριέχουν τον κώδικα HTML και CSS), καθώς και το όνομα των tags, μέσω των οποίων θα καλείται, οπουδήποτε μέσα στην εφαρμογή, το εν λόγω Component. Το δεύτερο τμήμα είναι η κυρίως κλάση, εντός της οποίας ορίζονται οι, απαραίτητες για τη σελίδα, μεταβλητές και λειτουργίες, με τα αποτελέσματά τους να περνάνε με κατάλληλο τρόπο ({{…}}, Event Binding, Property Binding) στον κώδικα HTML και επομένως να απεικονίζονται στη σελίδα που βλέπει ο χρήστης της εφαρμογής.

Συμπερασματικά από τα ανωτέρω, για οτιδήποτε εμείς θέλουμε να απεικονίσουμε στην οθόνη, πρέπει να φτιάξουμε και το κατάλληλο Component, δηλαδή μία κλάση και το αντίστοιχο HTML Template και στυλ CSS. Ένα Component δε σημαίνει ότι κατασκευάζει απαραίτητα μία ολόκληρη σελίδα, αλλά μπορεί να κατασκευάζει και ένα μικρό τμήμα της σελίδας, όπως για παράδειγμα, ένα ημερολόγιο.

app.component.ts

```
import { Component, OnInit }      from '@angular/core';
import { Router }                 from '@angular/router';
import { AuthenticationService }  from '../../Services/authentication.service';


@Component({
  moduleId:     module.id,
  selector:     'my-app',
  templateUrl:  './app.component.html',
  styleUrls:    [ './app.component.css' ]
})


export class AppComponent implements OnInit {
    days_labels = ['Κυριακή', 'Δευτέρα', 'Τρίτη', 'Τετάρτη', 'Πέμπτη', 'Παρασκευή', 'Σάββατο'];
    months_labels = ['Ιανουαρίου', 'Φεβρουαρίου', 'Μαρτίου', 'Απριλίου',
                     'Μαΐου', 'Ιουνίου', 'Ιουλίου', 'Αυγούστου',
                     'Σεπτεμβρίου', 'Οκτωβρίου', 'Νοεμβρίου', 'Δεκεμβρίου'];
    dateToShow: string = '';
    selectedTab: number;
    admin_selectedTab: number;
    doctor_selectedTab: number;


    constructor(private authservice: AuthenticationService,
                private router: Router) { }


    ngOnInit() {
      // Ανακτούμε το τρέχον URL και θέτουμε την κατάλληλη τιμή στη μεταβλητή
      // selectedTab, ώστε να αλλάξουμε χρώμα στην επιλεγμένη καρτέλα στη Navbar.
      this.router.events.subscribe(() => {
        let patientURL = this.router.routerState.snapshot.url;
        if (patientURL === '/make-appointment') {
          this.selectedTab = 1;
        } else if (patientURL === '/pending-appointments') {
          this.selectedTab = 2;
        } else if (patientURL === '/appointments-history') {
          this.selectedTab = 3;
        }
      });

      // Το ίδιο για τις καρτέλες του Admin.
      this.router.events.subscribe(() => {
        let adminURL = this.router.routerState.snapshot.url;
        if (adminURL === '/admin/handle-data') {
          this.admin_selectedTab = 1;
        } else if (adminURL === '/admin/ratings') {
          this.admin_selectedTab = 2;
        } else if (adminURL === '/admin/symptoms-statistics') {
          this.admin_selectedTab = 3;
        } else if (adminURL === '/admin/neural-network') {
```

Εικόνα 3.3: *Παράδειγμα ενός Component από την εφαρμογή μας. Αποτελείται από δύο τμήματα: τα μεταδεδομένα (@Component({…})) και την κυρίως κλάση.*

Τέλος, η Angular2 μας δίνει τη δυνατότητα να δημιουργήσουμε μία κλάση αυτόνομη, με την έννοια ότι δεν συνδέεται με κώδικα HTML. Αυτή η κλάση ονομάζεται *Service*. Επιτελεί κάποιες λειτουργίες και μπορεί να χρησιμοποιηθεί οπουδήποτε μέσα στην εφαρμογή μας. Στην εικόνα 3.4, απεικονίζεται ενδεικτικά ένα Service.

patient.service.ts

```
import { Injectable }           from '@angular/core';
import { Http, Headers }        from '@angular/http';
import { Observable }           from 'rxjs/Observable';
import { Patient }              from '../Classes/patient';
import { Status }               from '../Classes/status';
import 'rxjs/add/operator/map';


@Injectable()
export class PatientService {
    private headers = new Headers();

    constructor(private http: Http) { }


    // Get user by ID
    getUserById(id: string): Observable<Patient> {
        let token = sessionStorage.getItem('token');

        if (token) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + token });
        }

        return this.http
                   .get('http://localhost:3000/users/' + id, { headers: this.headers })
                   .map(response => response.json() as Patient);
    }


    // Save a new patient
    saveNewPatient(newUser: Patient): Observable<Status> {
        this.headers = new Headers({'Content-Type': 'application/json'});

        return this.http
                   .post('http://localhost:3000/users/new-user', newUser, {headers: this.headers})
                   .map(res => res.json() as Status);
    }


    // Update patient's information
    updatePatient(edittedUser: Patient, id: string): Observable<Status> {
        let token = sessionStorage.getItem('token');

        if (token) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + token });
            this.headers.append('Content-Type', 'application/json');
        }

        return this.http
                   .put('http://localhost:3000/users/update-user/' + id, JSON.stringify(edittedUser), {headers: this.headers})
                   .map(res => res.json() as Status);
    }
```

Εικόνα 3.4: *Παράδειγμα ενός Service από την εφαρμογή μας.*

Όσον αφορά στην εγκατάσταση της Angular2, την παρούσα στιγμή, λόγω της αναβάθμισής της, η διαδικασία έχει αλλάξει. Τώρα μπορεί κάποιος να ξεκινήσει πολύ εύκολα και γρήγορα ένα project μέσω της διεπαφής γραμμής εντολών «*Angular CLI*». Τα βήματα παρουσιάζονται αναλυτικά στην επίσημη ιστοσελίδα της Angular2 ([9]). Ωστόσο, τη στιγμή που άρχισε η ανάπτυξη της προτεινόμενης στην παρούσα διατριβή εφαρμογής, η διαδικασία ήταν διαφορετική και η οποία παρουσιάζεται παρακάτω:

α. Κατεβάσαμε ένα φάκελο από το *Github,* με όλα τα απαιτούμενα αρχεία για την έναρξη ενός νέου project, όπως μας καθοδήγησε η επίσημη ιστοσελίδα της Angular2 εκείνη τη χρονική περίοδο.

β. Δημιουργήσαμε το φάκελο της εφαρμογής μας με όνομα «*online-rantevou*».

γ. Αντιγράψαμε σε αυτόν όλα τα αρχεία που κατεβάσαμε στο πρώτο βήμα.

δ. Μέσα από το CMD των Windows μεταφερθήκαμε στο φάκελο του project μας με την εντολή «*cd C:\xampp\htdocs\online-rantevou*» και εκτελέσαμε την εντολή «*npm init*», η οποία κατέβασε και εγκατέστησε όλες τις απαραίτητες εξαρτώμενες βιβλιοθήκες και ουσιαστικά αρχικοποίησε το project μας.

ε. Με την εντολή «*npm install synaptic –save»* στο CMD, εγκαταστήσαμε την απαιτούμενη βιβλιοθήκη για τη χρήση του Νευρωνικού Δικτύου στην εφαρμογή μας.

Συνεχίζοντας, όσον αφορά στην εκκίνηση της εφαρμογής μας, πρέπει πρώτα να εκκινήσουμε το NodeJS Server, όπως περιγράφεται στην ανωτέρω υποενότητα. Στη συνέχεια, μέσα από το CMD των Windows, εκτελούμε τις κάτωθι εντολές:

α. «*cd C:\xampp\htdocs\online-rantevou*»

β. «*npm start*»

Κατόπιν, ανοίγει αυτόματα ο browser μας και απεικονίζεται η εφαρμογή μας. Όπως έχει προαναφερθεί, αυτή ανανεώνεται σε πραγματικό χρόνο. Αυτό σημαίνει ότι τη στιγμή που θα σώσουμε το αρχείο στο οποίο εργαζόμαστε, οι αλλαγές θα απεικονιστούν στον browser αμέσως.

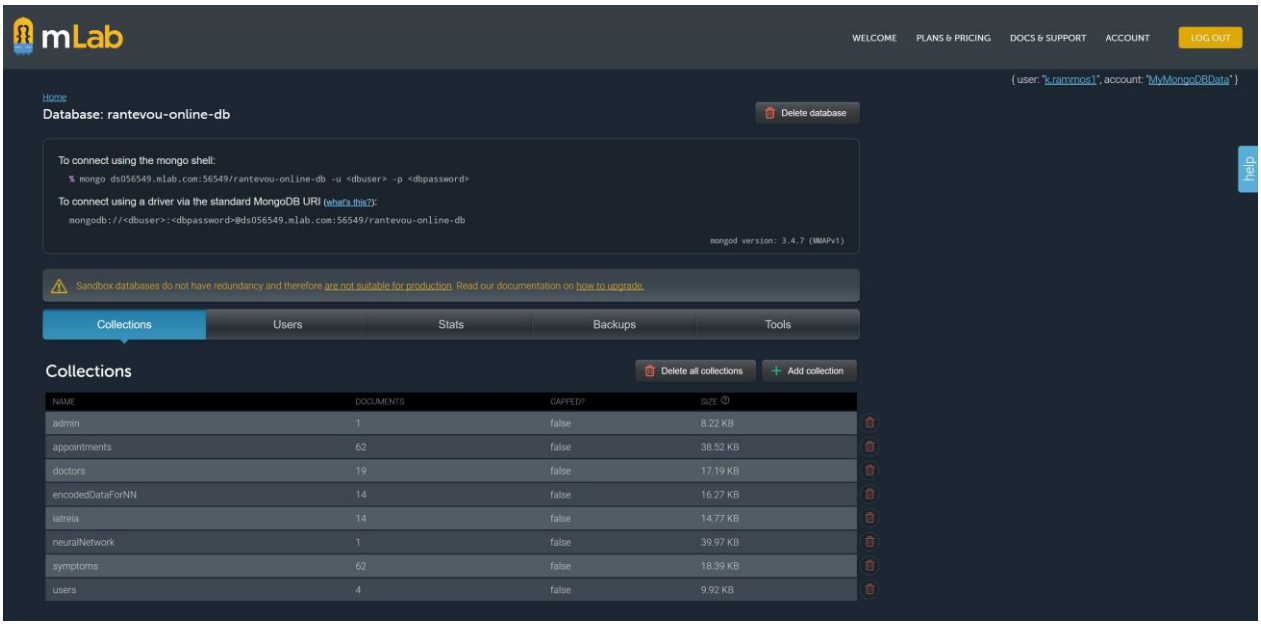
3.2.4 MongoDB

Ο τεράστιος όγκος δεδομένων, η μεγάλη διαφορετικότητα σε αυτά, η ανάγκη για συνεχή επικοινωνία με τη βάση, έχουν ως αποτέλεσμα, στις μέρες μας, την χαμηλή απόδοση και ταχύτητα εφαρμογών που χρησιμοποιούν σχεσιακές βάσεις δεδομένων (RDBMS), αλλά και τη δυσκολία στην συντήρηση αυτών των βάσεων. Για τους ανωτέρω λόγους, γνωρίζουν σιγά σιγά μεγάλη ανάπτυξη οι NoSQL βάσεις.

Οι προαναφερθείσες βάσεις υλοποιούν ένα μοντέλο δεδομένων σε ζεύγη «Πεδίο-Τιμή». Έτσι, μπορούν σε αυτές να αποθηκευτούν μη-δομημένα ή ημιδομημένα δεδομένα, μεγάλης ποσότητας, πυκνής ροής, εύκολα και γρήγορα. Ακόμα η ανάκτηση της ζητούμενης πληροφορίας γίνεται πολύ γρήγορα και με δυνατότητα για πολύπλοκα ερωτήματα.

Η *MongoDB* είναι μία βάση NoSQL, υψηλής απόδοσης. Αποτελείται από *Collections,* που, αν προσπαθούσαμε να τα παραλληλίσουμε με μία SQL βάση, θα λέγαμε ότι είναι οι αντίστοιχοι «πίνακες», και κάθε Collection από *Documents,* τις αντίστοιχες «εγγραφές» σε μία SQL βάση. Τα Documents αποτελούνται από ζεύγη δεδομένων (πεδίο-τιμή, όπως προαναφέρθηκε) και αποθηκεύονται σε μορφή *BSON* (Binary JSON). Η τιμή κάθε πεδίου μπορεί να είναι νούμερο, κείμενο, πίνακας νούμερων ή κειμένου, πίνακας άλλων Documents.

Για την εφαρμογή μας στήθηκε μία online βάση δεδομένων, την οποία ονομάσαμε «*rantevou-online-db*». Στην εικόνα 3.5, φαίνονται τα Collections από τα οποία αποτελείται.



Εικόνα 3.5: *Τα Collections της MongoDB βάσης της εφαρμογής μας.*

3.3 Διαγράμματα UML

3.3.1 Γενικά για τη γλώσσα UML

Η γλώσσα UML (Unified Modeling Language) δεν είναι μία γλώσσα προγραμματισμού, αλλά μία γλώσσα μοντελοποίησης και η οποία χρησιμοποιείται για τον προσδιορισμό, την απεικόνιση, την ανάπτυξη και την τεκμηρίωση των δομικών συστατικών ενός συστήματος, για παράδειγμα μίας εφαρμογής. Ουσιαστικά, αποτελείται από ένα σύνολο διαγραμμάτων, τα οποία μπορούν να καθορίσουν επακριβώς πώς λειτουργεί μία εφαρμογή και να συμβάλλουν στην ομαλή ροή της διαδικασίας «Ανάλυση Απαιτήσεων – Σχεδιασμός – Υλοποίηση – Τεκμηρίωση» αυτής. Τα εν λόγω διαγράμματα παρουσιάζονται παρακάτω:

α. Διαγράμματα Περιπτώσεων Χρήσης

β. Διαγράμματα Τάξεων

γ. Διαγράμματα Αντικειμένων

δ. Διαγράμματα Συνεργασίας

ε. Διαγράμματα Σειράς

στ Διαγράμματα Δραστηριοτήτων

ζ. Διαγράμματα Καταστάσεων

η. Διαγράμματα Εξαρτημάτων

θ. Διαγράμματα Διανομής

Στο πλαίσιο της παρούσας μεταπτυχιακής διατριβής, δεν είναι ο σκοπός μας η δημιουργία και παρουσίαση όλων των προαναφερθέντων διαγραμμάτων για όλες τις υποπεριπτώσεις λειτουργίας της εφαρμογής. Παρόλα αυτά, στο πνεύμα της επιδίωξης για καλύτερη κατανόησή της από τον αναγνώστη, θα παρουσιάσουμε μόνο τα διαγράμματα Περιπτώσεων Χρήσης.

3.3.2 Διαγράμματα Περιπτώσεων Χρήσης

Τα εν λόγω διαγράμματα περιγράφουν τη συμπεριφορά ενός συστήματος από την οπτική γωνία ενός χρήστη και απεικονίζουν τα όρια του συστήματος με το περιβάλλον του. Ουσιαστικά, παρουσιάζουν μία «υψηλού-επιπέδου» άποψη του συστήματος, μία απλή γραφική αναπαράσταση του τι μπορεί να κάνει το σύστημα. Έτσι, η χρήση τους ενδείκνυται για την παρουσίαση της εφαρμογής μας και την καλύτερη κατανόησή της.

Ένα διάγραμμα περιπτώσεων χρήσης αποτελείται από τις ίδιες τις περιπτώσεις χρήσης και έναν ή περισσότερους ενεργοποιούς (actors). Ο ενεργοποιός είναι ο εξωτερικός χρήστης, που δύναται να χρησιμοποιήσει ορισμένες από τις λειτουργίες του συστήματος και άρα να ενεργοποιήσει κάποιες από τις περιπτώσεις χρήσης. Για παράδειγμα, ενεργοποιός μπορεί να θεωρηθεί ο διαχειριστής ενός συστήματος, ο απλός επισκέπτης της εφαρμογής, ο συνδεδεμένος χρήστης, κ.λ.π. Ακόμα, ως ενεργοποιό, μπορούμε να θεωρήσουμε ένα άλλο σύστημα, που αλληλεπιδρά με το υπό εξέταση σύστημα, ή ακόμα και κάποια συσκευή hardware, που αποτελεί μέρος του συστήματος, χωρίς όμως να είναι ο κύριος υπολογιστής.

Επιπλέον, η αλληλεπίδραση μεταξύ των ενεργοποιών και των περιπτώσεων χρήσης ορίζεται από τη UML ως «σχέση». Υπάρχουν τρία (3) είδη σχέσεων:

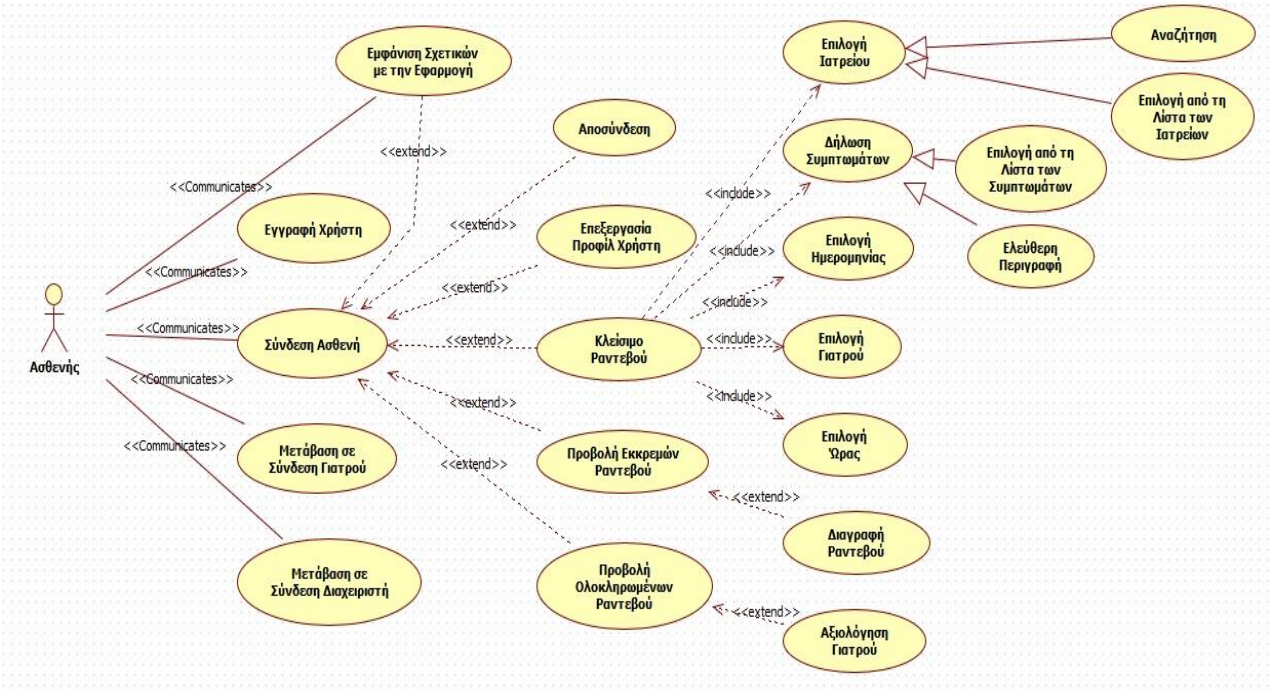
- Η σχέση «*επικοινωνεί (communicates)*», η οποία χρησιμοποιείται μόνο για τη σύνδεση ενός ενεργοποιού με κάποια περίπτωση χρήσης.

- Η σχέση «*χρησιμοποιεί (uses)*», η οποία συνδέει μία περίπτωση χρήσης με μία λειτουργία της εφαρμογής, όταν η επίτευξη της εν λόγω περίπτωσης, απαιτεί τη χρήση αυτής της λειτουργίας. Την εν λόγω λειτουργία την συμβολίζουμε ως μία ξεχωριστή περίπτωση χρήσης, γιατί αυτή μπορεί να χρησιμοποιείται από αρκετές ακόμα περιπτώσεις χρήσης και έτσι αποφεύγουμε να την σχεδιάσουμε πολλές φορές.

