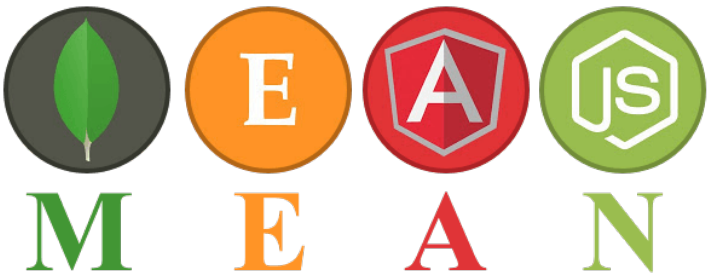
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ONLINE ΥΠΗΡΕΣΙΑΣ ΑΝΑΖΗΤΗΣΗΣ ΚΑΙ ΣΥΣΤΑΣΗΣ ΕΠΑΓΓΕΛΜΑΤΙΩΝ



Χαραλάμπους Βλάσιος

Επιβλέπων καθηγητής
Δρ. Χρυσόστομος Στύλιος

Άρτα,

Φεβρουάριος 2019

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ONLINE ΥΠΗΡΕΣΙΑΣ ΑΝΑΖΗΤΗΣΗΣ ΚΑΙ ΣΥΣΤΑΣΗΣ ΕΠΑΓΓΕΛΜΑΤΙΩΝ

Χαραλάμπους Βλάσιος

Επιβλέπων καθηγητής
Δρ. Χρυσόστομος Στύλιος

Άρτα,

Φεβρουάριος 2019

ONLINE SERVICE DESIGN AND IMPLEMENTATION FOR SEARCHING AND RECOMMEND PROFESSIONALS

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

2. Μέλος επιτροπής

3. Μέλος επιτροπής

Ο/Η Προϊστάμενος/η του Τμήματος

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία. Επίθετο, Όνομα Υπογραφή

Χαραλάμπους Βλάσιος

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ θερμά τον Δρ. Χρυσόστομο Στύλιο και τον διδάκτωρ ερευνητή Τζέριες Μπεσαρατ για την πολύτιμη υποστήριξη και καθοδήγηση καθ' όλη τη διάρκεια της συνεργασίας μας, στα πλαίσια της εκπόνησης της εν λόγω πτυχιακής εργασίας.

Επίσης, θέλω να ευχαριστήσω τους καθηγητές του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων, για την συμβολή τους στην ολοκλήρωση των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Δημιουργία online υπηρεσίας εύρεσης και σύστασης επαγγελματιών, με τη χρήση της Javascript και ειδικότερα του MEAN stack. Η εφαρμογή αναπτύχθηκε στο περιβάλλον ανάπτυξης (IDE) Webstorm.

Αρχικά, γίνεται μια παρουσίαση στις web τεχνολογίες του σήμερα καθώς και μια σύντομη ιστορική αναδρομή στην γλώσσα προγραμματισμού JavaScript. Στη συνέχεια, παρουσιάζονται εν συντομία οι τεχνολογίες που χρησιμοποιούνται στην εργασία. Τέλος, επεξηγείται ο κώδικας που γράφτηκε για να αναπτυχθεί η εφαρμογή.

ABSTRACT

Development of an online service for searching and reccomendation system, with Javascript and especially on MEAN stack approach.

At first, there is a presentation of today's web technologies and a brief historic reference to Javascript. In the second chapter there is a requirements analysis and in the next chapter a brief presentation of the project’s technologies. In the final chapter, the thw whole project’s code is getting explained.

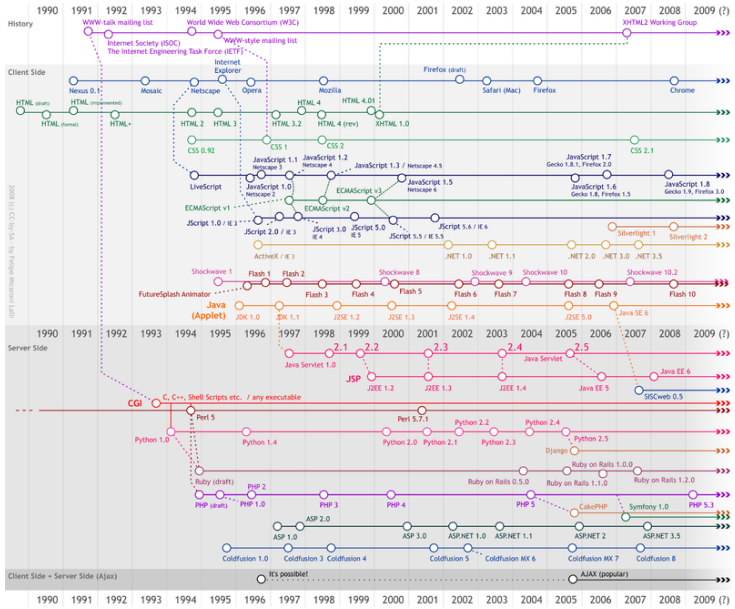
ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΙΣΑΓΩΓΗ

Το web development, ως κλάδος της Πληροφορικής, κάνει αλματώδη βήματα τα τελευταία χρόνια με το βλέμμα στραμμένο στο μέλλον. Πλέον, σχεδόν, όλες οι καθημερινές δραστηριότητες των ανθρώπων, σχετίζονται με τις web/mobile εφαρμογές. Ως web development, ορίζουμε την συγγραφή κώδικα εκ του οποίου δίνουμε λειτουργικότητα σε μια ιστοσελίδα/εφαρμογή, σύμφωνα με τις απαιτήσεις του ιδιοκτήτη. Κυρίως, αφορά τη μη-σχεδιαστική (non-design) πλευρά της ανάπτυξης ιστότοπων/εφαρμογών, το οποίο περιλαμβάνει προγραμματισμό και σύνταξη markup γλώσσας, όπως η HTML (HyperText Markup Language). Το web development, καλύπτει ένα ευρύ φάσμα, από τη δημιουργία ενός απλού αρχείου κειμένου, ως πολύπλοκες web εφαρμογές, εφαρμογές κοινωνικών δικτύων και εφαρμογές επιχειρήσεων.

Έχουν περάσει 30 χρόνια από τη μέρα που εφευρέθηκε το ίντερνετ από τον Tim-Berners Lee. Εκείνη την εποχή, εργαζόταν στο CERN ως μηχανικός λογισμικού, όπου αυτός και οι συνάδελφοί του έψαχναν ένα τρόπο να μοιραστούν τα αρχεία και δεδομένα των ερευνών τους σε διάφορους υπολογιστές. Τον Οκτώβριο του 1990 ο Lee, καθόρισε τις τρεις θεμελιώδης τεχνολογίες που αποτελούν, μέχρι και σήμερα, την βάση του διαδικτύου.

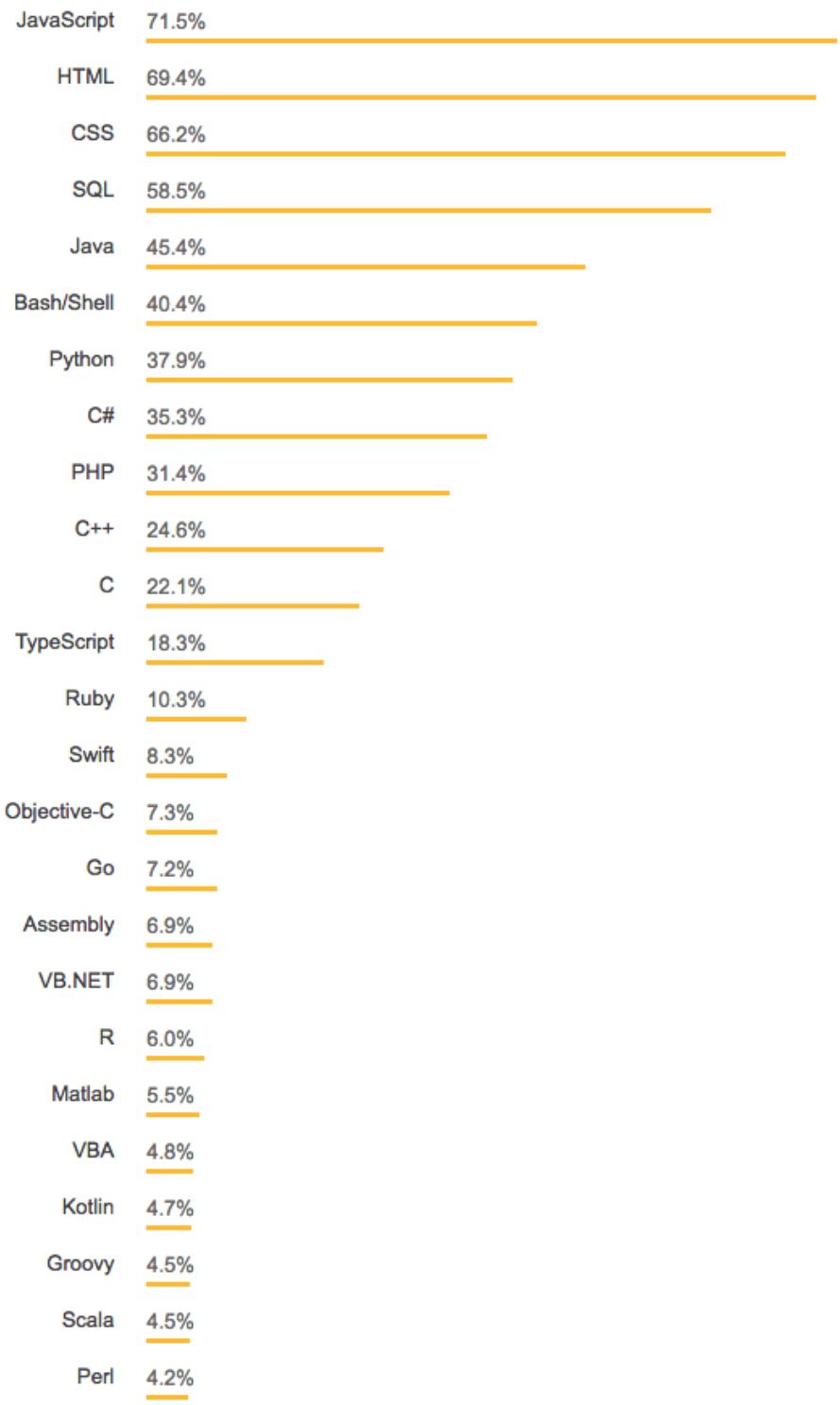
- **HTML: HyperText Markup Language.** Είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες και τα στοιχεία της είναι τα βασικά δομικά συστατικά των ιστοσελίδων.
- **URI: Uniform Resource Identifier.** Μια μορφή διεύθυνσης, η οποία είναι μοναδική για κάθε πόρο του διαδικτύου.
- **HTTP: HyperText Transfer Protocol.** Ένα πρωτόκολλο επικοινωνίας, το οποίο είναι το κύριο πρωτόκολλο που χρησιμοποιούν οι φυλλομετρητές του Παγκόσμιου Ιστού, ώστε να μεταφερθούν δεδομένα από ένα διακομιστή (server) σε έναν πελάτη (client).



Σχήμα 1.1 [1]

Programming, Scripting, and Markup Languages

All Respondents | Professional Developers



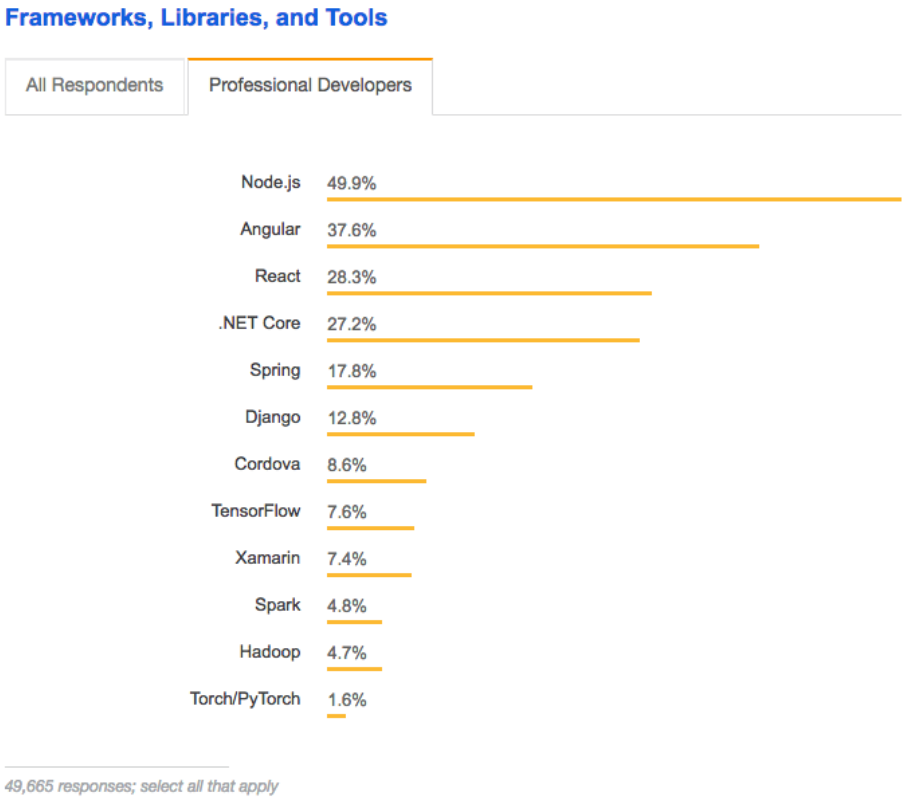
73,248 responses; select all that apply

Σχήμα 1.2 [2]

JavaScript

Όπως φαίνεται στο παραπάνω σχήμα (1.1), η Javascript έκανε την εμφάνισή της το 1995 και αναπτύχθηκε από Brendan Eich της Netscape Communication Corporation. Σκοπός της ήταν να υποστηρίξει τον Netscape Navigator, ο οποίος ήταν ένας web browser. Αρχικά ονομάσθηκε Mocha, αργότερα μετονομάσθηκε σε LiveScript και στο τέλος του 1995 πήρε το οριστικό της όνομα, JavaScript, με σκοπό να προβάλλει την υποστήριξη της JAVA από τον Netscape Navigator. Το 1996, εκτοξεύθηκε τόσο πολύ, ώστε συστάθηκε ένα σώμα με την ονομασία ECMA (European Computer Manufacturers Association) το οποίο είναι υπεύθυνο, μέχρι και σήμερα, για την συντήρηση και την εξέλιξη της JavaScript. Αυτή τη στιγμή κυκλοφορεί η EcmaScript 9, η οποία κυκλοφόρησε τον Ιούνιο του 2018.

Το 2005, ο Jesse James Garnett σύστησε στο κοινό τον όρο AJAX (Asynchronous Javascript and XML). Από εκείνη τη μέρα και έπειτα, η JavaScript έχει μια συνεχή ανοδική πορεία. Με το AJAX μπορούν να λειτουργήσουν ασύγχρονα οι εφαρμογές, οι οποίες μπορούν να εκτελέσουν πολλαπλές διεργασίες ταυτόχρονα, εν αντιθέσει με την σειριακή προσέγγιση, όπου κάθε διεργασία περιμένει να εκτελεστεί η αμέσως προηγούμενη για να εκτελεστεί. Δεκατέσσερα χρόνια μετά, η JavaScript είναι η πιο διαδεδομένη γλώσσα προγραμματισμού. Σύμφωνα με έρευνα του StackOverflow το 71,5% των επαγγελματιών developer, χρησιμοποιούν την JavaScript, ενώ στην κορυφή των frameworks/libraries, σύμφωνα με την ίδια έρευνα, είναι η Node.js και άλλα δύο frameworks που χρησιμοποιούν την TypeScript, οι Angular και η React. Η TypeScript είναι ένα superset της JavaScript, το οποίο αναφέρεται παρακάτω.



Σχήμα 1.3 [2]

1. Το Web development στην εποχή μας

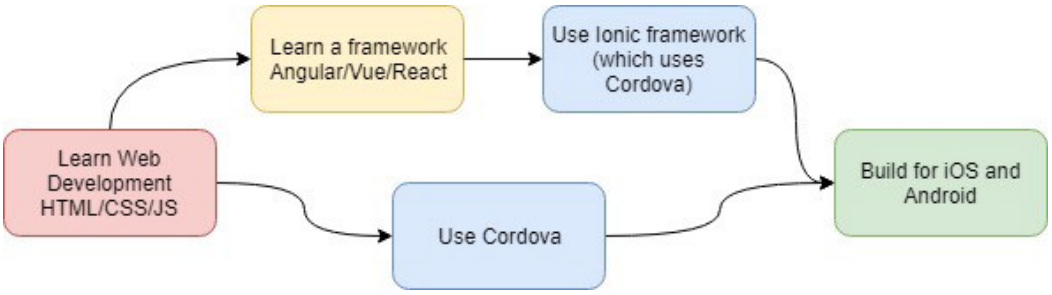
1.1 Οι τάσεις του web development

Ένα μεγάλο μερίδιο στις τεχνολογικές τάσεις του web development, όπως είδαμε παραπάνω, καλύπτει η JavaScript. Πριν μπούμε στο κυρίως μέρος της εργασίας θα αναφερθούμε σε μερικές από αυτές.

- **Node.js:** Είναι ένα runtime environment για την JavaScript, το οποίο τρέχει στον server. Η node.js έχει αναπτυχθεί πάνω στην V8 JavaScript engine του Google Chrome. Ήδη, από πολλούς θεωρείται ως βασική γνώση για έναν web developer και όπως φαίνεται και στο παραπάνω σχήμα 1.2, είναι η πρώτη σε στην κατηγορία Frameworks/Libraries.

- **Frontend Frameworks:** Διευκολύνουν την ανάπτυξη εφαρμογών στην πλευρά του client. Τα πιο δημοφιλή όπως φαίνεται στο σχήμα 1.2 είναι τα **Angular**, **React** και **Vue.** Μπορείς να φτιάξεις με ευκολία, αξιόπιστες και διαδραστικές εφαρμογές και να δομήσεις το UI σου ευκολότερα πέραν των υπόλοιπων χαρακτηριστικών που έχουν. Η **Angular** αναλύεται και παρακάτω, αφού είναι το framework που χρησιμοποιούμε για την υλοποίηση του project.

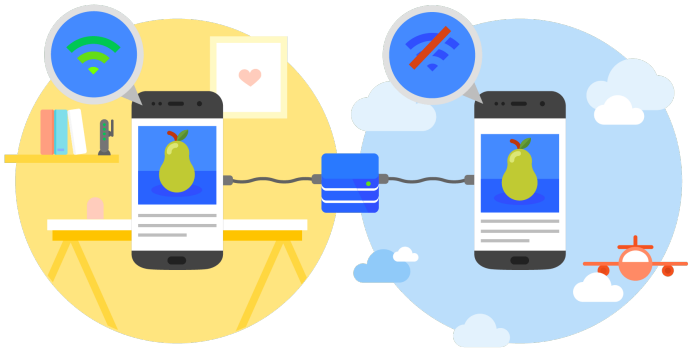
- **Hybrid Apps:** Frameworks όπως τα NativeScript, Ionic, Electron βοηθάνε στην ανάπτυξη mobile Apps χωρίς τη χρήση της native γλώσσας τους, όπως Android και Swift (iOS), αλλά γράφοντας σε JavaScript ή TypeScript, δημιουργώντας cross-platform εφαρμογές. Για παράδειγμα, η NativeScript υποστηρίζει μέσω plug-ins και τα Angular, Vue frameworks.



Ένα παράδειγμα για το οικοσύστημα της JS στο web development [3]

- **PWA (Progressive Web Apps):** Είναι εφαρμογές, όπως και οι hybrid, οι οποίες δεν αναπτύσσονται με τη native γλώσσα (Android, Swift), αλλά με JavaScript. Έχουν παρόμοια επίδοση με τις native εφαρμογές και τρέχουν στον browser. Κάποια προτερήματά τους είναι:
    - Υποστήριξη offline λειτουργίας
    - Push notifications

- Δεν χρειάζεται καταχώριση στο App/ Play Store



PWA (Progressive Web Apps) [4]

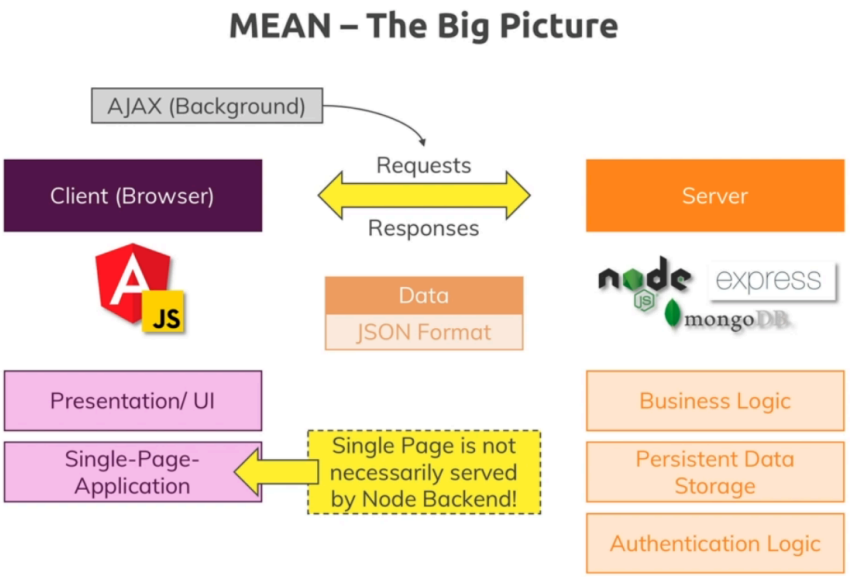
- **Serverless applications:** Είναι ένα cloud-based μοντέλο υπολογισμού, όπου ο πάροχος του cloud διαχειρίζεται δυναμικά την τροφοδοσία και την κατανομή των servers. Αυτές οι εφαρμογές είναι event-triggered και διαχειρίζονται, πλήρως, από τον πάροχο του cloud. Η τιμολόγηση βασίζεται στον αριθμό των εκτελέσεων και όχι βάσει της χωρητικότητας που αγοράζεις.

- **AI (Artificial Intelligence)/ ML (Machine Learning):** Χρόνο με το χρόνο, η τεχνητή νοημοσύνη και η μηχανική μάθηση κάνει πιο αισθητή την εμφάνιση της στο web development. Πλέον οι εφαρμογές τείνουν να γίνονται πιο «έξυπνες» και η χρήση αυτών των τεχνολογιών αρχίζουν να γίνονται απαραίτητες. Εφαρμογές αναγνώρισης εικόνας, συστήματα προτάσεων βάσει των συνηθειών του χρήστη, εφαρμογές που μαντεύουν τι θα πληκτρολογήσει ο χρήστης, βάσει αυτών που έχει πληκτρολογήσει.

1.2. Συνοπτική αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν στην πτυχιακή

Η εφαρμογή βασίζεται στην προσέγγιση του MEAN Stack. Είναι μια end-to-end προσέγγιση, για την δημιουργία δυναμικών ιστοσελίδων και εφαρμογών. Αποτελείται από:

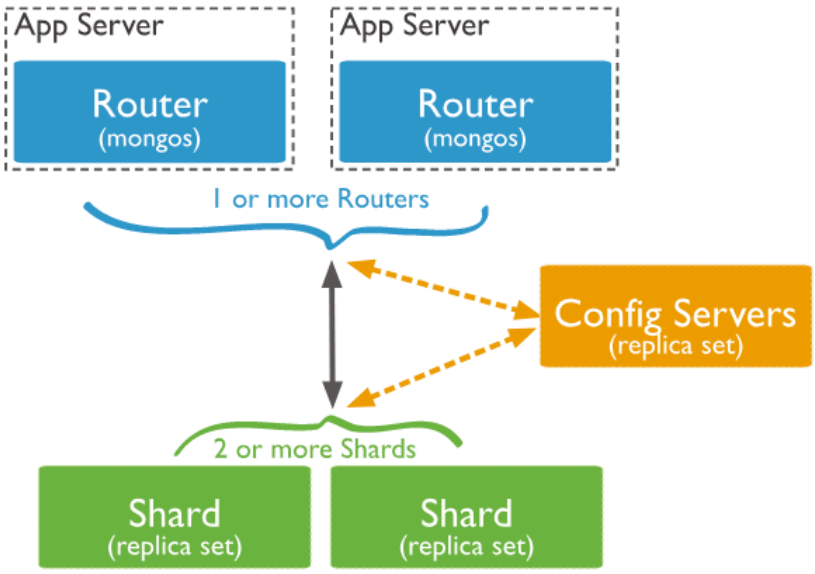
- **MongoDB**: Η βάση δεδομένων
- **Express.js**: Ένα framework της Node.js, το οποίο παρέχει ορισμένα σετ χαρακτηριστικών τα οποία διευκολύνουν την ανάπτυξη εφαρμογών και API's.
- **Angular.js:** Ένα JavaScript framework, το οποίο τρέχει στον browser
- **Node.js:** Ένα JavaScript runtime environment το οποίο τρέχει στον server.



The MEAN Stack [5]

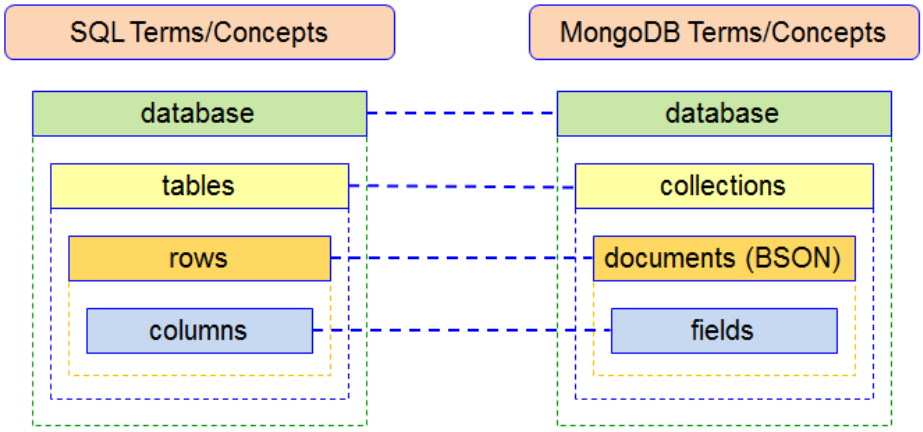
MongoDB

Η MongoDB είναι μια, NoSQL, document-oriented βάση δεδομένων η οποία χαρακτηρίζεται από την αποδοτικότητά της στο να χειρίζεται τεράστιες ποσότητες δεδομένων, εξού και το όνομά της προέρχεται από το Humongous (τεράστιος). Επίσης, είναι shema-less, που σημαίνει πως η μορφή ενός collection (table στην SQL), μπορεί να αλλάζει δυναμικά. Αυτά τα δύο χαρακτηριστικά, η ευκολία της να επεκτείνεται οριζόντια και όχι κάθετα όπως η SQL και η ευελιξία του να αλλάζεις το schema σου ανάλογα με τις ανάγκες της εφαρμογής σου την έχουν κάνει ιδιαίτερα δημοφιλή στις μέρες μας. Η οριζόντια επεκτασιμότητα (scalability) γίνεται μέσω της μεθόδου sharding, διανέμοντας τα δεδομένα σε πολλαπλά clusters.



MongoDB sharding [6]

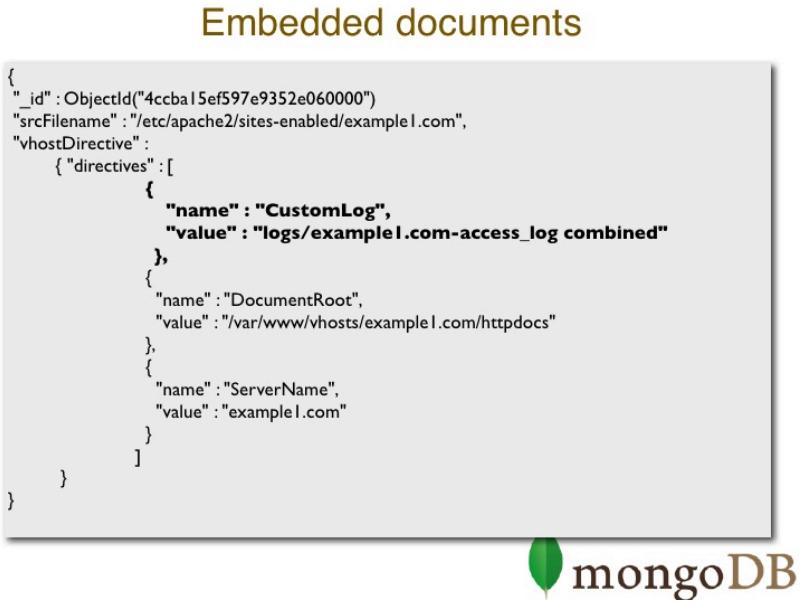
Η MongoDB (έτσι ονομάζεται και η εταιρεία) είναι, επίσης, και ένας database server που φιλοξενεί τη βάση.



Αντιστοίχιση όρων SQL με Mongo [7]

Παρόλο που είναι schema-less, δίνει τη δυνατότητα να δομήσεις το schema σου και να βάλεις περιορισμούς. Ο χρήστης χειρίζεται τα documents σε JSON (JavaScript Object Notation) μορφή, ενώ αποθηκεύονται σε BSON (Binary JSON), για λόγους απόδοσης. Ο χειρισμός των documents σε JSON διευκολύνει την αλληλεπίδραση με πολλές γλώσσες προγραμματισμού και πιο συγκεκριμένα, στην περίπτωσή μας, με την JavaScript.

Η Mongo δίνει τη δυνατότητα μεταξύ άλλων να αποθηκεύσεις εμφωλευμένα (embedded) documents τα οποία βοηθούν στην απόδοση, καθώς δεν χρειάζεται να προσκομίσεις δεδομένα από πολλούς διαφορετικούς πίνακες.