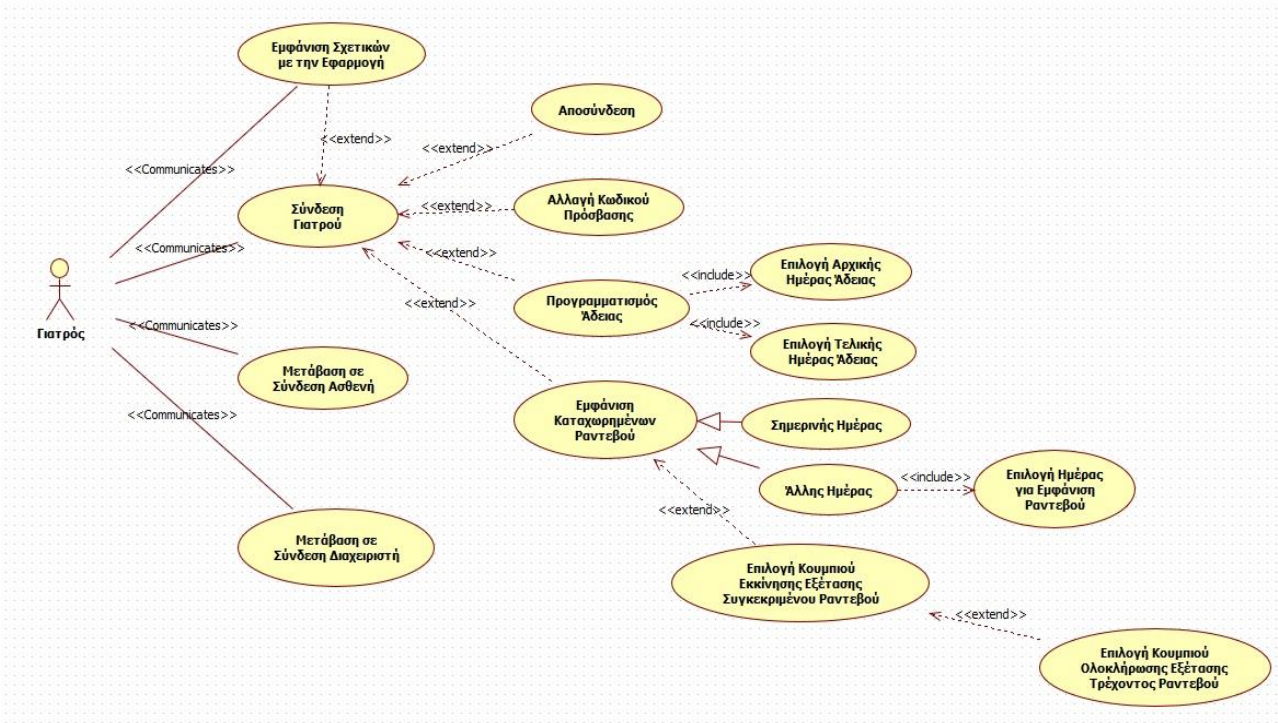
- Η σχέση «*επεκτείνει (extends)*», η οποία συνδέει μία περίπτωση χρήσης με μία άλλη περίπτωση χρήσης, η οποία δεν είναι σίγουρο ότι θα ενεργοποιηθεί. Η ενεργοποίηση αυτής εξαρτάται από τις επιλογές του χρήστη ή από την ύπαρξη κάποιας ειδικής συνθήκης στο σύστημα.

Ολοκληρώνοντας τη σύντομη περιγραφή μας πάνω στα διαγράμματα περιπτώσεων χρήσης, θα προχωρήσουμε στην παρουσίασή τους. Παραθέτουμε τρία (3) διαγράμματα με τις περιπτώσεις χρήσης

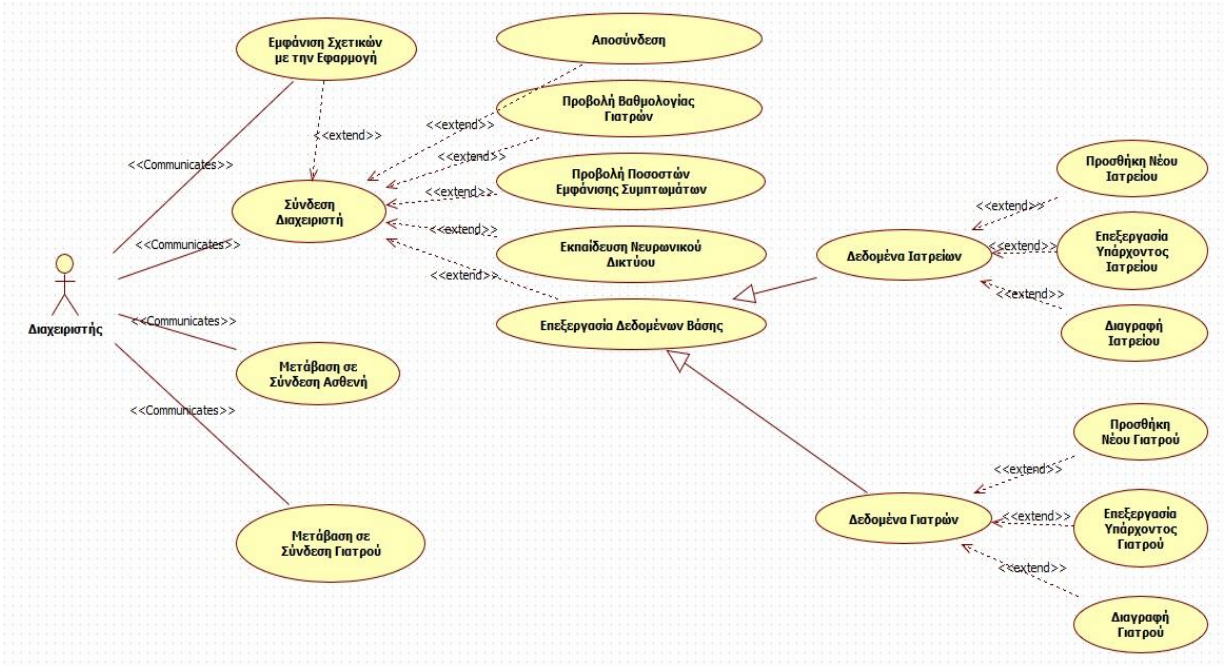
καθενός εκ των τριών (3) ενεργοποιών, που περιλαμβάνει το σύστημά μας, δηλαδή ένα διάγραμμα για τον Ασθενή, ένα για τον Γιατρό και ένα για το Διαχειριστή του συστήματος.



Εικόνα 3.6: *Διάγραμμα Περιπτώσεων Χρήσης για τον Ασθενή.*



Εικόνα 3.7: *Διάγραμμα Περιπτώσεων Χρήσης για τον Γιατρό.*



Εικόνα 3.8: *Διάγραμμα Περιπτώσεων Χρήσης για το Διαχειριστή.*

3.4 Δομή Αρχείων Εφαρμογής

3.4.1 Γενικά

Έχοντας ήδη αναφερθεί στις γλώσσες προγραμματισμού και στα εργαλεία που χρησιμοποιήσαμε για να αναπτύξουμε την εφαρμογή μας, αλλά και πάρει μία ιδέα από τον τρόπο λειτουργίας της, σε αυτή την ενότητα, θα αναλύσουμε τη δομή των αρχείων της και το περιεχόμενο καθενός από αυτά. Τόσο για τα αρχεία του Server, όσο και για τα αρχεία του Client, θα δούμε ποια είναι αυτά που παράγονται από την Angular2 και τον NodeJS Server αυτόματα, ποια είναι τα δικά μας αρχεία, πώς αυτά έχουν ταξινομηθεί σε φακέλους, τι ρόλο επιτελεί κάθε ένα και θα παρουσιάσουμε πίνακες που περιγράφουν τον τύπο και τη λειτουργία κάθε REST API του Server.

3.4.2 Δομή Αρχείων Server

Όλα τα αρχεία του Server βρίσκονται κάτω από τον φάκελο «*online-rantevou-server*» (εικόνα 3.9), όπως τον έχουμε ονομάσει. Παρακάτω, τα παρουσιάζουμε αναλυτικά:

- Ο φάκελος «*.vscode*», ο οποίος παράγεται από τον NodeJS Server αυτόματα κατά τη διαδικασία αρχικοποίησης μίας νέας εφαρμογής, περιλαμβάνει το αρχείο «*settings.json*». Σε αυτό μπορούμε, σε μορφή ζευγαριών «κλειδί-τιμή», να συμπεριλάβουμε κάποιες ρυθμίσεις που αφορούν στην εφαρμογή μας και οι οποίες θα αντικαταστήσουν τις προεπιλεγμένες. Ο σκοπός αυτού του αρχείου είναι να έχουμε τις custom ρυθμίσεις μας συγκεντρωμένες σε ένα μέρος.

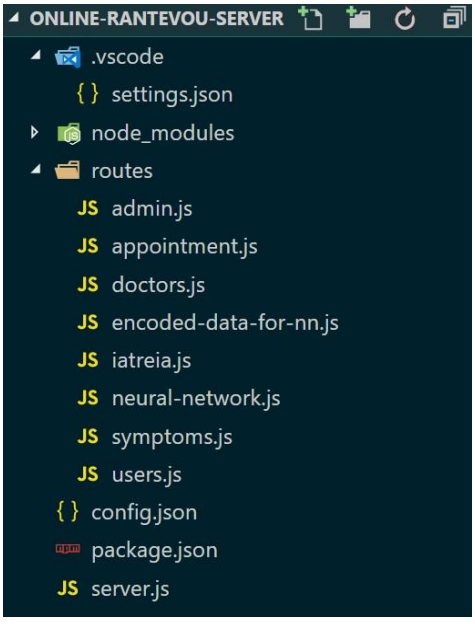
- Ο φάκελος «*node_modules*» περιλαμβάνει όλες τις βιβλιοθήκες του NodeJS και εγκαθίσταται αυτόματα κατά την αρχικοποίηση ενός νέου project.

- Ο φάκελος «*routes*» αποτελείται από τα δικά μας αρχεία, που επιτελούν όλες τις λειτουργίες τις εφαρμογής. Με άλλα λόγια, όλα τα REST API's του Server μας ορίζονται στα αρχεία μέσα σε αυτό το φάκελο. Κάθε ένα από αυτά συνδέεται με ένα συγκεκριμένο collection της βάσης. Αναλυτικά, το περιεχόμενο κάθε αρχείου, θα το δούμε παρακάτω.

- Το αρχείο «*config.json*» δημιουργήθηκε από εμάς και ορίζει ένα μυστικό κλειδί, το οποίο χρησιμοποιείται από την βιβλιοθήκη «*jsonwebtoken*» για την παραγωγή του token, που θα εξυπηρετήσει στην αυθεντικοποίηση κάθε χρήστη της εφαρμογής.

- Το αρχείο «*package.json*» δημιουργήθηκε αυτόματα και ορίζει σημαντικές παραμέτρους, χωρίς τις οποίες η εφαρμογή μας δε θα μπορούσε να λειτουργήσει. Ενδεικτικά αναφέρουμε ότι ορίζει το όνομα του φακέλου που εμπεριέχει την εφαρμογή μας, την έκδοσή της, το αρχείο που θα εκτελέσει πρώτα ο Server, τις εξαρτώμενες βιβλιοθήκες κ.α.

- Το αρχείο «*server.js*» είναι το αρχείο που θα εκτελεστεί πρώτα κατά την εκκίνηση του Server. Δημιουργήθηκε από εμάς. Ορίζει την πόρτα του «*http://localhost*», στην οποία θα τρέξει ο Server (έχουμε ορίσει την πόρτα *3000*), ορίζει όλα τα αρχεία που περιλαμβάνουν τα REST API's της εφαρμογής μας, εκτελεί τη διαδικασία της αυθεντικοποίησης του χρήστη μέσω της βιβλιοθήκης «*express-jwt*» σε κάθε εισερχόμενο request και ταυτόχρονα εξαιρεί από αυτή τη διαδικασία τα requests για login και register, αφού εξ' ορισμού ο χρήστης, εκείνη τη στιγμή, δε θα έχει δημιουργημένο token. Τέλος εκκινεί την εφαρμογή με τη βοήθεια του middleware Express και τη θέτει σε κατάσταση «Listening» για κάθε εισερχόμενο request.



Εικόνα 3.9: *Η δομή των αρχείων κάτω από το φάκελο «online-rantevou-server».*

Θα παρουσιάσουμε σε αυτό το σημείο αναλυτικά τι λειτουργία επιτελεί κάθε αρχείο του προαναφερθέντος φακέλου «*routes*»:

Αρχείο	admin.js	
Collection στη ΒΔ, που αντιστοιχεί	admin	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/admin/login	POST	Αυθεντικοποίηση του διαχειριστή της εφαρμογής και παραγωγή JSON Web Token. Οι δοθείσες τιμές username και password βρίσκονται μέσα στο σώμα (body) του request.

Αρχείο	users.js	
Collection στη ΒΔ, που αντιστοιχεί	users	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/user/login	POST	Αυθεντικοποίηση του χρήστη της εφαρμογής και παραγωγή JSON Web Token. Οι δοθείσες τιμές βρίσκονται μέσα στο σώμα του request.
/users/:id	GET	Επιστροφή όλων των δεδομένων του χρήστη με το id που δόθηκε.
/users/new-user	POST	Αποθήκευση νέου χρήστη στη βάση. Τα δεδομένα βρίσκονται στο body του request. Ο κωδικός μετατρέπεται σε hash μορφή.
/users/update-user/:id	PUT	Αποθήκευση πάνω στα υπάρχοντα πεδία του χρήστη με το id, που δόθηκε, των τροποποιημένων τιμών (βρίσκονται στο body). Ο κωδικός μετατρέπεται σε hash μορφή.

Αρχείο	doctors.js	
Collection στη ΒΔ, που αντιστοιχεί	doctors	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/doctors/login	POST	Αυθεντικοποίηση του γιατρού και παραγωγή JSON Web Token. Τιμές στο body του request.
/doctors/:iatreio/:date/:month/:year	GET	Επιστροφή των πεδίων name, lastname και iatreio των γιατρών του συγκεκριμένου ιατρείου, που δίνεται από την παράμετρο iatreio, οι οποίοι δεν έχουν άδεια την ημέρα που δίνεται στις υπόλοιπες παραμέτρους.
/doctors/:id/:firstDayOfLeave/:lastDay OfLeave	PUT	Ανανέωση των πεδίων startingDate και endingDate του γιατρού με το id που δίνεται, τα οποία δηλώνουν την άδειά τους.
/doctors/:name/:rate	PUT	Εύρεση συγκεκριμένου γιατρού ανάλογα με το ονοματεπώνυμό του και αύξηση της συνολικής βαθμολογίας του κατά rate και του πλήθους των ψήφων κατά 1.
/doctors/:iatreio	DELETE	Διαγραφή όλων των γιατρών του ιατρείου που δίνεται στην παράμετρο.
/doctors	GET	Επιστροφή όλων των γιατρών της βάσης.
/getDoctorNameAndLastname/:username	GET	Επιστροφή του ονοματεπωνύμου του γιατρού με username αυτό που δίνεται στην παράμετρο.

/new-doctor	POST	Εισαγωγή ενός νέου γιατρού στη βάση. Τα δεδομένα βρίσκονται στο body του request. Ο κωδικός μετατρέπεται σε hash μορφή. Η αρχική τιμή των πεδίων startingDate, endingDate, rating.totalVotes και rating.totalRate ορίζεται 0.
/doctorsById/:id	DELETE	Διαγραφή του συγκεκριμένου γιατρού με το id της παραμέτρου.
/update-doctor	PUT	Αποθήκευση πάνω στα υπάρχοντα πεδία του γιατρού με το id, που δόθηκε, των τροποποιημένων τιμών από το body του request. Ο κωδικός μετατρέπεται σε hash μορφή.
/doctor/change-password	PUT	Εύρεση του γιατρού με username αυτό που δίνεται στο body, και αντικατάσταση του password. Αποθηκεύεται σε hash μορφή.

Αρχείο	iatreia.js	
Collection στη ΒΔ, που αντιστοιχεί	iatreia	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/iatreia	GET	Επιστροφή όλων των πληροφοριών των ιατρείων.
/iatreia/:iatreio	GET	Επιστροφή όλων των πληροφοριών του ιατρείου με όνομα αυτό που δίνεται στην παράμετρο.
/iatreio-by-id/:id	GET	Επιστροφή όλων των πληροφοριών του ιατρείου με id αυτό που δίνεται στην παράμετρο.

/new-iatreio	POST	Αποθήκευση ενός νέου ιατρείου με τα δεδομένα από το body του request.
/iatreia/:id	DELETE	Διαγραφή του ιατρείου με id αυτό που δίνεται στην παράμετρο.
/iatreia	PUT	Αποθήκευση πάνω στα υπάρχοντα πεδία του ιατρείου με το id, που δόθηκε, των τροποποιημένων τιμών από το body του request.
/iatreiaAndDoctorsAggregate	GET	Εκτέλεση Aggregate στα collections «Iatreia» και «Doctors», ώστε να επιστραφεί το όνομα, το επώνυμο, η βαθμολογία και το πλήθος ψήφων κάθε γιατρού για όλα τα ιατρεία της βάσης.
/iatreiaAndSymptomsAggregate	GET	Εκτέλεση Aggregate στα collections «Iatreia» και «Symptoms», ώστε να επιστραφεί το σύμπτωμα και το πλήθος εμφανίσεών του για κάθε ιατρείο της βάσης.

Αρχείο	appointments.js	
Collection στη ΒΔ, που αντιστοιχεί	appointments	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/appointment	POST	Καταχώρηση νέου ραντεβού στη βάση. Τα δεδομένα περνάνε από το body του request.
/appointment/current/:iatreio/ :doctor/:date/:month/:year	GET	Επιστροφή ραντεβού για συγκεκριμένο ιατρείο, γιατρό και ημερομηνία, σύμφωνα με τις παραμέτρους.

/appointment/pending/:user/ :date/:month/:year	GET	Επιστροφή όλων των ραντεβού ενός συγκεκριμένου χρήστη με ημερομηνία μετά από τη δεδομένη στις παραμέτρους ημερομηνία.
/appointment/past/:user/ :date/:month/:year	GET	Επιστροφή όλων των ραντεβού ενός συγκεκριμένου χρήστη με ημερομηνία προγενέστερη από τη δεδομένη στις παραμέτρους.
/appointment/past-all/:date/:month/:year	GET	Επιστροφή όλων των ραντεβού για όλους τους χρήστες με ημερομηνία προγενέστερη από τη δεδομένη στις παραμέτρους
/appointment/:id	DELETE	Διαγραφή του ραντεβού με id αυτό που δίνεται ως παράμετρος.
/appointment/update-vote-field/:id/:rate	PUT	Ανανέωση της βαθμολογίας του γιατρού που εκτέλεσε το ραντεβού με id αυτό που δίνεται στις παραμέτρους.
/appointment/save-duration/:id/:duration	PUT	Ανανέωση της διάρκειας του ραντεβού με id αυτό που δίνεται στις παραμέτρους.
/appointment/total-number-of-appointments	GET	Επιστροφή του συνολικού αριθμού των αποθηκευμένων ραντεβού στη βάση.
/appointmentsAndUsersAggregate/ :doctorName/:doctorLastname/ :date/:month/:year	GET	Εκτέλεση Aggregate στα collections «Appointments» και «Users», ώστε να επιστραφούν τα ραντεβού του γιατρού και της ημέρας, που δίνονται στις παραμέτρους, μαζί με τα στοιχεία του χρήστη που έκλεισε το εν λόγω ραντεβού. Γίνεται ταξινόμηση των επιστρεφόμενων αποτελεσμάτων κατά ώρα.

Αρχείο	symptoms.js	
Collection στη ΒΔ, που αντιστοιχεί	symptoms	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/symptoms/:symptoms	PUT	Αύξηση κατά 1 του μετρητή κάθε συμπτώματος από αυτά που περνάνε με την παράμετρο symptoms (μπορούν να περάσουν πολλά συμπτώματα χωρισμένα μεταξύ τους με τελεία).
/symptoms	POST	Καταχώρηση νέου συμπτώματος στη βάση. Οι πληροφορίες περνάνε από το body του request.
/symptoms/:iatreio	DELETE	Διαγραφή όλων των συμπτωμά-των του ιατρείου με όνομα αυτό που δίνεται ως παράμετρος.
/symptoms/change-iatreio-label/:id	PUT	Αλλαγή του ονόματος του ιατρείου στα καταχωρημένα συμπτώματα. Παλιό και νέο όνομα περνάμε μέσω body του request.

Αρχείο	neural-network.js	
Collection στη ΒΔ, που αντιστοιχεί	neuralNetwork	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/save-neural-network	PUT	Αποθήκευση του εκπαιδευμένου ΝΔ στη βάση, σε JSON μορφή. Το ΝΔ περνάει μέσα από το body του request.
/retrieve-neural-network	GET	Επιστροφή του αποθηκευμένου στη βάση εκπαιδευμένου ΝΔ.

Αρχείο	encoded-data-for-nn.js	
Collection στη ΒΔ, που αντιστοιχεί	encodedDataForNN	
REST API's		
URI	**ΤΥΠΟΣ**	**ΛΕΙΤΟΥΡΓΙΑ**
/encoded-data-for-nn	DELETE	Διαγραφή όλων των δεδομένων κωδικοποίησης των εισόδων του ΝΔ.
/encoded-data-for-nn	POST	Αποθήκευση όλων των δεδομένων κωδικοποίησης του ΝΔ, όπως περνάνε από το body του request.
/encoded-data-for-nn/:iatreio	GET	Επιστροφή των δεδομένων κωδικοποίησης του ιατρείου με όνομα αυτό που δίνεται στην παράμετρο.

3.4.3 Δομή Αρχείων Client

Όλα τα αρχεία που αφορούν στην υλοποίηση του front-end της εφαρμογής μας βρίσκονται στο φάκελο «*online-rantevou*» και παρουσιάζονται παρακάτω:

- Ο φάκελος «*.vscode*» δημιουργείται αυτόματα και περιλαμβάνει το αρχείο «*settings.json*», στο οποίο ο χρήστης μπορεί να έχει συγκεντρωμένες όλες τις ρυθμίσεις του πάνω στην εφαρμογή, που θα αντικαταστήσουν τις προεπιλεγμένες.

- Ο φάκελος «*node_modules*» περιλαμβάνει όλες τις απαραίτητες βιβλιοθήκες της Angular2.

- Τα αρχεία «*.editorconfig*», «*bs-config.json*», «*debug.log*», «*package.json*» και «*tslint.json*» δημιουργούνται αυτόματα και αφορούν στη λειτουργία της εφαρμογής.

- Ο φάκελος «*src*» περιλαμβάνει τον πηγαίο κώδικα της εφαρμογής. Πιο συγκεκριμένα περιλαμβάνει:

  - Το αρχείο «*index.html*», το γνωστό δηλαδή αρχείο κάθε web εφαρμογής ή ιστοσελίδας, που ψάχνει ο Server για να τρέξει πρώτο. Σε αυτό ορίζονται ο τίτλος και το εικονίδιο της καρτέλας του browser που θα ανοίξει η εφαρμογή μας, διάφορα μεταδεδομένα, συνδέονται τα απαιτούμενα CSS και Javascript αρχεία. Τέλος, μέσα στο <body> του αρχείου, καλείται το *<my-app></my-app>*, το οποίο, χάρη στη φιλοσοφία και δομή της Angular2, κρύβει από πίσω τον κώδικα με τις λειτουργίες και το UI της εφαρμογής μας.

- Το φάκελο «*app*», ο οποίος περιλαμβάνει τα αρχεία που υλοποιούν τον κώδικα που κρύβεται πίσω από το <*my-app*></*my-app*> και τον οποίο θα δούμε αναλυτικά παρακάτω.

- Το φάκελο «*css*», ο οποίος περιλαμβάνει όλα τα απαιτούμενα αρχεία CSS για τη λειτουργία του Bootstrap, καθώς και ένα custom αρχείο CSS, που καθορίζει την εμφάνιση της σελίδας Index της εφαρμογής μας.

- Το φάκελο «*fonts*», ο οποίος περιλαμβάνει τα αρχεία που ορίζουν τα «*glyphicons*» της εφαρμογής μας.

- Το φάκελο «*img*», ο οποίος περιλαμβάνει όλες τις εικόνες και τα εικονίδια που χρησιμοποιούνται στην εφαρμογή μας.

- Το φάκελο «*js*», ο οποίος περιλαμβάνει όλα τα απαιτούμενα αρχεία Javascript της εφαρμογής μας, όπως αρχεία για τη λειτουργία του Bootstrap και της Angular2.

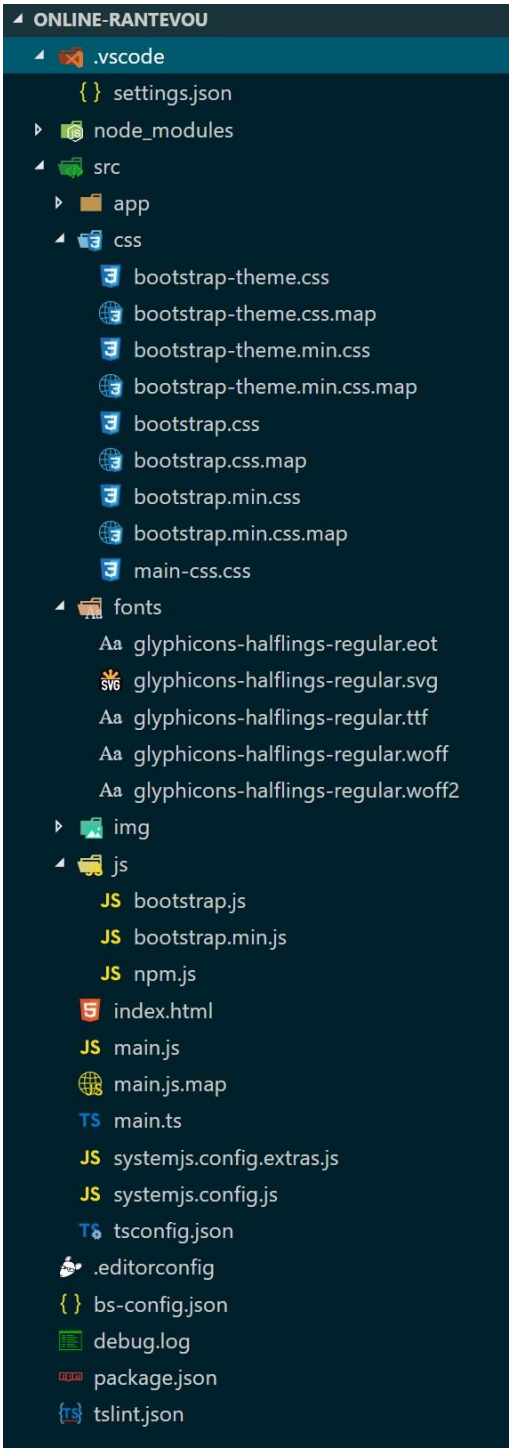
- Το αρχείο «*main.ts*», το οποίο δημιουργείται αυτόματα και είναι το σημείο εισόδου στην εφαρμογή μας. Εκκινεί το root module («*AppModule*») της εφαρμογής μας, ώστε αυτή να τρέξει στον browser. Τα αρχεία «*main.js*» και «*main.js.map*» παράγονται αυτόματα από την Angular2 και προκύπτουν από τη μετατροπή του αρχείου «*main.ts*» σε Javascript.

- Το αρχείο «*systemjs.config.js*», το οποίο αντιστοιχεί τα ονόματα των βιβλιοθηκών της Angular2, όπως τα χρησιμοποιούμε στα αρχεία μας (πχ @angular/core), με την πραγματική τους διαδρομή μέσα στο φάκελο node_modules (πχ node_modules/@angular/core/bundles/core.umd.js).

- Το αρχείο «*tsconfig.json*», το οποίο παράγεται αυτόματα και περιλαμβάνει ρυθμίσεις, σε μορφή json, για το compile της εφαρμογής μας.

Πρακτικά, η ουσία της Angular2 έγκειται στο ότι κάθε οθόνη της εφαρμογής μας και οι αντίστοιχες λειτουργίες, που αυτή επιτελεί, παράγονται από τα αρχεία «*.ts*», «*.html*» και «*.css*», που βρίσκονται μέσα στο φάκελο «*app*». Να σημειωθεί σε αυτό το σημείο ότι για κάθε αρχείο με κατάληξη «*.ts*», η Angular2 αυτόματα παράγει τα αντίστοιχα αρχεία «*.js»* και «*.js.map»,* τα οποία απαιτούνται, ώστε η εφαρμογή μας να τρέχει στον browser του Client. Οπότε, ο προαναφερθείς φάκελος «*app*» περιλαμβάνει:

- Το αρχείο «*app.module.ts*», το οποίο ουσιαστικά συνδέει τη σελίδα Index, την οποία εκκινεί πρώτα ο Server, με τα αρχεία «*.ts*», «*.html*» και «*.css*» του φακέλου «*app*», που έχουμε φτιάξει εμείς και που υλοποιούν τις λειτουργίες και το UI της εφαρμογής μας. Στο εν λόγω αρχείο δηλώνονται όλα τα «*Modules*», τα «*Components*» τα «*Services*», που θα χρησιμοποιήσει η εφαρμογή, καθώς και το Component, το οποίο θα τρέξει πρώτο.



Εικόνα 3.10: *Η δομή των αρχείων κάτω από το φάκελο «online-rantevou».*

• Το αρχείο «*app-routing.module.ts*», το οποίο ορίζει ένα route σε κάθε Component, δηλαδή ποιο URI θα μας οδηγεί στο συγκεκριμένο Component. Να σημειωθεί ότι η σειρά που έχουν οριστεί τα routes παίζει ρόλο, καθώς το σύστημα τα αναζητεί διαδοχικά από πάνω προς τα κάτω και αν δεν βρει ένα συγκεκριμένο Component, συνεχίζει στο αμέσως επόμενο.

• Το φάκελο «*Classes*». Ο υπόψη φάκελος περιλαμβάνει αρχεία, που κάθε ένα από αυτά ορίζει μία κλάση, που έχουμε κατασκευάσει εμείς και χρησιμοποιούμε στην εφαρμογή μας. Κάθε μία από αυτές τις κλάσεις ορίζει μόνο μεταβλητές και όχι μεθόδους. Κατά κύριο λόγο, οι μεταβλητές μιας κλάσης αντιστοιχούν ακριβώς στα πεδία της αντίστοιχης json απάντησης από το Server. Συνεπώς, δημιουργώντας αντικείμενα των εν λόγω κλάσεων, μπορούμε να διαχειριστούμε μέσα στην εφαρμογή μας τα δεδομένα από τις απαντήσεις του Server.

• Το φάκελο «*Animations*». Εδώ αποθηκεύονται τα αρχεία που υλοποιούν τα animations της εφαρμογής μας. Προς το παρόν, έχουμε δημιουργήσει μόνο ένα animation, το πεδίο επιλογής ιατρείου στο UI του ασθενή, το οποίο υλοποιείται από το αρχείο «*move-iatreia-div.animation.ts*».

• Το φάκελο «*Services*». Σε αυτό το φάκελο περιλαμβάνονται τα Services της εφαρμογής, δηλαδή, όπως έχει προαναφερθεί και στην περιγραφή της λειτουργίας της Angular2, των βοηθητικών αρχείων μόνο για την υλοποίηση λειτουργιών της εφαρμογής μας και όχι την υλοποίηση UI (ρόλος των Components). Ουσιαστικά, μέσα σε αυτά τα αρχεία ορίζονται κλάσεις, οι οποίες περιλαμβάνουν διάφορες μεθόδους. Κάθε μία μέθοδος, κατά κύριο λόγο, επιτελεί τις εξής λειτουργίες: ανακτά το αντίστοιχο token (του γιατρού, του ασθενή ή του admin, ανάλογα το request), δημιουργεί Authorization Header με το ανακτηθέν token και τελικά πραγματοποιεί ένα request σε κάποιο REST API του Server. Κάθε μία από τις ορισθέντες μεθόδους καλείται μία ή περισσότερες φορές μέσα στα Components. Συνεπώς, αφού η εν λόγω μέθοδος εκτελέσει το request και λάβει την απάντηση από το Server, την επιστρέφει σε μορφή JSON πίσω στο Component, για κατάλληλη διαχείρισή της. Να σημειωθεί σε αυτό το σημείο ότι για την υλοποίηση των requests χρησιμοποιούνται αντικείμενα της κλάσης «*Http*» της Angular2, οι μέθοδοι «*get()*», «*post()*», «*put()*», και «*delete()*» της εν λόγω κλάσης για επιστροφή, καταχώρηση, επεξεργασία ή διαγραφή δεδομένων, αντίστοιχα, και η μέθοδος «*map()*» για τη μετατροπή της απάντησης σε μορφή JSON και αντιστοίχισή της σε κάποια κλάση του φακέλου «*Classes*». Τέλος, να επισημανθεί ότι ο λόγος που η υλοποίηση των requests γίνεται μέσα στα Services, και όχι μέσα στα Components, είναι η φιλοσοφία να μη μπλέκουμε πολλές λειτουργίες μαζί σε μία μόνο μέθοδο. Κάθε μέθοδός μας επιδιώκουμε να επιτελεί μία γενικευμένη λειτουργία, ώστε αυτή να είναι επαναχρησιμοποιήσιμη, αλλά και τυχόν αλλαγές στο σχεδιασμό της εφαρμογής, να οδηγούν σε αλλαγές στον κώδικα σε ένα συγκεκριμένο τμήμα του και όχι σε πολλά, γεγονός που θα δυσκόλευε τη συντήρησή του.

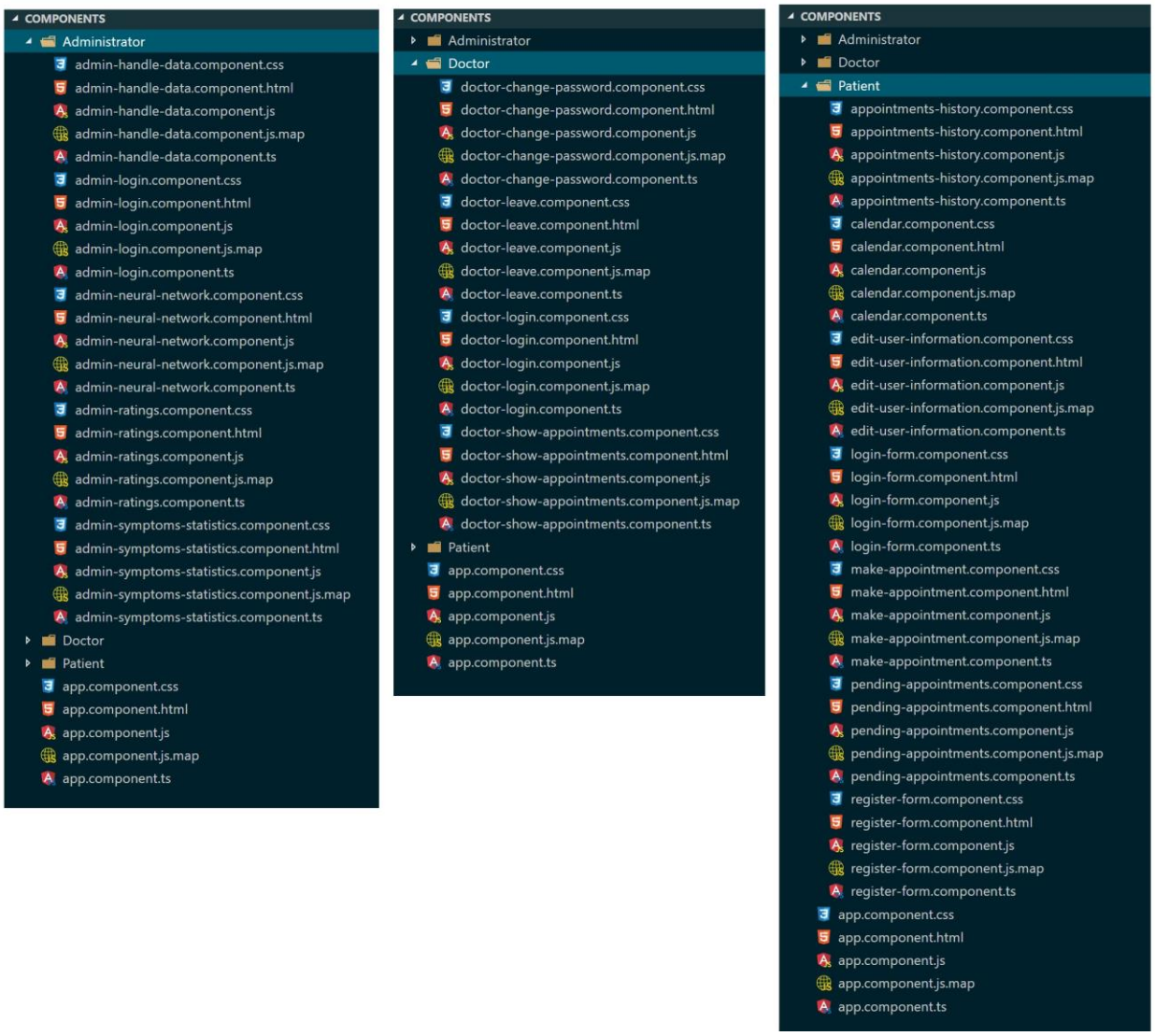
• Το φάκελο «*Components*». Αυτός περιλαμβάνει όλα τα απαιτούμενα αρχεία «*.ts*», «*.html*» και «*.css*», που έχουμε φτιάξει εμείς, για την υλοποίηση του UI και των αντίστοιχων λειτουργιών πίσω από αυτό, κάθε κατηγορίας χρήστη της εφαρμογής. Ειδικότερα, ο εν λόγω φάκελος περιλαμβάνει:

- Τα αρχεία «*app.component.ts*», «*app.component.html*» και «*app.component.css*». Αυτά είναι τα αρχεία που εκτελούνται, όταν η σελίδα «*index.html*» καλεί το *<my-app></my-app>.* Ενώ το αρχείο «*index.html*» υλοποιεί τη σελίδα που θα πατήσει πάνω η υπόλοιπη εφαρμογή και ορίζει την εικόνα στο background αυτής, τα προαναφερθέντα αρχεία έρχονται να προσθέσουν το «*Header*» στη σελίδα, το «*Footer*», τη «*Navigation Bar*», ό,τι δηλαδή τμήμα της σελίδας παραμένει σταθερό και αμετάβλητο σε κάθε οθόνη οποιουδήποτε χρήστη. Η χρήση του *<router-outlet></router-outlet>* είναι υπεύθυνη να εισάγει στο μεταβλητό τμήμα της σελίδας το κατάλληλο κάθε φορά Component για την αλληλεπίδραση της εφαρμογής με το χρήστη.

- Τον υποφάκελο «*Patient*». Αυτός περιλαμβάνει τα αρχεία που υλοποιούν κάθε Component του UI του ασθενή. Συγκεκριμένα, υλοποιούνται τρία (3) Components για τις τρεις καρτέλες της Navigation Bar («*make-appointment.component*», «*pending-appointments.component*», «*appointments-history.component*»), ένα (1) Component για τη σύνδεση του χρήστη («*login-form.component*»), ένα (1) Component για την εγγραφή του χρήστη («*register-form.component*»), ένα (1) Component για την επεξεργασία του προφίλ του χρήστη («*edit-user-information.component*») και τέλος ένα (1) Component για την υλοποίηση ενός ημερολογίου («*calendar.component*»), το οποίο χρησιμοποιείται μέσα στο «*make-appointment.component*», αλλά και επαναχρησιμοποιείται μέσα στο UI του γιατρού, αποδεικνύοντας με αυτό τον τρόπο τη σημασία της δημιουργίας γενικευμένων και επαναχρησιμοποιήσιμων Components.

- Τον υποφάκελο «*Doctor*». Αυτός περιλαμβάνει τα αρχεία που υλοποιούν κάθε Component του UI του γιατρού, δηλαδή τρία (3) Components για τις τρεις καρτέλες της Navigation Bar («*doctor-show-appointments.component*», «*doctor-leave.component*», «*doctor-change-password.component*») και ακόμα ένα (1) Component για τη σύνδεση του γιατρού («*doctor-login.component*»).

- Τον υποφάκελο «*Administrator*». Αυτός περιλαμβάνει τα αρχεία που υλοποιούν κάθε Component του UI του διαχειριστή της εφαρμογής, δηλαδή τέσσερα (4) Components για τις τέσσερις καρτέλες της Navigation Bar («*admin-handle-data.component*», «*admin-symptoms-statistics.component*», «*admin-ratings.component*», «*admin-neural-network.component*») και ακόμα ένα (1) Component για τη σύνδεση του διαχειριστή («*admin-login.component*»).



Εικόνα 3.11: *Η δομή των αρχείων κάτω από το φάκελο «Components».*

3.5 Υλοποίηση του Κώδικα

3.5.1 Δομή Κώδικα Αρχείων του Φακέλου «*Routes*» στο Server

Στην προηγούμενη ενότητα, παρουσιάστηκαν σε πίνακα όλα τα REST API's κάθε αρχείου του φακέλου «*routes*» του Server και είδαμε κάθε ένα τι λειτουργία επιτελεί. Σε αυτή την ενότητα, θα εξετάσουμε τον τρόπο με τον οποίο είναι κατασκευασμένο κάθε ένα από τα προαναφερθέντα αρχεία, ώστε να επιτελεί τις απαιτούμενες λειτουργίες του. Όλα τα αρχεία έχουν όμοια δομή. Ενδεικτικό παράδειγμα δίνεται στην εικόνα 3.12. Οπότε, σε κάθε αρχείο:

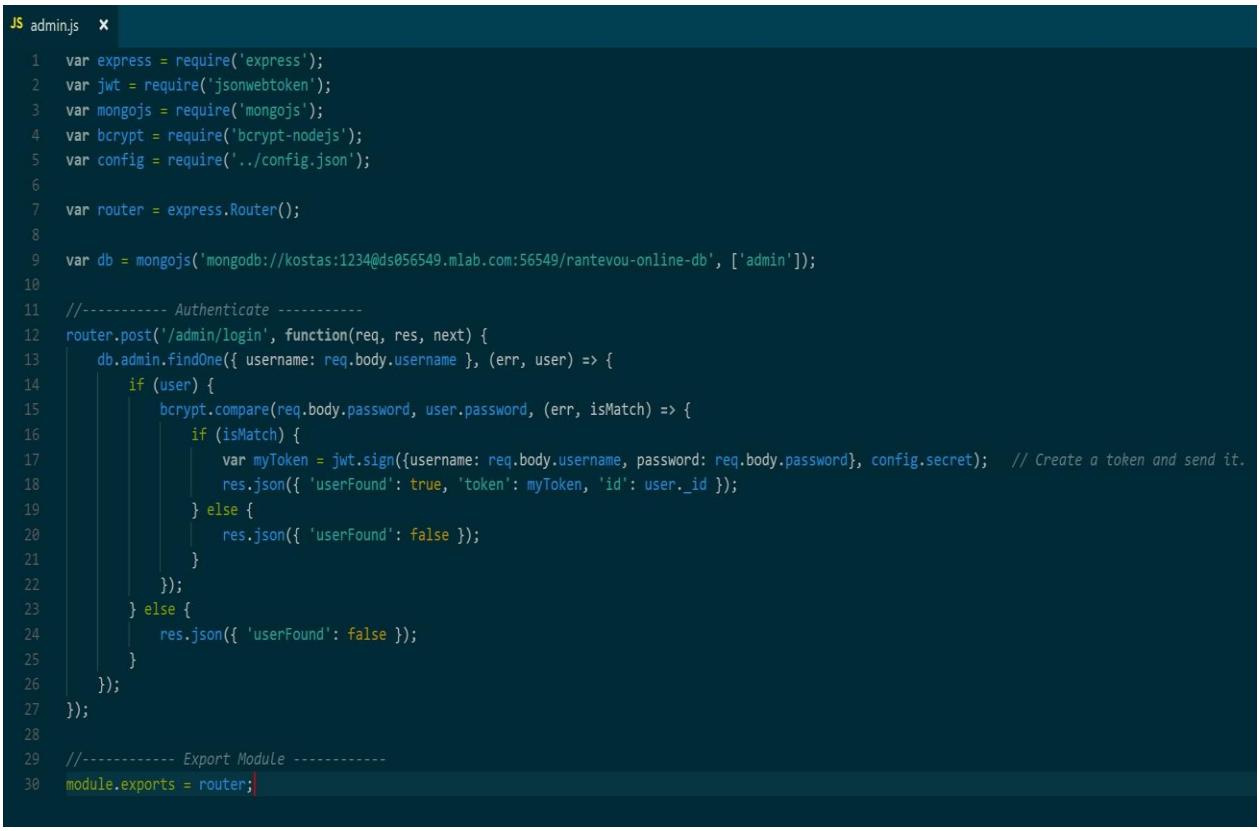
• Αρχικά, δημιουργούνται αντικείμενα, που συνδέουν το αρχείο με όλες τις απαιτούμενες για τη λειτουργία του βιβλιοθήκες. Για παράδειγμα, «*var express = require('express');*».