Ενσωματωμένο document [8]

Express.js

Το Express.js είναι ένα framework της Node.js το οποίο μας βοηθά, παρέχοντας μας ορισμένες μεθόδους, να επικεντρωθούμε στο business logic της εφαρμογής μας και να μην ασχολούμαστε με το server-side logic, το οποίο κάποιες φορές είναι πολύπλοκο και χρονοβόρο. Ένα παράδειγμα είναι εξαγωγή δεδομένων από τον browser. Το express μέσω της βιβλιοθήκης body-parser δίνει τη δυνατότητα με ελάχιστες γραμμές κώδικα να πάρουμε το body από τον browser. Θα το δούμε αναλυτικότερα στο τρίτο κεφάλαιο.

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('hello world')
})
```

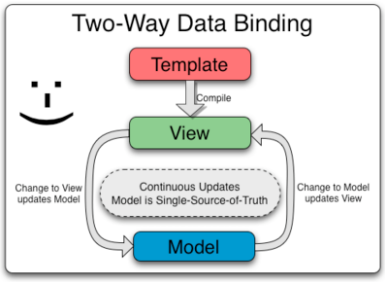
Με τις παραπάνω γραμμές κώδικα, ο server στέλνει στον home directory του browser το “hello world”. Στο τρίτο κεφάλαιο θα δούμε πιο αναλυτικά με ποιό τρόπο δουλεύει το express.

Angular

Το Angular είναι ένα JavaScript client-side framework το οποίο μας επιτρέπει να αναπτύσσουμε SPA (Single Page Applications), οι οποίες δίνουν την αίσθηση πως πρόκειται για mobile εφαρμογές. Αυτό οφείλεται στο ότι χειρίζεται τα πάντα στον browser και δεν χρειάζεται reload για να ανακτήσει δεδομένα.
Η Angular, μεταξύ άλλων, τροφοδοτεί το UI (User Interface) με δυναμικά δεδομένα, χειρίζεται δεδομένα που εισάγει ο χρήστης, και επικοινωνεί με το backend.

Κάποια χαρακτηριστικά που κάνουν αυτό το framework δημοφιλές είναι:

- **Two way data binding:** Με αυτή τη μέθοδο, κάθε αλλαγή σε ένα property του component, μεταδίδεται και ενημερώνεται και το UI. Αντιστρόφως, όταν το UI ενημερώνεται, οι τιμές διαδίδονται και στον component.

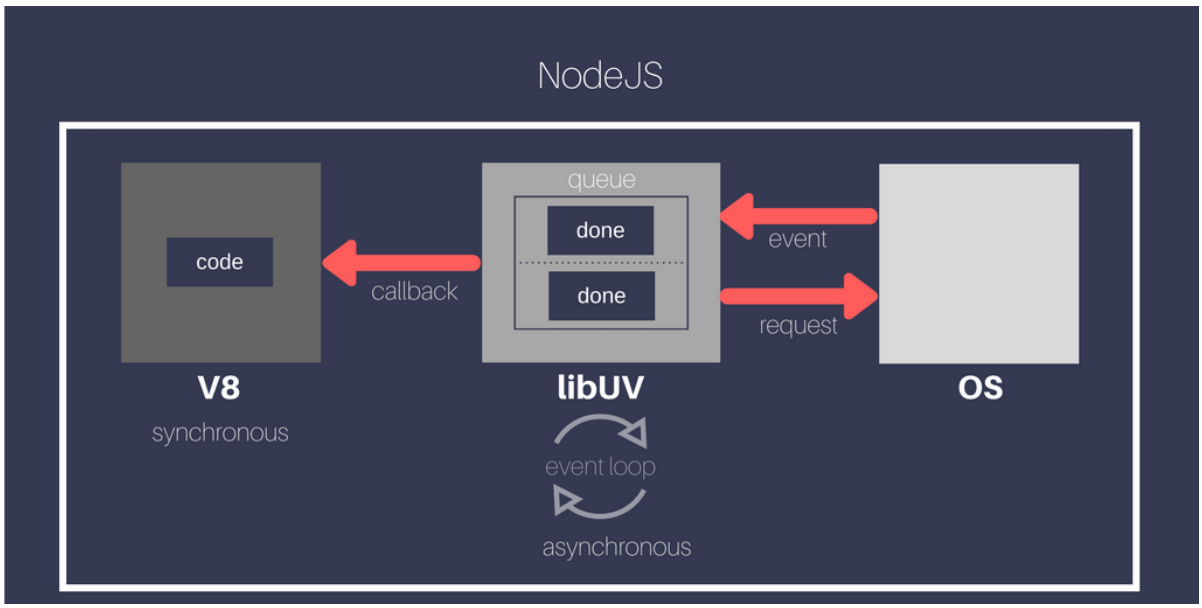


Angular two way data binding [9]

- **MVC (Model-View-Controller):** Η συγκεκριμένη μέθοδος ανάπτυξης λογισμικού, χωρίζει την εφαρμογή σε Model, που είναι οι Servers (Node), σε View που είναι η HTML/CSS, και στον Controller που είναι η Angular. Δεν χρειάζεται να γράψεις έξτρα κώδικα για να συνδέσεις τα επίπεδα του MVC, κάτι που έχει ως αποτέλεσμα λιγότερες γραμμές κώδικα.
- **Templates:** Μπορείς να φτιάξεις UI views, πολύ γρήγορα.
- **Angular CLI:** Μέσω του CLI μπορείς να δημιουργήσεις components, να εκκινήσεις τον server κ.α.
- **RxJS:** Με την χρήση αυτής της βιβλιοθήκης υπάρχουν αρκετές διευκολύνσεις στην επικοινωνία client-server, όπως για παράδειγμα μέσω του HttpClient, του οποίου οι μέθοδοι επιστρέφουν observables, τα οποία συμπεριφέρονται σαν stream από τιμές. Αναλύονται παρακάτω στο κεφάλαιο 2 και 3.

Node.js

Η Node είναι ένα asynchronous non-blocking runtime environment της JavaScript. Η single-threaded, event-driven αρχιτεκτονική, επιτρέπει στη Node να τρέχει ταυτόχρονα, πολλές συνδέσεις αποδοτικά. Οι πιο πολλές γνωστές πλατφόρμες δημιουργούν ένα επιπλέον thread για κάθε νέο request, χρησιμοποιώντας την RAM όσο χρειάζεται για να ολοκληρωθεί, ενώ η Node λειτουργεί σε ένα thread χρησιμοποιώντας το event loop και τα callbacks για I/O λειτουργίες, αναθέτοντας διεργασίες όπως λειτουργίες βάσης δεδομένων όσο το δυνατόν γρηγορότερα.



NodeJs asynchronous [10]

Η Node είναι ανεπτυγμένη πάνω στην V8 Chrome Engine (γραμμένη στην C++) και στην ουσία είναι επέκταση της JavaScript ώστε να τρέχει στον server. Πλέον μπορούμε να δημιουργήσουμε full stack εφαρμογές μόνο με JavaScript. Αυτό μας διευκολύνει πάρα πολύ και ενισχύει το οικοσύστημα της JavaScript. Με την Node έχουμε και το NPM (Node

Package Manager), ένα εργαλείο το οποίο περιέχει ολόκληρο το πακέτο των modules στο οποίο μπορούμε να έχουμε πρόσβαση αν πάσα στιγμή μέσω του CLI.

Node Modules που χρησιμοποιήσαμε στο project:

- **body-parser:** Είναι ένα middleware το οποίο μας βοηθά να κάνουμε ανάλυση τις τιμές από εισερχόμενα requests.
- **cors (Cross Origin Resource Sharing):** Αυτό το module επιτρέπει σε μια εφαρμογή που τρέχει σε ένα συγκεκριμένο domain να έχει πρόσβαση σε ένα άλλο server. Στην εφαρμογή μας, για παράδειγμα, ο server (Node) τρέχει στο **http://localhost:3000** ενώ ο client (Angular) στη διεύθυνση **http://localhost:4200.**
- **mongoose:** Ένα εργαλείο που μας επιτρέπει να μοντελοποιούμε και να τρέχουμε την Mongo στο ασύγχρονο περιβάλλον της Node.
- **bcrypt:** Είναι ένα module το οποίο μας επιτρέπει να κρυπτογραφήσουμε τους κωδικούς που εισάγει ο χρήστης στο σύστημα μας.
- **jsonwebtoken:** Είναι ένα module που χρησιμοποιεί το JWT standard, το οποίο απότελεί μια αυτούσια λύση για να μεταδίδεις πληροφορίες μεταξύ δυο μερών. Στην περίπτωση μας το χρησιμοποιούμε για την αυθεντικοποίηση του χρήστη κατά τη σύνδεση.

TypeScript

Η TypeScript είναι ένα superset της JavaScript, η οποία σου δίνει τη δυνατότητα να γράψεις JavaScript με τον τρόπο που θέλεις. Είναι μια αντικειμενοστραφής γλώσσα με κλάσεις, όπως η C# και η Java. Η Angular είναι γραμμένη σε TypeScript. Ο πιο στατικός τρόπος γραφής της TypeScript, μας βοηθά να αποφύγουμε τα runtime errors που προκύπτουν καθώς η εφαρμογή μεγαλώνει. Αυτό δεν μας εμποδίζει να γράψουμε την Angular εφαρμογή μας σε JavaScript, αν δεν το επιθυμούμε. Εν τέλει, η TypeScript μεταγλωττίζεται σε JavaScript.

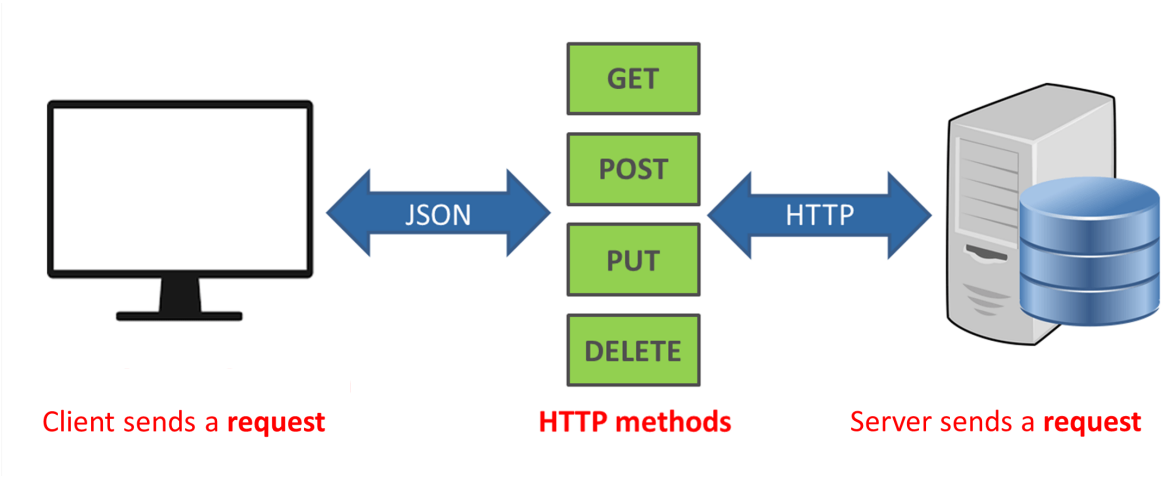
Reactive Programming-RxJS

Η RxJS είναι μια βιβλιοθήκη με την οποία αναπτύσσουμε ασύγχρονα και event-driven προγράμματα χρησιμοποιώντας τις ακολουθίες των observables. Παρέχει ένα κύριο τύπο, το Observable, κάποιους τύπους-δορυφόρους, όπως οι Observers, Schedulers, Subjects. Ένα observable είναι ένα stream από τιμές. Στο project μας, όταν κάνουμε Http request στον server, παίρνουμε ως response ένα observable, στο οποίο κάνοντας subscription λαμβάνουμε κάθε νέο response από τον server.

Angular Material

Είναι μια βιβλιοθήκη η οποία περιέχει ένα σετ από επαναχρησιμοποιήσιμα, δοκιμασμένα και προσβάσιμα UI components για την Angular, η οποία εφαρμόστηκε πάνω στο Material Design της Google. Με αυτή τη βιβλιοθήκη μπορούμε να δημιουργούμε και να χρησιμοποιούμε έτοιμα components στην εφαρμογή μας.

REST API



REST API [11]

Η REST (Representational State Transfer) είναι μια αρχιτεκτονική λογισμικού η οποία μας βοηθά να μεταφέρουμε δεδομένα από τον client στον server και αντίστροφα. Χρησιμοποιείται στις SPA εφαρμογές γιατί δεν λειτουργεί όπως μια παραδοσιακή web API, όπου σε κάθε request γίνεται render μια νέα HTML σελίδα, αλλά έχουμε ένα αρχικό request που κάνει render την HTML σελίδα και έπειτα, για κάθε request στέλνονται μέσω εντολών της JavaScript στην περίπτωση μας, οδηγίες για χειρισμό του DOM της σελίδας, το οποίο ενημερώνεται δυναμικά. Στην συγκεκριμένη αρχιτεκτονική πρέπει να είναι διαχωρισμός μεταξύ client-server (Separation of Concerns). Η REST API δεν πρέπει να «ενδιαφέρεται» για το UI. Επίσης, δεν αποθηκεύεται περιεχόμενο του client στον server καθώς είναι stateless. Όλος ο χειρισμός της κατάστασης (state) της εφαρμογής γίνεται στον client.

bcrypt

Το bcrypt είναι μια βιβλιοθήκη που μας παρέχει κάποιες μεθόδους ώστε να κρυπτογραφούμε τους κωδικούς του χρήστη που αποθηκεύονται στη βάση. Αυτό μας προστατεύει σε περίπτωση που κάποιο κακόβουλο λογισμικό αποκτήσει πρόσβαση στη βάση να μην έχει πρόσβαση στους κωδικούς των χρηστών. Κάθε φορά που ένας χρήστης εισάγει ένα κωδικό, ο κωδικός περνάει από τη μέθοδο hash του bcrypt και δημιουργείται μια τυχαία συμβολοσειρά η οποία, είναι όσο μεγάλη είναι τα hashing rounds που έχουμε θέσει στη μέθοδο genSalt. Καθώς κάθε round είναι και ένα επιπλέον επίπεδο κρυπτογράφησης, που σημαίνει και μεγαλύτερο χρονικό κόστος. Στην εφαρμογή χρησιμοποιείται κρυπτογράφηση με 10 rounds.

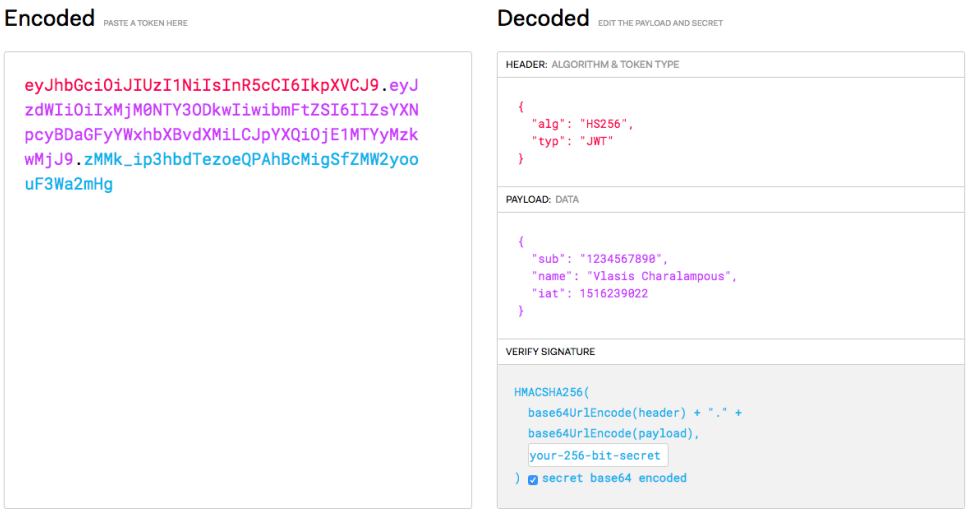
jsonwebtoken

Το jsonwebtoken είναι μια βιβλιοθήκη που χρησιμοποιεί το standard JWT (JwonWebToken), το οποίο μας βοήθα στο authentication system. Στις SPA (Single Page Application) εφαρμογές, ο server δεν αποθηκεύει πληροφορίες για τον client, αφού η

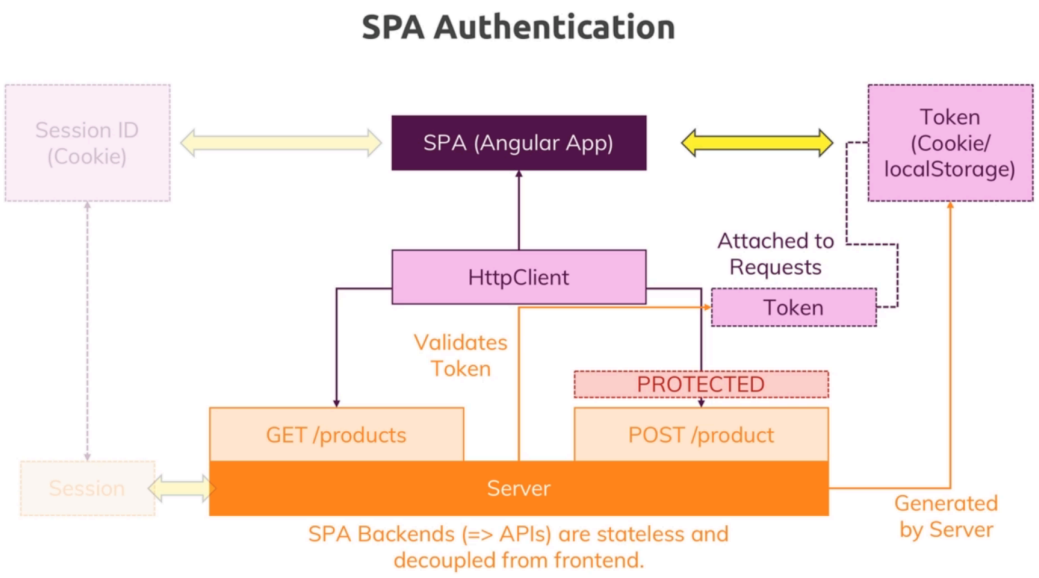
stateless αρχιτεκτονική τους διαχωρίζει. Με το jsonwebtoken, κάθε φορά που ο χρήστης κάνει σύνδεση, ο server στέλνει ένα token το οποίο ο client μπορεί να αποθηκεύσει στο localStorage ή στο cookieStorage. Το validation για το token γίνεται από τον server καθώς εκεί δημιουργήθηκε. Ένα JWT είναι μια συμβολοσειρά που αποτελείται από ένα header, ένα payload και το signature.

Header: Αποτελείται από δύο κομμάτια. Τον τύπο του token και τον signing αλγόριθμο.

Payload: Αποτελείται από τα claims τα οποία είναι δεδομένα του χρήστη, ή πρόσθετα μεταδεδομένα.
Signature: Το πιο σημαντικό κομμάτι του token καθώς περιέχει τα header, payload κωδικοποιημένα.



Ένα JsonWebToken κωδικοποιημένο και αποκωδικοποιημένο [12]



Ο κύκλος ζωής του SPA authentication, όπου στο αριστερό μέρος υπάρχουν οι παραδοσιακοί τρόποι αυθεντικοποίησης όπου ο server αποθηκεύει δεδομένα(session id, session) για την αυθεντικοποίηση. [13]

2. Ανάλυση απαιτήσεων

Εγγραφή χρήστη (Πελάτη/ Επαγγελματία)

Προαπαιτούμενη συνθήκη: Ο χρήστης δεν είναι εγγεγραμμένος

- Ο χρήστης επιλέγει να κάνει εγγραφή.
- Η εφαρμογή εμφανίζει την φόρμα εγγραφής.
- Ο χρήστης πρέπει να επιλέξει αν θα εγγραφεί ως πελάτης ή ως επαγγελματίας.
- Αν ο χρήστης επιλέξει να εγγραφεί ως επαγγελματίας πρέπει να καταχωρήσει επιπλέον την κατηγορία που ειδικεύεται, την πόλη που καλύπτει, όπως και το brand name.
- Ο χρήστης έχει τη δυνατότητα να εμφανίσει ή να κρύψει τον κωδικό που πληκτρολογεί.
- Η εφαρμογή ελέγχει δυναμικά την είσοδο των στοιχείων του χρήστη και εμφανίζει μηνύματα λάθους αλλά και μηνύματα για το τι πρέπει να πληκτρολογήσει ο χρήστης, αν οι είσοδοι δεν είναι στις προδιαγραφές της φόρμας
- Μέχρι να συμπληρωθούν όλα τα πεδία και να είναι έγκυρα, το κουμπί αποθήκευσης είναι απενεργοποιημένο.
- Αν τα στοιχεία είναι έγκυρα, στέλνονται και ο server κάνει την επαλήθευση. Αν το email υπάρχει ή αν ο κωδικός είναι λανθασμένος εμφανίζεται ένα αναδυόμενο παράθυρο που ενημερώνει τον χρήστη για το σφάλμα.
- Αν τα στοιχεία επαληθευτούν από τον server τότε ο χρήστης εγγράφεται επιτυχώς και ο χρήστης ανακατευθύνεται στη φόρμα σύνδεσης (login).
- Οι εγγραφές των κωδικών κρυπτογραφούνται πριν αποθηκευτούν στη βάση δεδομένων.

Σύνδεση χρήστη (Πελάτη/ Επαγγελματία)

Προαπαιτούμενη συνθήκη: Ο χρήστης δεν είναι συνδεδεμένος

- Ο χρήστης επιλέγει να κάνει σύνδεση.
- Η εφαρμογή εμφανίζει την φόρμα εγγραφής.
- Ο χρήστης πρέπει να επιλέξει αν θα συνδεθεί ως πελάτης ή ως επαγγελματίας.
- Αν τα στοιχεία είναι έγκυρα, στέλνονται και ο server κάνει την επαλήθευση. Αν το email υπάρχει ή αν ο κωδικός είναι λανθασμένος εμφανίζεται ένα αναδυόμενο παράθυρο που ενημερώνει τον χρήστη για το σφάλμα.
- Μέχρι να συμπληρωθούν όλα τα πεδία και να είναι έγκυρα, το κουμπί αποθήκευσης είναι απενεργοποιημένο.
- Αν τα στοιχεία επαληθευτούν από τον server τότε ο χρήστης συνδέεται επιτυχώς και ο χρήστης ανακατευθύνει τον χρήστη στο προφίλ του. Η μπάρα περιήγησης (navbar) ενημερώνεται δυναμικά ανάλογα με τον ρόλο του χρήστη. Αν ο χρήστης είναι πελάτης τότε ανακατευθύνει στο προφίλ, αλλιώς ανακατευθύνεται στο dahshboard του επαγγελματία.
- Κατά τη σύνδεση η γραμμή περιήγησης αλλάζει και προστίθενται το κουμπί logout. Επίσης κρύβονται τα κουμπιά σύνδεση, εγγραφή όσο ο χρήστης είναι συνδεδεμένος και εμφανίζεται αυτό της προσθήκης αίτησης, μόνο σε περίπτωση που ο χρήστης είναι πελάτης.

Καταχώρηση αίτησης

Προαπαιτούμενη συνθήκη: Ο χρήστης είναι συνδεδεμένος και είναι πελάτης

- Ο χρήστης επιλέγει να κάνει μια αίτηση.
- Εμφανίζεται η φόρμα συμπλήρωσης της αίτησης.
- Η εφαρμογή ελέγχει δυναμικά την είσοδο των στοιχείων του χρήστη και εμφανίζει μηνύματα λάθους αλλά και μηνύματα για το τι πρέπει να πληκτρολογήσει ο χρήστης, αν οι είσοδοι δεν είναι στις προδιαγραφές της φόρμας
- Αν οι είσοδοι της φόρμας είναι έγκυροι, τότε στέλνονται στον server αποθήκευση.
- Αφού αποθηκευτεί, η αίτηση «τοποθετείται» στην καρτέλα εκκρεμείς αιτήσεις.
- Ο χρήστης στην καρτέλα εκκρεμείς αιτήσεις έχει τη δυνατότητα να επεξεργαστεί ή να διαγράψει την αίτηση του. Επίσης έχει τη δυνατότητα να δει στον τίτλο των αιτήσεων τον αριθμό των προσφορών από τους επαγγελματίες.

Διάδραση πελάτη-επαγγελματία (Κύκλος ζωής της αίτησης)

Προαπαιτούμενη συνθήκη: Ο πελάτης έχει κάνει μι αίτηση. Ο επαγγελματίας που συνδέεται στην πλατφόρμα, δραστηριοποιείται στην ίδια πόλη και κατηγορία που είναι καταχωρημένη η αίτηση.

- Ο χρήστης επιλέγει να κάνει μια αίτηση.
- Ο επαγγελματίας βλέπει την αίτηση στις προτεινόμενες αιτήσεις και έχει τη δυνατότητα να κάνει μια προσφορά.
- Αφού κάνει την προσφορά εμφανίζεται το παράθυρο των καταχωρημένων προσφορών.
- Μόλις καταχωρείται η προσφορά έχει ένα προεπιλεγμένο στάτους (Σε αναμονή), μέχρι να δώσει μια απάντηση ο χρήστης.
- Αφού ο επαγγελματίας έχει κάνει την προσφορά του, το προφίλ του πελάτη ενημερώνεται και βλέπει τις προσφορές του
- Ο πελάτης έχει τη δυνατότητα να αποδεχθεί ή να απορρίψει την προσφορά του επαγγελματία.
- Αν ο χρήστης αποδεχθεί την προσφορά του το στάτους της αίτησης στο προφίλ του επαγγελματία ενημερώνεται σε αποδεκτή.
- Αφού έχει γίνει αποδεκτή η προσφορά, στον επαγγελματία εμφανίζεται το κουμπί επιβεβαίωση, που σημαίνει πως επιβεβαιώνει την αποδοχή και θα δρομολογήσει την διαδικασία επισκευής. Αφού πατήσει το κουμπί επιβεβαίωσης, εμφανίζεται άλλο ένα, το οποίο τερματίζει την αίτηση και σημαίνει πως η επισκευή ολοκληρώθηκε.
- Αφού ο επαγγελματίας επιλέξει να κλείσει την αίτηση, η αίτηση μεταφέρεται στην καρτέλα με τις ανενεργές αιτήσεις, κάτι το οποίο συμβαίνει και στο προφίλ του πελάτη. Ταυτόχρονα, ανοίγει και η φόρμα αξιολόγησης του επαγγελματία από τον πελάτη.
- Ο πελάτης έχει τη δυνατότητα να κάνει μια αξιολόγηση. Αφού ο πελάτης πραγματοποιήσει μια κριτική, οι καρτέλες κριτικές του πελάτη και του επαγγελματία ενημερώνονται.

Ανώνυμη περιήγηση

- Ένας μη εγγεγραμμένος/συνδεδεμένος χρήστης έχει τη δυνατότητα να περιηγηθεί στη σελίδα Περιήγηση.
- Στη σελίδα, αρχικά εμφανίζονται όλοι οι επαγγελματίες που είναι εγγεγραμμένοι στην εφαρμογή.
- Ο χρήστης έχει τη δυνατότητα να κάνει αναζήτηση το όνομα ενός επαγγελματία.
- Ο χρήστης έχει τη δυνατότητα να κάνει ταξινόμηση τους επαγγελματίες αλφαβητικά.
- Ο χρήστης έχει τη δυνατότητα να χρησιμοποιήσει φίλτρα ώστε να εμφανίσει τους επαγγελματίες ανά πόλη ή ανά κατηγορία.

3. Ανάλυση κώδικα του project (Server Side)

3.1. Σχεδιασμός βάσης δεδομένων και REST API

Για τη βάση δεδομένων (MongoDB) χρειαστήκαμε 6 collections, τα οποία μοντελοποιήθηκαν στον φάκελο model της REST API.

- **users:** εδώ αποθηκεύονται οι χρήστες της εφαρμογής
- **pros:** εδώ αποθηκεύονται οι επαγγελματίες της εφαρμογής
- **issues:** εδώ αποθηκεύονται οι αιτήσεις των χρηστών
- **reviews:** εδώ αποθηκεύονται οι κριτικές των χρηστών
- **cities:** εδώ έχουμε αποθηκεύσει τις πόλεις τις οποίες χρησιμοποιεί η εφαρμογή
- **categories:** εδώ έχουμε αποθηκεύσει τις κατηγορίες επαγγελματιών τις οποίες χρησιμοποιεί η εφαρμογή

3.1.1. User collection

Το user collection αποτελείται από τα εξής documents:

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: String,
    email: String,
    password: String,
    createdAt: {type: Date, default: Date.now},
    role: {type: String, default: 'user'}
});

module.exports = mongoose.model('user', userSchema, 'users');
```

Στην πρώτη γραμμή εισάγεται το module της mongoose. Έπειτα δημιουργείται ένα mongoose schema (userSchema) το οποίο μας επιτρέπει να έχουμε ένα πρότυπο για τη δημιουργία του χρήστη, από τη φόρμα εγγραφής.

- **name:** Όνομα του χρήστη
- **email:** Email χρήστη
- **password:** Ο κωδικός του χρήστη
- **createdAt:** ένα document το οποίο καταχωρεί την ημερομηνία που εγγράφηκε ο χρήστης. Το πεδίο default σημαίνει πως η βάση δίνει μια προεπιλεγμένη τιμή, η οποία στη συγκεκριμένη περίπτωση είναι η ημερομηνία που έγινε η εγγραφή
- **role:** αυτό το document μας δίνει την προεπιλεγμένη τιμή του χρήστης, σε κάθε εγγραφή που γίνεται και μας βοηθά στο να ενημερώνεται δυναμικά η εφαρμογή ανάλογα με τον τύπο χρήστη που συνδέεται (χρήστης ή επαγγελματίας).

- **_id:** Η mongo δημιουργεί αυτόματα αυτό το πεδίο και είναι μια μοναδική συμβολοσειρά για κάθε document και έχει τη μορφή: "_id" : ObjectId("5c50dd8870066314a63ab8e7")

Στην γραμμή 11 εξάγουμε το schema μας, μέσω της μεθόδου mongoose.model. Το πρώτο argument της μεθόδου είναι το όνομα που εξάγουμε το schema μας, το δεύτερο είναι το schema που δημιουργήσαμε παραπάνω και το τρίτο είναι το collection στο οποίο αναφέρεται το συγκεκριμένο schema.

3.1.2. Pro collection

Το pro collection αποτελείται από τα εξής documents:

```
const mongoose = require('mongoose');

const proSchema = new mongoose.Schema({
    p_name: String,
    p_surname: String,
    p_email: String,
    p_password: String,
    p_category: String,
    p_locationCoverage: String,
    p_brandName: String,
    createdAt: {type: Date, default: Date.now},
    role: {type: String, default: 'pro'}
});


module.exports = mongoose.model('pro', proSchema, 'pros');
```

- **p_name:** Όνομα του επαγγελματία
- **p_surname:** Επώνυμο επαγγελματία
- **p_email:** Email επαγγελματία
- **p_password:** Ο κωδικός του επαγγελματία
- **p_category:** Κατηγορία στην οποία δραστηριοποιείται ο επαγγελματίας
- **p_locationCoverage:** Περιοχή στην οποία δραστηριοποιείται ο επαγγελματίας
- **p_brandName:** Επωνυμία εταιρείας του επαγγελματία
- **createdAt:** ένα document το οποίο καταχωρεί την ημερομηνία που εγγράφηκε ο επαγγελματίας. Το πεδίο default σημαίνει πως η βάση δίνει μια προεπιλεγμένη τιμή, η οποία στη συγκεκριμένη περίπτωση είναι η ημερομηνία που έγινε η εγγραφή
- **role:** αυτό το document μας δίνει την προεπιλεγμένη τιμή χρήστης, σε κάθε εγγραφή που γίνεται και μας βοηθά στο να ενημερώνεται δυναμικά η εφαρμογή ανάλογα με τον τύπο χρήστη που συνδέεται (χρήστης ή επαγγελματίας).
- **_id:** Η mongo δημιουργεί αυτόματα αυτό το πεδίο και είναι μια μοναδική συμβολοσειρά για κάθε document και έχει τη μορφή: "_id" : ObjectId(“5c50dd8870066314a63ab8e7”)

Στην τελευταία γραμμή εξάγεται το schema μας, μέσω της μεθόδου mongoose.model. Το πρώτο argument της μεθόδου είναι το όνομα που εξάγεται το schema μας, το δεύτερο είναι το schema που δημιουργήσαμε παραπάνω και το τρίτο είναι το collection στο οποίο αναφέρεται το συγκεκριμένο schema.

3.1.3. Issue collection

Το issue collection αποτελείται από τα εξής documents:

```
const mongoose = require('mongoose');

const issueSchema = new mongoose.Schema({
    id: String,
    title: String,
    description: String,
    creator: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User"
    },
    createdAt: {type: Date, default: Date.now},
    price:String,
    closed: {type: Boolean, default: false},
    category: String,
    bids:[{
        amount: Number,
        creator: String,
        createdAt: {type: Date, default: Date.now},
        status: String,
        confirmed: Boolean,
        closed: {type: Boolean, default: false}
    }],
    address: {
        city: String,
        homeAddress: String
    },
    phone: Number,
    availability: Date,
    image: String

});

module.exports = mongoose.model('issue', issueSchema, 'issues');
```

- **title:** Ο τίτλος της αίτησης
- **description:** Περιγραφή του ζητήματος για το οποίο κάνουμε την αίτηση

- **creator:** το _id του χρήστη που κάνει την αίτηση
- **creatorEmail:** το email του χρήστη που κάνει την αίτηση
- **createdAt:** η ημερομηνία καταχώρησης της αίτησης
- **price:** το ποσό που είναι διατεθειμένος να δώσει ο χρήστης
- **category:** κατηγορία στην οποία έγκειται η αίτηση
- **bids :** Δημιουργούμε ένα embedded document που θα αποθηκεύει τις πληροφορίες των προσφορών που στέλνουν οι επαγγελματίες
  - **amount:** το ποσό της προσφοράς
  - **creator:** το _id του επαγγελματία που κατέθεσε την προσφορά
  - **createdAt:** ημερομηνία καταχώρησης της προσφοράς
  - **status:** εδώ ελέγχουμε αν η πρόταση έχει γίνει αποδεκτή ή όχι (accepted/rejected)
  - **confirmed:** αυτό το πεδίο ενεργοποιείται όταν ο επαγγελματίας επιβεβαιώσει την αποδοχή της αίτησης από το χρήστη
  - **closed:** αυτό το πεδίο ενεργοποιείται όταν ο επαγγελματίας κλείσει την αίτηση, αφού ολοκληρωθεί
- **address:** Διεύθυνση που καταχωρεί ο χρήστης για το συγκεκριμένο ζήτημα
- **phone:** Τηλέφωνο επικοινωνίας
- **availability:** Ημερομηνία διαθεσιμότητας του χρήστη

3.1.4. Review collection

```
const mongoose = require('mongoose');
let Schema = mongoose.Schema;
let ObjectId = Schema.Types.ObjectId;

let ReviewsSchema = new Schema({
    title: {
        type: String,
    },
    description: {
        type: String,
    },
    stars: {
        type: Number,
    },
    user: {
        type: String, ref: 'User',
    },
    pro: {
        type: ObjectId, ref: 'Pro',
    },
    created_at: {
        type: Date, default: Date.now    }
});
```

```
module.exports = mongoose.model('Reviews', ReviewsSchema);
```

- **title:** Ο τίτλος της κριτικής
- **description:** Περιγραφή στην κριτική που κάνουμε
- **stars:** Η βαθμολογία του χρήστη
- **user:** το _id του χρήστη που κάνει την κριτική
- **pro:**το _id του επαγγελματία τον οποίο αφορά η κριτική
- **created_at:** ημερομηνία καταχώρησης της κριτικής

3.1.5. Cities/Categories collections

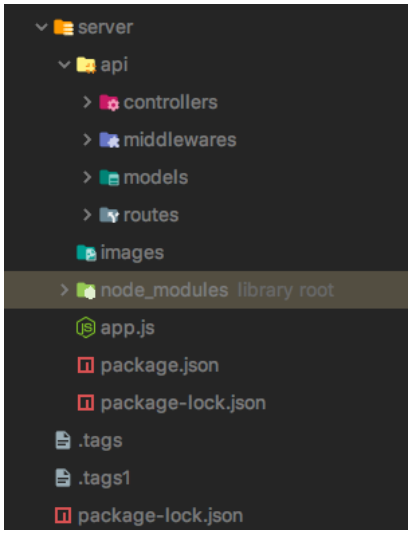
Οι δύο αυτές collections είναι στατικές, και δημιουργήθηκαν με σκοπό να τροφοδοτούν την εφαρμογή με τις κατηγορίες και τις πόλεις των επαγγελματιών.

Αποτελούνται από τα:
_id: το που δημιουργείται αυτόματα από τη mongo
value: Η τιμή που έχει το document
viewValue: Η τιμή που εμφανίζεται στον browser

* Τα collections αυτά, παρόλο και τα δύο documents έχουν ίδιες τιμές, σχεδιάστηκε αρχικά με την σκέψη πως ίσως χρειαζόταν να διαφοροποιήσουμε στον browser αυτά που αποθηκεύουμε με αυτά που βλέπει ο χρήστης.

City Collection

Category Collection



3.2.1. Δομή του server

- **controllers:** σε αυτό το φάκελο αποθηκεύουμε τα controllers, τα οποία περιέχουν τον κώδικα για να χειριζόμαστε τον server
- **middlewares:** Τα middlewares είναι μέθοδοι οι οποίες έχουν πρόσβαση στο request και response object όπως και στη μέθοδο next() του request/response κύκλου ζωής. Μπορούμε να τα χρησιμοποιήσουμε ως μια ενδιάμεση κατάσταση μεταξύ του request και response.
- **models:** Σε αυτό το φάκελο αποθηκεύονται τα μοντέλα που θα χειρίζεται η βάση δεδομένων μέσω της mongoose.
- **routes:** Σε αυτό το φάκελο αποθηκεύονται τα routes που θα επικοινωνεί ο client.
- **app.js:** Σε αυτό το αρχείο εκτελούνται κάποιες εντολές που είναι απαραίτητες για την λειτουργία της Node και της Express.

3.2.2. Αρχείο app.js

*** Το κείμενο που ξεκινά με διπλή διαγώνιο (//) είναι τα σχόλια επεξήγησης του κώδικα**

```
//εισαγωγή του express και των υπόλοιπων modules που είναι απαραίτητα
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const port = 3000;
const mongoose = require('mongoose');

//εισαγωγή των routes από τον φακελο routes
const mongoose = require('mongoose');
```

```
const userRoutes = require('./api/routes/user.routes');
const proRoutes = require('./api/routes/pro.routes');
const issueRoutes = require('./api/routes/issue.routes');
const browseRoutes = require('./api/routes/browse.routes');

//Ορίζουμε το url της βάσης δεδομένων
const db = "mongodb://127.0.0.1:27017/fixair3_0";

//Με την εντολή .use θέτουμε ένα middleware ως προεπιλεγμένο σε κάθε request/response
app.use(cors());
//Με την εντολή .json ορίζουμε τον body-parser να αναλύει και να εξάγει δεδομένα μόνο requests, όπου το content/type είναι application/json
app.use(bodyParser.json());


//θέτουμε νέα προεπιλεγμένο μονοπάτι για κάθε route. Για παράδειγμα, κάθε userRoute θα ξεκινά το μονοπάτι του από το http://localhost/api/user. Ομοίως και τα υπόλοιπα.
app.use('/api/user', userRoutes);
app.use('/api/pro', proRoutes);
app.use('/api/issue', issueRoutes);
app.use('/api/browse', browseRoutes);


// Ρυθμίζουμε ένα deprecation warning της mongoose το οποίο έχει να κάνει με τη findOneAndUpdate μέθοδο
mongoose.set('useFindAndModify', false);

//Εκκινούμε τη βάση δεδομένων. Argument είναι το url που έχουμε ορίσει παραπάνω, και εκτελούμε και έλεγχο λάθους μέσω ενός callback.
*με το {useNewUrlParser: true} χειριζόμαστε ένα warning της mongoose που έχει να κάνει με το νέο τρόπο που έχει προσθέσει στο να κάνει parsing συμβολοσειρές.
mongoose.connect(db, {useNewUrlParser: true}, err => {
  if (err) {
    console.log(err);
  } else {
    console.log('Mongo Connected');
  }
});

// Με τη μέθοδο listen εκκινούμε τον server μας, στο port που έχουμε θέσει παραπάνω
app.listen(port, ()=>{
  console.log('App started on port ' + port);
});
```

3.3. Φάκελος controllers

Σε αυτό το φάκελο υλοποιείται το κύριο μέρος του application logic του server. Εδώ δημιουργούμε τις μεθόδους τις οποίες εξάγουμε, ώστε να είναι επαναχρησιμοποιήσιμες, και τις «καταναλώνουν» τα αρχεία του φακέλου routes που θα δούμε παρακάτω

3.3.1 Αρχείο user.controller.js

Σε αυτό το αρχείο εκτελείται η, βασική για την εφαρμογή, λειτουργία του authentication system. Εδώ αναπτύσσονται οι μέθοδοι εγγραφής και σύνδεσης του χρήστη αλλά και μια μέθοδος ανάκτησης των δεδομένων του χρήστη. Δύο βιβλιοθήκες που μας επιτρέπουν να το πετύχουμε αυτό είναι η jswonwebtoken και η bcrypt.

```
//Εισάγουμε τα δύο μοντέλα (User, Pro) από τον φάκελο models, καθώς και τις απαραίτητες βιβλιοθήκες (jwt, bcrypt) που θα μας βοηθήσουν για να πετύχουμε το σκοπό μας

const User = require('../models/user.model');
const Pro = require('../models/pro.model');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');


//Μέθοδος για την εγγραφή χρήστη
let u_register = (req, res) => {

//Στέλνεται request στη βάση ώστε να ελεγχθεί, με τη μέθοδο findOne της mongoose, αν υπάρχει ήδη το email. Το email εξάγεται από τον client μέσω του bodyParser. Αν το email υπάρχει, στέλνεται μήνυμα στον client πως το email υπάρχει ήδη αλλιώς προχωράμε στο κομμάτι εγγραφής. Με την εντολη req.body.email το body-parser εξάγεται από τη φόρμα userRegistrationForm η είσοδος που καταχωρείται από τον χρήστη στο πεδίο email
  User.findOne({email: req.body.email})
   .then(user => {
      if (user) {
        res.status(401).send({message: 'Το email δεν είναι διαθέσιμο./nΔιαλέξτε κάποιο άλλο'});
      } else {
            // Αποθηκεύουμε στη μεταβλητή userData το body της φόρμας εγγραφής.
let userData = req.body;
            // Δημιουργούμε ένα νέο instance του Use model, στο οποίο εισάγουμε τις τιμές της userData
let user = new User(userData);
            // ξεκινάμε τη διαδικασία κρυπτογράφησης, θέτοντας το επίπεδο κρυπτογράφησης στα 10 rounds
```

```
bcrypt.genSalt(10, (err, salt) => {
```

// Δίνουμε ως όρισμα στην hash μέθοδο του bcrypt, τον κωδικό από το user object, καθώς και hash που μας έχει επιστρέψει η genSalt. Το callback επιστρέφει error αν υπάρχει, αλλιώς αποθηκεύουμε το hash στη μεταβλητη user.password

```
            bcrypt.hash(user.password, salt, (err, hash) => {
   if (err) throw err;
   user.password = hash;
```

//Η μέθοδος save είναι η μέθοδος της mongoose για να αποθηκεύει δεδομένα στη βάση. Στο callback επιστρέφουμε στον client είτε πιθανό λάθος, είτε τον εγγεγραμμένο χρήστη καθώς και το μήνυμα 'User registered succesfully'

```
   user.save((err, registeredUser) => {
                        if (err)
                            console.log(err);
                        else {
                            // let token = jwt.sign({_id: this._id}, 'secret');
                            // let payload = {subject : registeredUser._id };
                            // let token = jwt.sign(payload, 'secret');
                            res.status(200).send({
                                // token: token,
                                // userId: registeredUser._id,
                                registeredUser: registeredUser,
                                message: 'User registered succesfully'
                            });
                        }
                    });
                })
            });
        }
    })
};
```

//Η μεθοδολογία για την εγγραφή του επαγγελματία είναι πανομοιότυπη.

```
let p_register = (req, res) => {

   Pro.findOne({p_email: req.body.p_email})
      .then(pro => {
         if (pro) {
            console.log(pro);
            res.status(401).send({message: 'To email δεν είναι διαθέσιμο. Διαλέξτε κάποιο
άλλο'});
         } else {
            let proData = req.body;
            let pro = new Pro(proData);
            bcrypt.genSalt(10, (err, salt) => {
```

```
                bcrypt.hash(pro.p_password, salt, (err, hash) => {
                    pro.p_password = hash;
                    pro.save((err, registeredPro) => {
                        if (err)
                            console.log(err);
                        else {
                            res.status(200).send({
                                // token: token,
                                registeredPro: registeredPro,
                                message: 'Pro registered succesfully'
                            });
                        }
                    });
                })
            });
        }
    })
};

//Μέθοδος για την σύνδεση του χρήστη
let u_login = (req, res) => {

//Δημιουργία μεταβλητής στην οποία θα ανατεθεί η τιμή του user instance
let fetchedUser;

//Ελεγχος για την ύπαρξη του email στη βάση
  User.findOne({email: req.body.email})
    .then(user => {
      if (!user) {
        res.status(401).send({message: 'Το email δεν υπάρχει.'});
      }
      fetchedUser = user;
        //μέσω της μεθόδου compare του bcrypt, συγκρίνεται ο κωδικός που εξάγουμε από τον client με το password του user (req.body.password) που προσπάθησε να κάνει εγγραφή.
        return bcrypt.compare(req.body.password, fetchedUser.password);
})
      //αν η μέθοδος compare δεν επιστρέφει αποτέλεσμα, τότε στέλνεται στον client μήνυμα ότι ο κωδικός είναι λάθος.
    .then(result => {
  if (!result) {
    res.status(401).send({message: 'Λάθος κωδικός. Προσπαθήστε ξανά!'});
  }
        // Αν επιστρέφει αποτέλεσμα, τότε προχωράει στην δημιουργία token. Στο token αποθηκεύεται το id και το email του χρήση για μελλοντική χρήση.
        let token = jwt.sign(
  {email: fetchedUser.email, userId: fetchedUser._id},
```

```
    'secret',
    {expiresIn: '1h'}
);
```
//Στέλνεται το token και τα στοιχεία του χρήστη στον client, καθώς και ένα μήνυμα ότι ο χρήστης συνδέθηκε επιτυχώς.
```
          res.status(200).send({
        token: token,
        fetchedUser: fetchedUser,
        message: "Log in successful"
    })
})
```
// Εφαρμόζεται catch σε τυχόν λάθος.
```
         .catch(err => {
          res.status(401).send(err);
        })
};
```

//Η μεθοδολογία για την σύνδεση του επαγγελματία είναι πανομοιότυπη.

```
let p_login = (req, res) => {
    let fetchedPro;
    Pro.findOne({p_email: req.body.p_email})
        .then(pro => {
            if (!pro) {
                res.status(401).send({message: 'Το email δεν υπάρχει.'});
            }
            fetchedPro = pro;
            return bcrypt.compare(req.body.p_password, fetchedPro.p_password);
        })
        .then(result => {
            if (!result) {
                res.status(401).send({message: 'Λάθος κωδικός. Προσπαθήστε ξανά!'});
            }
            let token = jwt.sign(
                {email: fetchedPro.p_email, userId: fetchedPro._id},
                'secret',
                {expiresIn: '1h'}
            );
            res.status(200).send({
                token: token,
                fetchedPro: fetchedPro,
                message: "Log in successful"
            })
        })
        .catch(err => {
            res.status(401).send(err);
        })
```

```
};
```

// Μέθοδος που προσπελάζει τη βάση, συγκρίνει το email του χρήστη και το συγκρίνει με το email που είναι αποθηκευμένο στο token. Τα δεδομένα του token τα περνάμε στη μέθοδο, μέσω του middleware verifyToken, το οποίο χειρίζεται αυτά τα δεδομένα.

```
let getProfile = (req,res) => {
    console.log(req.userData);
    User.findOne({email: req.userData.email})
        .then(user=> {
            res.status(200).send({
                message: "User fetched",
                fetchedUser: user
            });
        })
};


module.exports = {
    u_register: u_register,
    p_register: p_register,
    u_login: u_login,
    p_login: p_login,
    getProfile: getProfile
};
```

3.3.2 Αρχείο issue.controller.js

Σε αυτό το αρχείο χειριζόμαστε, κυρίως, θέματα που αφορούν τις αιτήσεις (issue) του χρήστη.

```
//Εισαγωγή μοντέλου Issue
const Issue = require('../models/issue.model');

// Μέθοδος αποθήκευσης ενός issue
let addIssue = ((req, res) => {
```

// Δημιουργία ενός instance του Issue και σε αυτό αποθηκεύουμε τις τιμές που μας εξάγει το body-parser αλλά και αυτές (creator, creatorEmail) που έχει αποθηκευμένες middleware verifyToken, το οποίο χρησιμοποιούμε στο route μας και έτσι έχουμε πρόσβαση στις μεταβλητές αυτές.

```
  const issues = new Issue({
    title: req.body.title,
    description: req.body.description,
    creator: req.userData.userId,
```

```
    price: req.body.price,
    address: {
        city: req.body.city,
        homeAddress: req.body.homeAddress
    },
    category: req.body.category,
    phone: req.body.phone,
    availability: req.body.availability,
});
// Αποθηκεύουμε το issue και στέλνουμε στον client το issue, το id του issue καθώς και ένα μήνυμα
   issues.save().then(result => {
        res.status(201).send({
            message: "Post added successfully",
            issues: issues,
            issueId: result._id
        })
    });

});

// Ανακτώνται όλα τα αποθηκευμένα issues από τη βάση και επιστρέφονται στον client
let fetchIssues = (req, res) => {
    Issue.find()
        .then(documents => {
            res.status(200).send({
                issues: documents,
                message: "Issue added successfully"
            })
        })
};

// Διαγράφεται ένα συγκεκριμένο issue του οποίου το μοναδικό id εξάγεται από το middleware verifyToken()
let deleteIssue = (req, res) => {
    Issue.deleteOne({_id: req.params.id})
        .then(result => {
            console.log(result);
            res.status(200).send({
                message: 'Issue deleted'
            })
        });
};
```

// Ανακτώνται τα issues ενός συγκεκριμένου χρήστη τα οποία δεν έχουν ακόμη κλείσει και επιστρέφονται στον client. Το μοναδικό id εξάγεται από το middleware verifyToken()

```
let getUserIssues = (req, res) => {
    Issue.find({
        creator: req.userData.userId,
        closed: false
    }).exec((err, issue) => {
        if (err) console.log(err);
        else {
            res.send({
                message: "Posted fetched succesfully",
                issue: issue,
            });
        }
    })
};
```

// Ανακτώνται τα issues ενός συγκεκριμένου χρήστη τα οποία έχουν κλείσει επιστρέφονται στον client. Το μοναδικό id εξάγεται από το middleware verifyToken()

```
let getClosedUserIssues = (req, res) => {
    Issue.find({
        creator: req.userData.userId,
        closed: true
    }).exec((err, issue) => {
        if (err) console.log(err);
        else {
            res.send({
                message: "Posted fetched succesfully",
                issue: issue,
            });
        }
    })
};
```

// Ανακτώνται τα issues ενός συγκεκριμένου επαγγελματία τα οποία έχουν κλείσει και επιστρέφονται στον client. Το μοναδικό id εξάγεται από το middleware verifyToken()

```
let getClosedProIssues = (req, res) => {
    Issue.find({
        closed: true,
        "bids.creator": req.userData.userId}).exec((err, issue) => {
        if (err) console.log(err);
        else {
            res.send({
                message: "Closed issues fetched succesfully",
```

```
            issue: issue,
        });
    }
  })
};

// Ανακτώνται τα issues ενός συγκεκριμένου χρήστη τα οποία δεν έχουν κλείσει και επιστρέφονται στον client. Το μοναδικό id εξάγεται από το middleware verifyToken()
let getOpenProIssues = (req, res) => {
  Issue.find({
    closed: false,
    "bids.creator": req.userData.userId
    }).exec((err, issue) => {
    if (err) console.log(err);
    else {
      res.send({
        message: "Open issues fetched succesfully",
        issue: issue,
      });
    }
  })
};

// Ανακτώνται τα issues τα οποία θα εμφανίζονται στο προφίλ του επαγγελματία και είναι αυτά που θα αφορούν την περιοχή και την κατηγορία που δραστηριοποιείται.
let getIssuesForPro = (req, res) => {
  Issue.find({
    "address.city": req.query.city,
    category: req.query.category,
    closed: false
  }).exec((err, issue) => {
    if (err) console.log(err);
    else {
      res.send({
        message: "Issues for pro fetched succesfully",
        issue: issue,
      });
    }
  })
};
```

```
// Ανακτώνται τα issues ενός συγκεκριμένου χρήστη και επιστρέφονται στον client. Το μοναδικό id εξάγεται από το middleware verifyToken()
let getIssuesById = (req, res) => {
    Issue.find({_id: req.query.id}).then(issue => {
        console.log(issue);
        res.status(200).send({
            issue: issue,
            message: "Issue fetched successfully"
        });
    })
};


//Με την μέθοδο αυτή πραγματοποιείται ενημέρωση στα στοιχεία του issue όταν ο χρήστης επιλέξει edit. Το id εξάγεται από το req.query του url που στέλνει ο client
let updateIssue = (req, res) => {
    const issue = new Issue(req.body);
    Issue.findOneAndUpdate({_id: req.query.id}, {
        $set: {
            title: issue.title,
            description: issue.description,
            price: issue.price,
            address: {
                city: req.body.city,
                homeAddress: req.body.homeAddress
            },
            phone: issue.phone,
            availability: issue.availability,
        }
    }, issue).then(result => {
        res.status(201).send({
            result: result,
            message: 'Bid posted'
        })
    })
};


// Με τη μέθοδο αυτή προστίθεται στο array των bids η προσφορά που κάνει ο επαγγελματίας. Εδώ, στέλνεται στη βάση μόνο το id του επαγγελματία που κάνει την προσφορά και το ποσό, τα οποία εξάγονται από το req.query και το middleware αντίστοιχα.

let makeBid = (req, res) => {
    Issue.findOneAndUpdate({_id: req.query.id}, {
        $push: {
            bids: {
```

```
            creator: req.userData.userId,
            amount: req.query.bid
        }
    }
  }).then((result, err) => {
    if (err) console.log(err);
    res.status(201).send({
      result: result,
      message: 'Bid posted'
    })
  })
};
```

// Με τη μέθοδο αυτή ενημερώνεται το status ενός bid. Τα δεδομένα του request εξάγονται από το url και την μέθοδο req.query

```
let respondToBId = (req, res) => {

  Issue.findOneAndUpdate(
    {"_id": req.query.issueId, "bids._id": req.query.bidId},
    {
      $set:
        {
          "bids.$.status": req.query.response
        }
    })
    .then((result, err) => {
      if (err) console.log(err);
      res.status(201).send({
        result: result,
        message: 'Responded to bid'
      })
    })
};
```

// Με τη μέθοδο αυτή ενημερώνεται το confirmed document ενός bid. Όταν ο επαγγελματίας επιβεβαιώνει μια αποδεκτή πρόταση από ένα πελάτη.

```
let confirmIssue = (req, res) => {

  Issue.findOneAndUpdate(
    {"_id": req.query.issueId, "bids._id": req.query.bidId},
    {
      $set:
        {
          "bids.$.confirmed": true,
        }
```

```
    })
    .then((result, err) => {
      if (err) console.log(err);
      res.status(201).send({
        result: result,
        message: 'Bid confirmed'
      })
    })
};
```

// Με τη μέθοδο αυτή ενημερώνεται το closed document ενός bid. Όταν ο επαγγελματίας κλείνει μια αίτηση εφόσον έχει ολοκληρωθεί

```
let closeIssue = (req, res) => {
  console.log('&&' + req.query.response);
  console.log('bidId: ' + req.query.bidId);
  console.log('issueId: ' + req.query.issueId);

  Issue.findOneAndUpdate(
    {"_id": req.query.issueId, "bids._id": req.query.bidId},
    {
      $set:
        {
          "bids.$.closed": true,
          closed: true
        }
    })
    .then((result, err) => {
      if (err) console.log(err);
      res.status(201).send({
        result: result,
        message: 'Issue closed'
      })
    })
};

};
```

// Με τη μέθοδο αυτή ανακτώνται από τη βάση τα issues ανά πόλη.

```
let getIssuesByCity = (req, res) => {
  Issue.findOne({"address.city": req.query.city}).then(issue => {
    if (issue === null) {
      console.log('no issues')
    } else {
      res.status(200).send({
```

```
            issue: issue,
            message: "Issues fetched successfully"
        });
    }

  })
};

//Εξαγωγή των μεθόδων
module.exports = {
    fetchIssues: fetchIssues,
    addIssue: addIssue,
    deleteIssue: deleteIssue,
    getUserIssues: getUserIssues,
    getIssuesById: getIssuesById,
    updateIssue: updateIssue,
    makeBid: makeBid,
    respondToBId: respondToBId,
    confirmIssue: confirmIssue,
    closeIssue: closeIssue,
    getIssuesByCity: getIssuesByCity,
    getClosedUserIssues: getClosedUserIssues,
    getOpenProIssues: getOpenProIssues,
    getClosedProIssues: getClosedProIssues,
    getIssuesForPro: getIssuesForPro

};
```

3.3.3. Αρχείο browse.controller.js

Σε αυτό το αρχείο αναπτύσσονται, κυρίως, οι μέθοδοι οι οποίες χρησιμοποιούνται για την ανώνυμη περιήγηση από τον χρήστη.

```
//Εισάγουμε τα απαιτούμενα μοντέλα
const Categories = require('../models/categories.model');
const Cities = require('../models/cities.model');
const Pro = require('../models/pro.model');

//Με τη μέθοδο αυτή ανακτώνται οι κατηγορίες από τη βάση ώστε να «καταναλωθούν» από τον client
let getCategories = (req,res) => {
    Categories.find().then(cat=> {
        res.status(200).send({
            cat: cat,
            message: "Categories fetched successfully"
```

```
        });
    })
};
```

//Με τη μέθοδο αυτή ανακτώνται οι πόλεις από τη βάση ώστε να «καταναλωθούν» από τον client.

```
let getCities = (req,res) => {
    Cities.find().sort({value: 1}).then(city=> {
        res.status(200).send({
            city: city,
            message: "Cities fetched successfully"
        });
    })
};
```

//Με τη μέθοδο αυτή ανακτώνται οι επαγγελματίες από τη βάση με αλφαβητική σειρά, βάσει του επωνύμου, ώστε να «καταναλωθούν» από τον client

```
let getProsSortedByName = (req,res) => {
    Pro.find().sort({p_surname: req.query.sort}).then( sortedProsByName => {
            console.log(sortedProsByName);
            res.status(200).send({
                pros: sortedProsByName,
                message: 'Pros sorted by name'
            })
        }
    )
};

module.exports = {
    getCategories: getCategories,
    getCities: getCities,
    getProsSortedByName: getProsSortedByName
};
```

3.3.4. Αρχείο reviews.controller.js

Σε αυτό το αρχείο αναπτύσσονται, κυρίως, οι μέθοδοι που χρησιμοποιούνται για τις κριτικές.

```
//Εισάγουμε τα απαιτούμενα μοντέλα
const Review = require('../models/reviews.model');
const mongoose = require('mongoose');
```

//Μέθοδος για την αποθήκευση ενός review. Η περιγραφή και η βαθμολογία (stars) εξάγονται μέσω του bodyparser, το id του χρήστη που καταχωρεί την κριτική από το middleware.verifyToken και το id του επαγγελματία που αφορά η κριτική το παίρνουμε από το url.

```
let addReview = (req, res) => {
    const review = new Review({
        title: req.body.title,
        description: req.body.description,
        stars: req.body.stars,
        user: req.userData.email,
        pro: req.query.bidCreator
    });

    review.save().then((re) => {
        console.log(re);
        res.status(200).send(re)
    })
};
```

// Ανακτώνται όλες τις κριτικές του επαγγελματία. Το id εξάγεται από το middleware verifyToken()

```
let getProReviews = (req,res) => {
    Review.find({pro: req.userData.userId}).then(review=> {
        res.status(200).send({
            message: "Review fetched",
            review: review
        });
    })
};
```