• Εν συνεχεία, καλούμε τη μέθοδο «*Router()*» του αντικειμένου «*express*», που ορίσαμε προηγουμένως, δημιουργώντας το αντικείμενο «*router*». Η διαδικασία αυτή απαιτείται από το χρησιμοποιούμενο middleware Express, με απώτερο σκοπό να μας διευκολύνει στη συνέχεια στην επικοινωνία με τη βάση μας.

• Επίσης, υπό την προϋπόθεση ότι έχουμε συμπεριλάβει τη βιβλιοθήκη «*mongojs*», ορίζουμε τη μεταβλητή «*db*», με την εντολή «*var db = mongojs('mongodb://kostas:1234@ds056549.mlab.com:56549/rantevou-online-db', ['admin']);*» την οποία χρησιμοποιούμε για τη σύνδεση με συγκεκριμένο Collection της βάσης. Στο συγκεκριμένο παράδειγμα, το Collection που χρησιμοποιείται είναι το «*admin*», αλλά στη θέση του μπορούσε να μπει οποιοδήποτε. Να επισημανθεί ότι κάθε αρχείο του φακέλου «*routes*» κάνει σύνδεση με μόνο ένα Collection, άρα η παραγόμενη μεταβλητή «*db*» χρησιμοποιείται σε όλη την έκταση του αρχείου.

• Αφού ολοκληρωθεί η απαιτούμενη προεργασία σε κάθε αρχείο, ακολουθούν τα REST API's, που περιλαμβάνει αυτό. Κάθε ένα από αυτά έχει συγκεκριμένη δομή. Πρώτα, καλείται από τη μεταβλητή «*router*» μία μέθοδος εκ των τεσσάρων: get(), για ανάκτηση πληροφορίας από τη βάση, post(), για εγγραφή πληροφορίας στη βάση, update(), για επεξεργασία δεδομένων της βάσης και τέλος delete() για διαγραφή δεδομένων. Κάθε μία από τις προαναφερθείσες μεθόδους δέχεται δύο παραμέτρους. Η πρώτη είναι το URI, μέσω του οποίου θα καλείται το εκάστοτε API από τα Services του Client. Αυτό μπορεί να είναι απλό, όπως της εικόνας 3.12, «*/admin/login*», ή να περιλαμβάνει και παράμετρο, όπως για παράδειγμα «*/iatreia/:id*». Η δεύτερη παράμετρος είναι μία callback συνάρτηση, η οποία δέχεται τρεις παραμέτρους, τη «*req*» με πληροφορίες του εισερχόμενου request, τη «*res*» με πληροφορίες της εξερχόμενης απάντησης και τη «*next*». Μέσω της «*req*» μπορούμε να ανακτήσουμε την παράμετρο του URI, με την εντολή «*var param = req.params.id;*» και ακόμη πληροφορίες από το σώμα (body) του request με την εντολή «*var iatreio = req.body;*». Στη συνέχεια, αφού έχουμε εξάγει όλες τις απαραίτητες πληροφορίες, είτε από το body του request, είτε από το URI, συνεχίζουμε με την επικοινωνία με τη βάση μας. Για την υλοποίηση, λοιπόν, των ερωτημάτων στη βάση, χρησιμοποιείται η σύνταξη της MongoDB, όπως αυτή περιγράφεται στο [13], με τη διαφορά ότι χρησιμοποιούμε επιπλέον μία callback συνάρτηση, ώστε με το που ολοκληρωθεί το ερώτημα, να επιστραφεί η απάντηση στον Client ή μήνυμα λάθους, αν κάτι πάει στραβά.

• Τέλος, κλείνουμε το αρχείο μας με την εντολή «*module.exports = router*». Μέσω αυτής εξάγουμε όλες τις λειτουργίες που υλοποιήσαμε.



```
JS admin.js ×

1   var express = require('express');
2   var jwt = require('jsonwebtoken');
3   var mongojs = require('mongojs');
4   var bcrypt = require('bcrypt-nodejs');
5   var config = require('../config.json');
6
7   var router = express.Router();
8
9   var db = mongojs('mongodb://kostas:1234@ds056549.mlab.com:56549/rantevou-online-db', ['admin']);
10
11  //----------- Authenticate -----------
12  router.post('/admin/login', function(req, res, next) {
13      db.admin.findOne({ username: req.body.username }, (err, user) => {
14          if (user) {
15              bcrypt.compare(req.body.password, user.password, (err, isMatch) => {
16                  if (isMatch) {
17                      var myToken = jwt.sign({username: req.body.username, password: req.body.password}, config.secret);   // Create a token and send it.
18                      res.json({ 'userFound': true, 'token': myToken, 'id': user._id });
19                  } else {
20                      res.json({ 'userFound': false });
21                  }
22              });
23          } else {
24              res.json({ 'userFound': false });
25          }
26      });
27  });
28
29  //------------ Export Module ------------
30  module.exports = router;
```

Εικόνα 3.12: *Ενδεικτικό παράδειγμα της δομής των αρχείων του φακέλου «routes».*

3.5.2 Ανάλυση Κώδικα Υλοποίησης των API's του Server

Στην προηγούμενη παράγραφο, πήραμε μία γενική ιδέα για τη δομή του κώδικα καθενός αρχείου του φακέλου «*routes*» του Server, μέσα στα οποία υλοποιούνται όλα τα χρησιμοποιούμενα REST API's. Ας εστιάσουμε τώρα σε κάποιες συγκεκριμένες υλοποιήσεις κώδικα, που παρουσιάζουν ενδιαφέρον και αξίζει να αναφερθούν.

Στο αρχείο «*doctors.js*» ενδιαφέρον παρουσιάζει η υλοποίηση του API «*/doctors/:iatreio/:date/:month/:year*», η οποία παρουσιάζεται στην παρακάτω εικόνα. Το εν λόγω API χρησιμοποιείται για την ανάκτηση όλων των γιατρών μία συγκεκριμένη ημέρα, που θα είναι παρόντες στο νοσοκομείο και δεν θα έχουν άδεια. Αρχικά παίρνουμε τις παραμέτρους που περνάνε μέσω του URI και τις αποθηκεύουμε σε τοπικές μεταβλητές. Στη συνέχεια μέσω της μεθόδου «*getTime()*» της κλάσης «*Date*» μετατρέπουμε την ημερομηνία σε ακέραιο. Έπειτα, μέσα στο collection «*doctors*» αναζητούμε όλους τους γιατρούς του συγκεκριμένου ιατρείου, ελέγχοντας ταυτόχρονα και αν η τρέχουσα ημερομηνία βρίσκεται στο διάστημα των τιμών που ορίζουν τα πεδία «*startingDate*» και «*endingDate*». Έτσι επιστρέφονται τα πεδία «*name*», «*lastname*» και «*iatreio*» μόνο των γιατρών που θα είναι παρόντες τη συγκεκριμένη μέρα.

```
//---------- Get Doctors without Leave for a given Iatreio ----------
router.get('/doctors/:iatreio/:date/:month/:year', function(req, res, next) {
    var paramIatreio = req.params.iatreio;
    var paramDate = parseInt(req.params.date);
    var paramMonth = parseInt(req.params.month);
    var paramYear = parseInt(req.params.year);

    // Getting Date as an Integer
    var date = new Date(paramYear, paramMonth - 1, paramDate);
    var dateInt = date.getTime();

    db.doctors.find(
        {
            iatreio : paramIatreio,
            $or: [  {startingDate: {$exists: false}, endingDate: {$exists: false}},
                    {startingDate: {$gt: dateInt}},
                    {endingDate: {$lt: dateInt}} ]
        },
        {name: 1, lastname: 1, iatreio: 1},     // Ορίζω να επιστρέψουν από τη βάση μόνο τα πεδία name, lastname, iatreio.
        (err, doctor) => {
            if(err) { res.send(err); }
            res.json(doctor);
        }
    );
});
```

Εικόνα 3.13: Υλοποίηση του API «*/doctors/:iatreio/:date/ :month/:year*» του αρχείου «*doctors.js*».

Επίσης, στο ίδιο αρχείο, ενδιαφέρουσα είναι και η υλοποίηση του API «*/doctors/:name/:rate*». Ως «*name*» περνάει το ονοματεπώνυμο του γιατρού. Με χρήση της μεθόδου «*split()*» αποθηκεύουμε ξεχωριστά το όνομα και το επώνυμο του γιατρού, στη θέση 0 και 1 του πίνακα «*paramNameArray*», αντίστοιχα. Στη συνέχεια, στο collection «*doctors*», εντοπίζουμε το συγκεκριμένο γιατρό και αυξάνουμε το πεδίο «*totalRate*» κατά παράγοντα «*paramRate*», που πήραμε από το URI, και το πεδίο «*totalVotes*» κατά 1.

```
//---------- Update Doctor's Rating ----------
router.put('/doctors/:name/:rate', function(req, res, next) {
    var paramNameArray = req.params.name.split(' ');
    var paramRate = parseInt(req.params.rate);

    // Αυξάνουμε το totalRate κατά "paramRate" και το totalVotes κατά 1.
    db.doctors.update(
        { name: paramNameArray[0], lastname: paramNameArray[1] },
        { $inc: { "rating.totalRate": paramRate, "rating.totalVotes": 1 } },
        (err, data) => {
            if (err) { res.send(err) }
            res.json({status: 'ok'});
    });
});
```

Εικόνα 3.14: Υλοποίηση του API «*/doctors/:name/:rate*» του αρχείου «*doctors.js*».

Στο αρχείο «*users.js*», ας αναλύσουμε το REST API «*/users/login*». Να σημειωθεί ότι η ίδια μεθοδολογία χρησιμοποιείται και στο login του γιατρού και του διαχειριστή. Το username και το password που έδωσε ο χρήστης περνάνε μέσα από το body του request. Αρχικά, μέσα στο collection «*users*» εντοπίζεται ο χρήστης που δόθηκε. Αν δεν υπάρχει, επιστρέφεται πίσω ένα αντικείμενο «*{ 'userFound' : false }*». Αν βρεθεί ο χρήστης, τότε συγκρίνονται τα hashes των κωδικών. Αν δεν ταιριάξουν, επιστρέφεται πάλι το προαναφερθέν αντικείμενο. Αν ταιριάξουν, τότε ο Server παράγει ένα μοναδικό token για τον χρήστη, με σκοπό την αυθεντικοποίησή του σε κάθε επόμενο request, και του το επιστρέφει.

```
//---------- Authenticate ----------
router.post('/users/login', function(req, res, next) {
    db.users.findOne({ username: req.body.username }, (err, user) => {
        if (user) {
            bcrypt.compare(req.body.password, user.password, (err, isMatch) => {
                if (isMatch) {
                    var myToken = jwt.sign({username: req.body.username, password: req.body.password}, config.secret);   // Create a token and send it.
                    res.json({ 'userFound': true, 'token': myToken, 'id': user._id });
                } else {
                    res.json({ 'userFound': false });
                }
            });
        } else {
            res.json({ 'userFound': false });
        }
    });
});
```

Εικόνα 3.15: Υλοποίηση του API «*/users/login*» του αρχείου «*users.js*».

Εν συνεχεία, ας εξετάσουμε το REST API «*/appointment/pending/:user/:date/:month/:year*» του αρχείου «*appointments.js*». Σκοπός του υπόψη API είναι να επιστρέψει όλα τα ραντεβού ενός συγκεκριμένου χρήστη, προγραμματισμένα σε ημερομηνία μεταγενέστερη από την τρέχουσα. Με απλά λόγια, επιστρέφονται τα ραντεβού που εκκρεμούν και δεν έχουν διεξαχθεί ακόμα. Από το URI, παίρνουμε και αποθηκεύουμε σε τοπικές μεταβλητές, το username του χρήστη και την σημερινή ημερομηνία (date, month, year). Η συνθήκη, που χρησιμοποιείται στην εντολή find() της MongoDB για την επίτευξη του ανωτέρω στόχου, φαίνεται στην εικόνα 3.16. Συγκεκριμένα, ελέγχονται όλα τα καταχωρημένα ραντεβού μέσα στο collection «*appointments*» και επιστρέφονται μόνο αυτά που ανήκουν στον χρήστη που λήφθηκε ως παράμετρος από το URI και που ισχύει μία εκ των κάτωθι συνθηκών:

- Το έτος, στο οποίο έχει προγραμματιστεί το ραντεβού, είναι μεγαλύτερο από το τρέχον έτος.
- Αν τα έτη είναι ίδια, τότε ο μήνας, στον οποίον έχει προγραμματιστεί το ραντεβού, είναι μεγαλύτερος από τον τρέχοντα μήνα.
- Αν οι μήνες είναι ίδιοι, τότε η ημέρα, στην οποία έχει προγραμματιστεί το ραντεβού, είναι μεγαλύτερη από την τρέχουσα ημέρα.

Εικόνα 3.16: Υλοποίηση του API «*/appointment/pending/:user/:date/:month/:year*» του αρχείου «*appointments.js*».

Τέλος, θα δούμε το REST API «*/iatreiaAndDoctorsAggregate*» του αρχείου «*iatreia.js*», το οποίο, θα μπορούσαμε να πούμε, είναι ένα LEFT JOIN, σε γλώσσα SQL, δύο collections, του collection «*iatreia*» και του collection «*doctors*». Αυτό χρησιμοποιείται από τη διεπαφή του διαχειριστή για την ανάκτηση των βαθμολογιών των γιατρών κάθε ιατρείου. Ο σκοπός μας ήταν τα αποτελέσματα που θα παίρναμε από τη βάση να ήταν στη μορφή που παρουσιάζεται στην εικόνα 3.17. Όμως, τα ιατρεία ήταν αποθηκευμένα σε ένα collection, ενώ οι, σχετικές με τους γιατρούς, πληροφορίες σε άλλο collection. Χρησιμοποιώντας τη μέθοδο «*aggregate()*» της MongoDB και το «*$lookup*», παίρνουμε κάθε εγγραφή του collection «*iatreia*» και ενώνουμε σε αυτή τις εγγραφές του collection «*doctors*», για τις οποίες ταυτίζονται τα πεδία «*name*» και «*iatreio*» των δύο collections. Στη συνέχεια, με τη χρήση του «*$project*», σχηματίζουμε τη μορφή που θέλουμε να πάρουν τα αποτελέσματα και τέλος τα επιστρέφουμε.

Εικόνα 3.17: *Επιθυμητή μορφή αποτελεσμάτων από τη βάση, όταν καλούμε το REST API «/iatreiaAndDoctorsAggregate».*

```
// Με αυτό το URI, σχηματίζουμε την απάντηση του server για την καρτέλα του admin "Βαθμολογία Γιατρών".
// Αρχικά, ανακτούμε όλα τα ιατρεία και πραγματοποιούμε ένα LEFT JOIN με το collection "DOCTORS".
// Στη συνέχεια, προσθέτουμε σε κάθε ιατρείο ένα array με όλες τις καταχωρήσεις
// από το collection DOCTORS, που αντιστοιχούν στο εκάστοτε ιατρείο ($lookup).
// Έπειτα με το $project και το $map, διαμορφώνουμε το δεδομένα εξόδου στη μορφή που θέλουμε.
router.get('/iatreiaAndDoctorsAggregate', function(req, res, next) {
    db.iatreia.aggregate([
        {$lookup: {
                    from: "doctors",
                    localField: "name",
                    foreignField: "iatreio",
                    as: "doctor_docs"
                  }
        },
        {$project: {
                     _id: 0,
                     iatreioLabel: "$name",
                     ratingData: {
                        $map: {
                                input: "$doctor_docs",
                                as: "result",
                                in: {
                                    doctorName: "$$result.name",
                                    doctorLastname: "$$result.lastname",
                                    totalRate: "$$result.rating.totalRate",
                                    totalVotes: "$$result.rating.totalVotes",
                                }
                        }
                     }
                   }
        }
    ], (err, iatreia) => {
        if (err) { res.send(err); }
        res.json(iatreia);
    });
});
```

Εικόνα 3.18: Υλοποίηση του API «*/iatreiaAndDoctorsAggregate*» του αρχείου «*iatreia.js*».

3.5.3 Δομή Κώδικα Μεθόδων του Φακέλου «*Services*» στον Client

Ας περάσουμε τώρα στον Client. Κάθε μέθοδος των αρχείων που περιλαμβάνονται στο φάκελο «*Services*» του Client, όπως έχει προαναφερθεί, παίζει το ρόλο του ενδιάμεσου ανάμεσα στα Components και τη βάση. Εκάστη μέθοδος καλείται από κάποιο Component και αυτή με τη σειρά της καλεί το αντίστοιχο URI του Server. Αφού λάβει την απάντηση από αυτόν, την επιστρέφει πίσω στο Component. Όλες οι μέθοδοι έχουν κοινή δομή. Συγκεκριμένα:

- Πρώτα δημιουργείται ένα αντικείμενο τύπου Headers με την εντολή «*let headers = new Headers();*».

- Στη συνέχεια, ανακτάται το token από τις αποθηκευμένες πληροφορίες στο Session, με την εντολή «*let token = sessionStorage.getItem('token');*». Αν η συνάρτηση χρησιμοποιείται στο UI του

ασθενή, ανακτάται το «*token*», αν χρησιμοποιείται στο UI του διαχειριστή, ανακτάται το «*adminToken*» και τέλος αν χρησιμοποιείται στο UI του γιατρού, ανακτάται το «*doctorToken*».

• Αν βρεθεί το αντίστοιχο token, δημιουργείται το κατάλληλο Authorization Header. Χρησιμοποιείται η εντολή «*headers = new Headers({ 'Authorization' : 'Bearer ' + token});*».

• Επιπρόσθετα, με την εντολή «*headers.append('Content-Type', 'application/json');*», προστίθεται το υπόψη header, όπου αυτό είναι απαραίτητο, δηλαδή στην περίπτωση που μεταφέρονται δεδομένα στο Server, που σχετίζονται με το request (όπως μία εγγραφή δεδομένων, μία τροποποίηση κ.τ.λ.) και πρέπει ο Server να γνωρίζει ότι έχουν μορφή JSON.

• Τέλος, γίνεται η κλήση στο αντίστοιχο URI του Server και η επιστροφή της απάντησης στον Client. Χρησιμοποιείται το αντικείμενο της κλάσης «*Http*», που έχει παραχθεί στον Constructor του Service. Αυτό περιλαμβάνει τέσσερις (4) μεθόδους: την get(), την post(), την put() και την delete(). Κατόπιν, γίνεται κλήση στη συνάρτηση «*map()*», η οποία μετατρέπει την απάντηση του Server σε JSON μορφή και την αντιστοιχεί σε μία από τις ορισμένες κλάσεις του φακέλου «*Classes*» του Client. Επίσης, επισημαίνουμε τα κάτωθι:

- Η μέθοδος get() αφορά στην ανάκτηση πληροφοριών από τη βάση. Ως πρώτη παράμετρο δέχεται το αντίστοιχο URI του Server και ως δεύτερη, ένα αντικείμενο με τη μεταβλητή «*header*» που έχουμε σχηματίσει. Αν χρειάζεται να περάσουμε στο Server μία μεταβλητή, την προσθέτουμε στο URI. Λόγου χάρη, «*return this.http.get('http://localhost:3000/iatreio-by-id/" + id, { headers: headers }). map(response => response.json() as Iatreio);*».

- Η μέθοδος delete() αφορά στην διαγραφή πληροφοριών από τη βάση. Ως πρώτη παράμετρο δέχεται το αντίστοιχο URI του Server και ως δεύτερη, ένα αντικείμενο με τη μεταβλητή «*header*» που έχουμε σχηματίσει. Για παράδειγμα, «*return this.http.delete('http://localhost:3000/ appointment/" + id, { headers: headers }).map(response => response.json() as Status);*».

- Η μέθοδος post() αφορά στην εγγραφή δεδομένων στη βάση. Ως πρώτη παράμετρο δέχεται το αντίστοιχο URI του Server. Ως δεύτερη παράμετρο δέχεται ένα αντικείμενο με τα δεδομένα που θέλουμε να περάσουμε (περνάνε μέσα από το body του request). Ως τρίτη παράμετρο, δέχεται ένα αντικείμενο με τη μεταβλητή «*header*» που έχουμε σχηματίσει. Παράδειγμα, «*return this.http.post('http:// localhost:3000/appointment', JSON.stringify(newAppointment), { headers: headers }).map(response => response.json() as Status);*».

- Η μέθοδος put() αφορά στην επεξεργασία δεδομένων στη βάση. Ως πρώτη παράμετρο δέχεται το αντίστοιχο URI του Server, με προσαρτημένη την πληροφορία για το πού πρέπει να γίνει η τροποποίηση. Ως δεύτερη παράμετρο δέχεται ένα αντικείμενο με τα δεδομένα που θέλουμε να περάσουμε (περνάνε μέσα από το body του request). Ως τρίτη παράμετρο, δέχεται ένα αντικείμενο με τη μεταβλητή «*header*» που έχουμε σχηματίσει. Παράδειγμα, «*return this.http.put('http:// localhost:3000/users/update-*

user/' + id, JSON.stringify(edittedUser), { headers: headers }).map(response => response.json() as Status);».

```
addAppointment(newAppointment: Appointment): Observable<Appointment> {
    let patientHeaders = new Headers();
    let token = sessionStorage.getItem('token');

    if (token) {
        patientHeaders = new Headers({ 'Authorization': 'Bearer ' + token });
        patientHeaders.append('Content-Type', 'application/json');
    }

    return this.http
              .post('http://localhost:3000/appointment', JSON.stringify(newAppointment), {headers : patientHeaders})
              .map(res => res.json());
}
```

Εικόνα 3.19: *Ενδεικτικό παράδειγμα της δομής μίας συνάρτησης ενός Service του Client.*

3.6 Νευρωνικό Δίκτυο

3.6.1 Γενικά για τα Νευρωνικά Δίκτυα

Όπως έχει ήδη αναφερθεί και στην παρουσίαση της εφαρμογής, γίνεται χρήση μέσα σε αυτή ενός Νευρωνικού Δικτύου. Για να το κατανοήσουμε, θα εξηγήσουμε πρώτα τον όρο *Μηχανική Μάθηση*, που ακούμε πολύ συχνά στις μέρες μας. Θα μπορούσαμε να ορίσουμε τον όρο αυτό ως το πεδίο μελέτης που δίνει σε ένα υπολογιστικό σύστημα την ικανότητα να δημιουργήσει μοντέλα ή πρότυπα από ένα σύνολο δεδομένων. Ως κλάδος της Τεχνητής Νοημοσύνης, η Μηχανική Μάθηση ασχολείται με τη μελέτη αλγορίθμων που βελτιώνουν τη συμπεριφορά τους σε κάποια εργασία, που τους έχει ανατεθεί, χρησιμοποιώντας την εμπειρία τους.

Θα μπορούσαμε να εντάξουμε τις τεχνικές Μηχανικής Μάθησης, που έχουν αναπτυχθεί, σε τρεις μεγάλες κατηγορίες:

α. Μάθηση με Επιτήρηση (Supervised Learning)

β. Μάθηση χωρίς Επιτήρηση (Unsupervised Learning)

γ. Μάθηση με ημι-επιτήρηση (Semi-supervised Learning)

Στη Μάθηση με Επιτήρηση, η επιθυμητή σχέση/μοντέλο εισόδου-εξόδου δεν είναι γνωστή, αλλά έχουμε στη διάθεσή μας μερικά παραδείγματά της, δηλαδή ζευγάρια τιμών εισόδου στο σύστημα και του αντίστοιχου αποτελέσματος. Το σύστημα καλείται μέσα από αυτά να ανακαλύψει την κατάλληλη σχέση.

Στη Μάθηση χωρίς Επιτήρηση, δεν είναι γνωστή ούτε η σχέση εισόδου-εξόδου, αλλά ούτε και διαθέτουμε δείγματά της. Σε αυτή την περίπτωση, το σύστημα απλά καλείται να ανακαλύψει μόνο του συσχετίσεις ή ομάδες σε ένα σύνολο δεδομένων, δημιουργώντας πρότυπα, χωρίς να είναι γνωστό αν υπάρχουν, πόσα και ποια είναι.

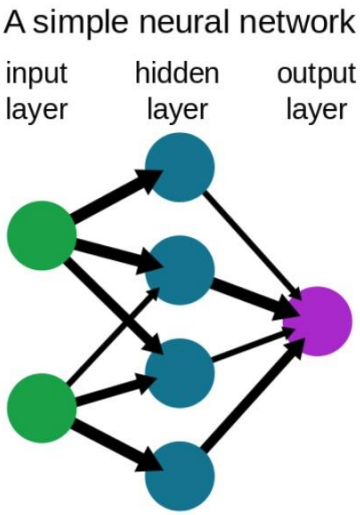
Στη Μάθηση με ημι-επιτήρηση, μέρος των δεδομένων αποτελούν δείγματα μίας επιθυμητής σχέσης εισόδου-εξόδου, ενώ άλλα όχι.

Μία ειδική κατηγορία των τεχνικών Μηχανικής Μάθησης αποτελούν τα Νευρωνικά Δίκτυα ([4]). Αυτά προσπαθούν να προσεγγίσουν την λειτουργία του ανθρώπινου εγκεφάλου σε μία μηχανή, μιμούμενα την αρχιτεκτονική των βιολογικών νευρώνων. Συνεπώς, ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) είναι ένα σύνολο από τεχνητούς νευρώνες, δηλαδή μονάδες επεξεργασίας, που συνδέονται μεταξύ τους. Οι νευρώνες χωρίζονται σε τρεις (3) κατηγορίες:

α. Στους «*νευρώνες εισόδου*». Αυτοί λαμβάνουν τις εισόδους τους από το περιβάλλον και απλά τις περνάνε στους υπολογιστικούς νευρώνες, χωρίς να επιτελούν κανέναν υπολογισμό. Οι εν λόγω νευρώνες αποτελούν το «*στρώμα εισόδου (input layer)*» του ΤΝΔ.

β. Στους «*νευρώνες εξόδου*». Αυτοί απλά διοχετεύουν στο περιβάλλον τις τελικές αριθμητικές εξόδους του δικτύου. Οι εν λόγω νευρώνες αποτελούν το «στρώμα εξόδου (output layer)» του ΤΝΔ.

γ. Στους «*υπολογιστικούς* ή *κρυμμένους νευρώνες*». Αυτοί αποτελούν το «*κρυφό στρώμα (hidden layer)*» του ΤΝΔ και η λειτουργία τους αναλύεται στην παρακάτω παράγραφο. Ένα ΤΝΔ μπορεί να αποτελείται από παραπάνω από ένα κρυφά στρώματα.



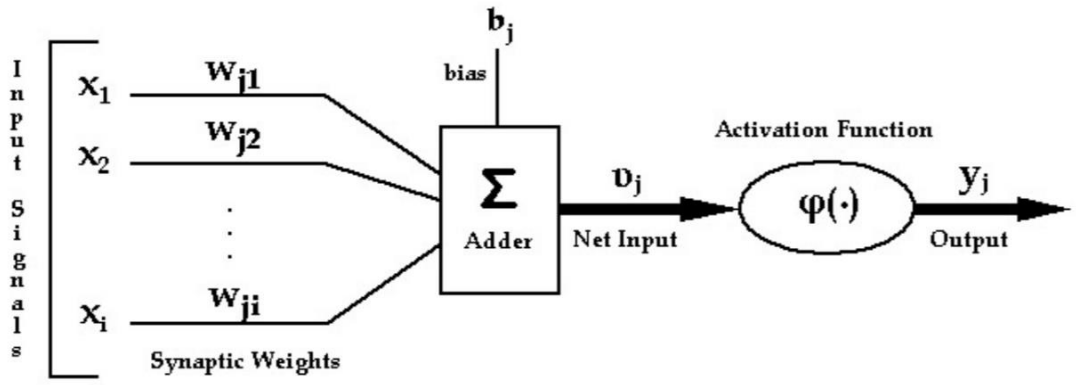
Εικόνα 3.20: *Απεικόνιση ενός απλού ΤΝΔ με ένα κρυφό στρώμα.*

Αναφορικά με τους υπολογιστικούς νευρώνες ενός τεχνητού Νευρωνικού Δικτύου, δίνεται η δομή τους στην εικόνα 3.21 και ο τρόπος λειτουργίας τους είναι ο εξής: Κάθε νευρώνας αυτού του είδους μπορεί να έχει πολλές εισόδους $x_1, x_2, ..., x_i$, αλλά μόνο μία έξοδο y_j, η οποία με τη σειρά της μπορεί να αποτελέσει την είσοδο για άλλους νευρώνες, αν το ΝΔ έχει παραπάνω από ένα κρυφά στρώματα. Κάθε μία από τις εισόδους του νευρώνα έχει διαφορετική σημαντικότητα, η οποία καθορίζεται από ένα

συντελεστή βάρους ($w_{j1}, w_{j2}, ..., w_{ji}$). Πολλαπλασιάζεται, λοιπόν, κάθε είσοδος με το αντίστοιχο συναπτικό βάρος και κατόπιν αθροίζονται όλα τα γινόμενα. Το αποτέλεσμα τροφοδοτείται ως όρισμα στη συνάρτηση ενεργοποίησης φ, που υλοποιεί εσωτερικά κάθε κόμβος. Η τιμή που λαμβάνει η συνάρτηση για το εν λόγω όρισμα είναι και η έξοδος του νευρώνα. Δηλαδή,

$$y_k = \varphi(\sum_{i=0}^{N} x_{ki} w_{ki})$$

Υπάρχουν αρκετές συναρτήσεις ενεργοποίησης που μπορούν να χρησιμοποιηθούν, αλλά οι πιο διαδεδομένες είναι η βηματική, η σιγμοειδής, η γραμμική, η συνάρτηση πρόσημου.



Εικόνα 3.21: *Απεικόνιση ενός τεχνητού νευρώνα ενός ΤΝΔ..*

Σε αυτό το σημείο πρέπει να σημειωθεί ότι υπάρχουν κάποια στάδια που πρέπει να ολοκληρωθούν έως ότου το Νευρωνικό μας Δίκτυο να είναι έτοιμο για χρήση στην εφαρμογή μας και τα αποτελέσματά του να είναι ακριβή. Το πρώτο και πολύ βασικό στάδιο της διαδικασίας είναι ο σωστός προσδιορισμός του προβλήματος και του τι θέλουμε επακριβώς να κάνουμε. Διαφορετικά, μία γενικευμένη αναζήτηση ή ένα όχι σωστά ορισμένο πρόβλημα, θα οδηγήσει σε λάθος αποτελέσματα. Εν συνεχεία, πρέπει να διαλέξουμε σωστά τις μεταβλητές που θα αποτελέσουν τις εισόδους και εξόδους του ΝΔ και να τις κωδικοποιήσουμε ορθά. Έπειτα, πρέπει να επιλέξουμε τον κατάλληλο για το πρόβλημά μας τύπο ΝΔ (απλής τροφοδότησης/feedforward, ανατροφοδότησης/feedback, κ.τ.λ.) και να κάνουμε σε αυτό τις σωστές ρυθμίσεις (κατάλληλη συνάρτηση ενεργοποίησης, κατάλληλο πλήθος κρυφών στρωμάτων, κατάλληλος αριθμός νευρώνων σε καθένα από αυτά, κατάλληλος ρυθμός εκμάθησης, τρόπος μέτρησης του σφάλματος, όριο αποδεκτού σφάλματος, μέγιστος αριθμός επαναλήψεων), ώστε να πάρουμε τα καλύτερα δυνατά αποτελέσματα. Το τελευταίο στάδιο της διαδικασίας αποτελεί η εκπαίδευση του ΝΔ.

Αναφορικά με τη διαδικασία της εκπαίδευσης, τροφοδοτούμε στο ΝΔ ένα σύνολο δειγμάτων, δηλαδή τις εισόδους και την απαιτούμενη έξοδό τους, και με μία επαναληπτική διαδικασία, αυτό προσπαθεί να καθορίσει κατάλληλα του συντελεστές βάρους, ώστε οι είσοδοι να δίνουν ως αποτέλεσμα

την επιθυμητή έξοδο. Ο ακριβής προσδιορισμός της εξόδου είναι αδύνατος, άρα αναζητούμε την καταλληλότερη διαμόρφωση του ΝΔ, ώστε τα αποτελέσματα να είναι όσο πιο κοντά στα πραγματικά γίνεται. Την ικανοποιητική τιμή του σφάλματος μεταξύ πραγματικών και προσεγγιστικών αποτελεσμάτων, την ορίζουμε εμείς. Παρόλα αυτά, πρέπει να επαναλάβουμε τα στάδια, που αναφέρθηκαν στην προηγούμενη παράγραφο, μέχρι να καταλήξουμε σε ένα ικανοποιητικό σφάλμα, διότι δεν μπορούμε να ξέρουμε εκ των προτέρων ποια είναι η πιο σωστή διαμόρφωση του ΝΔ.

3.6.2 Κωδικοποίηση Τιμών Εισόδου-Εξόδου του Νευρωνικού Δικτύου

Όπως προαναφέραμε, πολύ σημαντικό στάδιο για την παραγωγή από το Νευρωνικό Δίκτυο αποτελεσμάτων με ακρίβεια, είναι η κατάλληλη επιλογή των μεταβλητών που θα τροφοδοτηθούν σε αυτό και η μετατροπή τους στην καταλληλότερη μορφή. Η υπόψη διαδικασία ονομάζεται στην ξενόγλωσση βιβλιογραφία «*Feature Engineering*», είναι αντικείμενο εκτενούς μελέτης και είναι τεράστιας σημασίας, γιατί εξαρτάται από αυτό η απόδοση του ΝΔ ([6]).

Έχει αναφερθεί σε προηγούμενο κεφάλαιο ότι το Νευρωνικό Δίκτυο, που έχουμε συμπεριλάβει στην εφαρμογή μας, δέχεται δύο (2) εισόδους, το όνομα του ιατρείου και το σύμπτωμα του ασθενή. Το προφανές πρόβλημα σε αυτό το σημείο είναι ότι οι είσοδοι στο σύστημα δεν είναι αριθμοί, ώστε να μπορέσουν να γίνουν διάφοροι υπολογισμοί με αυτούς. Αντιθέτως, είναι «*περιγραφικές μεταβλητές* ή *categorical variables*».

Όσον αφορά στην κωδικοποίηση των περιγραφικών μεταβλητών ([5]), υπάρχουν διάφορες τεχνικές. Ενδεικτικά αναφέρουμε τις παρακάτω:

α. «*Onehot Encoding*». Δημιουργούμε ένα πίνακα με μήκος όσες και οι τιμές που μπορεί να πάρει η μεταβλητή μας. Για κάθε τιμή της μεταβλητής, θέτουμε με 1 μία συγκεκριμένη θέση του πίνακα και με 0 όλες τις υπόλοιπες. Για παράδειγμα, έστω ότι έχουμε τη μεταβλητή «*χώρα*» με τιμές «*Ελλάδα, Αγγλία, Ιταλία*». Η τιμή «*Ελλάδα*» μπορεί να κωδικοποιηθεί ως [1,0,0], η τιμή «*Αγγλία*» ως [0,1,0] και η τιμή «*Ιταλία*» ως [0,0,1].

β. «*Hash Encoding*». Παρόμοια με την προηγούμενη μέθοδο με τη διαφορά ότι η κωδικοποίηση κάθε τιμής της μεταβλητής είναι σταθερού μήκους. Η κάθε τιμή δίνεται ως όρισμα σε μία hash συνάρτηση, η οποία τη μετατρέπει σε αριθμό. Έπειτα, αυτός ο αριθμός διαιρείται με το επιθυμητό μήκος του διανύσματος εξόδου και παίρνουμε το υπόλοιπο της διαίρεσης R. Έτσι κάθε τιμή της μεταβλητής κωδικοποιείται ως ένα διάνυσμα με μηδενικά και την τιμή 1 στη θέση R.

γ. «*Label Encoding*». Κάθε τιμή της μεταβλητής αντιστοιχίζεται σε μία αριθμητική τιμή. Στο παράδειγμα της χώρας, θα μπορούσαμε να ονομάσουμε την «*Ελλάδα*» ένα (1), την «*Αγγλία*» δύο (2) και την «*Ιταλία*» τρία (3).

δ. «*Count Encoding*». Κάθε τιμή της μεταβλητής εισόδου αντικαθίσταται από το πλήθος εμφανίσεών της στο σύνολο δεδομένων εκπαίδευσης.

ε. «*LabelCount Encoding*». Μία μείξη των (γ) και (δ) τεχνικών κωδικοποίησης, κατά την οποία ταξινομούμε σε ομάδες τις τιμές της μεταβλητής εισόδου, ανάλογα με το πλήθος των εμφανίσεων κάθε τιμής στο σύνολο δεδομένων εκπαίδευσης, και έπειτα δίνουμε σε κάθε ομάδα ένα αριθμητικό ID.

Κατόπιν μελέτης διαφόρων περιπτώσεων χρήσης Νευρωνικού Δικτύου με εισόδους περιγραφικές μεταβλητές ([1], [2], [3]) και εκτέλεσης πλήθους δοκιμών και πάντα με στόχο την επίτευξη του μικρότερου δυνατού σφάλματος, αποφασίσαμε να κωδικοποιήσουμε τις μεταβλητές εισόδου (ιατρείο, σύμπτωμα) του Νευρωνικού Δικτύου, που συμπεριλάβαμε στην προτεινόμενη εφαρμογή, με μία παραλλαγή της προαναφερθείσας μεθόδου «*Label Encoding*». Κάθε τιμή της εκάστοτε περιγραφικής μεταβλητής εισόδου πήρε ένα αριθμητικό ID, το οποίο όμως αυξάνεται κατά 0,1 και όχι κατά 1. Να σημειώσουμε ότι την παρούσα στιγμή είναι καταχωρημένα στη βάση μας δεκατέσσερα (14) ιατρεία. Κάθε ένα από αυτά τα ιατρεία περιλαμβάνει ένα μεταβλητό αριθμό συμπτωμάτων με επτά (7) τα περισσότερα. Συνεπώς, η κωδικοποίηση πραγματοποιήθηκε ως εξής:

- «Αγγειολογικό» => 0,1
- «Αλλεργιολογικό» => 0,2
- «Γαστρεντερολογικό» => 0,3
- «Γυναικολογικό» => 0,4
- …
- «Ρευματολογικό» => 1,3
- «Ψυχιατρικό» => 1,4

Αντίστοιχα, για τα συμπτώματα του «Ρευματολογικού», για παράδειγμα (και ομοίως για τα υπόλοιπα ιατρεία), η κωδικοποίηση έγινε ως κάτωθι:

- «Οστεοπόρωση» => 0,1
- «Τενοντίτιδα» => 0,2
- «Ρευματοειδής αρθρίτιδα» => 0,3

Εν συνεχεία, θα δούμε την κωδικοποίηση της εξόδου του χρησιμοποιούμενου, στην εφαρμογή, ΝΔ, δηλαδή της εκτιμώμενης διάρκειας του ραντεβού του ασθενή. Και εδώ η αριθμητική τιμή που χρησιμοποιείται ανήκει στο διάστημα (0,1), έτσι ώστε να μην διαφοροποιείται η τάξη μεγέθους της κωδικοποίησης της μεταβλητής εξόδου σε σχέση με τις μεταβλητές εισόδου. Αναλυτικά:

- 0 έως 9 λεπτά => 0,1
- 10 έως 19 λεπτά => 0,2

- 20 έως 29 λεπτά => 0,3
- 30 έως 39 λεπτά => 0,4
- 40 έως 49 λεπτά => 0,5
- 50 έως 59 λεπτά => 0,6
- 60 έως 89 λεπτά => 0,7
- 90 έως 119 λεπτά => 0,8
- Παραπάνω από 120 λεπτά => 0,9

3.6.3 Υλοποίηση Νευρωνικού Δικτύου

Η εκτέλεση των υπολογισμών του Νευρωνικού Δικτύου της εφαρμογής μας, με άλλα λόγια ο βασικός του πυρήνας, υλοποιείται από τη βιβλιοθήκη «*synaptic.js*» ([7]). Η εν λόγω βιβλιοθήκη είναι μία javascript βιβλιοθήκη για το NodeJS και το Browser και περιλαμβάνει ένα γενικευμένο αλγόριθμο για την κατασκευή σχεδόν οποιουδήποτε ΝΔ, ανεξαρτήτως της αρχιτεκτονική του. Ωστόσο, περιλαμβάνει και κάποιες ήδη έτοιμες αρχιτεκτονικές, όπως πολυστρωματικά ΝΔ Perceptron, ΝΔ με μνήμη, όπως το Long-Short Term Memory (LSTM), το Hopfield, κ.α. Επιπλέον, διαθέτει και μία κλάση «*Trainer*», η οποία διευκολύνει την εκπαίδευση του ΝΔ.

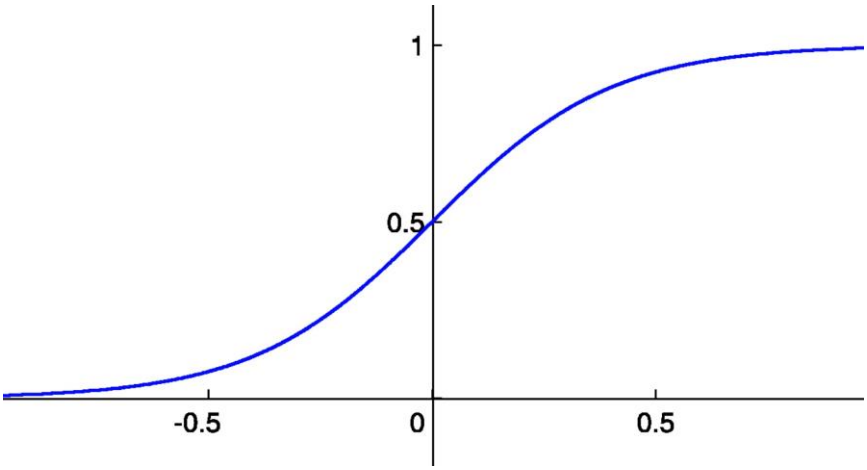
Ενώ η υλοποίηση του ΝΔ γίνεται μέσα από τη βιβλιοθήκη «*synaptic.js*», η χρήση του γίνεται μέσα από το αρχείο «*neural-network.service.ts*», το οποίο έχουμε κατασκευάσει εμείς. Ο κώδικας του συγκεκριμένου αρχείου παρατίθεται ολόκληρος στο Παράρτημα. Μέσα σε αυτόν ορίζεται η αρχιτεκτονική του ΝΔ, το πλήθος των κρυμμένων στρωμάτων, το πλήθος των νευρώνων κάθε στρώματος, ο αριθμός των επαναλήψεων του συνόλου των δεδομένων εκπαίδευσης, ο ρυθμός εκμάθησης (learning rate) κατά την εκπαίδευση, το αποδεκτό σφάλμα, η μέθοδος ενεργοποίησης κάθε νευρώνα και ο τρόπος υπολογισμού του σφάλματος. Ας δούμε κάθε μία από τις προαναφερθείσες ρυθμίσεις αναλυτικά.