// Ανακτούμε όλες τις κριτικές του χρήστη. Το email το παίρνουμε, ξανά, από το middleware

```
let getUserReviews = (req,res) => {
    Review.find({user: req.userData.email}).then(review=> {
        res.status(200).send({
            message: "Review fetched",
            review: review
        });
    })
};
```

```
module.exports = {
    addReview: addReview,
    getProReviews: getProReviews,
    getUserReviews: getUserReviews
};
```

3.4. Φάκελος middlewares

3.4.1 Αρχείο verifyToken.js

```
//Εισάγεται η βιβλιοθήκη jsonwebtoken
const jwt = require('jsonwebtoken');

//Μέθοδος που επαληθεύει το token από τον client.
let verifyToken = (req, res, next) => {

    //Ελέγχεται αν υπάρχει το authorization στα headers του request
if (!req.headers.authorization) {
return res.status(401).send('Unauthorized request')
    }
  //Αν υπάρχει, τότε εξάγεται, χωρίζεται το string του authorization μετά το κενό και αποθηκεύεται αυτό το string στη μεταβλητή token. Το πρώτο κομμάτι είναι το secret key που αποθηκεύει ο client με το TokenInterceptor.
let token = req.headers.authorization.split(' ')[1];

    //Αν δεν υπάρχει token στέλνεται μήνυμα, πως είναι unauthorized request
    if (token === 'null') {
    return res.status(401).send('Unauthorized request')
}

    // η μέθοδος verify επαληθεύει το token και περνάει σε αυτό ένα secret key
let payload = jwt.verify(token, 'secret');

    // στέλνεται στο request το userData, το οποίο περιέχει τα στοιχεία που έχουν αποθηκευθεί στο payload κατά τη διάρκεια του jwt.sign
req.userData = {email: payload.email, userId: payload.userId};

    //Αν δεν υπάρχει payload στέλνεται μήνυμα, πως είναι unauthorized request
if (!payload) {
    return res.status(401).send('Unauthorized request')
}
    //Καλείται η επόμενη μέθοδος ή route να εκτελεστεί
next();

};

module.exports = {
```

```
    verifyToken: verifyToken,
};
```

3.5. Φάκελος routes

3.5.1. Αρχείο browse.routes.js

```
// Εισάγεται το express για να ενεργοποιηθεί το router το οποίο επιτρέπει τη δημιουργία modules από routes έτσι ώστε να εξαχθούν σε άλλους φακέλους
const express = require('express');
const router = express.Router();

// Εισάγεται ο controller για τα browser routes
const ctrlBrowsing = require('../controllers/browse.controller');

// Ανατίθεται σε κάθε route η μέθοδος που θα εκτελέσει
router.get('/categories', ctrlBrowsing.getCategories);
router.get('/getProsSortedByName', ctrlBrowsing.getProsSortedByName);
router.get('/cities', ctrlBrowsing.getCities);

// Εξάγεται το router ώστε να είναι δυνατό τα routes να χρησιμοποιηθούν έξω από αυτό.
module.exports = router;
```

3.5.2. Αρχείο issue.routes.js

```
// Σε αυτό το αρχείο εισάγεται και το middleware το οποίο βοηθά στο authentication system αλλά και σε κάποιες άλλες μεθόδους που χρησιμοποιούνται

const express = require('express');
const router = express.Router();
let ctrlIssues = require('../controllers/issues.controller');
let ctrlReviews = require('../controllers/review.controller');
let middleware = require('../middlewares/verifyToken');

router.get('/', ctrlIssues.fetchIssues);
router.post('/add', middleware.verifyToken, ctrlIssues.addIssue);
router.post('/add/review', middleware.verifyToken, ctrlReviews.addReview);
router.put('/update', ctrlIssues.updateIssue);
router.put('/bid', middleware.verifyToken, ctrlIssues.makeBid);
router.put('/bidResponse', ctrlIssues.respondToBId);
router.put('/confirmIssue', ctrlIssues.confirmIssue);
router.put('/closeIssue', ctrlIssues.closeIssue);
router.get('/getUserIssues', middleware.verifyToken, ctrlIssues.getUserIssues);
router.get('/getClosedProIssues', middleware.verifyToken, ctrlIssues.getClosedProIssues);
router.get('/getOpenProIssues', middleware.verifyToken, ctrlIssues.getOpenProIssues);
```

```
router.get('/getClosedUserIssues', middleware.verifyToken,
ctrlIssues.getClosedUserIssues);
router.delete('/add:id', ctrlIssues.deleteIssue);
router.get('/getById', middleware.verifyToken, ctrlIssues.getIssuesById);
router.get('/getByCity', ctrlIssues.getIssuesByCity);
router.get('/getIssuesForPro', ctrlIssues.getIssuesForPro);

module.exports = router;
```

3.5.3. Αρχείο pro.routes.js

```
const express = require('express');
const router = express.Router();
let ctrlPros = require('../controllers/pros.controller');
let ctrlReviews = require('../controllers/review.controller');
const middlewares = require('../middlewares/verifyToken');

router.get('/', ctrlPros.getPros);
router.get('/profile',middlewares.verifyToken, ctrlPros.getProProfile);
router.get('/prosCity', ctrlPros.getProsByCity);
router.get('/prosCategory', ctrlPros.getProsByCategory);
router.get('/prosName', ctrlPros.getProsByName);
router.get('/reviews', middlewares.verifyToken, ctrlReviews.getProReviews);
router.get('/reviews/user', middlewares.verifyToken, ctrlReviews.getUserReviews);

module.exports = router;
```

3.5.4. Αρχείο user.routes.js

```
const express = require('express');
const router = express.Router();
const middlewares = require('../middlewares/verifyToken');

let ctrlUser = require('../controllers/user.controller');

router.get('/profile', middlewares.verifyToken, ctrlUser.getProfile);
router.post('/u_register', ctrlUser.u_register);
router.post('/p_register', ctrlUser.p_register);
router.post('/u_login', ctrlUser.u_login);
router.post('/p_login', ctrlUser.p_login);

module.exports = router;
```

4. Ανάλυση κώδικα του project (Client Side)

Η δομή του client χωρίζεται σε components, models, services.

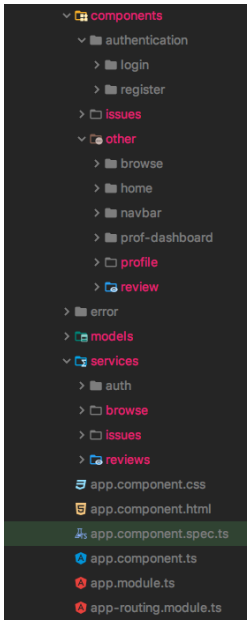
Components: Είναι τα κύρια συστατικά του client και Αποτελούνται από ένα typescript αρχείο, ένα html και ένα css. Ένα component μπορεί να επαναχρησιμοποιηθεί οπουδήποτε αλλού στο project, βάζοντας απλά τον selector του. Το template-url μας δηλώνει την html σελίδα που χρησιμοποιεί ο συγκεκριμένος component, όπως και το styleUrl.

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
```

Models: Είναι typescript αρχεία δημιουργούμε, τα οποία περιέχουν κάποιες κλάσεις, ώστε να μπούμε να ελέγχουμε καλύτερα κάποιες μεταβλητές που δημιουργούμε κάνοντας πιο strictly-typed based κώδικα.
Services: Είναι typescript αρχεία στα οποία αναπτύσσουμε μεθόδους οι οποίες μπορούν να γίνουν injected μέσω του constructor και να χρησιμοποιούν σε παραπάνω από ένα components. Αυτό αυξάνει το reusability του κώδικα του project.

```
//Παράδειγμα όπου κάνουμε inject to _issue τύπου IssueService στον contructor ενός component
constructor(private _issue: IssueService)
```

Δεξιά φαίνεται πως είναι δομημένο το app directory το οποίο περιέχει όλα τα βασικά συστατικά του client, τα οποία αναφέρθηκαν παραπάνω. Οι component με το αναγνωριστικό app, είναι τα αρχεία όπου οι αλλαγές σε αυτά εφαρμόζονται σε όλο το project. Για παράδειγμα, αν προστεθεί ένα component στο app.component.html, όπως έχουμε γίνει με τον component <app-navbar>, τότε αυτός θα εμφανίζεται σε όλα τα components του project. Το ίδιο ισχύει και για τα υπόλοιπα αρχεία με το αναγνωριστικό app.



4.1. Components

4.1.1. AppComponent

app.component.html

```
//Με την παρακάτω εντολή φορτώνεται η navbar, κάτι που σημαίνει πως θα εμφανίζεται σε όλες τις οθόνες του project
<app-navbar></app-navbar>
```

```
//Με το directive <router-outlet> ενημερώνουμε τον router που
να κάνει render τον περιεχόμενο
<router-outlet></router-outlet>
```

app.module.ts

Σε αυτό το αρχείο γίνεται εισαγωγή οποιοδήποτε module ή service χρειάζονται οι υπόλοιποι components.

app.routing-module.ts

Σε αυτό το αρχείο δηλώνονται τα routes του client, καθώς και σε ποιο component αντιστοιχούν. Σε κάποια από αυτά δηλώνεται και ένα id στο path έτσι ώστε να στέλνουμε δεδομένα στον server και να εξάγονται με τη μέθοδο req.params. Ta route guards που εφαρμόζονται σε κάποια από αυτά, είναι για να προστατεύουν κάποια routes, από το να έχουν πρόσβαση χρήστες που δεν είναι συνδεδεμένοι.

```
const routes: Routes = [
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'browse',
    component: BrowseComponent
  },

  {
    path: 'login',
    component: LoginComponent
  },
  {
    path: 'register',
    component: RegisterComponent
  },
  {
    path: 'profile',
    component: ProfileComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'profDashboard',
    component: ProfDashboardComponent,
    canActivate: [AuthGuard]
  },

  {
```

```
  path: 'addIssue',
  component: IssueCreateComponent,
  canActivate: [AuthGuard]
 },
 {
  path: 'edit/:issueId',
  component: IssueCreateComponent,
  canActivate: [AuthGuard]
 },
 {
  path: 'bid/:issueId',
  component: ProfDashboardComponent,
  canActivate: [AuthGuard]
 },
 {
  path: 'test',
  component: TestIssuesComponent,
 },


];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```

4.1.2. HomeComponent

Ο HomeComponent είναι ένας component που δεν τρέχει κάτι παραπάνω από την αρχική οθόνη και απλά χρησιμοποιούνται μερικές γραμμές κώδικα html και κάποιες παραπάνω από css ώστε να σχεδιάσουμε την αρχική σελίδα.

* Οι ετικέτες mat που υπάρχουν στον HTML κώδικά αναφέρονται σε pre-built components της Angular Material.

home.component.html

```
<div class="home">
 <div class="text-box">
  <h1 class="heading-primary">
   <span class="heading-primary-main">FIXAIR</span>
   <span class="heading-primary-sub">Lorem ipsum</span>
```

```
    </h1>
  </div>
</div>
```

home.component.css

Εφαρμόζονται κάποιες γραμμές css έτσι ώστε να οριστεί background στην αρχική σελίδα, να τοποθετηθούν και να μορφοποιηθούν οι τίτλοι, όπως επίσης και για τη δημιουργία καπόιων animations.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}


.home {
  height: 95vh;
  background-image: linear-gradient(to right bottom,
  rgba(30, 97, 204, 0.5),
  rgba(80, 144, 229, 0.8)),
  url("../../../../assets/el.jpg");
  background-size: cover;
  background-position: top;
  clip-path: polygon(0 0, 100% 0, 100% 75vh, 0 100%);
}

.text-box {
  position: absolute;
  top: 40%;
  left: 50%;
  transform: translate(-50%, -50%)
}

.heading-primary {
  color: white;
  text-transform: uppercase;
  backface-visibility: hidden;
}

.heading-primary-main {
  display: block;
  font-size: 60px;
  font-weight: 400;
  letter-spacing: 35px;
```

```
  animation-name: moveInLeft;
  animation-duration: 1s;
  animation-timing-function: ease-out;
  animation-iteration-count: 3;
  animation-delay: 3s;
}

.heading-primary-sub {
  display: block;
  font-size: 20px;
  font-weight: 700;
  letter-spacing: 17px;
  transform: translate(5%, 15%);
  animation: moveInRight 2s;
  backface-visibility: hidden;

}

.heading-primary:hover {
  animation: moveInLeft  1s ease-out;
}

@keyframes moveInLeft {
  0% {
    opacity: 0;
    transform: translateX(-100px) rotate(0deg);
  }

  80% {
    transform: translateX(10px) rotate(180deg);
  }

  100% {
    opacity: 1;
    transform: translateX(0);

  }
}

@keyframes moveInRight {
  0% {
    opacity: 0;
    transform: translateX(100px);
  }

  100% {
```

```
    opacity: 1;
    transform: translateX(0);

  }
}
```

4.1.3. NavBarComponent

navbar.component.ts

```
import {Component, OnInit} from '@angular/core';
import {AuthService} from "../../../services/auth/auth.service";
import {Router} from "@angular/router";
import {UserRegisterModel} from "../../../models/userRegister.model";
import {ProRegisterModel} from "../../../models/proRegister.model";

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
export class NavbarComponent implements OnInit {

  userRole;


 // Γίνεται inject ο AuthService και ο Router.
constructor(private _auth: AuthService,
            private router: Router) {}


 // Οι μέθοδοι που δηλώνονται μέσα στην ngOnInit εκτελούνται με το που ο component τίθεται σε λειτουργία.
ngOnInit() {

 // Εδώ καλείται η getProfilDataListener() που μας επιστρέφει ένα observable. Το observable με το που εκτελείται subscribe ενεργοποιείται και δημιουργεί ένα stream από τιμές. Η μέθοδος αυτή ανακτά από τη βάση τα δεδομένα του συγκεκριμένου χρήστη που έχει συνδεθεί και από το response του server αποθηκεύει τον ρόλο του χρήστη στην τιμή userRole που έχει δημιουργηθεί παραπάνω. Το ίδιο κάνει και η getProfProfileDataListener(), αλλά από την πλευρά του επαγγελματία αυτή τη φορά.
 this._auth.getProfileDataListener().subscribe(
    (res: UserRegisterModel[]) => {
     this.userRole = res.role
    }
  );
```

```
 this._auth.getProfProfileDataListener().subscribe((res: ProRegisterModel[]) => {
  this.userRole = res.role
 })
}

 //Με την μέθοδο isUser() ελέγχεται αν ο χρήστης είναι user και με αυτή την πληροφορία, τα components της navbar αλλάζουν δυναμικά, ανάλογα τον χρήστη.
isUser() {
 if (this.userRole === 'user') {
  return true;
 }
}
}
```

navbar.component.html

Χρησιμοποιείται Bootstrap για να γίνει η navbar responsive και χρησιμοποιείται το routerLink ώστε να δηλωθεί που θα μας ανακατωθούνε κάθε στοιχείο της navbar. Τα paths του routerlink έχουν δηλωθεί στο **app-routing-module.ts**

```
<nav class="navbar navbar-expand-lg navbar-light bg-light" >
 <a class="navbar-brand" href="#">FixAir</a>
 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
       aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
  <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav ml-auto navbar-right">
   <span class="spacer"></span>
   <li class="nav-item">
    <a class="nav-link" routerLink="/browse">Περιήγηση</a>
   </li>
   <li class="‘nav-item">

//Σε αυτό το link πραγματοποιείται έλεγχος αν ο χρήστης έχει συνδεθεί και είναι χρήστης (role == user). Η μέθοδος isAuthenticated() είναι μέρος του authService που έχουμε κάνει inject στον constructor αυτού του component. Η isUser() είναι δηλωμένη στον app.component.ts και ελέγχει αν ο χρήστης έχει role = user
      <a class="nav-link" routerLink="/profile" *ngIf="_auth.isAuthenticated() && isUser()">Προφίλ</a>
   </li>
   <li class="nav-item">
    <a class="nav-link" routerLink="/register"
```

```
        *ngIf="!_auth.isAuthenticated()">Εγγραφή</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" routerLink="/addIssue"
        *ngIf="_auth.isAuthenticated() && isUser()">Κάντε μια αίτηση</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" routerLink="/addIssue"
        *ngIf="_auth.isAuthenticated() && !isUser()">Dashboard</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" routerLink="/login"
        *ngIf="!_auth.isAuthenticated()">Σύνδεση</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" *ngIf="_auth.isAuthenticated()"
        style="cursor: pointer"
        (click)="_auth.logoutUser()">Απόσύνδεση</a>
    </li>

  </ul>

 </div>
</nav>
```

4.1.4. BrowseComponent

browse.component.ts

```
import {Component, OnInit} from '@angular/core';
import {BrowseService} from "../../../services/browse/browse.service";
import {ProRegisterModel} from "../../../models/proRegister.model";
import {Observable, Subscription} from "rxjs";
import {Router} from "@angular/router";
import {FormBuilder, FormGroup} from "@angular/forms";
import {map, startWith} from "rxjs/operators";
import {CategoriesModel} from "../../../models/categoriesModel";
import {CitiesModel} from "../../../models/cities.model";

@Component({
 selector: 'app-browse',
 templateUrl: './browse.component.html',
 styleUrls: ['./browse.component.css']
})
export class BrowseComponent implements OnInit {
```

```
constructor(private _browse: BrowseService,
            private router: Router,
            private formBuilder: FormBuilder) {
}

pros;
selectedCity: string = ' ';
filterPro: string = ' ';
searchForm: FormGroup;
name: String;
proCategories;
cities;
```

// Αυτό το array έχει δημιουργηθεί για να εξυπηρετήσει το mat-select στο html αρχείο, το οποίο ορίζει την ταξινόμηση κατ' αλφαβητική σειρά και εμφανίζει στον browser τις viewValue τιμές. Οι τιμές του value εξυπηρετούν ώστε να στείλει στον server τον τρόπο ταξινόμησης που θέλουμε να μας επιστρέψει.

```
filter = [
  {value: 1, viewValue: "Όνομα Όνομα (Α-Ω)"},
  {value: -1, viewValue: "Όνομα Όνομα (Ω-Α)"},
  {value: "Αξιολογήσεων", viewValue: "Αξιολογήσεων"},
];


ngOnInit() {
```

//Καλούμε την getCategories καθώς και την getCategoriesUpdated() η οποία επιστρέφει ένα subject. Αφού πραγματοποιηθεί subscribe στο subject τότε ανατίθενται οι τιμές από το response του server, στην μεταβλητή proCateogories. Αυτές τις κατηγορίες θα χρειαστούν για mat-select το οποίο θα επιστρέφει τους επαγγελματίες βάσει της κατηγορίας που δραστηριοποιούνται.

```
  this._browse.getCategories();
  this._browse.getCategoriesUpdated().subscribe(
    (res: CategoriesModel[]) => {
      // console.log(res);
      this.proCategories = res;
    }
  );
```

//Καλείται η getCities καθώς και την getCitiesUpdated() η οποία επιστρέφει ένα subject. Αφού πραγματοποιηθεί subscribe στο subject τότε ανατίθενται οι τιμές από το response του server, στην μεταβλητή cities. Αυτές οι πόλεις θα χρειαστούν για το select-bar το οποίο θα επιστρέφει τους επαγγελματίες βάσει της πόλης που δραστηριοποιούνται.

```
  this._browse.getCities();
  this._browse.getCitiesUpdated().subscribe(
    (res: CitiesModel[]) => {
```

```
      this.cities = res;
    }
  );
```

//Αυτή είναι μια μικρή reactive form που αφορά το search bar για να βρει ο χρήστης επαγγελματίες κατ' όνομα

```
  this.searchForm = this.formBuilder.group({
    'name': [this.name]
  });
```

//Καλείται η getAllPros καθώς και η getPros$() η οποία επιστρέφει ένα subject. Αφού πραγματοποιηθεί subscribe στο subject τότε ανατίθενται οι τιμές από το response του server, στην μεταβλητή pros. Οι επαγγελματίες θα επιστραφούν και θα εμφανιστούν στη σελίδα του browse.

```
  this._browse.getAllPros();
  this._browse.getPros$().subscribe(
    (res) => {
      this.pros = res;
    }
  );
}
```

//Αυτή η μέθοδος ενεργοποιείται μόλις ο χρήστης επιλέξει πόλη για να βρει τον επαγγελματία και παίρνει ως όρισμα την επιλεγμένη πόλη από το mat-select-bar. Μόλις ενεργοποιηθεί αυτή η μέθοδος καλείται την getProsByCity(selectedCity) καθώς και η getProsByCityUpdated() η οποία επιστρέφει ένα subject. Αφού πραγματοποιηθεί subscribe στο subject τότε ανατίθεται οι τιμές από το response του server, στην μεταβλητή pros.

```
onChangeCity(selectedCity: string) {
  this._browse.getProsByCity(selectedCity);
  this._browse.getProsByCityUpdated().subscribe(
    (pros: ProRegisterModel[]) => {
      this.pros = pros;
    }
  )
}
```

//Αυτή η μέθοδος ενεργοποιείται μόλις ο χρήστης επιλέξει κατηγορία για να βρει τον επαγγελματία και παίρνει ως όρισμα την επιλεγμένη κατηγορία από το mat-select-bar. Μόλις ενεργοποιηθεί αυτή η μέθοδος καλούμε την getProsByCategory(selectedCategory) καθώς και την getProsByCategoryUpdated() η οποία επιστρέφει ένα subject. Αφού κάνουμε subscribe στο subject τότε αναθέτουμε τις τιμές από το response του server, στην μεταβλητή pros.

```
onChangeCategory(selectedCategory: string) {
  this._browse.getProsByCategory(selectedCategory);
  this._browse.getProsByCategoryUpdated().subscribe((pros: ProRegisterModel[]) => {
    this.pros = pros;
```

```
  })
 }
```

//Αυτή η μέθοδος ενεργοποιείται μόλις ο χρήστης κάνει search για να βρει τον επαγγελματία με το όνομά του και παίρνει ως όρισμα το search input του χρήστη από το search-input. Μόλις ενεργοποιηθεί αυτή η μέθοδος καλούμε την getProsByName(name) καθώς και την getProsByNameUpdated() η οποία επιστρέφει ένα subject. Αφού κάνουμε subscribe στο subject τότε αναθέτουμε τις τιμές από το response του server, στην μεταβλητή pros.

```
 search(name) {
  this._browse.getProsByName(name);
  this._browse.getProsByNameUpdated().subscribe((pros:ProRegisterModel[]) => {
   this.pros = pros;
  });
 }
```

//Αυτή η μέθοδος ενεργοποιείται μόλις ο χρήστης επιλέξει να ταξινομήσει τους επαγγελματίες με αλφαβητική σειρά και παίρνει ως όρισμα την τιμή του value από το filterPro array. Μόλις ενεργοποιηθεί αυτή η μέθοδος καλείται η getProsSortedByName(filterPro.value) καθώς και η getProsSortedByNameUpdated() η οποία επιστρέφει ένα subject. Αφού κάνουμε subscribe στο subject τότε αναθέτουμε τις τιμές από το response του server, στην μεταβλητή pros.

```
 onFilterPros(filterPro: string) {
  this._browse.getProsSortedByName(filterPro.value);
  this._browse.getProsSortedByNameUpdated().subscribe((pros: ProRegisterModel) => {
   this.pros = pros;
  });
 }
}
```

browse.component.html

Reactive forms: είναι φόρμες που χρησιμοποιούν την model-driven προσέγγιση για να χειρίζονται inputs σε φόρμες τα οποία αλλάζουν με την πάροδο του χρόνου. Αυτές οι φόρμες αναπτύχθηκαν πάνω στα observable streams, όπου τα inputs και οι τιμές που εισάγονται παρέχονται σαν stream από input values, τα οποία είναι προσβάσιμα ασύγχρονα.

//Από αυτή τη φόρμα εξάγεται το search input του επαγγελματία και κάνει trigger την μέθοδο search(name). Αυτή η μέθοδος έχει όρισμα το name το οποίο έχει οριστεί στη φόρμα παρακάτω.

```
<form (ngSubmit)="search(name)" [formGroup]="searchForm" class="example-form"
style="width: 100%; text-align: center">
  <mat-form-field class="example-full-width">
    <input
      type="text"
      placeholder="Pick one"
      matInput
      formControlName="name"
      id="name"
      [(ngModel)]="name"
      [matAutocomplete]="auto">
    <mat-autocomplete #auto="‘matAutocomplete”>
```

// Πραγματοποιείται μια επανάληψη με την *ngFor στο array που επιστρέφει η βάση και το κάνουμε ανάκτηση με τη μέθοδο getAllPros(). Με το [value] αναθέτουμε τις τιμές που έχουμε επιλέξει και από κάτω με την τεχνική του string interpolation μπορούμε να εμφανίσουμε με τις διπλές αγκύλες τιμές που υπάρχουν στον component.ts.

```
      <mat-option *ngFor="let pro of pros " [value]="pro.p_name + ' ' + pro.p_surname">
        {{pro.p_name + ' ' + pro.p_surname}}
      </mat-option>
    </mat-autocomplete>
  </mat-form-field>
    <button
      type="submit"
      mat-raised-button
    >Search</button>
</form>
```

//Εφαρμόζεται και εδώ η ίδια τεχνική όπως και στις υπόλοιπες φόρμες αυτού του component.html

```
<mat-form-field style="width: 100%; text-align: center">
    <label > Επιλέξτε περιοχή
      <mat-select
        matNativeControl
        [(ngModel)]="selectedCity"
        (selectionChange)="onChangeCity($event)"
      >
        <mat-option *ngFor="let city of cities" [value]="city.value">
          {{city.viewValue}}
        </mat-option>
      </mat-select>
    </label>
  </mat-form-field>
```

```
<br>

<mat-form-field style="width: 100%; text-align: center">
  <label> Επιλέξτε κατηγορία επαγγελματία
    <mat-select matNativeControl
          [(ngModel)]="selectedCity"
          (selectionChange)="onChangeCategory($event)"
    >
      <mat-option *ngFor="let cat of proCategories" [value]="cat.value">
        {{cat.viewValue}}
      </mat-option>
    </mat-select>
  </label>
</mat-form-field>

<mat-form-field style="width: 100%; text-align: center">
  <label> Αναζήτησε βάσει
    <mat-select matNativeControl
          [(ngModel)]="filterPro"
          (selectionChange)="onFilterPros($event)"
    >
      <mat-option *ngFor="let f of filter" [value]="f.value">
        {{f.viewValue}}
      </mat-option>
    </mat-select>
  </label>
</mat-form-field>


<mat-accordion *ngFor="let pro of pros;">
  <mat-expansion-panel>
    <mat-expansion-panel-header class="title">
      <b>Όνομα</b>: {{ pro.p_surname + ' ' + pro.p_name  }}
    </mat-expansion-panel-header>
    <p><b>Ειδικότητα</b>:  {{ pro.p_category }}</p>
    <p><b>Περιοχή</b>:  {{ pro.p_locationCoverage }}</p>
    <p><b>Email</b>:  {{ pro.p_email }}</p>
  </mat-expansion-panel>
</mat-accordion>
```

4.1.5. RegisterComponent

register.component.ts

```
import { Component, OnInit } from '@angular/core';
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {UserRegisterModel} from "../../../models/userRegister.model";
import {AuthService} from "../../../services/auth/auth.service";
import {ProRegisterModel} from "../../../models/proRegister.model";
import {MatSnackBar} from "@angular/material";
import {BrowseService} from "../../../services/browse/browse.service";
import {CategoriesModel} from "../../../models/categoriesModel";
import {CitiesModel} from "../../../models/cities.model";

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
```

//Ορίζονται FormGroups για τη φόρμα εγγραφής του χρήστη και του επαγγελματία. Επίσης δημιουργούνται μεταβλητές τύπου UserRegisterModel, ProRegisterModel για να καταχωρηθούν τα inputs της φόρμας

```
  userRegistrationForm: FormGroup;
  proRegistrationForm: FormGroup;
  userRegistrationData: UserRegisterModel = new UserRegisterModel();
  proRegistrationData: ProRegisterModel = new ProRegisterModel();
  _selected  = 'customer';
```

//Δημιουργείται ένα array το οποίο κρατά τις τιμές για το radio-button όπου ο χρήστης θα επιλέγει αν είναι πελάτης η επαγγελματίας. Όπως και πριν το viewValue του array θα είναι αυτό που θα είναι εμφανές στον browser και το value θα είναι αυτό που θα στέλνεται στον server.

```
  roles = [
    {value: "customer", viewValue: "Πελάτης"},
    {value: "pro", viewValue: "Επαγγελματίας"},
  ];

  proCategories;
  cities;
```

//Πραγματοποείται inject του FormBuilder που χρειάζεται για την φόρμα μας, το MatSnackbar που επιτρέπει να εμφανίζουμε pop-up μηνύματα στην οθόνη αλλά και τα services για την επικοινωνία με τον server

```
  constructor(private formBuilder: FormBuilder,
              private _auth: AuthService,
              private snack: MatSnackBar,
              private _browse: BrowseService) {
```

```
  }

  ngOnInit() {
```

// Με τις παρακάτω μεθόδους πραγματοποιούνται requests στον server ώστε να ανακτήσουμε τις κατηγορίες και τις πόλεις από τη βάση δεδομένων για να τα χρησιμοποιήσουμε στη φόρμα μας. Αναθέτουμε τα responses της βάσης στις τιμές proCategories και cities

```
    this._browse.getCategories();
    this._browse.getCategoriesUpdated().subscribe(
      (res: CategoriesModel[]) => {
       this.proCategories = res;
      }
    );
    this._browse. getAllPros;
    this._browse.getCitiesUpdated().subscribe(
      (res: CitiesModel[]) => {
       this.cities = res;
      }
    );
```

//Ορίζεται στην reactive form μας που θα αποθηκεύονται οι είσοδοί της και εφαρμόζεται validation, βάζοντας περιορισμούς στο τι επιτρέπεται να εισάγει ο χρήστης στο κάθε input καθώς και τα πεδία που είναι απαραίτητα.

```
    this.userRegistrationForm = this.formBuilder.group({
      'name': [this.userRegistrationData.name, [
        Validators.required, // Απαιτούμενο πεδίο
        Validators.minLength(8),  //Μινιμουμ 8 χαρακτήρες
      ]],
      'email': [this.userRegistrationData.email, [
        Validators.required,
        Validators.email, // υποχρεατικά τύπου email
        Validators.minLength(5),
      ]],
      'password': [this.userRegistrationData.password, [
        Validators.required,
        Validators.minLength(5),
        Validators.maxLength(30), Μαξιμουμ 8 χαρακτήρες

      ]],
    });
```

//Φόρμα εγγραφής επαγγελματία

```
    this.proRegistrationForm = this.formBuilder.group({
      'p_name': [this.proRegistrationData.p_name, [
```

```
      Validators.required,
      Validators.minLength(3),
    ]],
    'p_surname': [this.proRegistrationData.p_name, [
      Validators.required,
      Validators.minLength(3),
    ]],

    'p_email': [this.proRegistrationData.p_email, [
      Validators.required,
      Validators.email,
      Validators.minLength(5),
    ]],
    'p_password': [this.proRegistrationData.p_password, [
      Validators.required,
      Validators.minLength(5),
      Validators.maxLength(30),
    ]],
    'p_phone': [this.proRegistrationData.p_phone, [
      Validators.required,
      Validators.minLength(9),
      Validators.maxLength(11),
      Validators.pattern("^[0-9]*$"),  //μόνο αριθμός απόδεκτός

    ]],
    'p_brandName': [this.proRegistrationData.p_brandName, [
      Validators.required,
    ]],
    'p_category': [this.proRegistrationData.p_category, [
      Validators.required,
    ]],
    'p_locationCoverage': [this.proRegistrationData.p_locationCoverage, [
      Validators.required,
    ]],

  });

 }
```

//Με τις μεθόδους registeruser(), registerPro() που παίρνουν ως όρισμα τις τιμές που μας επιστρέφουν οι reactive φόρμες, στέλνεται, μέσω του _auth service τα δεδομένα στον server για να τα ελέγξει και να τα αποθηκεύσει στη βάση δεδομένων

```
registerUser() {
  this._auth.registerUser(this.userRegistrationData);
}
```

```
 registerPro() {
   this._auth.registerPro(this.proRegistrationData);
   console.log(this.proRegistrationData)
 }
```

// Με τη μέθοδο αυτή ελέγχεται η τιμή του _selected, την οποία ενημερώνει το mat-radio-button. Αυτή η μέθοδος είναι υπεύθυνη να αλλάζει τις φόρμες ανάλογα με την επιλογή του χρήστη

```
 checkRole(){
   return this._selected === 'pro';
 }
```

//Με τις παρακάτω μεθόδους πραγματοποιείται έλεγχος για την εγκυρότητα του mail.

```
 get email() {
   return this.userRegistrationForm.get('email');
 }

 getEmailError() {
   return this.email.hasError('required') ? 'Εισάγετε ένα email' :
     this.email.hasError('email') ? 'Το email δεν είναι έγκυρο' : '';
 }

}
```

register.component.html

```
<div class="medium-list">
 <div class="list-viewer-wrapper">
   <div class="list-header-title">Register</div>
```

// Το radio button το οποίο επιλέγει ο χρήστης την ιδιότητά του, το οποίο είναι συνδεδεμένο με την μεταβλητή _selected, της οποίας την τιμή ελέγχει η μέθοδος checkRole()

```
   <div class="radio">
     <label> Είστε πελάτης ή επαγγελματίας;</label><br>
     <mat-radio-group labelPosition="after" [(ngModel)]="_selected">
       <mat-radio-button *ngFor="let role of roles" [value]="role.value">
         {{role.viewValue}}
       </mat-radio-button>
     </mat-radio-group>
   </div>

   <h3 style="text-align: center">{{!checkRole() ? 'Εγγραφή πελάτη' : 'Εγγραφή επαγγελματία'}}</h3>
```

// Η φόρμα εγγραφή του πελάτη η οποία εμφανίζεται μόνο όταν η μεταβλητή _selected δεν είναι pro. Το attribute novalidate απενεργοποιεί το native browser validation, καθώς κάνουμε custom validation με τους validators της reactive form.

```
  <form *ngIf="!checkRole()" [formGroup]="userRegistrationForm"
      (ngSubmit)="userRegistrationForm.valid && registerUser()" novalidate>
    <div class="form-container">
      <mat-form-field>
        <input id="name"
            matInput
            formControlName="name"
            [(ngModel)]="userRegistrationData.name"
            placeholder="Όνομα">
```

// Τα <mat-error> ελέγχουν τα validation patterns που έχουν οριστεί στον component και εμφανίζουν μηνύματα λάθους σε περίπτωση που οι είσοδοι δεν πληρούν τις προδιαγραφές των validators.

```
        <mat-error *ngIf="userRegistrationForm.controls['name'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
        <mat-error *ngIf="userRegistrationForm.controls['name'].errors?.minlength">Το πεδίο πρέπει να έχει μινιμουμ 8 χαρακτήρες</mat-error>
      </mat-form-field>
      <mat-form-field>
        <input id="email"
            matInput
            formControlName="email"
            [(ngModel)]="userRegistrationData.email"
            placeholder="Email">
```

// Με την τεχνική του string interpolation εφαρμόζουμε ένα custom validator που έχουμε στον component.

```
        <mat-error *ngIf="email.invalid">{{getEmailError()}}</mat-error>
      </mat-form-field>
      <mat-form-field>
        <input id="password"
            matInput
            formControlName="password"
            [(ngModel)]="userRegistrationData.password"
            placeholder="Κωδικός πρόσβασης"
            [type]="!hide ? 'password' : 'text' "/>
```

// Βάζουμε ένα icon το οποίο κρύβει ή εμφανίζει τον κωδικό καθώς τον δακτυλογραφούμε στην φόρμα

```
        <mat-icon matSuffix (click)="hide = !hide">
          {{ hide ? 'visibility' : 'visibility_off' }}
        </mat-icon>
```

```
        <mat-error *ngIf="userRegistrationForm.controls['name'].errors?.required">Το
πεδίο είναι υποχρεωτικό</mat-error>

      </mat-form-field>
      <button mat-raised-button
            color="primary"
            type="submit"
// Σε περίπτωση που η φόρμα δεν είναι valid το κουμπί Register είναι απενεργοποιημένο
και ως εκ τούτου δεν μπορεί να ολοκληρωθεί η εγγραφή
            [disabled]="!userRegistrationForm.valid">
        Register
      </button>
      <button mat-raised-button
            color="accent"
            type="submit"
// Ανακατευθύνεται ο χρήστης στον LoginComponent
            routerLink="/login"
            class="divider">
        Login
      </button>
    </div>
  </form>


// Φόρμα εγγραφής επαγγελματία. Παρόλο που εδώ η φόρμα είναι μεγαλύτερη και έχει
καπόια στοιχεία που δεν έχει η προηγούμενη, η λογική είναι η ίδια.
  <form
*ngIf="checkRole()" [formGroup]="proRegistrationForm" (ngSubmit)="proRegistrationF
orm.valid && registerPro()"
        novalidate>
    <div class="form-container">
      <mat-form-field>
        <input id="p_name"
            matInput
            formControlName="p_name"
            [(ngModel)]="proRegistrationData.p_name"
            placeholder="Όνομα">
        <mat-error *ngIf="proRegistrationForm.controls['p_name'].errors?.required">Το
πεδίο είναι υποχρεωτικό</mat-error>
      </mat-form-field>

<mat-form-field>
        <input id="p_surname"
            matInput
            formControlName="p_surname"
            [(ngModel)]="proRegistrationData.p_surname"
            placeholder="Επώνυμο">
```

```
        <mat-error *ngIf="proRegistrationForm.controls['p_surname'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
      </mat-form-field>


      <mat-form-field>
        <input id="p_email"
               matInput
               formControlName="p_email"
               [(ngModel)]="proRegistrationData.p_email"
               placeholder="Email">
        <mat-error *ngIf="proRegistrationForm.controls['p_email'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
      </mat-form-field>

      <mat-form-field>
        <input id="p_password"
               matInput
               formControlName="p_password"
               [(ngModel)]="proRegistrationData.p_password"
               placeholder="Κωδικός πρόσβασης"
               [type]="!hide ? 'password' : 'text' "/>
        <mat-icon matSuffix (click)="hide = !hide">
          {{ hide ? 'visibility' : 'visibility_off' }}
        </mat-icon>
        <mat-error *ngIf="proRegistrationForm.controls['p_password'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
      </mat-form-field>

      <mat-form-field>
        <input id="p_brandName"
               matInput
               formControlName="p_brandName"
               [(ngModel)]="proRegistrationData.p_brandName"
               placeholder="Επωνυμία εταιρείας">
        <mat-error
*ngIf="proRegistrationForm.controls['p_brandName'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>

      </mat-form-field>

      <mat-form-field>
        <input id="p_phone"
               matInput
               formControlName="p_phone"
               [(ngModel)]="proRegistrationData.p_phone"
               placeholder="Τηλέφωνο">
```

```
        <mat-error *ngIf="proRegistrationForm.controls['p_phone'].errors?.required">Το
πεδίο είναι υποχρεωτικό</mat-error>
        <mat-error
*ngIf="proRegistrationForm.controls['p_phone'].errors?.pattern">Παρακαλούμε δώστε ένα
έγκυρο αριθμό (Αριθμός)</mat-error>
        <mat-error *ngIf="proRegistrationForm.controls['p_phone'].errors?.minlength">Το
πεδίο πρέπει να έχει μινιμουμ 9 αριθμούς</mat-error>
        <mat-error *ngIf="proRegistrationForm.controls['p_phone'].errors?.maxlength">Το
πεδίο πρέπει να έχει μάξιμουμ 11 αριθμούς</mat-error>       </mat-form-field>

      <mat-form-field>
        <label> Επιλέξτε κατηγορία απασχόλησης

          <mat-select matNativeControl
                      [(ngModel)]="proRegistrationData.p_category"
                      formControlName="p_category">
            <mat-option *ngFor="let car of proCategories" [value]="car.value">
              {{car.viewValue}}
            </mat-option>
          </mat-select>
        </label>
        <mat-error *ngIf="proRegistrationForm.controls['p_category'].errors?.required">Το
πεδίο είναι υποχρεωτικό</mat-error>
      </mat-form-field>
      <mat-form-field>
        <label> Επιλέξτε κατηγορία κάλυψης
          <mat-select matNativeControl
                      [(ngModel)]="proRegistrationData.p_locationCoverage"
                      formControlName="p_locationCoverage">
            <mat-option *ngFor="let city of cities" [value]="city.value">
              {{city.viewValue}}
            </mat-option>
          </mat-select>
        </label>
        <mat-error
*ngIf="proRegistrationForm.controls['p_locationCoverage'].errors?.required">Το πεδίο
είναι υποχρεωτικό</mat-error>

      </mat-form-field>
      <button mat-raised-button
              color="primary"
              type="submit"
              [disabled]="!proRegistrationForm.valid">
        Register
      </button>
      <button mat-raised-button
              color="accent"
```

```
            type="submit"
            routerLink="/login"
            class="divider">
        Login
      </button>
    </div>
  </form>
 </div>
</div>
<div></div>
```

4.1.6. LoginComponent

login.component.ts

// Σε αυτόν τον component δεν εφαρμόζεται κάτι διαφορετικό σε σχέση με τον RegisterComponent. Η μόνη διαφορά είναι πως requests στον server για σύνδεση και όχι για εγγραφή.

```
import { Component, OnInit } from '@angular/core';
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {UserLoginModel} from "../../../models/userLogin.model";
import {AuthService} from "../../../services/auth/auth.service";
import {ProLoginModel} from "../../../models/proLogin.model";
import {MatSnackBar} from "@angular/material";

@Component({
 selector: 'app-login',
 templateUrl: './login.component.html',
 styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

 userLoginForm: FormGroup;
 proLoginForm: FormGroup;
 _selected  = 'customer';

 private userLoginData: UserLoginModel = new UserLoginModel();
 private proLoginData: ProLoginModel = new ProLoginModel();
```

```
 roles = [
   {value: "customer", viewValue: "Πελάτης"},
   {value: "pro", viewValue: "Επαγγελματίας"},
 ];

 constructor(private formBuilder: FormBuilder,
             private _auth: AuthService,
             private snack: MatSnackBar
             ) { }

 ngOnInit() {
   this.userLoginForm = this.formBuilder.group({
   'email': [this.userLoginData.email, [
     Validators.required,
     Validators.email
   ]],
   'password': [this.userLoginData.password, [
     Validators.required,
   ]],
 });
   this.proLoginForm = this.formBuilder.group({
   'p_email': [this.proLoginData.p_email, [
     Validators.required,
     Validators.email
   ]],
   'p_password': [this.proLoginData.p_password, [
     Validators.required,
   ]],
 })
}

 checkRole(){
   return this._selected === 'pro';
 }

 loginUser() {
  this._auth.loginUser(this.userLoginData);

 }

 loginPro() {
   this._auth.loginPro(this.proLoginData);
 }

}
```

login.component.html

```
<div class="medium-list">
 <div class="list-viewer-wrapper">
  <div class="list-header-title">Σύνδεση</div>

  <div class="radio">
   <label > Είστε πελάτης ή επαγγελματίας;</label><br>
   <mat-radio-group labelPosition="after" [(ngModel)]="_selected">
    <mat-radio-button *ngFor="let role of roles" [value]="role.value">
     {{role.viewValue}}
    </mat-radio-button>
   </mat-radio-group>
  </div>

  <h3 style="text-align: center">{{!checkRole() ? 'Σύνδεση πελάτη': 'Σύνδεση
επαγγελματία'}}</h3>



<form *ngIf="!checkRole()" [formGroup]="userLoginForm" (ngSubmit) =
"userLoginForm.valid && loginUser()" novalidate>
 <div class="form-container">
  <mat-form-field>
   <input id="email"
        matInput
        formControlName="email"
        [(ngModel)] = "userLoginData.email"
        placeholder="Email">
  </mat-form-field>

  <mat-form-field>
   <input id="password"
        matInput
        formControlName="password"
        [(ngModel)] = "userLoginData.password"
        placeholder="Password"
        [type]="hide ? 'password' : 'text' "/>
   <mat-icon matSuffix (click)="hide = !hide">
    {{ hide ? 'visibility' : 'visibility_off' }}
   </mat-icon>
  </mat-form-field>

  <button mat-raised-button
       color="primary"
       type="submit"
       [disabled]="!userLoginForm.valid">
```

```
    Login
  </button>
  <button mat-raised-button
          color="accent"
          type="submit"
          routerLink="/register"
          class="divider">
    Register
  </button>
 </div>
</form>

  <form *ngIf="checkRole()" [formGroup]="proLoginForm" (ngSubmit) =
"proLoginForm.valid && loginPro()" novalidate>
 <div class="form-container">
  <mat-form-field>
   <input id="p_email"
          matInput
          formControlName="p_email"
          [(ngModel)] = "proLoginData.p_email"
          placeholder="Email">
  </mat-form-field>
  <mat-form-field>
   <input id="p_password"
          matInput
          formControlName="p_password"
          [(ngModel)] = "proLoginData.p_password"
          placeholder="Password"
          [type]="hide ? 'password' : 'text' "/>
   <mat-icon matSuffix (click)="hide = !hide">
    {{ hide ? 'visibility' : 'visibility_off' }}
   </mat-icon>
  </mat-form-field>

  <button mat-raised-button
          color="primary"
          type="submit"
          [disabled]="!proLoginForm.valid">
    Login
  </button>
  <button mat-raised-button
          color="accent"
          type="submit"
          routerLink="/register"
          class="divider">
    Register
  </button>
```

```
  </div>
</form>

  </div>
</div>
```

4.1.7. IssueCreateComponent

issue.create.compoent.ts

Στο component αυτό ο χρήστης δημιουργεί την αίτηση. Κατά την δημιουργία της αίτησης χρησιμοποιούνται οι ίδιες τεχνικές με το registration. Το μόνο που διαφοροποιείται εδώ, είναι πως πραγματοποιείται έλεγχος αν η φόρμα βρίσκεται σε edit ή create form. Το κομμάτι κώδικα που εκτελείται αυτός ο έλεγχος είναι αυτό.

```
import {Component, OnInit} from '@angular/core';
import {FormBuilder, FormControl, FormGroup, Validators} from "@angular/forms";
import {IssueModel} from "../../../models/Issue.model";
import {IssueService} from "../../../services/issues/issue.service";
import {ActivatedRoute, ParamMap} from "@angular/router";
import {CitiesModel} from "../../../models/cities.model";
import {BrowseService} from "../../../services/browse/browse.service";
import {CategoriesModel} from "../../../models/categoriesModel";
@Component({
  selector: 'app-issue-create',
  templateUrl: './issue-create.component.html',
  styleUrls: ['./issue-create.component.css']
})



export class IssueCreateComponent implements OnInit {

  createIssueForm: FormGroup;
  issueData: IssueModel = new IssueModel();
  private mode;
  private issueId: string;
  issue: IssueModel;
  issueToUpdate: IssueModel[] = [];
  cities;
  categories;
  constructor(private formBuilder: FormBuilder,
              private _issues: IssueService,
              private a_route: ActivatedRoute,
              private _browse: BrowseService) {
```

```
    this.createIssueForm = this.formBuilder.group({
      'title': [new FormControl(this.issueData.title), [
        Validators.required,
        Validators.minLength(9),

      ]],
      'description': [new FormControl(this.issueData.description), [
        Validators.required,
        Validators.minLength(9)
      ]],
      'category': [new FormControl(this.issueData.category), [
        Validators.required,
      ]],
      'price': [new FormControl(this.issueData.price), [
        Validators.required,
        Validators.maxLength(5),
        Validators.pattern("^[0-9]*$")
      ]],
      'city': [new FormControl(this.issueData.city), [
        Validators.required,
        Validators.minLength(3)
      ]],
      'homeAddress': [new FormControl(this.issueData.homeAddress), [
        Validators.required,
        Validators.minLength(5)
      ]],

      'phone': [new FormControl(this.issueData.phone), [
        Validators.required,
        Validators.pattern("^[0-9]*$"),
        Validators.minLength(9),
        Validators.maxLength(11),

      ]],
      'availability': [new FormControl(this.issueData.availability), [
        Validators.required,
        Validators.minLength(6)
      ]],
    });
    this.createIssueForm.reset();
  }

  ngOnInit() {
// Ανακτώνται από τη βάση οι πόλεις και οι κατηγορίες. Η μεθοδολογία είναι η ίδια με προηγούμενα κομμάτι κώδικα
    this._browse.getCities();
```

```
  this._browse.getCitiesUpdated().subscribe(
    (res: CitiesModel[]) => {
     this.cities = res;
    }
  );
  this._browse.getCategories();
  this._browse.getCategoriesUpdated().subscribe(
    (res: CategoriesModel[]) => {
      this.categories = res;
    }
  );
```

// Μέσω του ActivatedRoute που έχουμε κάνει inject ελέγχουμε αν στο url υπάρχει το issueId. Αν υπάρχει το issueId ορίζουμε πως είμαστε σε edit mode. Το issueId το στέλνει ο χρήστης πατώντας το κουμπί edit.

```
  this.a_route.paramMap.subscribe((paramMap: ParamMap) => {
    if (paramMap.has('issueId')) {
      this.mode = 'edit';
// παίρνουμε το issueId από το URL και το χρησιμοποιούμε ως όρισμα στην getEditedIssue() του _isssue service
      this.issueId = paramMap.get('issueId');
this.issue = this._issues.getEditedIssue(this.issueId);
    } else {
      this.mode = 'create';
      this.issueId = null;
    }
  });
  console.log(this.mode);
```

// Εφόσον η σελίδα είναι σε edit mode καλείται η getIssueByIssueId(this.issueId) με όρισμα το issueId που πήραμε από το url. Αυτή μέθοδος μας επιστρέφει το συγκεκριμένο issue που θέλουμε να κάνουμε edit.

```
  if (this.mode === 'edit') {
   this._issues.getIssueByIssueId(this.issueId);
    this._issues.getIssueToBeUpdatedistener().subscribe(
      (resIssue) => {
```

// Μέσω της μεθόδου setValue των reactive forms, αναθέτουμε τιμές στη φόρμα. Έτσι, στη φόρμα τοποθετούνται τιμές που έχουμε πάρει ως response από τον server, ώστε κατά τη διάρκεια της διόρθωσης της φόρμας ο χρήστης να βλέπει τις προηγούμενες τιμές.

```
        this.createIssueForm.setValue({
          title: resIssue[0].title,
          description: resIssue[0].description,
          category: resIssue[0].category,
          price: resIssue[0].price,
```

```
          city: resIssue[0].address.city,
          //
          homeAddress: resIssue[0].address.homeAddress,
          phone: resIssue[0].phone,
          availability: resIssue[0].availability
        });
      }
    );
  }
}
```

```
// Αυτή η μέθοδος ελέγχει αν η σελίδα είναι σε edit ή create mode και αναλόγως καλείται η ανάλογη συνάρτηση. Στην updateIssue δίνεται το id του issue γιατί να ενημερωθεί το συγκεκριμένο issue.
addIssue() {
  if (this.mode === 'create') {
    this._issues.addIssues(this.issueData);
  } else if (this.mode === 'edit') {
    this._issues.updateIssue(this.issueId, this.issueData);
  }
}

}
```

issue-create.component.html

```
<mat-card>
// Ο τίτλος της mat-card ενημερώνεται ανάλογα με το mode που βρίσκεται η φόρμα με τους conditional operators του string interpolation.
<mat-card-title>
  {{mode === 'edit' ? 'Ενημερώστε': 'Αποθηκεύστε'}} την αίτησή σας
</mat-card-title>

// Ακολουθείται η ίδια μεθοδολογία με την registration form
<form [formGroup]="createIssueForm" (ngSubmit)="createIssueForm.valid && addIssue() " novalidate>
  <mat-form-field>
    <input
      matInput
      formControlName="title"
      placeholder="Τίτλος βλάβης"
      [(ngModel)]="issueData.title">
    <mat-error *ngIf="createIssueForm.controls['title'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
```

```
    <mat-error *ngIf="createIssueForm.controls['title'].errors?.minlength">Το πεδίο πρέπει να έχει μινιμουμ 9 χαρακτήρες</mat-error>
   </mat-form-field>
   <mat-form-field>
    <input
     matInput
     formControlName="description"
     placeholder="Περιγραφή βλάβης"
     [(ngModel)]="issueData.description">
    <mat-error *ngIf="createIssueForm.controls['description'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
    <mat-error *ngIf="createIssueForm.controls['description'].errors?.minlength">Το πεδίο πρέπει να έχει μινιμουμ 9 χαρακτήρες</mat-error>
   </mat-form-field>
   <mat-form-field>
    <label> Επιλέξτε κατηγορία απασχόλησης

     <mat-select matNativeControl
           [(ngModel)]="issueData.category"
           formControlName="category">
      <mat-option *ngFor="let cat of categories" [value]="cat.value">
       {{cat.viewValue}}
      </mat-option>
     </mat-select>
    </label>
    <mat-error *ngIf="createIssueForm.controls['category'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
   </mat-form-field>
   <mat-form-field>
    <input
     matInput
     formControlName="price"
     placeholder="Βάλτε το ποσό που μπορείτε να διαθέσετε (€)"
     [(ngModel)]="issueData.price">
    <mat-error *ngIf="createIssueForm.controls['price'].errors?.required">Το πεδίο είναι υποχρεωτικό</mat-error>
    <mat-error *ngIf="createIssueForm.controls['price'].errors?.maxlength">Το πεδίο πρέπει να έχει μάξιμουμ 5 αριθμούς</mat-error>
    <mat-error *ngIf="createIssueForm.controls['price'].errors?.pattern">Παρακαλούμε δώστε ένα έγκυρο ποσό (Αριθμός)</mat-error>
   </mat-form-field>
   <mat-form-field>
    <label> Επιλέξτε περιοχή
     <mat-select
      matNativeControl
      [(ngModel)] = "issueData.city"
      formControlName="city">
```

```
        <label>Νομός</label>
        <mat-option *ngFor="let cit of cities" [value]="cit.value"> {{cit.viewValue}}</
mat-option>
      </mat-select>
    </label>

  </mat-form-field>
  <mat-form-field>
    <input
     matInput
     formControlName="homeAddress"
     placeholder="Βάλτε τη διεύθυνση του χώρου που θα έρθει ο εκπρόσωπός μας"
     [(ngModel)]="issueData.homeAddress">
    <mat-error *ngIf="createIssueForm.controls['homeAddress'].errors?.required">Το
πεδίο είναι υποχρεωτικό</mat-error>
    <mat-error *ngIf="createIssueForm.controls['homeAddress'].errors?.minlength">Το
πεδίο πρέπει να έχει μινιμουμ 5 χαρακτήρες</mat-error>
  </mat-form-field>
  <mat-form-field>
    <input
     matInput
     formControlName="phone"
     placeholder="Τηλέφωνο επικοινωνίας"
     [(ngModel)]="issueData.phone">
    <mat-error *ngIf="createIssueForm.controls['phone'].errors?.required">Το πεδίο είναι
υποχρεωτικό</mat-error>
    <mat-error *ngIf="createIssueForm.controls['phone'].errors?.pattern">Παρακαλούμε
δώστε ένα έγκυρο αριθμό (Αριθμός)</mat-error>
    <mat-error *ngIf="createIssueForm.controls['phone'].errors?.minlength">Το πεδίο
πρέπει να έχει μινιμουμ 9 αριθμούς</mat-error>
    <mat-error *ngIf="createIssueForm.controls['phone'].errors?.maxlength">Το πεδίο
πρέπει να έχει μάξιμουμ 11 αριθμούς</mat-error>
  </mat-form-field>
  <mat-form-field>
    <input
     matInput
     formControlName="availability"
     [(ngModel)]="issueData.availability"
     placeholder="Πατήστε στο εικονίδιο ημερολόγιο για να διαλέξετε ημερομηνία"
     [matDatepicker]="picker" >
    <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
    <mat-datepicker #picker></mat-datepicker>
  </mat-form-field>
  <button
    mat-raised-button
    color="accent"
    type="“submit"
```

```
//Το κουμπί που κάνει submit τη φόρμα αλλάζει τον τίτλο του, δυναμικά, ανάλογα με mode που βρίσκεται η σελίδα. Αν η φόρμα δεν είναι valid το κουμπί είναι απενεργοποιημένο.
    [disabled]="!createIssueForm.valid">{{mode === 'edit' ? 'Ενημερώστε': 'Αποθηκεύστε'}}
   </button>
  </form>
</mat-card>
```

4.1.8. IssueListComponent

issue-list.component.ts

```
import {Component, OnDestroy, OnInit} from '@angular/core';
import {IssueService} from "../../../services/issues/issue.service";
import {IssueModel} from "../../../models/Issue.model";
import {Subscription} from "rxjs";
import {AuthService} from "../../../services/auth/auth.service";
import {UserRegisterModel} from "../../../models/userRegister.model";
import {Router} from "@angular/router";
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {ReviewModel} from "../../../models/reviewModel";
import {MatSnackBar} from "@angular/material";
import {MatDialog, MatDialogConfig} from "@angular/material";
import {ReviewComponent} from "../../other/review/review.component";
@Component({
  selector: 'app-issue-list',
  templateUrl: './issue-list.component.html',
  styleUrls: ['./issue-list.component.css']
})

export class IssueListComponent implements OnInit, OnDestroy {
  userId: '';

  issues: IssueModel[] = [];
  data: UserRegisterModel[] = [];
  private issuesSub: Subscription;
 addReviewForm: FormGroup;
  reviewData: ReviewModel = new ReviewModel();



  constructor(private _issue: IssueService,
              private _auth: AuthService,
```

```
            private router: Router,
            private formBuilder: FormBuilder,
            private snack: MatSnackBar,
            private dialog: MatDialog
            ) {
  }
```

```
ngOnInit() {
  // Αυτή είναι η δήλωση της φόρμας για το review του χρήστη. Η φόρμα θα ανοίξει σε αναδυόμενο παράθυρο τύπου MatDialog.

    this.addReviewForm = this.formBuilder.group({
  'description': [this.reviewData.description, [
    Validators.required
  ]],
  'stars': [this.reviewData.stars, [
    Validators.required
  ]]
});

  //Παρακάτω ανακτώνται τα στοιχεία και τα issues του χρήστη.
 this._auth.getProfile();
this._auth.getProfileDataListener()
  .subscribe(
    (data) => {
     this.data = data;
     this.userId = data._id;
    }
  );
this._issue.getUserIssues();
this._issue.getIssueUpdateListener()
  .subscribe(
    (issues: IssueModel[]) => {
      this.issues = issues;
});


  // Διαγράφεται ένα issue και ενημερώνεται η λίστα του χρήστη
  onDelete(issueId: String) {
    this._issue.deleteIssue(issueId);
    this._issue.getIssueByIssueId(this.userId);
  }
```

```
 // Ο χρήστης απαντά στην προσφορά του επαγγελματία. Τα ορίσματα για την μέθοδο αυτή στέλνονται από το click events των accept και reject buttons
 respondToBId(bidId, response, issueId, bid) {
   this._issue.respondToBid(bidId, response, issueId, bid);
   this._issue.getIssues();
   this._issue.getIssueUpdateListener().subscribe((issues: IssueModel[]) => {
     this.issues = issues;
     this.snack.open('Η απάντηση σας καταχωρήθηκε', "OK", {
       duration: 2000
     });
   });
 }

 // Ο χρήστης επιλέγει να προσθέσει μια κριτική. Το όρισμα bidCreator στέλνεται από το click event του add review button
 addReview(bidCreator) {
 this._issue.addReview(this.reviewData, bidCreator);
}

 // Ο χρήστης επιλέγει να προσθέσει μια κριτική. Το όρισμα bidCreator στέλνεται από το click event του add review button. Αυτό το click event κάνει trigger το άνοιγμα ενος αναδυόμενου παραθύρου, το οποίο είναι ένας component (ReviewComponent). Σε αυτό το dialog μέσω της μεθόδου dialogConfig.data, μεταφέρεται η τιμή στον ReviewComponent από τον IssueListComponent.
onReview(creator) {
 const dialogConfig = new MatDialogConfig();
  dialogConfig.disableClose = false;
  dialogConfig.autoFocus = true;
  dialogConfig.width = "60%";
  dialogConfig.data = creator;
  this.dialog.open(ReviewComponent, dialogConfig);
  this.dialog.afterAllClosed.subscribe(creator => {
    console.log(creator)
  })
} }}
```

issue-list.component.html

```
 <mat-accordion>
 // Με μια επανάληψη πραγματοποιείται η εμφάνιση των issues του user.
   <mat-expansion-panel *ngFor="let issue of issues">
     <mat-expansion-panel-header>
       <b style="color: black">{{issue.title}}</b>
     </mat-expansion-panel-header>
```

```
    <p><b>Περιγραφή</b>:  {{ issue.description }}</p>
    <p><b>Creator:  {{ issue.creator }}</b></p>
    <p><b>ID:  {{ issue._id }}</b></p>
    <p><b>Εύρος τιμής</b>:  {{ issue.price }}</p>
    <p><b>Πόλη</b>:  {{ issue.address.city }}</p>
    <p><b>Διεύθυνση</b>:  {{ issue.address.homeAddress }}</p>
    <p><b>Τηλέφωνο</b>:  {{ issue.phone }}</p>
    <p><b>Ημ/νία καταχώρησης: </b>: {{ issue.createdAt | date : 'medium'}}</p>
    <hr>

 // Γίνεται έλεγχος για το πλήθος των issues και αναλόγως ο τίτλος αλλάζει δυναμικά.
    <h3 style="text-align: center">{{issue.bids.length === 0? 'Δεν υπάρχουν προσφορές':
'Προσφορές επαγγελματιών'}}</h3>
 // Με μια επανάληψη πραγματοποιείται η εμφάνιση των bids του κάθε issue
   <mat-action-list *ngFor="let bid of issue.bids" style="border: #030303">
     <p><b>Ποσο: </b>{{bid.amount}}</p>
     <p><b>Ημ/νία Καταχώρησης: </b>{{bid.createdAt | date : 'medium'}}</p>
     <p><b>ID Επαγγελματία: </b>{{bid.creator}}</p>

          // Τα κουμπιά ενημερώνονται δυναμικά ανάλογα με τα δεδομένα που μας στέλνει
ο server.
          Αν το status του bid είναι accepted το κουμπί είναι disabled, και αντιστρόφως.
      <button [disabled]="bid.status === 'accepted'" mat-raised-button color="primary"
            (click)="respondToBId(bid._id, 'accepted', issue._id)"  onclick="dis"
value="accept">Accept
      </button>
      <button [disabled]="bid.status === 'reject' || bid.closed === true" mat-raised-button
color="warn"
            (click)="respondToBId(bid._id, 'reject', issue._id)">Reject
      </button>
      <hr>
          // Το όνομα του κουμπιού αλλάζει δυναμικά, ανάλογα με τα responses του server
      <mat-chip-list *ngIf="bid.confirmed === 'true' || bid.closed == true">
       <mat-chip color="accent" selected>{{bid.closed ? 'Issue closed' : 'Issue confirmed'}}
</mat-chip>
      </mat-chip-list>
 // Το κουμπί add review εμφανίζεται μόνο όταν το issue.closed είναι true.
      <button mat-raised-button color="primary" *ngIf="issue.closed ===
true" (click)="onReview(bid.creator)" style="margin-top: 1em">Add review</button>

    </mat-action-list>
    <mat-action-row>
      <button mat-raised-button color="primary" [routerLink]="['/edit/', issue._id]">Edit</
button>
      <button mat-raised-button color="warn" (click)="onDelete(issue._id)">Delete</
button>
    </mat-action-row>
```

```
  </mat-expansion-panel>
 </mat-accordion>
\
```

4.1.9. ClosedIssueListComponent

closed-issue-list.component.ts

Το component αυτό υλοποιείται με παρόμοιο τρόπο με το IssueListComponent, μόνο που τώρα το response του server αφορά τα issues που έχουν κλείσει.