Όσον αφορά την αρχιτεκτονική του ΝΔ μας, χρησιμοποιήσαμε την απλή αρχιτεκτονική Perceptron, δηλαδή ένα απλό δίκτυο, μη ανατροφοδοτούμενο. Κατόπιν δοκιμών και με γνώμονα την ελαχιστοποίηση του προκύπτοντος σφάλματος, καταλήξαμε στη χρήση ενός (1) κρυφού στρώματος στο δίκτυό μας, το οποίο να περιλαμβάνει είκοσι πέντε (25) νευρώνες. Ακόμα, ως γνωστόν, το στρώμα εισόδου έχει δύο νευρώνες εισόδου, για το ιατρείο και το σύμπτωμα, και το στρώμα εξόδου ένα (1) νευρώνα, για την εκτιμώμενη διάρκεια του ραντεβού. Κάθε νευρώνας εισόδου συνδέεται με κάθε έναν νευρώνα του κρυφού στρώματος και κάθε ένας από αυτούς συνδέεται με τον νευρώνα εξόδου.

Αφού ορίσαμε και κατασκευάσαμε το Νευρωνικό μας Δίκτυο, προχωρήσαμε στη δημιουργία ενός «*Trainer*», δηλαδή ενός βοηθητικού αντικειμένου της βιβλιοθήκης για την εκπαίδευσή του ΝΔ. Τα ορίσματα που απαιτούνται για τη δημιουργία του Trainer είναι το σύνολο δεδομένων εκπαίδευσης και ένα σύνολο ρυθμίσεων. Αναφορικά με το σύνολο δεδομένων εκπαίδευσης, κατεβάσαμε τα δεδομένα των

ιατρείων και των αντίστοιχων συμπτωμάτων τους από τη βάση, τα κωδικοποιήσαμε και συνθέσαμε αντικείμενα της μορφής «*{ input : [0.8, 0.1] , output : [0.1] }*». Αναφορικά με το σύνολο ρυθμίσεων, ορίσαμε το ρυθμό εκμάθησης του ΝΔ κατά την εκπαίδευσή του σε 0,15. το όριο των επαναλήψεων ανάμεσα στα δεδομένα εκπαίδευσης στις 1500000, το αποδεκτό σφάλμα στο 0,08. Αν επιτευχθεί το αποδεκτό σφάλμα ή φτάσουμε στο ανώτατο όριο επαναλήψεων, ολοκληρώνεται η εκπαίδευση του ΝΔ.

Σχετικά με τη συνάρτηση ενεργοποίησης κάθε νευρώνα, δεν ορίστηκε μέσα στο αρχείο μας κάτι διαφορετικό, οπότε χρησιμοποιήθηκε η προεπιλεγμένη συνάρτηση. Αυτή είναι η σιγμοειδής και παρουσιάζεται στην παρακάτω εικόνα.



Εικόνα 3.22: Σιγμοειδής συνάρτηση ενεργοποίησης κάθε νευρώνα του δικτύου.

Τέλος, η βιβλιοθήκη «*synaptic.js*» περιλαμβάνει τρεις (3) τρόπους υπολογισμού του σφάλματος του Νευρωνικού Δικτύου: «*CROSS_ENTROPY*», «*MSE*» και «*BINARY*». Εμείς χρησιμοποιήσαμε τη μέθοδο «*BINARY*», καθώς πειραματικά αποδείχτηκε ότι δίνει το μικρότερο σφάλμα και τα αποτελέσματα που προκύπτουν, όταν χρησιμοποιούμε το ΝΔ, είναι πάρα πολύ κοντά στα αναμενόμενα.

Αφού έχουν οριστεί όλα τα χαρακτηριστικά του Νευρωνικού Δικτύου, στη συνέχεια, το εν λόγω αρχείο «*neural-network.service.ts*», προχωράει στην υλοποίηση της εκπαίδευσής του. Πριν δούμε τη διαδικασία, πρέπει να επισημανθούν κάποιες σημαντικές λεπτομέρειες:

- Η εντολή εκκίνησης της εκπαίδευσης του Νευρωνικού Δικτύου, που είναι σχετικά μία χρονοβόρα διαδικασία, δίνεται από τον διαχειριστή της εφαρμογής μέσω ενός κουμπιού, που έχει τοποθετηθεί στο UI του. Η χρήση, από την άλλη, του ΝΔ γίνεται μέσα στο UI του ασθενή, όταν εμφανίζεται η εκτίμηση της διάρκειας του ραντεβού που έκλεισε. Εκείνη τη στιγμή, η πρόβλεψη του ΝΔ πρέπει να είναι άμεση, το οποίο συνεπάγεται ότι το ΝΔ πρέπει να είναι ήδη εκπαιδευμένο. Συμπερασματικά, αφού εκπαιδευτεί κατόπιν εντολής του διαχειριστή, αποθηκεύεται στη βάση σε μορφή JSON, ώστε να δύναται να χρησιμοποιηθεί αμέσως, όταν απαιτηθεί.

• Υπάρχει στην εφαρμογή μας η δυνατότητα πρόσθεσης και αφαίρεσης ιατρείων και συμπτωμάτων, μέσα από τη διεπαφή του διαχειριστή. Αυτό σημαίνει ότι οποιαδήποτε αλλαγή θα δημιουργήσει πρόβλημα στη συνοχή της κωδικοποίησης των μεταβλητών και επακόλουθα στην απόδοση και την ακρίβεια του Νευρωνικού Δικτύου. Για την αντιμετώπιση αυτού του προβλήματος, κάθε φορά που ο διαχειριστής δίνει εντολή για «Εκπαίδευση» του ΝΔ, πριν αυτή εκκινήσει, η εφαρμογή εκτελεί μία επικαιροποίηση της κωδικοποίησης των εισόδων και εξόδων του.

Ύστερα λοιπόν από τις ανωτέρω επισημάνσεις, ας δούμε τη διαδικασία που επιτελεί το αρχείο «*neural-network.service.ts*», κατόπιν της εντολής για «*Εκπαίδευση*» από το διαχειριστή:

• Κατεβάζει τις πληροφορίες των ιατρείων και των αντίστοιχων συμπτωμάτων τους από τη βάση.

• Κωδικοποιεί κάθε ιατρείο και κάθε σύμπτωμα εκ νέου, όπως αναφέρθηκε στην προηγούμενη υποενότητα.

• Αποθηκεύει τις κωδικοποιήσεις στο collection «*encodedDataForNN*», βάσει του οποίου κωδικοποιούνται στη διεπαφή του ασθενή τα δεδομένα εισόδου που θα τροφοδοτηθούν στο ΝΔ.

• Κατεβάζει όλα τα παρελθοντικά ραντεβού, όλων των χρηστών, με σκοπό τη δημιουργία του συνόλου εκπαίδευσης.

• Από κάθε ραντεβού του συνόλου εκπαίδευσης, δημιουργεί ένα αντικείμενο «*TrainingSetObj*», με μεταβλητές εισόδου το κωδικοποιημένο ιατρείο και το κωδικοποιημένο σύμπτωμα, και με μεταβλητή εξόδου την κωδικοποιημένη χρονική διάρκεια του ραντεβού.

• Χρησιμοποιώντας όλα τα «*TrainingSetObj*», που έχει δημιουργήσει, ξεκινάει την εκπαίδευση του ΝΔ, μέσω της μεθόδου «*train()*» της βιβλιοθήκης «*synaptic.js*».

• Αφού ολοκληρωθεί η εκπαίδευση, το εκπαιδευμένο ΝΔ αποθηκεύεται σε μορφή JSON στη βάση.

Η χρήση του ΝΔ, όπως προαναφέρθηκε, γίνεται μέσα στη διεπαφή του ασθενή. Καθώς αυτός κλείνει το ραντεβού του, το ιατρείο και το σύμπτωμα, που επέλεξε, κωδικοποιούνται βάσει της αντιστοιχίας ιατρείου – κωδικοποίησης και συμπτώματος – κωδικοποίησης από το collection «*encodedDataForNN*» της βάσης και ύστερα δίνονται ως είσοδοι στο ΝΔ. Αυτό, με τη χρήση της συνάρτησης «*activate()*», παράγει την εκτίμηση της διάρκειας του ραντεβού. Το αποτέλεσμα όμως είναι κωδικοποιημένο. Οπότε τελευταίο βήμα είναι η μετατροπή του σε λεπτά. Η εν λόγω διαδικασία παρουσιάζεται στην κάτωθι εικόνα.

```
// Ανάκτηση εκπαιδευμένου ΝΔ από τη βάση.
let iatreio = this.selectedIatreio.name;
let standardProblems = this.selectedStandardProblems;
this.nn
    .retrieveNeuralNetwork()
    .subscribe(exportedNN => {
        this.exportedNN = Network.fromJSON(exportedNN[0].nn);

        // Ανάκτηση κωδικοποίησης δεδομένων εισόδου-εξόδου από τη βάση.
        this.nn
            .getEncodedDataForNN(iatreio)
            .subscribe(encodedData => {
                let encodedIatreio = encodedData[0].encodeForNN;
                console.log('Encoded Iatreio: ' + encodedIatreio);

                for (let i = 0; i < standardProblems.length; i++) {
                  for (let j = 0; j < encodedData[0].symptoms.length; j++) {
                    if (standardProblems[i] === encodedData[0].symptoms[j].name) {
                      let encodedSymptom = encodedData[0].symptoms[j].encodeForNN;
                      console.log('Encoded Symptom: ' + encodedSymptom);
                      console.log('************************');

                      let encodedDuration = Math.round(10 * this.exportedNN.activate([encodedIatreio, encodedSymptom])) / 10;
                      console.log('Encoded Duration: ' + encodedDuration);

                      if ( encodedDuration === 0.1 ) { this.duration = 10; }
                      else if ( encodedDuration === 0.2 ) { this.duration = 20; }
                      else if ( encodedDuration === 0.3 ) { this.duration = 30; }
                      else if ( encodedDuration === 0.4 ) { this.duration = 40; }
                      else if ( encodedDuration === 0.5 ) { this.duration = 50; }
                      else if ( encodedDuration === 0.6 ) { this.duration = 60; }
                      else if ( encodedDuration === 0.7 ) { this.duration = 90; }
                      else if ( encodedDuration === 0.8 ) { this.duration = 120; }
                      else if ( encodedDuration === 0.9 ) { this.duration = 180; }

                      console.log('Real Duration: ' + this.duration + ' λεπτά.');
                      console.log('************************');
                    }
                  }
                }
```

Εικόνα 3.23: Χρήση του ΝΔ μέσα στη διεπαφή του ασθενή.

3.6.4 Αποτελέσματα Δοκιμών

Σε αυτή την παράγραφο, θα αναλύσουμε πώς καταλήξαμε στη διαμόρφωση του Νευρωνικού μας Δικτύου, δηλαδή πώς επιλέξαμε τον κατάλληλο αριθμό κρυφών στρωμάτων και νευρώνων σε καθένα από αυτά, τον κατάλληλο ρυθμό εκμάθησης, αριθμό επαναλήψεων και το αποδεκτό σφάλμα. Η επιλογή αυτών των παραμέτρων έγινε, κατόπιν εκτέλεσης πολλών δοκιμών εκπαίδευσης του ΝΔ μέσα από τη διεπαφή του διαχειριστή και παρατηρώντας κάθε φορά το προκύπτον σφάλμα, όπως αυτό απεικονίζεται στην κονσόλα του browser, μετά το πέρας κάθε εκπαίδευσης.

Η επιλογή της καταλληλότερης διαμόρφωσης πραγματοποιήθηκε σε τρία (3) στάδια. Αρχικά, έπρεπε να ξεκαθαρίσουμε ποιος θα είναι ο αριθμός των κρυφών στρωμάτων που θα χρησιμοποιήσουμε, και ποιος ο αριθμός των νευρώνων σε κάθε ένα από αυτά. Επίσης, έπρεπε να καταλήξουμε στην τάξη μεγέθους του ρυθμού εκμάθησης. Συνεπώς, στον κάτωθι πίνακα, φαίνονται όλες οι δοκιμές που κάναμε, έχοντας ως όρια τερματισμού της εκπαίδευσης, τις *500000* επαναλήψεις ή αποδεκτό σφάλμα *0,10*, ό,τι έρθει πρώτο.

Αριθμός Κρυμμένων Στρωμάτων	Αριθμός Νευρώνων 1ου Κρυμμένου Στρώματος	Αριθμός Νευρώνων 2ου Κρυμμένου Στρώματος	Αριθμός Νευρώνων 3ου Κρυμμένου Στρώματος	Rate	Ελάχιστο Σφάλμα	Αριθμός Επαναλήψεων	Χρόνος Ολοκλήρωσης Εκπαίδευσης σε ms
1	15	-	-	0,1	0,085	270443	23568
1	15	-	-	0,08	0,18	500000	43053
1	25	-	-	0,08	0,085	419538	56417
1	35	-	-	0,08	0,068	492403	91575
1	**25**	**-**	**-**	**0,1**	**0,068**	**70190**	**8996**
1	35	-	-	0,1	0,085	68181	31209
1	30	-	-	0,1	0,085	18946	3011
1	25	-	-	0,2	0,085	141696	28982
1	35	-	-	0,2	0,339	500000	91092
2	5	8	-	0,1	0,17	500000	44312
2	5	8	-	0,06	0,17	500000	44995
2	10	12	-	0,1	0,17	500000	81320
2	10	12	-	0,4	0,085	188453	31095
2	10	12	-	0,06	0,17	500000	81082
2	15	20	-	0,1	0,085	142383	43704
2	15	20	-	0,07	0,085	52966	16584
2	15	20	-	0,3	0,356	500000	156015
3	7	10	12	0,05	0,17	500000	106480
3	7	10	12	0,08	0,17	500000	106854
3	7	10	12	0,1	0,17	500000	107975
3	7	10	12	0,3	0,49	500000	107637

Παρατηρήσαμε, λοιπόν, από τις παραπάνω δοκιμές ότι η χρήση ενός (1) κρυφού στρώματος με είκοσι πέντε (25) νευρώνες και ρυθμός εκμάθησης της τάξης του 0,1, δίνει ως αποτέλεσμα το μικρότερο σφάλμα από όλες τις υπόλοιπες περιπτώσεις. Οπότε, στη συνέχεια, περάσαμε στο δεύτερο στάδιο, που είχε ως στόχο τον ακριβή προσδιορισμό του ρυθμού εκμάθησης, ώστε να προκύπτει το ελάχιστο σφάλμα. Με ίδιο όριο επαναλήψεων, μειώσαμε το όριο αποδεκτού σφάλματος σε *0,08* και εκτελέσαμε πέντε (5) δοκιμές για κάθε μία από τις εξής τιμές ρυθμού εκμάθησης κοντά στο 0,1: [*0,05*, *0,10*, *0,12*, *0,15*, *0,20*].

Ρυθμός Εκμάθησης	Αριθμός Δοκιμής	Αριθμός Επαναλήψεων που τερμάτισε η Εκπαίδευση	Παρατηρήσεις
0,05	1	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	2	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	3	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	4	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	5	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
0,10	1	**383433**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	2	**104238**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	3	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	4	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	5	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
0,12	1	**481846**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	2	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	3	**120433**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	4	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	5	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
0,15	1	**38085**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	2	**279204**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	3	**176555**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	4	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	5	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
0,20	1	**245551**	**Επιτεύχθηκε το επιθυμητό σφάλμα**
	2	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	3	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	4	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα
	5	500000	Δεν επιτεύχθηκε το επιθυμητό σφάλμα

Παρατήσαμε ότι στις πέντε (5) δοκιμές κάθε μίας από τις παραπάνω περιπτώσεις, ο αλγόριθμος δεν κατάφερε σε όλες να εκπαιδευτεί κατάλληλα, ώστε το σφάλμα να είναι μικρότερο από το όριο που είχαμε θέσει, οπότε η διαδικασία σταμάτησε στις 500000 επαναλήψεις. Για ρυθμό εκπαίδευσης *0,15*, επιτεύχθηκε σφάλμα μικρότερο από το όριο των *0,08*, περισσότερες φορές από κάθε άλλη περίπτωση, οπότε θεωρήσαμε το ρυθμό εκμάθησης *0,15* ικανοποιητικό και τον επιλέξαμε.

Στο τρίτο και τελικό στάδιο, είχαμε ως σκοπό να επιβεβαιώσουμε ότι το ελάχιστο σφάλμα (*0,068*), που καταφέραμε να πετύχουμε σε τρεις (3) από τις πέντε (5) προηγούμενες προσπάθειες στις 500000 επαναλήψεις, μπορεί να επιτευχθεί σχεδόν πάντα μέσα σε 1500000 επαναλήψεις. Οπότε, με ένα (1) κρυφό στρώμα των είκοσι πέντε (25) νευρώνων, ρυθμό εκμάθησης *0,15*, και όρια σφάλματος και επαναλήψεων, *0,08* και *1500000*, αντίστοιχα, πραγματοποιήθηκαν οι κάτωθι δοκιμές:

Αριθμός Δοκιμής	Αριθμός Επαναλήψεων που τερμάτισε η Εκπαίδευση	Χρόνος Ολοκλήρωσης Εκπαίδευσης σε ms
1	82475	10377
2	174499	22875
3	210340	27580
4	106555	13575
5	661909	83932
6	657490	80520
7	370457	47274
8	1254824	157762
9	476365	63046
10	305109	36785

Συμπερασματικά, η διαμόρφωση, που αναφέρθηκε στην προηγούμενη παράγραφο, είναι η ιδανική διαμόρφωση, καθώς καταφέραμε να εκπαιδεύσουμε το Νευρωνικό μας Δίκτυο με το ελάχιστο δυνατό σφάλμα του *0,068* και στις δέκα (10) δοκιμές που εκτελέσαμε.

3.7 Ασφάλεια Εφαρμογής

Στην εποχή που ζούμε οι κακόβουλες ενέργειες στο Διαδίκτυο έχουν αυξηθεί ραγδαία. Αυτό συνεπάγεται ότι μία ολοκληρωμένη διαδικτυακή εφαρμογή οφείλει να υλοποιεί μέτρα προστασίας από όλων των ειδών τις επιθέσεις και απειλές με σκοπό να προστατέψει τους χρήστες της, την ιδιωτικότητά τους και τα δεδομένα τους. Στις πιο συχνές επιθέσεις κατά διαδικτυακών εφαρμογών ανήκουν οι επιθέσεις CSRF, XSS, Injection κ.τ.λ., τις οποίες αντιμετωπίζει επιτυχώς η εφαρμογή μας.

Κατά την επίθεση CSRF (Cross-Site Request Forgery), ο επιτιθέμενος ξεγελάει το θύμα να ανοίξει ένα εικονικό website ή να πατήσει ένα σύνδεσμο από e-mail του, το οποίο έχει ως αποτέλεσμα την εκτέλεση κακόβουλου κώδικα στην ιστοσελίδα-στόχο, ο οποίος φαίνεται να εκτελείται από το θύμα, εφόσον είναι ήδη συνδεδεμένος σε αυτή. Ο τρόπος αντιμετώπισης της συγκεκριμένης επίθεσης είναι με web tokens. Ο Server, κατά το login του χρήστη, δημιουργεί ένα token με τη μέθοδο JSON Web Token, όπως περιγράφεται αναλυτικά στο [14], και του το στέλνει. Ο Client αποθηκεύει το token σε μία session μεταβλητή και σε κάθε request που κάνει, από εκείνο το σημείο και μετά, το στέλνει πίσω στο Server. Αν ο

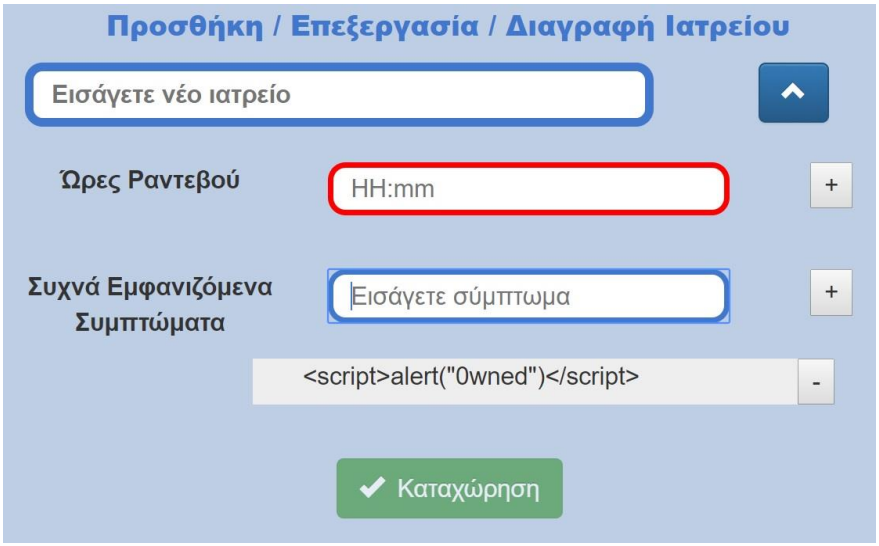
Server επαληθεύσει το token και αυθεντικοποιήσει το χρήστη, θεωρεί το request έγκυρο και το εκτελεί. Σε διαφορετική περίπτωση, δεν το εκτελεί. Όμοια διαδικασία με διαφορετικά tokens συμβαίνει για τους γιατρούς και το διαχειριστή του συστήματος.

```
addAppointment(newAppointment: Appointment): Observable<Appointment> {
    let patientHeaders = new Headers();
    let token = sessionStorage.getItem('token');

    if (token) {
        patientHeaders = new Headers({ 'Authorization': 'Bearer ' + token });
        patientHeaders.append('Content-Type', 'application/json');
    }

    return this.http
              .post('http://localhost:3000/appointment', JSON.stringify(newAppointment), {headers : patientHeaders})
              .map(res => res.json());
}
```

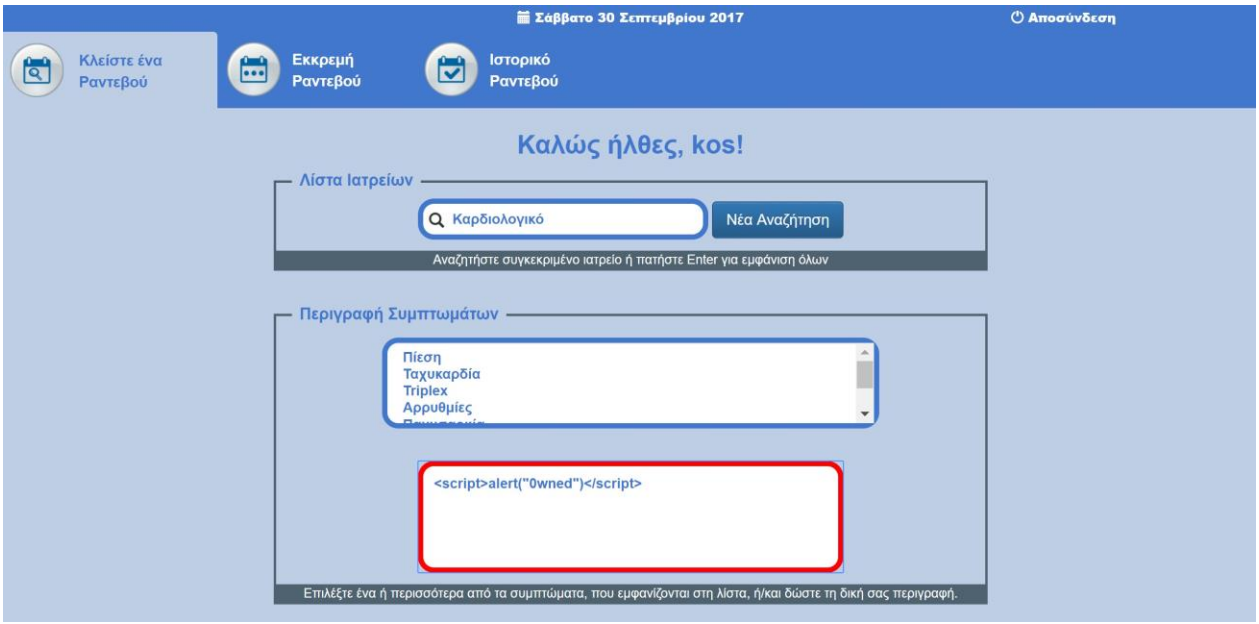
Εικόνα 3.24: Χρήση web token σε κάθε request που πραγματοποιεί η εφαρμογή μας.

Κατά την επίθεση XSS (Cross-Site Scripting), ο επιτιθέμενος καταφέρνει να γράψει client-side script κώδικα σε μία νόμιμη ιστοσελίδα, ο οποίος εκτελείται στον browser ενός άλλου χρήστη της ιστοσελίδας, όταν την ανοίξει ή προβεί σε κάποια συγκεκριμένη ενέργεια. Παράδειγμα σε ένα blog, όχι σωστά προστατευμένο, ο επιτιθέμενος θα μπορούσε να γράψει ένα κακόβουλο script στο πεδίο του μηνύματος και να το αποθηκεύσει. Από εκείνη τη στιγμή και μετά, αν κάποιος χρήστης άνοιγε το blog, θα έτρεχε το script στο μηχάνημά του. Ο τρόπος αντιμετώπισης σε αυτή την περίπτωση είναι να γίνει «*Input Sanitization*». Με τον όρο αυτό εννοούμε τη διαδικασία καθαρισμού του κειμένου, που γράφει ο χρήστης σε κάποιο πεδίο. Το framework της Angular2 έχει ενσωματωμένες μεθόδους για το σωστό και ολοκληρωμένο sanitization κάθε εισαγωγής του χρήστη, αλλά ως επιπλέον μέτρο προστασίας, έχουμε εισάγει και custom sanitization στην εφαρμογή μας.

Στις παρακάτω εικόνες παρουσιάζουμε ενδεικτικά παραδείγματα. Στην εικόνα 3.25 παρουσιάζεται η περίπτωση που ένας κακόβουλος χρήστης προσπαθήσει να εισάγει στην εφαρμογή ένα script και η Angular2 το χειρίζεται σαν κείμενο και δεν το εκτελεί. Στην εικόνα 3.26, φαίνεται ένα πεδίο της διεπαφής του ασθενή, που θα μπορούσε να είναι ευάλωτο. Το πρώτο και βασικό μέτρο προστασίας, που έχει ληφθεί, είναι ο custom έλεγχος της εισόδου του χρήστη από εμάς, ώστε να περιέχει μόνο γράμματα, αριθμούς και ορισμένα σύμβολα (εικόνα 3.27). Τα σύμβολα «<, >» δε θεωρούνται επιτρεπτά, οπότε το πλαίσιο του πεδίου εισαγωγής «κοκκινίζει» και δεν ενεργοποιείται το κουμπί επιβεβαίωσης του ραντεβού. Ωστόσο, αν ο επιτιθέμενος με κάποιο τρόπο καταφέρει να παρακάμψει αυτόν τον έλεγχο, τότε περνάμε στο δεύτερο μέτρο προστασίας, το sanitization από την Angular2, όπως φαίνεται στην εικόνα 3.28. Σε αυτή απεικονίζεται η αυτόματη αφαίρεση των «<script></script>».



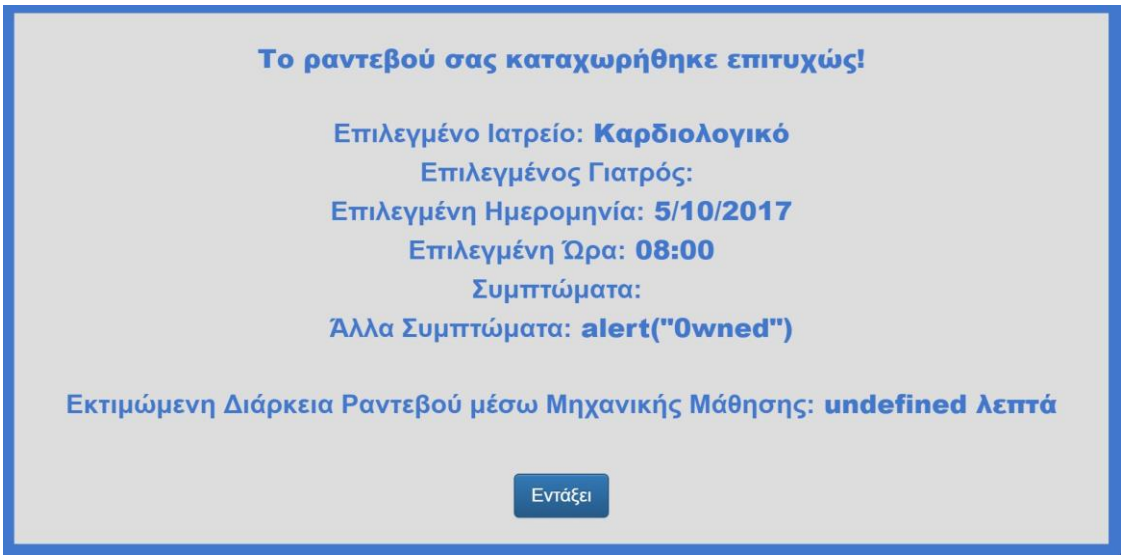
Εικόνα 3.25: Το framework Angular2 πραγματοποιεί sanitization σε κάθε είσοδο του χρήστη. Σε αυτή την περίπτωση, χειρίζεται την εισαγωγή του χρήστη σαν κείμενο.



Εικόνα 3.26: Εισαγωγή κακόβουλου κώδικα. Η εφαρμογή μας διαθέτει έλεγχο των εισαγωγών του χρήστη και δεν ενεργοποιεί το κουμπί της επιβεβαίωσης. Ακόμα και να παρακαμπτόταν αυτός ο έλεγχος, η Angular2 πραγματοποιεί αυτόματα sanitization.

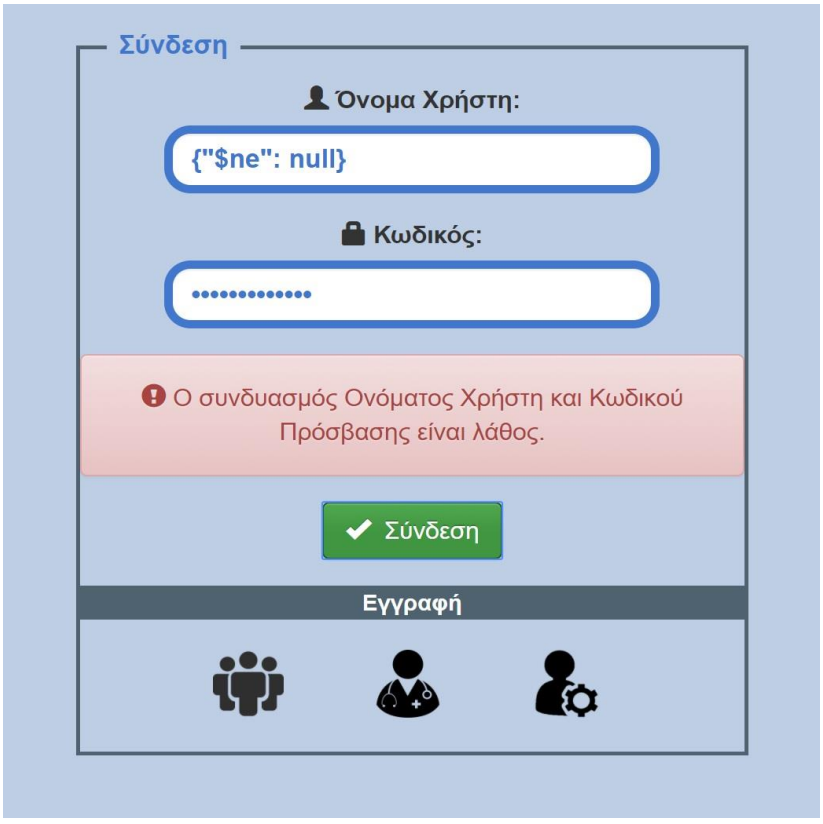
```
addAppointment(): void {
  let regExForProblemDesc = new RegExp(/^[ΑΕΗΙΥΟΩάέώίήύόΑ-Ωα-ωA-Za-z0-9\s\?\,\.\:\;\[\]\{\}\-\(\)\_\=\+\!]*$/);

  if (  this.selectedHour &&
        this.selectedDoctor._id !== '0' &&
        ((this.selectedStandardProblems.length > 0 && this.problemDescription === '') ||
         (this.selectedStandardProblems.length > 0 && this.problemDescription !== '' && regExForProblemDesc.test(this.problemDescription)) ||
         (this.selectedStandardProblems.length === 0 && this.problemDescription !== '' && regExForProblemDesc.test(this.problemDescription)))
    ) {
```

Εικόνα 3.27: *Custom sanitization στην εισαγωγή του χρήστη στο πεδίο «Περιγραφή Συμπτώματος».*



Εικόνα 3.28: Αυτόματο sanitization από την Angular2. Τα «<script></script>» αφαιρέθηκαν.

Επιπλέον, η εφαρμογή μας είναι προστατευμένη και από επιθέσεις «NoSQL Injections» ([15], [16]), χάρη στις ενσωματωμένες μεθόδους της Angular2 για sanitization, όπως προαναφέραμε. Κάθε ερώτημα που φτάνει στη βάση είναι σε μορφή κειμένου, οπότε ακόμα και να περιλαμβάνει ύποπτους χαρακτήρες, όπως «”, $, {, }», δεν εκτελείται κάποια μη επιτρεπόμενη ενέργεια σε αυτή.



Εικόνα 3.29: *Προσπάθεια για «NoSql Injection» χωρίς αποτέλεσμα.*

Τέλος, αν ένας επιτιθέμενος, με κάποιο τρόπο, καταφέρει να αποκτήσει πρόσβαση στη βάση, δεν θα μπορέσει να δει τους κωδικούς κάθε χρήστη της εφαρμογής, καθώς αυτοί είναι αποθηκευμένοι σε *hash* μορφή. Για την μετατροπή των κωδικών σε hash, χρησιμοποιήσαμε την βιβλιοθήκη «*bcrypt-nodejs*». Κατά τη μετατροπή, η εν λόγω βιβλιοθήκη χρησιμοποιεί αυτόματα και «*salt*».


```
{
   "_id": {
      "$oid": "590d86cbf36d281fc3b9067e"
   },
   "name": "Κωνσταντίνος",
   "lastname": "Ράμμος",
   "username": "kos",
   "password": "$2a$10$CSmyMlwfTSsLryQBbLxfsOoGCZCNWaqGbl9jNd1CzG.Vw5DdRnhZm",
   "gender": "male",
   "dateOfBirth": "06-10-1990",
   "email": "kos@gmail.com",
   "tel": "2109955111"
}
```


Εικόνα 3.30: *Οι κωδικοί των χρηστών αποθηκεύονται στη βάση σε hash μορφή.*

ΚΕΦΑΛΑΙΟ 4 –

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ

ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Η σελίδα αφέθηκε σκόπιμα κενή

ΚΕΦΑΛΑΙΟ 4 - ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Ανακεφαλαιώνοντας, στο πλαίσιο της παρούσας μεταπτυχιακής διατριβής, αναπτύξαμε μία εφαρμογή για να κλείνουν οι ασθενείς, διαδικτυακά, ραντεβού σε ένα νοσοκομείο και ακόμα, οι γιατροί, από την πλευρά τους, να έχουν τη δυνατότητα να παρακολουθούν αυτά τα ραντεβού. Αρχικά, αναφερθήκαμε σε ήδη υπάρχουσες παρεμφερείς εφαρμογές και επισημάναμε τα πλεονεκτήματα της δικιάς μας εφαρμογής σε σχέση με αυτές. Στη συνέχεια, κάναμε μία παρουσίαση των δυνατοτήτων της εφαρμογής μας και συντάξαμε το εγχειρίδιο χρήσης. Έπειτα, αναλύσαμε την αρχιτεκτονική της. Πιο συγκεκριμένα, καλύψαμε το θεωρητικό υπόβαθρο των γλωσσών προγραμματισμού που χρησιμοποιήσαμε και περιγράψαμε τη διαδικασία εγκατάστασής τους, σχεδιάσαμε τα UML διαγράμματα Περιπτώσεων Χρήσης, για την καλύτερη κατανόηση της εφαρμογής μας, παρουσιάσαμε τη δομή των αρχείων της, τόσο στη μεριά του Server, όσο και στη μεριά του Client, αναλύσαμε τον τρόπο υλοποίησης αξιοσημείωτων τμημάτων κώδικα και επίσης την μελετήσαμε από την πλευρά της Ασφάλειας Λογισμικού, ώστε να αποδείξουμε ότι είναι ασφαλής για διαδικτυακή χρήση. Τέλος, αναπτύξαμε τη θεωρία πίσω από τη Μηχανική Μάθηση και τα Νευρωνικά Δίκτυα, τον τρόπο κωδικοποίησης των μεταβλητών εισόδου και εξόδου του Νευρωνικού Δικτύου, που χρησιμοποιήσαμε στην εφαρμογή, και τη διαδικασία που ακολουθήσαμε για να φτάσουμε στην κατάλληλη διαμόρφωσή του.

Η εφαρμογή μας δεν αναπτύχθηκε με μοναδικό σκοπό την ολοκλήρωση της παρούσας μεταπτυχιακής διατριβής, αλλά αναπτύχθηκε υπό το πρίσμα ότι μελλοντικά θα χρησιμοποιηθεί σε κάποιο νοσοκομείο για να αναβαθμίσει πλήρως τη διαδικασία κλεισίματος των ραντεβού, από την πλευρά των ασθενών, και παρακολούθησής τους, από την πλευρά των γιατρών. Συνεπώς, είναι μία custom, αλλά ταυτόχρονα πλήρως υλοποιημένη εφαρμογή, έτοιμη για κυκλοφορία, με πολλές δυνατότητες και λειτουργίες, για τις οποίες έχουν προβλεφθεί και υλοποιηθεί όλες οι περιπτώσεις χρήσης. Ακόμα, η ενσωμάτωση του Νευρωνικού Δικτύου, την καθιστά, όχι μόνο μια απλή εφαρμογή, αλλά μια «ευφυή» εφαρμογή, που θα εκτοξεύσει τη διαδικασία των ραντεβού σε ένα άλλο επίπεδο.

Επιπροσθέτως, θα ήθελα να αναφέρω ότι η εν λόγω εφαρμογή έχει αναπτυχθεί σε Javascript από την πλευρά του Server και από την πλευρά του Client, χρησιμοποιήθηκε η Angular2. Ως βάση δεδομένων, χρησιμοποιήθηκε η MongoDB. Για όλες αυτές τις γλώσσες και τεχνολογίες, δεν είχα καμία προηγούμενη γνώση και τις έμαθα και χρησιμοποίησα κατευθείαν στην παρούσα διατριβή. Ο λόγος είναι ότι ήθελα να προσπεράσω τις καθιερωμένες πια και πολυχρησιμοποιημένες γλώσσες και τεχνολογίες, όπως για παράδειγμα η PHP και η MySQL βάση, και να κατευθυνθώ προς νέες τεχνολογίες, όχι ακόμα ευρέως χρησιμοποιημένες, αλλά αναπτυσσόμενες. Εξάλλου, η Angular2 έχει ένα χαρακτηριστικό που την καθιστά όλο και πιο αγαπητή ανάμεσα στους προγραμματιστές: μεταφέρει το φόρτο των περισσοτέρων λειτουργιών της εφαρμογής στο *hardware* του Client, ελαφραίνοντας τον Server, κάτι το οποίο στις μέρες μας, την εποχή δηλαδή των Μεγάλων Δεδομένων, είναι ζωτικής σημασίας.

Κλείνοντας, επισημαίνουμε ότι, αν και έχουν ενσωματωθεί στην εφαρμογή μας πάρα πολλές λειτουργίες και δυνατότητες, υπάρχουν ακόμα ορισμένες ιδέες, που θα μπορούσαν να υλοποιηθούν σε μια δεύτερη έκδοση αυτής. Ενδεικτικά, αναφέρουμε τις κάτωθι:

- Η πιο βασική αναβάθμιση πάνω στην εφαρμογή μας είναι η χρήση του Νευρωνικού Δικτύου, για την επίτευξη πιο συμπυκνωμένης δόμησης των time slots των ραντεβού, με σκοπό την μείωση του χρόνου αναμονής των ασθενών και του κενού χρόνου, μεταξύ των ραντεβού, των γιατρών. Την παρούσα στιγμή, τα time slots κάθε ιατρείου, ορίζονται από το διαχειριστή του συστήματος και είναι σταθερά. Θα μπορούσαμε να τα μετατρέψουμε σε δυναμικά. Θα ορίζαμε το πρώτο time slot του ιατρείου στις 08:00, για παράδειγμα. Όταν έκλεινε κάποιος ασθενής ραντεβού στις 08:00, θα δήλωνε και τα συμπτώματά του. Το ΝΔ θα εκτιμούσε τη διάρκεια του εν λόγω ραντεβού, λόγου χάρη 10 λεπτά, και θα άνοιγε το επόμενο time slot στις 08:10. Συνεπώς, θα περιορίζαμε το νεκρό χρόνο στο ελάχιστο, με αποτέλεσμα την άρτια λειτουργία του νοσοκομείου και την άμεση εξυπηρέτηση των ασθενών.

- Ακόμα, πολύ χρήσιμη για την αναβάθμιση της εφαρμογής μας θα ήταν η περαιτέρω αξιοποίηση της Μηχανικής Μάθησης. Θα μπορούσαμε, για παράδειγμα, να χρησιμοποιήσουμε τον αλγόριθμο «*k-means*». Αυτός δεν ανήκει στην κατηγορία «*Μάθηση με Επιτήρηση*», όπως τα Νευρωνικά Δίκτυα, αλλά στην κατηγορία «*Μάθηση χωρίς Επιτήρηση*». Θα εισαγόταν στον εν λόγω αλγόριθμο ένα σύνολο δεδομένων με πληροφορίες των ολοκληρωμένων ραντεβού, όπως η ηλικία του ασθενή, το φύλο του και το ιατρείο που δήλωσε, και θα προέκυπταν, ως αποτέλεσμα, στατιστικά στοιχεία σχετικά με ποιες ομάδες ασθενών, αναφορικά με το φύλο τους και την ηλικία τους, επισκέπτονται το εκάστοτε ιατρείο.

- Επιπροσθέτως, θα ήταν πολύ χρηστικό να υλοποιηθεί στο μέλλον μία φόρμα επικοινωνίας για τους ασθενείς και τους γιατρούς, είτε για να επικοινωνούν κατευθείαν με το διαχειριστή του συστήματος για αναφορά τυχόν προβλημάτων, είτε να επικοινωνούν μεταξύ τους για συμβουλές ή διευκρινήσεις ιατρικού περιεχομένου.

- Επίσης, ένα εν δυνάμει πρόβλημα της εφαρμογής είναι στην επιλογή γιατρού από κάποιον ασθενή. Προς το παρόν, ο ασθενής μπορεί να επιλέξει όποιο γιατρό θέλει, από αυτούς που ανήκουν στο επιλεγμένο ιατρείο, και υπό την προϋπόθεση ότι δεν είναι κλεισμένες οι ώρες του και δεν έχει άδεια. Το πρόβλημα, στη συγκεκριμένη περίπτωση, είναι το εξής: έστω ότι ο ένας γιατρός ενός ιατρείου είναι πολύ καλός και ο άλλος όχι και τόσο. Οι ασθενείς, που θέλουν να κλείσουν ραντεβού στο συγκεκριμένο ιατρείο, θα επιλέγουν τον πρώτο γιατρό, με αποτέλεσμα τα ραντεβού αυτού να γεμίζουν, ενώ του δεύτερου να μένουν κενά. Θα μπορούσε, λοιπόν, να υλοποιηθεί ένας αλγόριθμος, ο οποίος να επέτρεπε σε κάθε χρήστη να επιλέξει κατάλληλα γιατρό, ώστε τα ραντεβού και των δύο να αυξάνονται ομοιόμορφα.