```
import { Component, OnInit } from '@angular/core';
import {IssueModel} from "../../../models/Issue.model";
import {UserRegisterModel} from "../../../models/userRegister.model";
import {Subscription} from "rxjs";
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {ReviewModel} from "../../../models/reviewModel";
import {IssueService} from "../../../services/issues/issue.service";
import {AuthService} from "../../../services/auth/auth.service";
import {Router} from "@angular/router";
import {MatDialog, MatDialogConfig, MatSnackBar} from "@angular/material";
import {ReviewComponent} from "../../other/review/review.component";

@Component({
 selector: 'app-closed-issues',
 templateUrl: './closed-issues.component.html',
 styleUrls: ['./closed-issues.component.css']
})
export class ClosedIssuesComponent implements OnInit {

 userId: '';
 issues: IssueModel[] = [];
 data: UserRegisterModel[] = [];
 private issuesSub: Subscription;
 // isClicked = true;
 addReviewForm: FormGroup;
 reviewData: ReviewModel = new ReviewModel();

 constructor(private _issue: IssueService,
             private _auth: AuthService,
             private router: Router,
             private formBuilder: FormBuilder,
             private snack: MatSnackBar,
             private dialog: MatDialog
 ) {
 }
```

```
ngOnInit() {
  this.addReviewForm = this.formBuilder.group({
    'description': [this.reviewData.description, [
      Validators.required
    ]],
    'stars': [this.reviewData.stars, [
      Validators.required
    ]]
  });
  this._auth.getProfile();
  this._auth.getProfileDataListener()
    .subscribe(
      (data) => {
       this.data = data;
       this.userId = data._id;
      }
    );
  this._issue.getClosedUserIssues();
  this._issue.getClosedIssuesUpdated().subscribe((res)=>{
    this.issues = res.issue;
  });
}


onDelete(issueId: String) {
  this._issue.deleteIssue(issueId);
  this._issue.getIssueByIssueId(this.userId);
}


respondToBId(bidId, response, issueId, bid) {
   this._issue.respondToBid(bidId, response, issueId, bid);
  this._issue.getIssues();
  this._issue.getIssueUpdateListener().subscribe((issues: IssueModel[]) => {
    this.issues = issues;
    this.snack.open('Η απάντηση σας καταχωρήθηκε', "OK", {
      duration: 2000
    });
  });
}

addReview(bidCreator) {
  this._issue.addReview(this.reviewData, bidCreator);
}
```

//Με το που γίνει trigger η μέθοδος onReview με το click event, ενεργοποιείται ένα αναδυόμενο παράθυρο το οποίο αντιστοιχεί στον ReviewComponent

```
 onReview(creator) {
  const dialogConfig = new MatDialogConfig();
   dialogConfig.disableClose = false;
   dialogConfig.autoFocus = true;
   dialogConfig.width = "60%";
   dialogConfig.data = creator;
   this.dialog.open(ReviewComponent, dialogConfig);
   this.dialog.afterAllClosed.subscribe(creator => {
  })
  }
}
```

closed-issue-list.component.html

```
<mat-accordion>
 <mat-expansion-panel *ngFor="let issue of issues">
   <mat-expansion-panel-header>
     <b style="color: black">{{issue.title}}</b>
   </mat-expansion-panel-header>
   <p><b>Περιγραφή</b>:  {{ issue.description }}</p>
   <p><b>Creator:  {{ issue.creator }}</b></p>
   <p><b>ID:  {{ issue._id }}</b></p>
   <p><b>Εύρος τιμής</b>:  {{ issue.price }}</p>
   <p><b>Πόλη</b>:  {{ issue.address.city }}</p>
   <p><b>Διεύθυνση</b>:  {{ issue.address.homeAddress }}</p>
   <p><b>Τηλέφωνο</b>:  {{ issue.phone }}</p>
   <p><b>Ημ/νία καταχώρησης: </b>: {{ issue.createdAt | date : 'medium'}}</p>
   <hr>

   <h3 style="text-align: center">{{issue.bids.length === 0? 'Δεν υπάρχουν προσφορές':
'Προσφορές επαγγελματιών'}}</h3>
   <mat-action-list *ngFor="let bid of issue.bids" style="border: #030303">
     <p><b>Ποσο: </b>{{bid.amount}}</p>
     <p><b>Ημ/νία Καταχώρησης: </b>{{bid.createdAt | date : 'medium'}}</p>
     <p><b>ID Επαγγελματία: </b>{{bid.creator}}</p>
     <button [disabled]="bid.status === 'accepted'" mat-raised-button color="primary"
          (click)="respondToBId(bid._id, 'accepted', issue._id)"  onclick="dis"
value="accept">Accept
     </button>
     <button [disabled]="bid.status === 'reject' || bid.closed === true" mat-raised-button
color="warn"
          (click)="respondToBId(bid._id, 'reject', issue._id)">Reject
     </button>
     <hr>
     <mat-chip-list *ngIf="bid.confirmed === 'true' || bid.closed == true">
       <mat-chip color="accent" selected>{{bid.closed ? 'Issue closed' : 'Issue confirmed'}}
</mat-chip>
```

```
    </mat-chip-list>

    <button mat-raised-button color="primary" *ngIf="bid.closed ===
true" (click)="onReview(bid.creator)" style="margin-top: 1em">Add review</button>

   </mat-action-list>
   <mat-action-row>
    <button mat-raised-button color="primary" [routerLink]="['/edit/', issue._id]">Edit</
button>
    <button mat-raised-button color="warn" (click)="onDelete(issue._id)">Delete</
button>
   </mat-action-row>
  </mat-expansion-panel>
</mat-accordion>
```

4.1.10. ProfileComponent

profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import {BrowseService} from "../../../services/browse/browse.service";
import {AuthService} from "../../../services/auth/auth.service";
import {UserRegisterModel} from "../../../models/userRegister.model";
import {HttpClient} from "@angular/common/http";
import {IssueService} from "../../../services/issues/issue.service";

@Component({
 selector: 'app-profile',
 templateUrl: './profile.component.html',
 styleUrls: ['./profile.component.css']
})
export class ProfileComponent implements OnInit {

 data: UserRegisterModel[] = [];
 constructor(private _browse: BrowseService,
          private _issue: IssueService,
          private _auth: AuthService,
          private http: HttpClient) { }

 ngOnInit() {
 // Ανάκτηση δεδομένων χρήστη από τον server.
  this._auth.getProfile();
  this._auth.getProfileDataListener()
   .subscribe(
```

```
      (data) => {
        this.data = data;
      }
    )
  }



}
```

profile.component.html

```
<h2 style="text-align:center">User Profile Card</h2>
// Τα στοιχεία εμφανίζονται στην κάρτα προφίλ με την μέθοδο του string interpolation
<div class="card">
  <img src="../../../../assets/Hipster-Developer-Dice.jpg" alt="John" style="width:100%">
  <h1>{{data.name}}</h1>
  <p class="title">CEO & Founder, Example</p>
  <p>ID: {{data._id}}</p>
  <div style="margin: 24px 0;">
    <a href="#"><i class="fa fa-twitter"></i></a>
    <a href="#"><i class="fa fa-linkedin"></i></a>
    <a href="#"><i class="fa fa-facebook"></i></a>
  </div>
</div>

//Δημιουργείται ένα mat-tab-group όπου στον καθένα από αυτά βάζουμε ένα component
<mat-tab-group>
  <mat-tab label="Εκκρεμείς αιτήσεις">
    <app-issue-list></app-issue-list>
  </mat-tab>
  <mat-tab label="Ολοκληρωμένες αιτήσεις">
    <app-closed-issues></app-closed-issues>
  </mat-tab>
  <mat-tab label="Κριτικές">
    <app-review-list></app-review-list>
  </mat-tab>

</mat-tab-group>
```

4.1.11. ProfDashboardComponent

prof-dashboard.component.ts

```
import {Component, OnInit} from '@angular/core';
```

```
import {BreakpointObserver} from '@angular/cdk/layout';
import {AuthService} from "../../../services/auth/auth.service";
import {ProRegisterModel} from "../../../models/proRegister.model";
import {IssueService} from "../../../services/issues/issue.service";
import {IssueModel} from "../../../models/Issue.model";
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {ActivatedRoute, ParamMap} from "@angular/router";
import {BrowseService} from "../../../services/browse/browse.service";
import {CitiesModel} from "../../../models/cities.model";
import {CategoriesModel} from "../../../models/categoriesModel";
import {ReviewModel} from "../../../models/reviewModel";

@Component({
  selector: 'app-prof-dashboard',
  templateUrl: './prof-dashboard.component.html',
  styleUrls: ['./prof-dashboard.component.css']
})
export class ProfDashboardComponent implements OnInit {

  profileData: ProRegisterModel[] = [];
  openIssues:[];
  closedIssues:[];
  issues:[];
  phone;
  bidForm: FormGroup;
  bid;
  cities;
  proCategories;
  private mode;
  private issueId: string;
  selectedCity: string = ' ';
  showBid: Boolean = false;
  reviews;
  private category: string;
  private city: any;
  profileIssues: any;


  constructor(private breakpointObserver: BreakpointObserver,
              private _auth: AuthService,
              private _issues: IssueService,
              private formBuilder: FormBuilder,
              private a_route: ActivatedRoute,
              private _browse: BrowseService) {
  }

  ngOnInit() {
```

```
// Παρακάτω ανακτώνται όλα τα στοιχεία που χρειάζονται για το προφίλ του επαγγελματία με τον ίδιο τρόπο που περιγράφηκε παραπάνω.
    this._auth.getProfProfile();
    this._auth.getProfProfileDataListener().subscribe(
      (data: ProRegisterModel[]) => {
        this.profileData = data;
        this._issues.getIssuesForPro(this.profileData[0].p_locationCoverage,
this.profileData[0].p_category);
        this._issues.getIssuesForProfile().subscribe((res)=>{
          this.profileIssues = res;
        })
      }
    );
    this._browse.getCities();
    this._browse.getCitiesUpdated().subscribe((res)=>{
      this.cities = res;
    });


    this._issues.getIssues();
    this._issues.getIssueUpdateListener().subscribe((res)=>{
     this.issues = res;
    });

    this._issues.getOpenProIssues();
    this._issues.getOpenIssuesUpdated().subscribe((res)=>{
     this.openIssues = res;
    });

    this._issues.getClosedProIssues();
    this._issues.getClosedProIssuesUpdated().subscribe((res)=> {
     this.closedIssues = res;
    });

    this._browse.getReviews();
    this._browse.getReviewsUpdated().subscribe((review: ReviewModel[]) => {
        this.reviews = review;
      }
    );

//Όταν ο επαγγελματίας αποφασίσει να κάνει μια προσφορά εφαρμόζουμε την ίδια μέθοδο με την edit issue που κάνει ο χρήστης, αφού χρησιμοποιείται το update στη βάση δεδομένων
    this.a_route.paramMap.subscribe((paramMap: ParamMap) => {
      if (paramMap.has('issueId')) {
        this.mode = 'edit';
        this.issueId = paramMap.get('issueId');
```

```
      this.issue = this._issues.getEditedIssue(this.issueId);
    }
   });

   if (this.mode === 'edit') {
    this._issues.getIssueByIssueId(this.issueId);
     this._issues.getIssueToBeUpdatedistener().subscribe(
      (resIssue) => { }
     ); }

// Η φόρμα που ελέγχει την είσοδο του bid
   this.bidForm = this.formBuilder.group({
    'bid': [this.bid, [
      Validators.required
    ]]  }) }

// Η μέθοδος που στέλνει μέσω του _issues service, το id του issue και το ποσό του bid,
στον server ώστε να καταχωρηθούν στη βάση. Αυτή η μέθοδος γίνεται triggered από το
click event του bid button
makeBid(issueId) {
  this._issues.makeBid(issueId, this.bid);
  this._issues.getOpenProIssues();
  this._auth.getProfProfile();
}

// Η μέθοδος που στέλνει μέσω του _issues service, το id του issue και το ποσό και το id
του bid και το boolean true ώστε να αποθηκευθεί στο bid.confrmed, στον server ώστε να
καταχωρηθούν στη βάση. Αυτή η μέθοδος γίνεται triggered από το click event του confirm
button
confirmIssue(bidId, confirm, issueId, bid) {
  this._issues.confirmIssue(bidId, confirm, issueId, bid);
  this._issues.getOpenProIssues();
  this._auth.getProfProfile();
}

// Ομοίως στέλνει τα κατάλληλα δεδομένα στον server για να κλείσει ένα issue
closeIssue(bidId, close, issueId, bid) {
  this._issues.closeIssue(bidId, close, issueId, bid);
  this._issues.getOpenProIssues();
  this._issues.getClosedProIssues();
  this._auth.getProfProfile();
}
// Κάνει toggle το κουμπί που εμφανίζει τις προσφορές
  showBids() {
   this.showBid = !this.showBid;
  }}
```

prof-dashboard.component.html

```
// Ανακτώνται τα στοιχεία του επαγγελματία με μια επανάληψη (*ngFor)
<div class="grid-container" *ngFor="let prof of profileData">
  <h2 style="text-align:center">Professional Profile Card</h2>

  <div class="card">
    <img src="../../../../assets/plumber_cartoon.png" alt="John" style="width:100%">
    <h1>{{prof.p_name}}</h1>
    <p class="title">{{prof.p_brandName}}</p>
    <p>ID: {{prof._id}}</p>
    <div style="margin: 24px 0;">
      <a href="#"><i class="fa fa-twitter"></i></a>
      <a href="#"><i class="fa fa-linkedin"></i></a>
      <a href="#"><i class="fa fa-facebook"></i></a>
    </div>
  </div>

// Αυτό το tab group χωρίζει τις αιτήσεις σε ενεργές (δεν έχουν κλείσει), ανενεργές (έχουν κλείσει), σε προτεινόμενες για τον επαγγελματία και στις κριτικές.
  <mat-tab-group>
    <mat-tab label="Ενεργές αιτήσεις">
      <mat-accordion>
// Με την παρακάτω επανάληψη εμφανίζονται τα ανοιχτά issues του  επαγγελματία.
        <mat-expansion-panel *ngFor="let issue of openIssues">
          <mat-expansion-panel-header>
            <b style="color: blue">Τίτλος</b>:  {{issue.description}}
          </mat-expansion-panel-header>
          <p><b>Περιγραφή</b>:  {{ issue.description }}</p>
          <p><b>Creator:  {{ issue.creator }}</b></p>
          <p><b>ID:  {{ issue._id }}</b></p>
          <p><b>Εύρος τιμής</b>:  {{ issue.price }}</p>
          <p><b>Πόλη</b>:  {{ issue.address.city }}</p>
          <p><b>Διεύθυνση</b>:  {{ issue.address.homeAddress }}</p>
          <p><b>Τηλέφωνο</b>:  {{ issue.phone }}</p>
          <p><b>Ημ/νία καταχώρησης: </b>: {{ issue.createdAt | date : 'medium'}}</p>

// Το παρακάτω κουμπί κάνει trigger την showBids() μέθοδο και αλλάζει τίτλο δυναμικά, ανάλογα με το αν υπάρχουν issues.
          <button mat-raised-button [disabled]="issue.bids.length  === 0"
                (click)="showBids()">{{issue.bids.length === 0 ? 'Καμία προσφορά' :
'Προσφορές'}}</button>
// Το παρακάτω κομμάτι κώδικα εμφανίζεται μόνο αν το showBid === true και εμφανίζει τα bids.
          <div *ngIf="showBid == true">
            <mat-action-list *ngFor="let bid of issue.bids" style="border: #030303">
              <p><b style="color: blue">Ποσο: </b>{{bid.amount}}</p>
```

```
          <p><b style="color: blue">Ημ/νία Καταχώρησης: </b>{{bid.createdAt | date : ‘medium'}}</p>
// Με τον *ngIf έλεγχο, εμφανίζεται ένα από τα τρία παρακάτω μηνύματα και εικονίδια ανάλογα με την απάντηση του χρήστη.
          <p><b style="color: blue">ID Επαγγελματία: </b>{{bid.creator}}</p>
          <div *ngIf="bid.status === 'accepted'">
            <h3 style="color: fuchsia">Accepted</h3>
            <i class="material-icons">
              sentiment_satisfied_alt
            </i>
          </div>
          <div *ngIf="bid.status === 'reject'">
            <h3 style="color: #BD362F">Rejected</h3>
            <i class="material-icons">
              sentiment_very_dissatisfied
            </i>
          </div>
          <div *ngIf="bid.status == null">
            <h3 style="color: #51A351">Waiting</h3>
            <i class="material-icons">
              sentiment_dissatisfied
            </i>
          </div>

          <button
            mat-raised-button
            color="primary"
            type="submit"
            (click)="confirmIssue(bid._id, true, issue._id)"
            [disabled]="bid.status == null || bid.status === 'reject' || bid.confirmed === true"
          >{{bid.confirmed === true ? 'Confirmed' : 'Confirm'}}
          </button>

          <button
            *ngIf="bid.confirmed === true"
            mat-raised-button
            color="primary"
            type="submit"
            (click)="closeIssue(bid._id, true, issue._id)"
            [disabled]="issue.closed == true"
          >{{issue.closed === true ? 'Closed' : 'Close'}}
          </button>
        </mat-action-list>
      </div>
      <hr>
    </mat-expansion-panel>
  </mat-accordion>
```

```
  </mat-tab>
// Σε αυτό το tab εμφανίζονται οι ανενεργές αιτήσεις
  <mat-tab label="Ανενεργές αιτήσεις">
    <mat-accordion>
      <mat-expansion-panel *ngFor="let issue of closedIssues">
        <mat-expansion-panel-header>
          <b style="color: blue">Τίτλος</b>:  {{issue.description}}
        </mat-expansion-panel-header>
        <p><b>Περιγραφή</b>:  {{ issue.description }}</p>
        <p><b>Creator:  {{ issue.creator }}</b></p>
        <p><b>ID:  {{ issue._id }}</b></p>
        <p><b>Εύρος τιμής</b>:  {{ issue.price }}</p>
        <p><b>Πόλη</b>:  {{ issue.address.city }}</p>
        <p><b>Διεύθυνση</b>:  {{ issue.address.homeAddress }}</p>
        <p><b>Τηλέφωνο</b>:  {{ issue.phone }}</p>
        <p><b>Ημ/νία καταχώρησης: </b>: {{ issue.createdAt | date : 'medium'}}</p>
        <hr>
      </mat-expansion-panel>
    </mat-accordion>
  </mat-tab>

// Σε αυτό το tab εμφανίζονται οι προτεινόμενες αιτήσεις για τον επαγγελματία βάσει της πόλης και της κατηγορίας του.
  <mat-tab label="Αιτήσεις" (click)="getProfileIssues()">
    <br>
    <mat-accordion>
      <mat-expansion-panel *ngFor="let issue of profileIssues">
        <mat-expansion-panel-header>
          <b style="color: blue">Τίτλος</b>:  {{issue.title}}
        </mat-expansion-panel-header>
        <p><b style="color: blue">Πόλη</b>:  {{ issue.address.city}}</p>

        <p><b style="color: blue">Περιγραφή</b>:  {{ issue.description }}</p>
        <p><b style="color: blue">ID:  {{ issue._id }}</b></p>
        <p><b style="color: blue">Εύρος τιμής</b>:  {{ issue.price }}</p>
        <p><b style="color: blue">Διεύθυνση</b>:  {{ issue.address.homeAddress }}</p>
        <p><b style="color: blue">Τηλέφωνο</b>:  {{ issue.phone }}</p>
        <p><b>Created at: </b>: {{ issue.createdAt | date : 'medium'}}</p>

        <button mat-raised-button [disabled]="issue.bids.length  ===
0" (click)="showBids()">{{issue.bids.length === 0 ? 'Καμία προσφορά' : 'Προσφορές'}}</
button>
        <div *ngIf="showBid == true">
          <mat-action-list *ngFor="let bid of issue.bids" style="border: #030303">
            <p><b style="color: blue">Ποσο: </b>{{bid.amount}}</p>
            <p><b style="color: blue">Ημ/νία Καταχώρησης: </b>{{bid.createdAt | date :
'medium'}}</p>
```

```
          <p><b style="color: blue">ID Επαγγελματία: </b>{{bid.creator}}</p>
          <div *ngIf="bid.status === 'accepted'">
            <h3 style="color: fuchsia">Accepted</h3>
            <i class="material-icons">
              sentiment_satisfied_alt
            </i>
          </div>
          <div *ngIf="bid.status === 'reject'">
            <h3 style="color: #BD362F">Rejected</h3>
            <i class="material-icons">
              sentiment_very_dissatisfied
            </i>
          </div>
          <div *ngIf="bid.status == null">
            <h3 style="color: #51A351">Waiting</h3>
            <i class="material-icons">
              sentiment_dissatisfied
            </i>
          </div>
        // Εδώ ο επαγγελματίας καταχωρεί προσφορά
          <button
            mat-raised-button
            color="primary"
            type="submit"
            (click)="confirmIssue(bid._id, true, issue._id)"
            [disabled]="bid.status == null || bid.status === 'reject' || bid.confirmed === true"
          >{{bid.confirmed === true ? 'Confirmed' : 'Confirm'}}
          </button>
          <button
            *ngIf="bid.confirmed === true"
            mat-raised-button
            color="primary"
            type="submit"
            (click)="closeIssue(bid._id, true, issue._id)"
            [disabled]="bid.closed == true"
          >{{bid.closed === true ? 'Closed' : 'Close'}}
          </button>
        </mat-action-list>
      </div>
      <mat-action-row>
// Σε αυτό το tab εμφανίζονται οι ανενεργές αιτήσεις
        <form
          [formGroup]="bidForm"
          (ngSubmit)="bidForm.valid && makeBid()"
          novalidate
        >
          <label>
```

```
            <input
              style="background-color: antiquewhite"
              matInput
              [(ngModel)]="bid"
              formControlName="bid"
            >
          </label>
          <p>{{bid}}</p>
          <button
            mat-raised-button
            color="primary"
            type="submit"
            [routerLink]="['/bid', issue._id]">Bid</button>
        </form>
      </mat-action-row>
    </mat-expansion-panel>
  </mat-accordion>
</mat-tab>

// // Σε αυτό το tab εμφανίζονται οι κριτικές του επαγγελματία
  <mat-tab label="Κριτικές">
    <mat-accordion>
      <mat-expansion-panel *ngFor="let rev of reviews">
        <mat-expansion-panel-header>
          <b style="color: blue">Τίτλος</b>:  {{rev.description}}
        </mat-expansion-panel-header>
        <b style="color: blue">Βαθμολογία:</b> {{rev.stars}}/5 <br>
        <b style="color: blue">Ημερομηνία Καταχώρησης:</b> {{ rev.created_at | date :
'medium'}}<br>
        <b style="color: blue">Χρήστης</b> {{ rev.user}}

      </mat-expansion-panel>
    </mat-accordion>
  </mat-tab>

 </mat-tab-group>
</div>
```

4.1.12. ReviewComponent

review.component.ts

```
import {Component, Inject, inject, OnInit} from '@angular/core';
import {FormBuilder, FormControl, FormGroup, Validators} from "@angular/forms";
import {ReviewModel} from "../../../models/reviewModel";
import {IssueService} from "../../../services/issues/issue.service";
import {MAT_DIALOG_DATA} from "@angular/material";

@Component({
  selector: 'app-review',
  templateUrl: './review.component.html',
  styleUrls: ['./review.component.css']
})
export class ReviewComponent implements OnInit {

  reviewForm: FormGroup;
  reviewData: ReviewModel = new ReviewModel();

  reviews = [
    {value: 1, viewValue: "Κακός"},
    {value: 2, viewValue: "Μέτριος"},
    {value: 3, viewValue: "Καλός"},
    {value: 4, viewValue: "Πολύ καλός"},
    {value: 5, viewValue: "Εξαιρετικός"},
  ];

// Γίνεται inject το MAT_DIALOG_DATA ώστε να ανακτηθεί η τιμή που έχει σταλεί από το configuration του MatDialog
  constructor(@Inject(MAT_DIALOG_DATA) public data: any,
              private formBuilder: FormBuilder,
              private _issues: IssueService) {
  }

//Αρχικοποίηση φόρμας review
  ngOnInit() {
    this.reviewForm = this.formBuilder.group({
      'description': [new FormControl(this.reviewData.description)],
      'stars': [new FormControl(this.reviewData.stars), [Validators.required]],
    })
  }

//Μέθοδος που στέλνει τα δεδομένα στον server μέσω του service
  onReview() {
    this._issues.addReview(this.reviewData, this.data);
  }
```

```
}
```

review.component.html

// Η φόρμα του review εκτελείται με τον ίδιο τρόπο όπως όλες οι φόρμες του project που εξηγήθηκαν παραπάνω.

```
<mat-card>
  <mat-card-title>
    Γράψτε την κριτική σας
  </mat-card-title>
  <form [formGroup]="reviewForm" (ngSubmit)="reviewForm.valid && onReview() "
novalidate>

    <mat-form-field style="width: 100%">
      <label> Αξιολογήστε τον επαγγελματία
        <mat-select
          matNativeControl
          [(ngModel)] = "reviewData.stars"
          formControlName="stars">
          <mat-option *ngFor="let rev of reviews" [value]="rev.value"> {{rev.viewValue}}</
mat-option>
        </mat-select>
      </label>
    </mat-form-field>

    <mat-form-field style="width: 100%">
      <input
        matInput
        formControlName="description"
        placeholder="Περιγράψτε την εμπειρία σας"
        [(ngModel)]="reviewData.description">
    </mat-form-field>

    <button
      mat-raised-button
      color="accent"
      type="submit"
      [disabled]="!reviewForm.valid"> Υποβολή κριτικής
    </button>
  </form>
</mat-card>
```

4.1.13 ReviewListComponent

review-list.component.ts

```
import {Component, OnInit} from '@angular/core';
import {ReviewModel} from "../../../../models/reviewModel";
import {ReviewService} from "../../../../services/reviews/review.service";

@Component({
  selector: 'app-review-list',
  templateUrl: './review-list.component.html',
  styleUrls: ['./review-list.component.css']
})
export class ReviewListComponent implements OnInit {

  reviews;

  constructor(private _review: ReviewService) {
  }
// Ανακτούμε τις κριτικές του χρήστη και του επαγγελματία
  ngOnInit() {
    this._review.getProReviews();
    this._review.getProReviewsUpdated().subscribe((review: ReviewModel[]) => {
        this.reviews = review;
      }
    );
    this._review.getUserReviews();
    this._review.getUserReviewsUpdated().subscribe((review: ReviewModel[]) => {
        this.reviews = review;
      }
    );  }
}
```

review-list.component.html

```
<mat-accordion>
  <mat-expansion-panel *ngFor="let rev of reviews">
    <mat-expansion-panel-header>
      <b style="color: blue">Τίτλος</b>:  {{rev.description}}
    </mat-expansion-panel-header>
    <b style="color: blue">Βαθμολογία:</b> {{rev.stars}}/5 <br>
```

```
  <b style="color: blue">Ημερομηνία Καταχώρησης:</b> {{ rev.created_at | date :
'medium'}}<br>
  <b style="color: blue">Χρήστης</b> {{ rev.user}}

 </mat-expansion-panel>
</mat-accordion>
```

4.1.14. ErrorComponent

error.component.ts

Αυτός ο component είναι, απλά, ένα MatDialog το οποίο αναδύεται μόλις στείλει response ο server ότι προέκυψε λάθος. Το message στέλνεται από τον TokenInteceptor μέσω του config.data του dialog, όπως νωρίτερα με το **review**

```
import {Component, Inject, OnInit} from '@angular/core';
import {MAT_DIALOG_DATA} from "@angular/material";

@Component({
 selector: 'app-error',
 templateUrl: './error.component.html',
 styleUrls: ['./error.component.css']
})
export class ErrorComponent implements OnInit {

 message = 'An unknown error occured';

 constructor(@Inject(MAT_DIALOG_DATA)
          public data:
            { message: string }) {
 }

 ngOnInit() {
 }
```

error.component.html

```
<h1 mat-dialog-title>Ουπς! Κάτι πήγε στραβά!</h1>
<div mat-dialog-content>
 <p class="mat-body-1">{{ data.message}}</p></div>
<div mat-dialog-actions>
 <button mat-button mat-dialog-close>Okay</button>
</div>
```

error-interceptor.service.ts

Μέσω αυτού του interceptor όλα τα http requests φιλτράρονται και γίνεται έλεγχος αν υπήρχε κάποιο λάθος τύπου HttpError όπως φαίνεται παρακάτω. Αν υπάρχει τότε εμφανίζεται το MatDialog παράθυρο το οποίο έχει ως κείμενο το μήνυμα που στέλνει ο server σε περίπτωση σφάλματος.

```
import {HttpErrorResponse, HttpInterceptor} from "@angular/common/http";
import {catchError} from "rxjs/operators";
import {throwError} from "rxjs";
import {Injectable} from "@angular/core";
import {MatDialog} from "@angular/material";
import {ErrorComponent} from "../../error/error.component";

@Injectable({
  providedIn: 'root'
})
export class ErrorInterceptor implements HttpInterceptor {

  constructor(private dialog: MatDialog) {
  }

  intercept(req, next) {

    return next.handle(req).pipe(
      catchError((error: HttpErrorResponse) => {
        let errorMessage = 'Unknown error occured';
        if (error.error.message){
          errorMessage = error.error.message
        }
        this.dialog.open(ErrorComponent, {data: {message: errorMessage}});
        return throwError(error);
      })
    )
  }
}
```

token-interceptor.service.ts

```
import {Injectable, Injector} from '@angular/core';
import {HttpInterceptor} from "@angular/common/http";
import {AuthService} from "./auth.service";

@Injectable({
  providedIn: 'root'
```

```
})
export class TokenInterceptorService implements HttpInterceptor {

 constructor(private _auth: AuthService) {
 }

 intercept(req, next) {
// πραγματοποιείται clone του request και θέτουμε τα headers που χρειάζονται
   let tokenizeReq = req.clone({
     setHeaders: {
//Η τιμή των headers θα είναι Bearer ακολουθούμενη από το token που είναι
αποθηκευμένο στο localStorage του browser.
       Authorization: `Bearer ${this._auth.getToken()}`
     }
   });
  return next.handle(tokenizeReq)
 }
}
```

auth.guard.ts

Είναι ένα interface που χρησιμοποιείται για να περιορίσει την πρόσβαση σε συγκεκριμένα routes.