- Τέλος, σε μία μελλοντική αναβάθμιση της εφαρμογής, θα μπορούσαμε να τροποποιήσουμε κατάλληλα τον κώδικα, ώστε, αν όλα τα ραντεβού μίας ημέρας έχουν κλείσει, η ημέρα να γίνεται κόκκινη πάνω στο ημερολόγιο. Την παρούσα στιγμή, ο ασθενής πρέπει πρώτα να επιλέξει ημέρα και γιατρό και στη συνέχεια θα εμφανιστούν τα time slots. Αν είναι όλα κλεισμένα, ο ασθενής θα πρέπει να ξαναεπιλέξει ημέρα και γιατρό, κάτι το οποίο δεν είναι φιλικό προς το χρήστη.

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΠΗΓΕΣ

Η σελίδα αφέθηκε σκόπιμα κενή

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΠΗΓΕΣ

[1] Hsiang-Fu and al. Feature Engineering and Classifier Ensemble for KDD Cup 2010. Taiwan. Department of Computer Science and Information Engineering, National Taiwan University. *JMLR: Workshop and Conference Proceedings* 1: 1-16. 2010

[2] Phil Brierley, David Vogel, Randy Axelrod. How we did it – Team Market Makers. *Heritage Provider Network Health Prize, Round 1 Milestone Prize*. 2011

[3] Jonathan P. F. Strahl. Patient Appointment Scheduling System with supervised learning prediction. Espoo. Aalto University, School of Science. *Master's Thesis. 2015*

[4] Νευρωνικό Δίκτυο – Βικιπαίδεια, URL: https://el.wikipedia.org/wiki/Νευρωνικό_δίκτυο

[5] Feature Engineering - Slideshare, URL: https://www.slideshare.net/HJvanVeen/feature-engineering-72376750

[6] Discover Feature Engineering, How to engineer features and how to get good at it, URL: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

[7] GitHub-cazala/synaptic: architecture-free neural network, URL: https://github.com/cazala/synaptic/wiki

[8] TIOBE Index – The Software Quality Company, URL: https://www.tiobe.com/tiobe-index/

[9] Angular2, URL: https://angular.io/

[10] Typescript – Javascript that scales, URL: https://www.typescriptlang.org/

[11] NodeJS, URL: https://nodejs.org/en/

[12] Express – NodeJS web application framework, URL: https://expressjs.com/

[13] MongoDB, URL: https://www.mongodb.com/

[14] JSON Web Tokens, URL: https://jwt.io

[15] NoSQL Injection in MongoDB, URL: https://zanon.io/posts/nosql-injection-in-mongodb

[16] Hacking NodeJS and MongoDB, URL: https://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

[17] Search Engine Optimization Starter Guide – Google, URL: https://static.googleusercontent.com/media/www.google.com/el//webmasters/docs/search-engine-optimization-starter-guide.pdf

[18] DocASAP – Διαδικτυακή Εφαρμογή για Ραντεβού σε Γιατρούς, URL: https://docasap.com/

[19] DoctorAnytime – Εφαρμογή για Ραντεβού σε Γιατρούς, URL: https://www.doctoranytime.gr/

[20] NowDoctor – Διαδικτυακή Εφαρμογή για Ραντεβού σε Γιατρούς, URL: https://www.nowdoctor.gr/

Η σελίδα αφέθηκε σκόπιμα κενή

ΠΑΡΑΡΤΗΜΑ

Η σελίδα αφέθηκε σκόπιμα κενή

ΠΑΡΑΡΤΗΜΑ

neural-network.service.ts

```
import { Http, Headers }            from '@angular/http';
import { Injectable }               from '@angular/core';
import { Observable }               from 'rxjs/Observable';
import { Layer, Network, Trainer }  from 'synaptic';
import { Status }                   from '../Classes/status';
import { Iatreio }                  from '../Classes/iatreio';
import { Appointment }              from '../Classes/appointment';
import { IatreioService }           from '../Services/iatreio.service';
import { AppointmentService }       from '../Services/appointment.service';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/switchMap';

// ******************** Βοηθητικές Κλάσεις ********************
// Ορισμός Νευρωνικού Δικτύου με 2 κρυφά επίπεδα.
export class Perceptron extends Network {
    constructor(input: number, hidden: number, output: number) {
        super();

        // Create the layers
        let inputLayer = new Layer(input);
        let hiddenLayer = new Layer(hidden);
        let outputLayer = new Layer(output);

        // Connect the layers
        inputLayer.project(hiddenLayer);
        hiddenLayer.project(outputLayer);

        // Set the layers
        this.set({
            input: inputLayer,
            hidden: [ hiddenLayer ],
            output: outputLayer
        });
    }
}

// Ορισμός κλάσης κωδικοποιημένων δεδομένων εισόδου του ΝΔ.
export class EncodedDataForNN {
    iatreio: string;
    encodeForNN: number;
    symptoms: EncodedSymptomForNN[];
}

// Ορισμός κλάσης κωδικοποιημένων συμπτωμάτων.
export class EncodedSymptomForNN {
    name: string;
    encodeForNN: number;
}

// Ορισμός κλάσης κάθε στοιχείου του Training Set.
export class TrainingSetObj {
    input: number[];
    output: number[];
}
```

```
// ********************** Κυρίως Κλάση **********************
@Injectable()
export class NeuralNetworkService {
    private headers = new Headers();

    constructor(private iatreioService: IatreioService,
                private appointmentService: AppointmentService,
                private http: Http) { }

    // Σύνθεση και αποθήκευση στη βάση, της κωδικοποίησης των δεδομένων εισόδου του ΝΔ.
    // Εκπαίδευση του Νευρωνικού Δικτύου.
    updateDataEncodingAndTrainNN(): Observable<Status> {
        let iatreia: Iatreio[] = [];
        let encodedDataForNN: EncodedDataForNN[] = [];
        let symptoms: EncodedSymptomForNN[] = [];
        let appointments: Appointment[] = [];
        let trainingSet: TrainingSetObj[] = [];

        // Define number of neurons
        let noOfNeuronsInput = 2;
        let noOfNeuronsHidden = 25;
        let noOfNeuronsOutput = 1;

        // Create the neural network
        let myPerceptron = new Perceptron(noOfNeuronsInput, noOfNeuronsHidden, noOfNeuronsOutput);

        // Define a trainer for training the network
        let myTrainer = new Trainer(myPerceptron);

        // Training options
        let trainingOptions = {
            rate: 0.15,
            iterations: 1500000,
            error: 0.08,
            shuffle: true,
            log: 10000,
            cost: Trainer.cost.BINARY  // or CROSS_ENTROPY or MSE
        };
```

```
        // Downloading information of Iatreia Collection and encoding data.
        // switchMap() is used in order each function to be called after the previous one has finished.
        return this.iatreioService
                   .getIatreia()
                   .switchMap(returnedIatreia => {
                       iatreia = returnedIatreia;
                       for (let i = 0; i < iatreia.length; i++) {
                           let encodedDataForNNObj = new EncodedDataForNN();
                           encodedDataForNNObj.iatreio = iatreia[i].name;
                           encodedDataForNNObj.encodeForNN = (i + 1) / 10; // Κωδικοποίηση ιατρείων.

                           symptoms = [];
                           for (let j = 0; j < iatreia[i].problems.length; j++) {
                               let encodedSymptomForNNObj = new EncodedSymptomForNN();
                               encodedSymptomForNNObj.name = iatreia[i].problems[j];
                               encodedSymptomForNNObj.encodeForNN = (j + 1) / 10; // Κωδικοποίηση συμπτωμάτων.
                               symptoms[j] = encodedSymptomForNNObj;
                           }

                           encodedDataForNNObj.symptoms = symptoms;
                           encodedDataForNN[i] = encodedDataForNNObj;
                       }
                       // Delete old encoding data from DB.
                       return this.deleteEncodedDataForNN();

                   }).switchMap(status => {
                       // Insert the new encoding data into the DB.
                       return this.insertNewEncodedDataForNN(encodedDataForNN);

                   }).switchMap(status => {
                       console.log('Η κωδικοποίηση των δεδομένων εισόδου-εξόδου του Νευρωνικού Δικτύου αποθηκεύτηκε επιτυχώς στη βάση.');

                       // Compose Training Set according to the completed appointments
                       let today = new Date();
                       let currentDate = today.getDate();
                       let currentMonth = today.getMonth() + 1;
                       let currentYear = today.getFullYear();

                       // Call next service function after completion of the previous one.
                       return this.appointmentService.getAllPastAppointments(currentDate, currentMonth, currentYear);

                   }).switchMap(returnedAppointments => {
                       appointments = returnedAppointments;

                       for (let i = 0; i < appointments.length; i++) {
                           let findIatreioInEncodedData = this.findInEncodedDataArray(appointments[i].iatreio, encodedDataForNN);
                           // Ελέγχουμε να έχει γίνει το ραντεβού, άρα να έχει μη-μηδενική διάρκεια.
                           // Επίσης ελέγχουμε να μην έχει απομείνει όνομα ιατρείου στα καταχωρημένα ραντεβού,
                           // το οποίο δεν υπάρχει πλέον στο collection IATREIA.
                           if (findIatreioInEncodedData.exists && appointments[i].duration !== 0) {
                               for (let j = 0; j < appointments[i].standardProblems.length; j++) {
                                   let findSymptomInEncodedSymptoms = this.findInEncodedSymptomsArray(appointments[i].standardProblems[j], encodedDataForNN[findIatreioInEncodedData.index].symptoms);
                                   if (findSymptomInEncodedSymptoms.exists) {
                                       let obj = new TrainingSetObj();
                                       obj.input = [];
                                       obj.output = [];
                                       obj.input.push(encodedDataForNN[findIatreioInEncodedData.index].encodeForNN);
                                       obj.input.push(encodedDataForNN[findIatreioInEncodedData.index].symptoms[findSymptomInEncodedSymptoms.index].encodeForNN);
                                       obj.output.push(appointments[i].duration);
                                       trainingSet.push(obj);
                                   }
                               }
                           }
                       }
```

```
                    // Εκπαίδευση Νευρωνικού Δικτύου και απεικόνιση αποτελεσμάτων.
                    console.log('******* Δεδομένα εκπαίδευσης *******');
                    console.log(trainingSet);
                    console.log('      ');
                    console.log('******** Εκκίνηση εκπαίδευσης ********');
                    let trainingResults = myTrainer.train(trainingSet, trainingOptions);
                    console.log('******* Ολοκλήρωση εκπαίδευσης *******');
                    console.log('      ');
                    console.log('************ Αποτελέσματα ************');
                    console.log('   Τελικό Σφάλμα: ' + trainingResults.error);
                    console.log('   Αριθμός Επαναλήψεων: ' + trainingResults.iterations);
                    console.log('   Διάρκεια Εκπαίδευσης: ' + trainingResults.time + ' ms');
                    console.log('**************************************');
                    console.log('      ');

                    // Call next service function after completion of the previous one.
                    return this.saveNeuralNetwork(myPerceptron.toJSON());

                }).switchMap(status => {
                    console.log('Το εκπαιδευμένο ΝΔ αποθηκεύτηκε επιτυχώς στη βάση.');
                    return Observable.of<Status>(status);
                });
    } // end function


    // Βοηθητική μέθοδος για τη σύνθεση των δεδομένων εισόδου στο ΝΔ.
    findInEncodedDataArray(value: string, ObjArray: EncodedDataForNN[]): {exists: boolean, index: number} {
        for (let i = 0; i < ObjArray.length; i++) {
            if (value === ObjArray[i].iatreio) {
                return {exists: true, index: i};
            }
        }
        return {exists: false, index: -1};
    }


    // Βοηθητική μέθοδος για τη σύνθεση των δεδομένων εισόδου στο ΝΔ.
    findInEncodedSymptomsArray(value: string, ObjArray: EncodedSymptomForNN[]): {exists: boolean, index: number} {
        for (let i = 0; i < ObjArray.length; i++) {
            if (value === ObjArray[i].name) {
                return {exists: true, index: i};
            }
        }
        return {exists: false, index: -1};
    }


    // Διαγραφή των κωδικοποιημένων δεδομένων από τη βάση.
    deleteEncodedDataForNN(): Observable<Status> {
        let adminToken = sessionStorage.getItem('adminToken');

        if (adminToken) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + adminToken });
        }

        return this.http
                   .delete('http://localhost:3000/encoded-data-for-nn', {headers: this.headers})
                   .map(res => res.json() as Status);
    }
```

```
    // Εισαγωγή των νέων κωδικοποιημένων δεδομένων στη βάση.
    insertNewEncodedDataForNN(newEncodedData: EncodedDataForNN[]): Observable<Status> {
        let adminToken = sessionStorage.getItem('adminToken');

        if (adminToken) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + adminToken });
            this.headers.append('Content-Type', 'application/json');
        }

        return this.http
                    .post('http://localhost:3000/encoded-data-for-nn', newEncodedData, {headers: this.headers})
                    .map(res => res.json() as Status);
    }


    // Ανάκτηση κωδικοποιημένων δεδομένων.
    getEncodedDataForNN(iatreio: string): Observable<EncodedDataForNN[]> {
        let token = sessionStorage.getItem('token');

        if (token) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + token });
        }

        return this.http
                    .get('http://localhost:3000/encoded-data-for-nn/' + iatreio, {headers: this.headers})
                    .map(res => res.json() as EncodedDataForNN[]);
    }


    // Αποθήκευση του εκπαιδευμένου ΝΔ στη βάση.
    saveNeuralNetwork(neuralNetworkAtJson: any): Observable<Status> {
        let adminToken = sessionStorage.getItem('adminToken');

        if (adminToken) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + adminToken });
            this.headers.append('Content-Type', 'application/json');
        }

        return this.http
                    .put('http://localhost:3000/save-neural-network', neuralNetworkAtJson, {headers: this.headers})
                    .map(res => res.json() as Status);
    }


    // Ανάκτηση του εκπαιδευμένου ΝΔ από τη βάση.
    retrieveNeuralNetwork(): Observable<Object> {
        let token = sessionStorage.getItem('token');

        if (token) {
            this.headers = new Headers({ 'Authorization': 'Bearer ' + token });
        }

        return this.http
                    .get('http://localhost:3000/retrieve-neural-network', {headers: this.headers})
                    .map(res => res.json());
    }

} // end of Service
```

Εικόνα 1: *Αρχείο «neural-network.service.ts» για την υλοποίηση του Νευρωνικού Δικτύου της εφαρμογής μας.*

```
<div class="col-xs-12 appointment-data">
    <div id="container2">
        <div id="container1">
            <div id="col1">
                <div class="col-sm-6 col-xs-12 inside-box label">Γιατρός :</div>
                <div class="col-sm-6 col-xs-12 inside-box">{{ appointment.doctor }}</div>
                <div class="col-sm-6 col-xs-12 inside-box label">Ημερομηνία :</div>
                <div class="col-sm-6 col-xs-12 inside-box">{{ appointment.date[0] }}/{{ appointment.date[1] }}/{{ appointment.date[2] }}</div>
                <div class="col-sm-6 col-xs-12 inside-box label">Ώρα :</div>
                <div class="col-sm-6 col-xs-12 inside-box">{{ appointment.time }}</div>
                <div class="col-xs-12 border-line-for-small-devices"></div>
            </div>
            <div id="col2">
                <div class="col-xs-12 inside-box label">Συμπτώματα:</div>
                <div class="col-xs-12 inside-box">{{ appointment.standardProblems }}</div>
                <div class="col-xs-12 inside-box">{{ appointment.problemDescription }}</div>
            </div>
        </div>
    </div>
</div>
```

Εικόνα 2: *Τμήμα κώδικα HTML, που υλοποιεί το κόλπο για ίσο ύψος στηλών με δυναμικό περιεχόμενο, δηλαδή περιεχόμενο που δεν είναι γνωστό εκ των προτέρων.*

```
#container2 {
    float:left;
    width:100%;
    position:relative; /*this is to solve an Internet Explorer bug that stops the overflow:hidden from working*/
    overflow:hidden;
    background:#fff;
}
#container1 {
    float:left;
    width:100%;
    position:relative;
    right: 55%;  /* Πλάτος δεξιάς στήλης */
    background:#fff;
    border-right: 2px solid #4e6270;
}
#col1 {
    float:left;
    width:43%;  /* Πλάτος αριστερής στήλης - 2% αριστερό-δεξί περιθώριο */
    position:relative;
    left:56%;  /* Πλάτος δεξιάς στήλης + 1% αριστερό περιθώριο */
    overflow:hidden;
    padding-top: 15px;
}
#col2 {
    float:left;
    width:53%;  /* Πλάτος δεξιάς στήλης - 2% αριστερό-δεξί περιθώριο */
    position:relative;
    left:58%;  /* Πλάτος δεξιάς στήλης + 3% περιθώριο (αριστερό-δεξί αριστερής στήλης & αριστερό περιθώριο δεξιάς στήλης */
    overflow:hidden;
    padding-bottom: 15px;
}
```

Εικόνα 3: *Τμήμα κώδικα CSS, που υλοποιεί το κόλπο για ίσο ύψος στηλών με δυναμικό περιεχόμενο, δηλαδή περιεχόμενο που δεν είναι γνωστό εκ των προτέρων.*

Collection "admin"


```
{
    "_id": {
        "$oid": "593ae0e2734d1d683a03c93d"
    },
    "username": "admin",
    "password": "$2a$10$lZzuB5ea67wiUWtJuLRTlejX1ebY5x50DZWyal1OtjpRGAxc.8cie"
}
```


Collection "users"


```
{
    "_id": {
        "$oid": "590d86cbf36d281fc3b9067e"
    },
    "name": "Κωνσταντίνος",
    "lastname": "Ράμμος",
    "username": "kos",
    "password": "$2a$10$CSmyMlwfTSsLryQBbLxfsOoGCZCNWaqGbl9jNd1CzG.Vw5DdRnhZm",
    "gender": "male",
    "dateOfBirth": "06-10-1990",
    "email": "kos@gmail.com",
    "tel": "2109955111"
}
```


Collection "doctors"


```
{
    "_id": {
        "$oid": "59732fc2d0da7a23548c3de6"
    },
    "iatreio": "Παθολογικό",
    "lastname": "Παπανικολάου",
    "name": "Γεώργιος",
    "username": "geopa",
    "password": "$2a$10$VTdQIrwXgEiSLxWaYPCqoe5CX3BrlAv7zcyzo7.b7KFA23VJKxLHa",
    "email": "geopa@hotmail.com",
    "tel": "2107463108",
    "startingDate": 0,
    "endingDate": 0,
    "rating": {
        "totalRate": 4,
        "totalVotes": 1
    }
}
```


Collection "iatreia"


```
{
    "_id": {
        "$oid": "595a73079195541ab08ddf26"
    },
    "name": "Καρδιολογικό",
    "appointment_hours": [
        "08:00",
        "08:30",
        "09:00",
        "09:30",
        "10:00",
        "10:30",
        "11:00",
        "11:30",
        "12:00",
        "12:30",
        "13:00"
    ],
    "problems": [
        "Πίεση",
        "Ταχυκαρδία",
        "Triplex",
        "Αρρυθμίες",
        "Παχυσαρκία",
        "Δοκιμασία κόπωσης",
        "Holter ρυθμού 24ώρου"
    ]
}
```


Collection "appointments"


```
{
    "_id": {
        "$oid": "59c631320aef8333f0d4ba4d"
    },
    "user": "kos",
    "iatreio": "Παθολογικό",
    "doctor": "Γεώργιος Παπανικολάου",
    "date": [
        25,
        10,
        2017
    ],
    "time": "08:00",
    "standardProblems": [
        "Λοιμώξεις"
    ],
    "problemDescription": "Έχω ακόμα βήχα και πυρετό. Επίσης, είναι κλειστή η μύτη μου και βγάζω πολλά φλέματα. Ο πυρετός δεν πέφτει, αν και έχω πάρει διάφορα αντιπυρετικά (Depon, Ponstan).",
    "vote": 0,
    "helpArrayForRatingStars": [
        false,
        false,
        false,
        false,
        false
    ],
    "duration": 0.2
}
```


Collection "symptoms"


```
{
    "_id": {
        "$oid": "5960fee36ae83c2084c2fd29"
    },
    "label": "Λοιμώξεις",
    "iatreio": "Παθολογικό",
    "count": 11
}
```


Collection "encodedDataForNN"


```
{
    "_id": {
        "$oid": "59c695a990f03b370003fa95"
    },
    "iatreio": "Καρδιολογικό",
    "encodeForNN": 0.5,
    "symptoms": [
        {
            "name": "Πίεση",
            "encodeForNN": 0.1
        },
        {
            "name": "Ταχυκαρδία",
            "encodeForNN": 0.2
        },
        {
            "name": "Triplex",
            "encodeForNN": 0.3
        },
        {
            "name": "Αρρυθμίες",
            "encodeForNN": 0.4
        },
        {
            "name": "Παχυσαρκία",
            "encodeForNN": 0.5
        },
        {
            "name": "Δοκιμασία κόπωσης",
            "encodeForNN": 0.6
        },
        {
            "name": "Holter ρυθμού 24ώρου",
            "encodeForNN": 0.7
        }
    ]
}
```


Εικόνα 4: *Μία ενδεικτική καταχώρηση για κάθε collection της βάσης.*