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import {AuthService} from "./auth.service";

@Injectable({
 providedIn: 'root'
})

export class AuthGuard implements CanActivate {

 constructor(private _auth: AuthService,
          private router: Router){}
```

//Ελέγχει μέσω τις μεθόδου isAuthenticated από το AuthService αν ο χρήστης είναι συνδεδεμένος και επιστρέφει true ή false. Αν είναι false, ο χρήστης ανακατωθείτε στο login. Για το συγκεκριμένο project είναι ένα επίπεδο ασφαλείας επιπλέον, αφού τα routes που η πρόσβαση τους περιορίζεται απο τον guard, δεν εμφανίζονται στον client, αφού κρύβονται δυναμικά. Παρόλα αυτά είναι προσβάσιμα, αν ο χρήστης πληκτρολογήσει το μονοπάτι του component στον browser

```
 canActivate():boolean{
   if (this._auth.isAuthenticated()){
```

```
    return true
   } else {
    this.router.navigate(['/login']);
    return false
   }
  }
}
```

4.2. Services

auth.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient, HttpErrorResponse} from "@angular/common/http";
import {UserLoginModel} from "../../models/userLogin.model";
import {ProRegisterModel} from "../../models/proRegister.model";
import {UserRegisterModel} from "../../models/userRegister.model";
import {Router} from "@angular/router";
import {environment} from "../../../environments/environment";
import {Subject} from "rxjs";
import {MatSnackBar} from "@angular/material";
import {shareReplay} from "rxjs/operators";

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

 API_URL = environment.apiUrl;

  private userRole;

  private $userRole = new Subject();
  private $profileData = new Subject<UserRegisterModel[]>();
  private $profProfileData = new Subject<ProRegisterModel[]>();

  constructor(private http: HttpClient,
              private  router: Router,
              private snack: MatSnackBar) {
  }

 // Μέθοδος εγγραφής χρήστη, η οποία έχει ως όρισμα τα δεδομένα εγγραφής του χρήστη.
Πραγματοποιείται ένα post request στον server ο οποίος με τη σειρά του επιστρέφει ένα
observable. Ορίζουμε πως το response θα περιέχει τον registeredUser τύπου
RegisterModel και ένα message τύπου string. Η post request παίρνει ως ορίσματα τη
```

διεύθυνση του server που θα γίνει το request, όπως και τα δεδομένα που θα αποθηκεύσει. Εφόσον, η εγγραφή ολοκληρωθεί με επιτυχία, εμφανίζεται ένα MatSnackBar με το μήνυμα που φαίνεται παρακάτω και ο χρήστης ανακατευθύνεται στη φόρμα σύνδεσης (login)

```
 registerUser(registrationData) {
   return this.http.post<{registeredUser: UserRegisterModel[], message: string}
>(this.API_URL + 'user/u_register', registrationData).subscribe(
     (__registrationData) => {
       this.snack.open('Εγγραφήκατε επιτυχώς. Μπορείτε να συνδεθείτε.', "OK", {
         duration: 5000
       });
      this.router.navigate(['/login']);
     },
     error => {
       console.log(error)
     }
   )
 }

//Μέθοδος για την εγγραφή επαγγελματία με τον ίδιο τρόπο όπως του χρήστη, παραπάνω.
 registerPro(registrationData) {
   return this.http.post<{registeredPro: ProRegisterModel[], message: string}
>(this.API_URL + 'user/p_register', registrationData).subscribe(
     (__registrationData) => {
       this.snack.open('Εγγραφήκατε επιτυχώς. Μπορείτε να συνδεθείτε.', "OK", {
         duration: 5000
       });
      this.router.navigate(['/login']);
     }
   )
 }
```

//Μέθοδος για την σύνδεση χρήστη με τον ίδιο τρόπο όπως του χρήστη, παραπάνω. Ξανά, παρόμοια μέθοδος με την εγγραφή χρήστη/επαγγελματία, αλλά αυτή τη φορά ο server επιστρέφει και το token το οποίο αποθηκεύεται στο localStorage του browser. Τώρα ο χρήστης ανακατευθύνεται στο προφίλ του.

```
 loginUser(loginData) {
   return this.http.post<{token: string, fetchedUser: UserLoginModel[], message: string}
>(this.API_URL + 'user/u_login', loginData)
     .subscribe(
       (__loginData) => {
         localStorage.setItem('token', __loginData.token);
         this.router.navigate(['/profile']);
         this.snack.open('Συνδεθήκατε επιτυχώς.', "OK", {
           duration: 2000
         });
```

```
      })
  }
//Μέθοδος για την σύνδεση επαγγελματία με τον ίδιο τρόπο όπως του χρήστη, παραπάνω.
  loginPro(loginData) {
    return this.http.post<{token: string, fetchedPro: ProRegisterModel[], message: string}
>(this.API_URL + 'user/p_login', loginData)
      .subscribe(
        (__loginData) => {
          console.log(__loginData);
          localStorage.setItem('token', __loginData.token);
          this.router.navigate(['/profDashboard']);
          this.snack.open('Συνδεθήκατε επιτυχώς.', "OK", {
            duration: 2000
          });
        })
  }

// Ανάκτηση δεδομένων χρήστη. Πραγματοποιείται μια get request στον server, ο οποίος επιστρέφει ένα response με τα στοιχεία του χρήστη τα οποία τα αποθηκεύουμε σε ένα Subject.
  getProfile(){
    return this.http.get<{message: string, fetchedUser: UserRegisterModel[]}
>(this.API_URL + 'user/profile').subscribe(
      (res)=> {
        this.$profileData.next(res.fetchedUser);
      },
      err => {
        if (err instanceof HttpErrorResponse) {
          if (err.status === 401) {
            this.router.navigate(['/login'])
          }
        }
      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getProfileDataListener() {
    return this.$profileData.asObservable();
  }

//Ανάκτηση δεδομένων επαγγελματία με τον ίδιο τρόπο που έγινε και για τον χρήστη, παραπάνω.
  getProfProfile(){
    return this.http.get<{fetchedPro: ProRegisterModel[], token: string, message: string}
>(this.API_URL + "pro/profile").subscribe(
      (res) => {
```

```
      this.$profProfileData.next(res.fetchedPro);
    }
  )
 }

 getProfProfileDataListener(){
  return this.$profProfileData.asObservable();
 }

// Διαγράφει τα δεδομένα που είναι αποθηκευμένα στο localStorage και ανακατεθύνει τον χρήστη στον home component
 logoutUser(){
  localStorage.removeItem('token');
  localStorage.removeItem('userId');
  this.router.navigate(['/']);
 }

// Ελέγχει αν είναι ο χρήστης συνδεδεμένος. Επιστρέφει boolean και τσεκάρει αν υπάρχει token στο localstorage
 isAuthenticated(){
  return !!localStorage.getItem('token');
 }

// Επιστρέφει το token που είναι αποθηκευμένο στο localStorage
 getToken(){
  return localStorage.getItem('token');
 }


 isUser(){
  return this.$userRole.asObservable();
 }


}

// To URL template που είναι αποθηκευμένο στο environment.ts

export const environment = {
 production: false,

 apiUrl: 'http://localhost:3000/api/',

};
```

issues.service.ts

```
import {Injectable} from '@angular/core';
import {IssueModel} from "../../models/Issue.model";
import {Subject} from "rxjs";
import {HttpClient} from "@angular/common/http";
import {environment} from "../../../environments/environment";
import {Router} from "@angular/router";
import {MatSnackBar} from "@angular/material";
import {ReviewModel} from "../../models/reviewModel";
import {ProRegisterModel} from "../../models/proRegister.model";
import { catchError, map, tap } from 'rxjs/operators';
@Injectable({
  providedIn: 'root'
})
export class IssueService {
  API_URL = environment.apiUrl;

  private issues: IssueModel[] = [];
  private issueToUpdate: IssueModel[] = [];
  private issuesByCity;
  private closedIssues;
  private openIssues;
  private proIssues;

  private $issuesToUpdate = new Subject<IssueModel[]>();
  private $issuesByCity = new Subject();
  private $closedIssues = new Subject();
  private $closedProIssues = new Subject();
  private $openIssues = new Subject();
  private issuesUpdated = new Subject<IssueModel[]>();
  private $profileIssues = new Subject<IssueModel[]>();


  constructor(private http: HttpClient,
              private router: Router,
              private snack: MatSnackBar) {
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues που υπάρχει στη βάση και αποθηκεύεται σε ένα subject.
  getIssues() {
    this.http.get<{ issues: IssueModel[], message: String }>(this.API_URL +
'issue').subscribe(
      (res) => {
        this.issues = res.issues;
        this.issuesUpdated.next(this.issues);
```

```
    }
  )
 }

// Επιστρέφει ένα observable με τη χρήση του subject
 getIssueUpdateListener() {
  return this.issuesUpdated.asObservable();
 }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues του χρήστη που είναι ανοιχτά και αποθηκεύεται σε ένα subject.

 getUserIssues(){
  this.http.get<{message: String, issue: IssueModel[]}>(this.API_URL + 'issue/getUserIssues').subscribe(
    (res) => {
      this.issues = res.issue;
      this.issuesUpdated.next(this.issues);
    }
  )
 }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues του χρήστη που είναι κλειστά και αποθηκεύεται σε ένα subject.
 getClosedUserIssues(){
  this.http.get<{message: String, issue: IssueModel[]}>(this.API_URL + 'issue/getClosedUserIssues').subscribe(
    (res)=>{
      this.closedIssues = res;
      this.$closedIssues.next(this.closedIssues)
    }
  )
 }

// Επιστρέφει ένα observable με τη χρήση του subject
getClosedIssuesUpdated() {
  return this.$closedIssues.asObservable();
 }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues του επαγγελματία που είναι ανοικτά και αποθηκεύεται σε ένα subject.

 getOpenProIssues(){
  this.http.get<{message: string, issues: IssueModel}>(this.API_URL + 'issue/getOpenProIssues').subscribe(
    (issues) => {
      console.log(issues);
```

```
      this.openIssues = issues.issue;
       this.$openIssues.next(this.openIssues)
      }
    )
  }
// Επιστρέφει ένα observable με τη χρήση του subject
  getOpenIssuesUpdated() {
    return this.$openIssues.asObservable();
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues του επαγγελματία που είναι κλειστά και αποθηκεύεται σε ένα subject.

  getClosedProIssues() {
    this.http.get<{ message: string, issues: IssueModel }>(this.API_URL + 'issue/getClosedProIssues').subscribe(
      (issues) => {
        console.log(issues);
       this.issues = issues.issue;
        this.$closedProIssues.next(this.issues);
      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getClosedProIssuesUpdated() {
    return this.$closedProIssues.asObservable();
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getIssueToBeUpdatedistener() {
    return this.$issuesToUpdate.asObservable();
  }

//Μέθοδος αποθήκευσης ενός issue με τον ίδιο τρόπο που γίνεται το registration
  addIssues(issueDataAdded: any) {
    const issue: IssueModel = issueDataAdded;
    this.http.post<{ message: string, issueId: string }>(this.API_URL + 'issue/add', issue).subscribe(
      (__issueData) => {
        const id = __issueData.issueId;
        issue._id = id;
        this.issues.push(issue);
        this.issuesUpdated.next([...this.issues]);
        this.snack.open('Η αίτηση σας καταχωρήθηκε', "OK", {
          duration: 2000
        });
```

```
      this.router.navigate(['/profile']);
    }
  );
 }
```

//Μέθοδος διαγραφής ενός issue. Αφού ολοκληρωθεί το DELETE request στον server, αποθηκεύεται στην μεταβλητή updatedIssues, όλα τα issues εκτός από αυτό με το id που στάλθηκε στον server. Έπειτα, τα issues αυτά αποθηκεύονται σε ένα subject.

```
 deleteIssue(issueId: String) {
  this.http.delete(this.API_URL + 'issue/add' + issueId)
    .subscribe(() => {
      const updatedIssues = this.issues.filter(issue => issue._id !== issueId);
      this.issues = updatedIssues;
      this.snack.open('Η αίτηση σας διαγράφηκε', "OK", {
        duration: 2000
      });
      this.issuesUpdated.next([...this.issues]);
    })
 }
```

//Με τη μέθοδο αυτή, με τη χρήση του spread operator, επιστρέφεται ένα νέο object το οποίο αναζητά το id που ενημερώθηκε.

```
getEditedIssue(id: string) {
 return {...this.issues.find(is => is._id === id)}
}
```

// Ανάκτηση issue από το id του. Πραγματοποιείται μια GET request στον server για την ενημέρωση του issue με id που αποστέλλεται μέσω url στον server.

```
 getIssueByIssueId(issueId: string) {
  return this.http.get<any>(this.API_URL + '/issue/getById?id=' + issueId).subscribe((res)
=> {
    this.issueToUpdate = res.issue;
    this.$issuesToUpdate.next([...this.issueToUpdate]);
  })
 }
```

//Ενημέρωση ενός issue. Πραγματοποιείται μια PUT request στον server για την ενημέρωση του issue με id που αποστέλλεται μέσω url στον server.

```
 updateIssue(issueId, issueDataAdded: any) {
  const issue: IssueModel = issueDataAdded;
```

```
    this.http.put<{ result: IssueModel, message: string }>(this.API_URL + '/issue/update?
id=' + issueId, issue).subscribe(
      (res) => {
      this.issues = res.result;
        this.issuesUpdated.next(this.issues);
        this.router.navigate(['/profile']);
      }
    )
  }

//Απόστολή μια προσφοράς από τον επαγγελματία. Πραγματοποιείται μια POST request
στον server για την ενημέρωση του issue με id και ποσό που αποστέλλεται μέσω url στον
server.
  makeBid(issueId, bid: any) {
    const issueBid: IssueModel = bid;
    this.http.put<{ result: IssueModel[], message: string }>(this.API_URL + '/issue/bid?id='
+ issueId + "&bid=" + issueBid, issueBid).subscribe(
      (res) => {  }
    );
  }

//Ενημέρωση ενός issue. Πραγματοποιείται μια PUT request στον server για την
ενημέρωση του issue με id που αποστέλλεται μέσω url.
  respondToBid(bidId: String, response: String, issueId, bid) {
    const issueBid: IssueModel = bid;
    this.http.put<{ result: IssueModel[], message: string }>(this.API_URL + '/issue/
bidResponse?response=' + response + '&bidId=' + bidId + '&issueId=' + issueId,
issueBid).subscribe(
      (res) => { }
    ) }

//Επιβεβαίωση της αποδοχής της προσφοράς ενός issue. Πραγματοποιείται μια PUT
request στον server για την ενημέρωση του issue με id που αποστέλλεται μέσω url. Επίσης
αποστέλλονται τα τα id του bid, το ποσό του bid καθώς και το true το οποίο αποθηκεύεται
στον bid.confirmed της βάσης.
  confirmIssue(bidId: String, confirm: boolean, issueId, bid) {
    const issueBid: IssueModel = bid;
    this.http.put<{ result: IssueModel[], message: string }>(this.API_URL + 'issue/
confirmIssue?confirm=' + confirm + '&bidId=' + bidId + '&issueId=' + issueId,
issueBid).subscribe(
      (res) => { }
    )
  }

//Κλείσιμο ενός issue. Πραγματοποιείται μια PUT request στον server για την ενημέρωση
του bid με id και ποσό που αποστέλλεται μέσω url, καθώς και του close που κάνει το
bid.closed στη βάση, true.
```

```
  closeIssue(bidId: String, close: boolean, issueId, bid) {
    const issueBid: IssueModel = bid;
    this.http.put<{ result: IssueModel[], message: string }>(this.API_URL + 'issue/
closeIssue?confirm=' + close + '&bidId=' + bidId + '&issueId=' + issueId,
issueBid).subscribe(
      (res) => { }
    )
  }


//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των issues που
ανήκουν στην περιοχή και κατηγορία του επαγγελματία και αποθηκεύεται σε ένα subject.
  getIssuesForPro(city,category){
    console.log("City: " + city + "\nCategory: " + category);
    this.http.get<{ issue: IssueModel[], message: string }>(this.API_URL + "issue/
getIssuesForPro?city=" + city +"&category=" + category).subscribe((res)=>{
      console.log(res);
      this.proIssues = res.issue;
      this.$profileIssues.next(this.proIssues)
    });
  }

// Επιστρέφει ένα observable με τη χρήση του subject
getIssuesForProfile(){
    return this.$profileIssues.asObservable();
  }

}
```

browse.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient, HttpErrorResponse} from "@angular/common/http";
import {environment} from "../../../environments/environment";
import {Router} from "@angular/router";
import {Observable, Subject} from "rxjs";
import {ProRegisterModel} from "../../models/proRegister.model";
import {CategoriesModel} from "../../models/categoriesModel";
import {CitiesModel} from "../../models/cities.model";
import {ReviewModel} from "../../models/reviewModel";


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

```
 API_URL = environment.apiUrl;
 private pros;
 private $prosUpdated = new Subject();
 private $prosByCity = new Subject();
 private $prosByCategory = new Subject();
 private $prosByName = new Subject();
 private $categories = new Subject();
 private $cities = new Subject();
 private $prosSortedByName = new Subject();
 private $reviews = new Subject();


 private prosByCity;
 private prosByCategory;
 private prosByName;
 private categories;
 private cities;
 private prosSortedByName;
 private reviews;
 searchOption = [];

 constructor(private http: HttpClient,
             private router: Router) {
 }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των
επαγγελματιών και αποθηκεύεται σε ένα subject.
 getAllPros() {
   return this.http.get<{ pros: ProRegisterModel[], message: string }>(this.API_URL +
'pro')
     .subscribe(
       (__pros) => {
         // @ts-ignore
         this.pros = __pros.pro;
         this.$prosUpdated.next(this.pros);
       }
     )
 }

// Επιστρέφει ένα observable με τη χρήση του subject
 getPros$(): Observable<any> {
   return this.$prosUpdated.asObservable();
 }
```

```
//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των επαγγελματιών στη συγκριμένη πόλη που αποστέλλεται μέσω url και αποθηκεύεται σε ένα subject.
  getProsByCity(selectedCity: String) {
    return this.http.get<{ pro: ProRegisterModel[], message: string }>(this.API_URL + "pro/prosCity?city=" + selectedCity.value).subscribe(
      res => {
        this.prosByCity = res.pro;
        this.$prosByCity.next([...this.prosByCity]);

      }
    )
  }


// Επιστρέφει ένα observable με τη χρήση του subject
  getProsByCityUpdated() {
    return this.$prosByCity.asObservable();
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει το σύνολο των επαγγελματιών στη συγκριμένη κατηγορεί που αποστέλλεται μέσω url και αποθηκεύεται σε ένα subject.
  getProsByCategory(selectedCategory: string) {
    return this.http.get<{ pro: ProRegisterModel[], message: string }>(this.API_URL + "pro/prosCategory?category=" + selectedCategory.value)
      .subscribe(
        res => {
          this.prosByCategory = res.pro;
          this.$prosByCategory.next([...this.prosByCategory])
        }
      )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getProsByCategoryUpdated() {
    return this.$prosByCategory.asObservable();
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει τον επαγγελματία με το συγκεκριμένο όνομα που αποστέλλεται μέσω url και αποθηκεύεται σε ένα subject.
  getProsByName(name: string) {
    return this.http.get<{ pro: ProRegisterModel[], message: string }>(this.API_URL + "pro/prosName?name=" + name.split(' ')[1])
      .subscribe(
        res => {
          this.prosByName = res.pro;
```

```
        this.$prosByName.next([...this.prosByName]);
      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getProsByNameUpdated() {
    return this.$prosByName.asObservable();
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει τον επαγγελματία με το συγκεκριμένο όνομα που αποστέλλεται μέσω url και αποθηκεύεται σε ένα subject.
  getCategories() {
    return this.http.get<{ cat: CategoriesModel[], message: string }>(this.API_URL + 'browse/categories').subscribe(
      (res) => {
        this.categories = res.cat;
        this.$categories.next([...this.categories]);
    }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getCategoriesUpdated() {
    return this.$categories.asObservable();
  }

//Πραγματοποιείται μια GET request στον server, επιστρέφει τις πόλεις και αποθηκεύεται σε ένα subject.
  getCities() {
    return this.http.get<{ city: CitiesModel[], message: string }>(this.API_URL + 'browse/cities').subscribe(
      (res) => {
        this.cities = res.city;
        this.$cities.next([...this.cities]);

      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getCitiesUpdated() {
    return this.$cities.asObservable();
  }
```

```
//Πραγματοποιείται μια GET request στον server και επιστρέφει τους επαγγελματίες με αλφαβητική σειρά, ανάλογα με το input που αποστέλλεται (1, -1) που αποστέλλεται μέσω url και αποθηκεύεται σε ένα subject.
  getProsSortedByName(filterPro: number) {
    return this.http.get<{ pros: ProRegisterModel, message: string }>(this.API_URL + 'browse/getProsSortedByName?sort=' + filterPro).subscribe(
      (res) => {
        console.log(res);
        this.prosSortedByName = res.pros;
        this.$prosSortedByName.next([...this.prosSortedByName]);

      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getProsSortedByNameUpdated() {
    return this.$prosSortedByName.asObservable();
  }


  getReviews() {
    return this.http.get<{ review: ReviewModel[], message: string }>(this.API_URL + 'pro/reviews').subscribe(
      (res) => {
        this.reviews = res.review;
        this.$reviews.next([...this.reviews]);

      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getReviewsUpdated() {
    return this.$reviews.asObservable();
  }



}
```

review.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Router} from "@angular/router";
```

```
import {MatSnackBar} from "@angular/material";
import {environment} from "../../../environments/environment";
import {Subject} from "rxjs";
import {ReviewModel} from "../../models/reviewModel";

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

  API_URL = environment.apiUrl;
  private $reviews = new Subject();
  private $userReviews = new Subject();
  private reviews;


  constructor(private http: HttpClient,
              private router: Router,
              private snack: MatSnackBar) {
  }

//Πραγματοποιείται μια POST request στον server και αποθηκεύει μια κριτική στη βάση.
  addReview(reviewDataAdded: ReviewModel, bidCreator) {
    const reviewAdded: ReviewModel = reviewDataAdded;
    this.http.post(this.API_URL + 'issue/add/review?bidCreator=' + bidCreator,
reviewAdded).subscribe(
      (res) => {
        console.log(res);
        this.snack.open('Η κριτική σας καταχωρήθηκε', "OK", {
          duration: 2000
        });
        this.router.navigate(['/profile']);
      }
    )
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει τις κριτικές που αφορούν
τον επαγγελματία και αποθηκεύεται σε ένα subject.
  getProReviews() {
    return this.http.get<{ review: ReviewModel[], message: string }>(this.API_URL + 'pro/
reviews').subscribe(
      (res) => {
        this.reviews = res.review;
        this.$reviews.next([...this.reviews]);

      }
    )
```

```
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getProReviewsUpdated() {
    return this.$reviews.asObservable();
  }

//Πραγματοποιείται μια GET request στον server και επιστρέφει τις κριτικές που αφορούν τον χρήστη και αποθηκεύεται σε ένα subject.
  getUserReviews() {
    return this.http.get<{ review: ReviewModel[], message: string }>(this.API_URL + 'pro/reviews/user').subscribe(
      (res) => {
        this.reviews = res.review;
        this.$userReviews.next([...this.reviews]);

      }
    )
  }

// Επιστρέφει ένα observable με τη χρήση του subject
  getUserReviewsUpdated() {
    return this.$userReviews.asObservable();
  }


}
```

4.3. Models

Τα μοντέλα (κλάσεις) που χρησιμοποιήθηκαν για να περιοριστούν οι τύποι μεταβλητών που θα ήταν αποδεκτοί σε κάποια σημεία του Project.

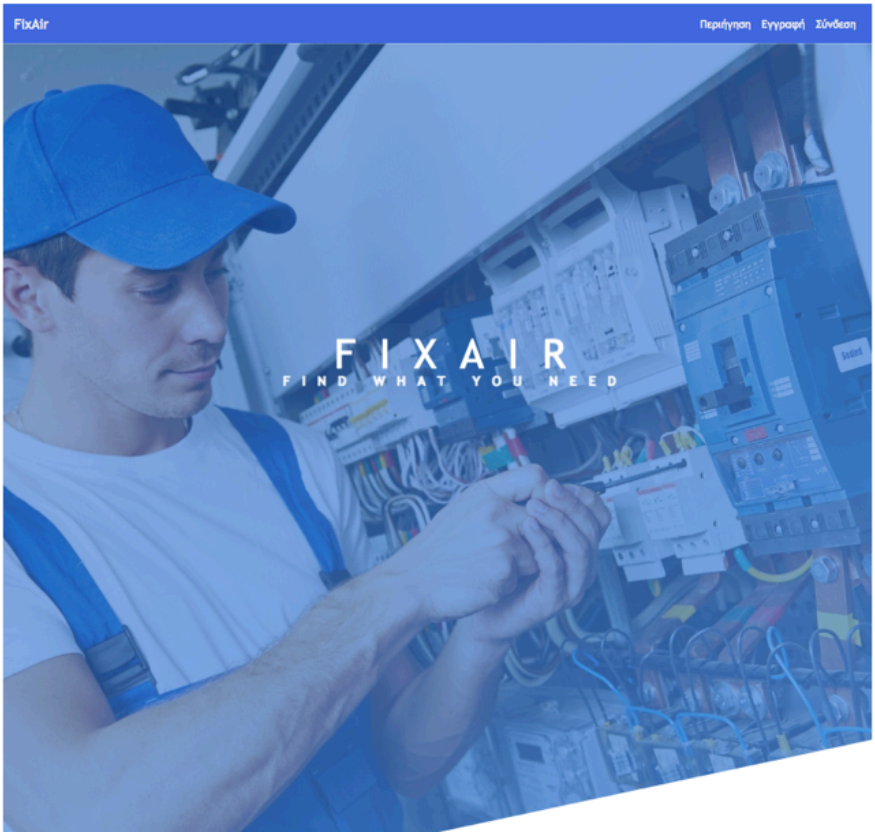
```
export class UserRegisterModel {
  _id: String
  name: String;
  email: String;
  password: String;
  createdAt: String;
}

export class UserLoginModel {
  email: String;
  password: String;
```

```
}

export class ReviewModel {
  _id: string;
  stars: string;
  description: string;
  date: Date;
}

export class ProRegisterModel {
  role: string;
  p_name: string;
  p_surname: string;
  p_email: string;
  p_password: string;
  p_brandName: string;
  p_category: string;
  p_locationCoverage: string;
  p_phone: string;
}

export class ProLoginModel {
  p_email: String;
  p_password: String;
}

export class IssueModel {
  _id: string;
  title: string;
  description: string;
  category: string;
  price: string;
  city: string;
  homeAddress: string;

  bids:[{
    amount: number,
    createdAt: Date,
    creator: string
    status: {
      type: string,
    }
  }];

  phone: number;
  availability: string;
  address: any;
```

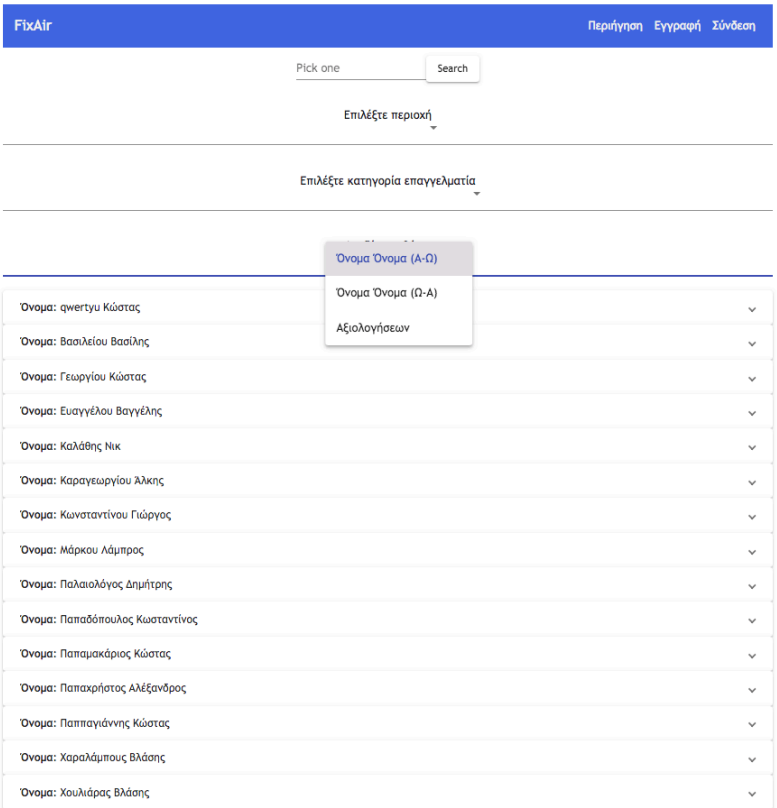
```
}

export class CitiesModel {
  _id: string;
  value: string;
  viewValue: string;
}

export class CategoriesModel {
  _id: string;
  value: string;
  viewValue: string;
}
```

5. Η εφαρμογή & μελλοντικές προσθήκες

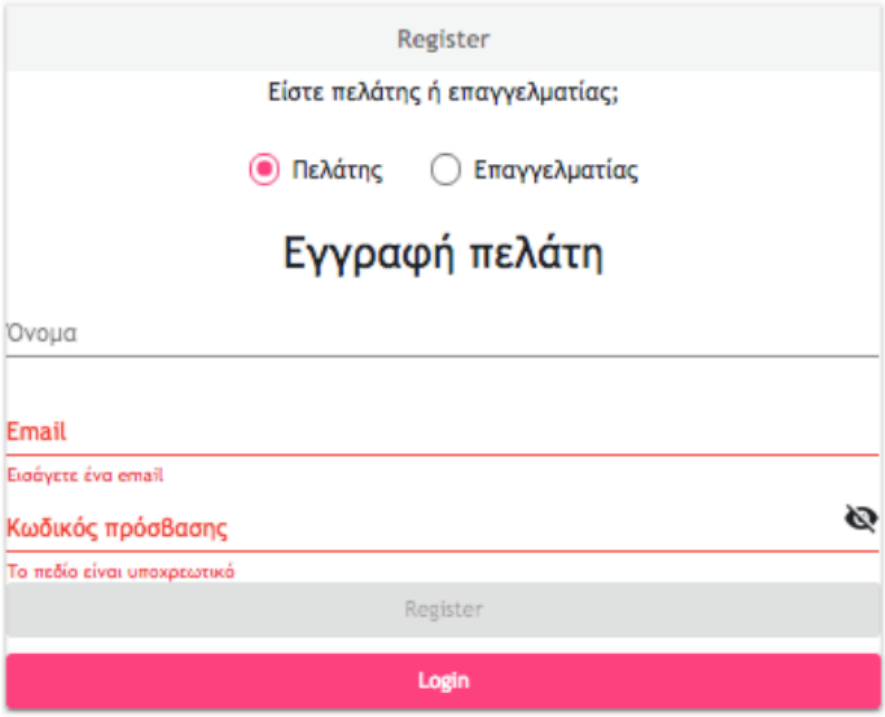
5.1) Αρχική σελίδα



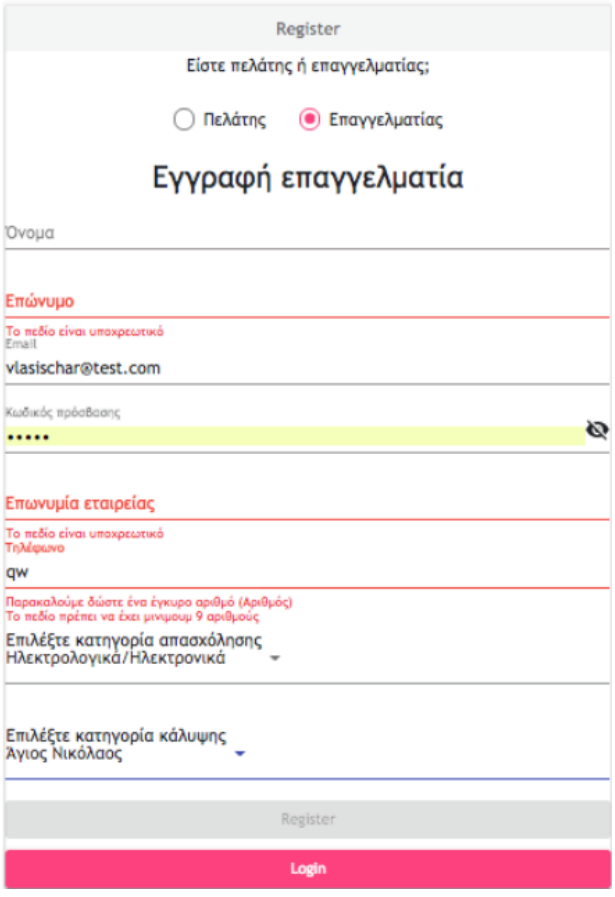
5.2) Ανώνυμη περιήγηση



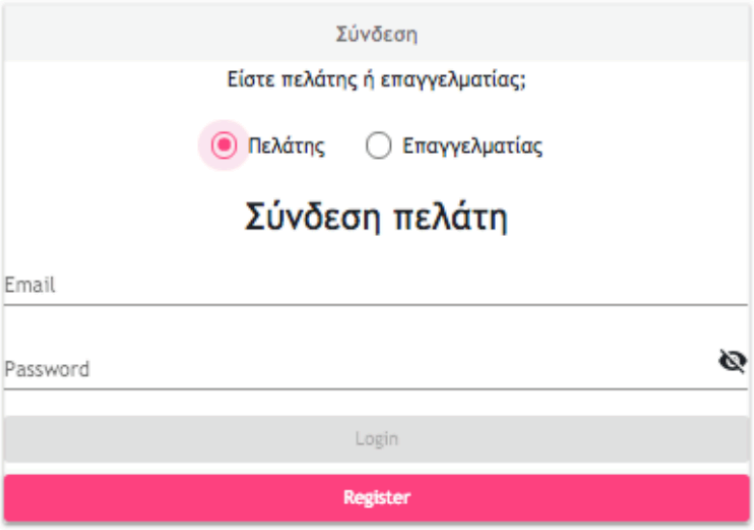
5.3) Εγγραφή πελάτη



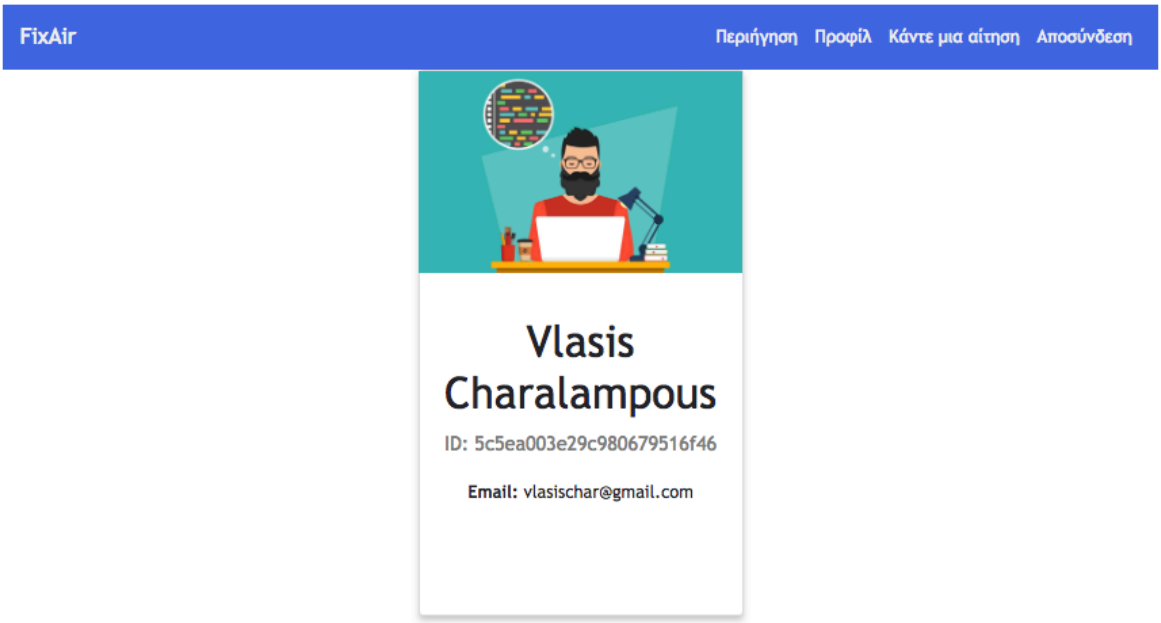
5.4) Εγγραφή επαγγελματία



5.5) Σύνδεση πελάτη

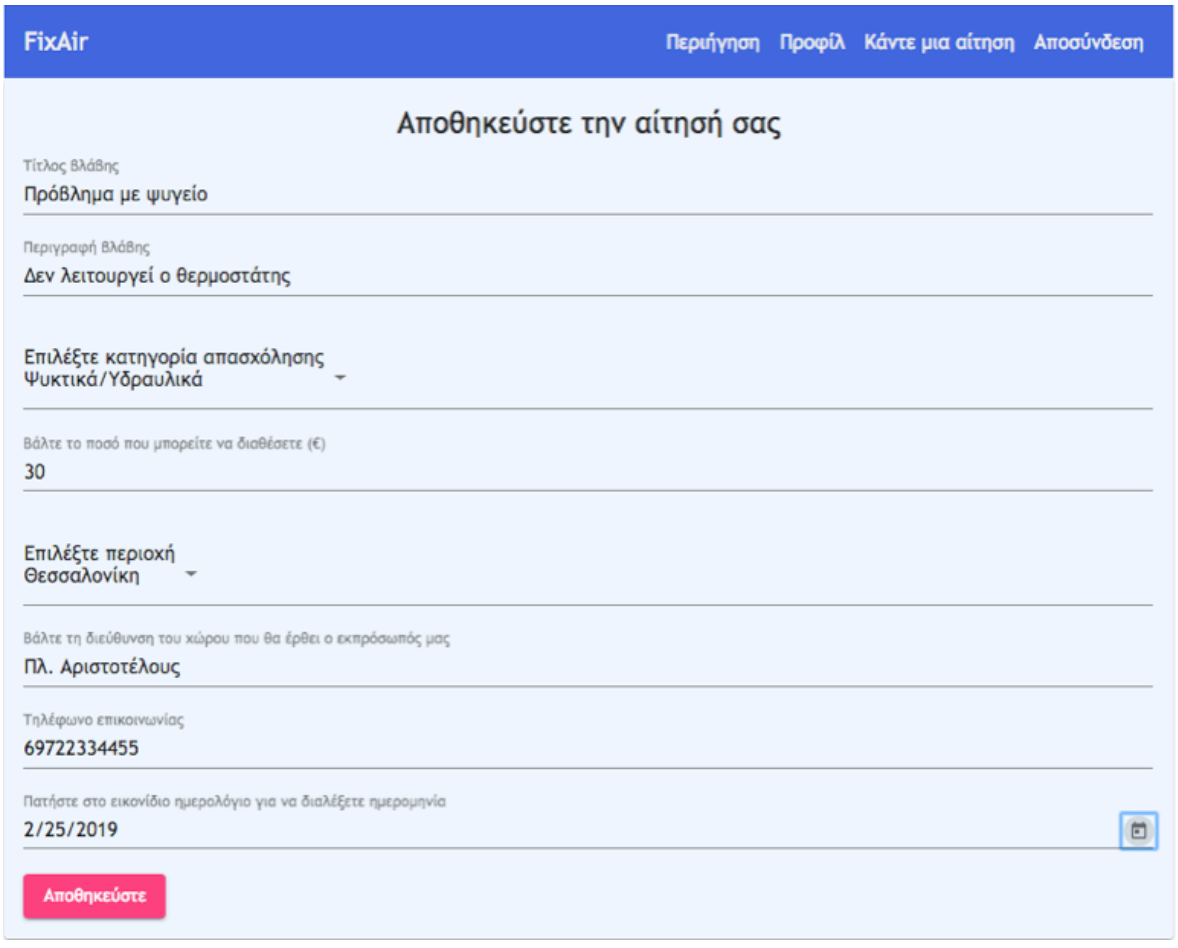


5.6) Προφίλ πελάτη

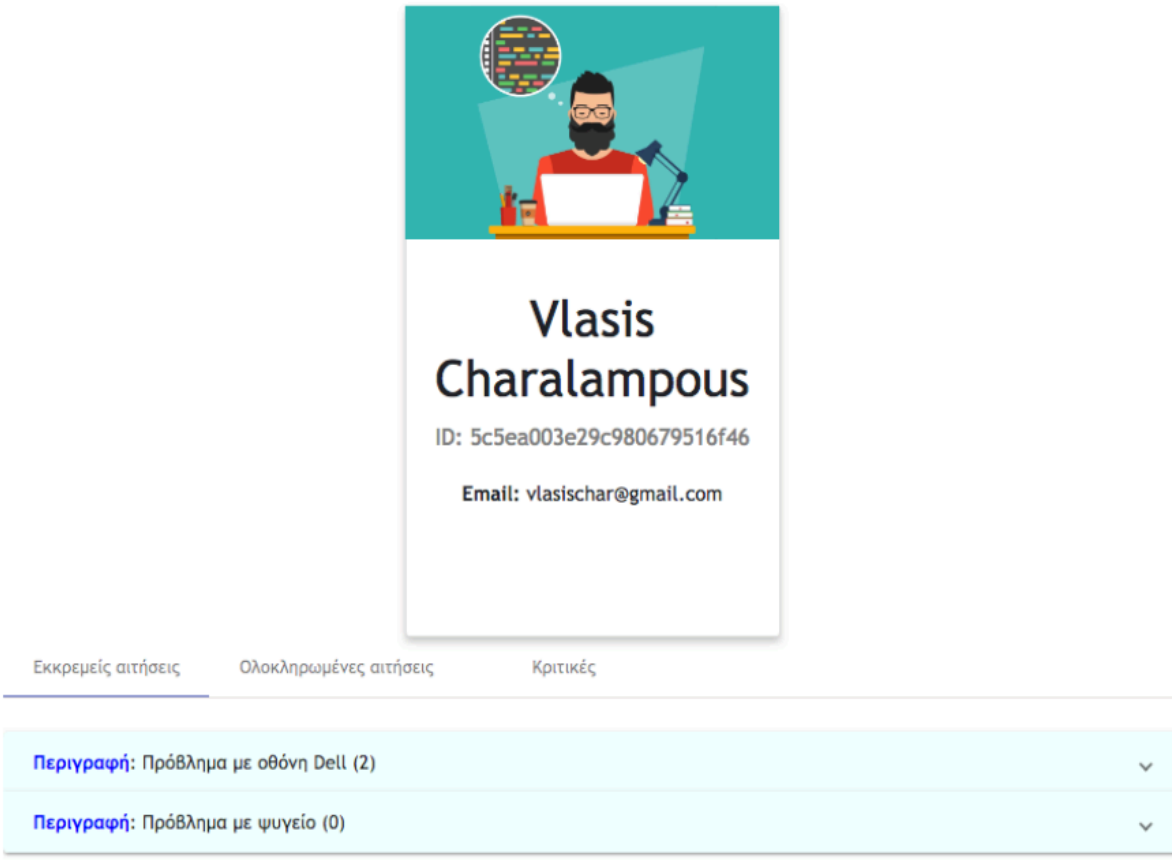


5.7) Προφίλ επαγγελματία

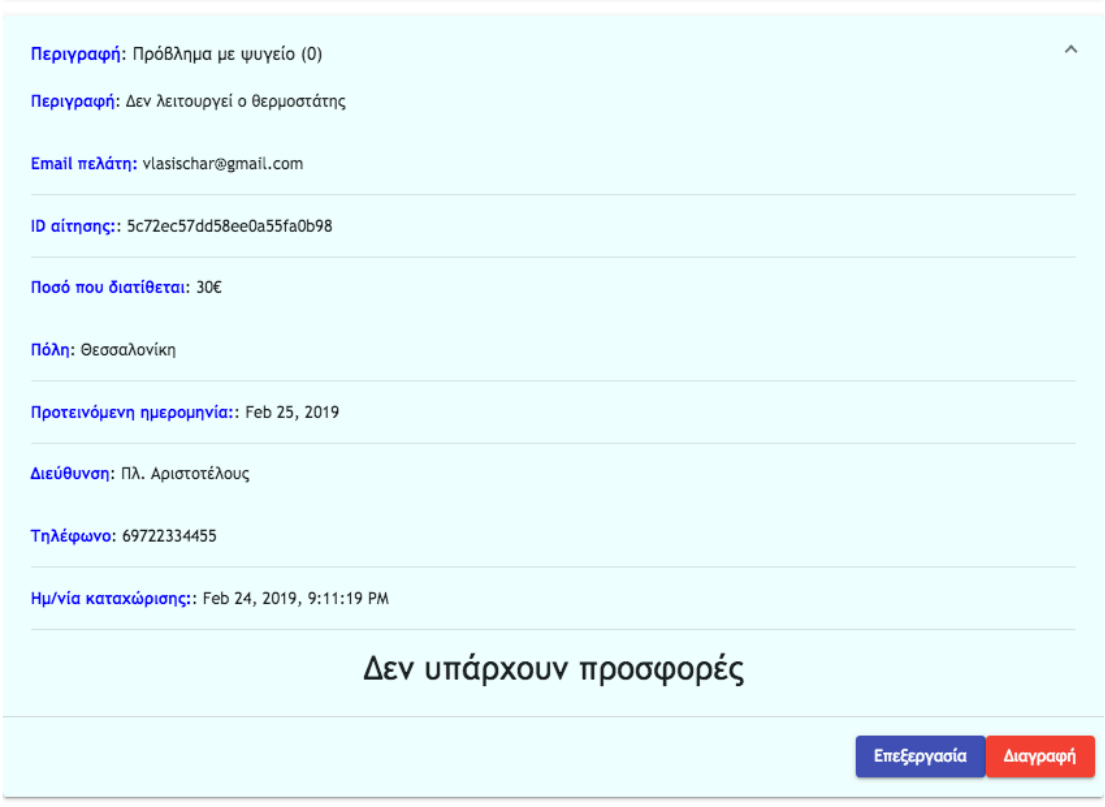
5.8) Καταχώρηση αίτησης



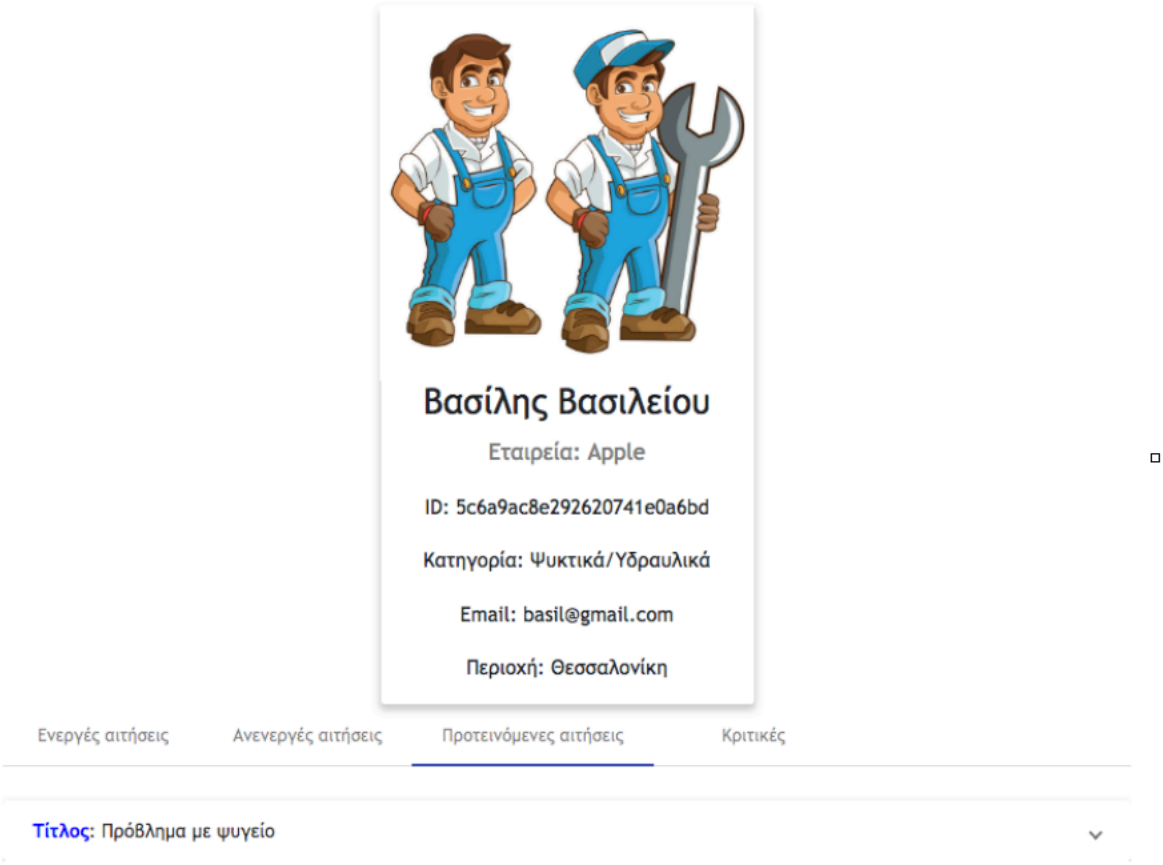
5.9) Καταχωρημένη αίτηση στο προφίλ πελάτη



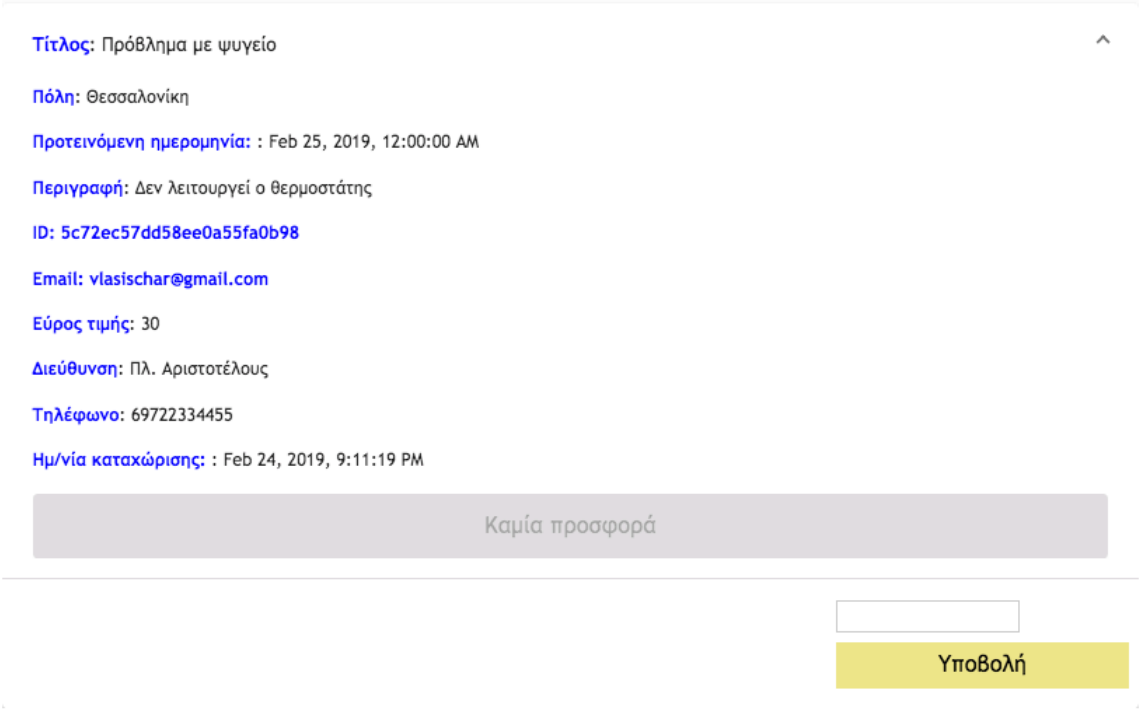
5.10) Λεπτομέρειες αίτησης



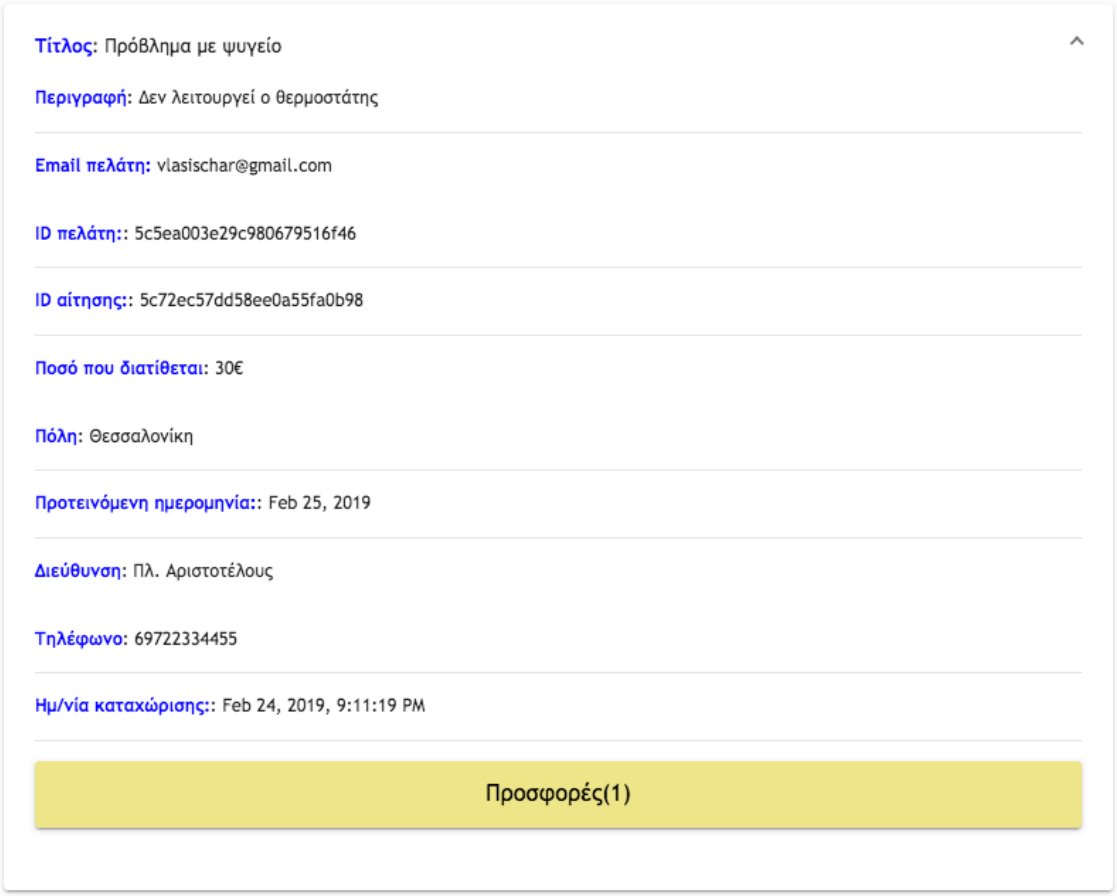
5.11) Η αίτηση στα προτεινόμενα του επαγγελματία



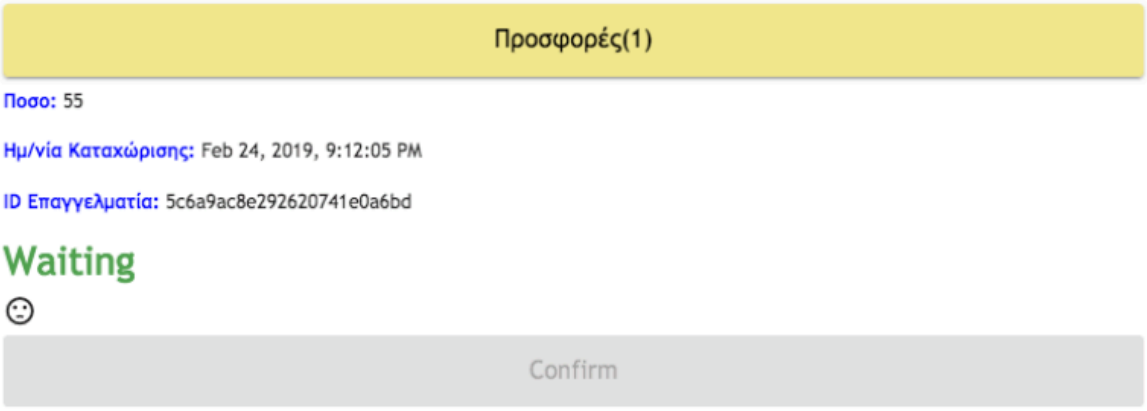
5.12) Λεπτομέρειες αίτησης για επαγγελματία (Πριν την προσφορά)



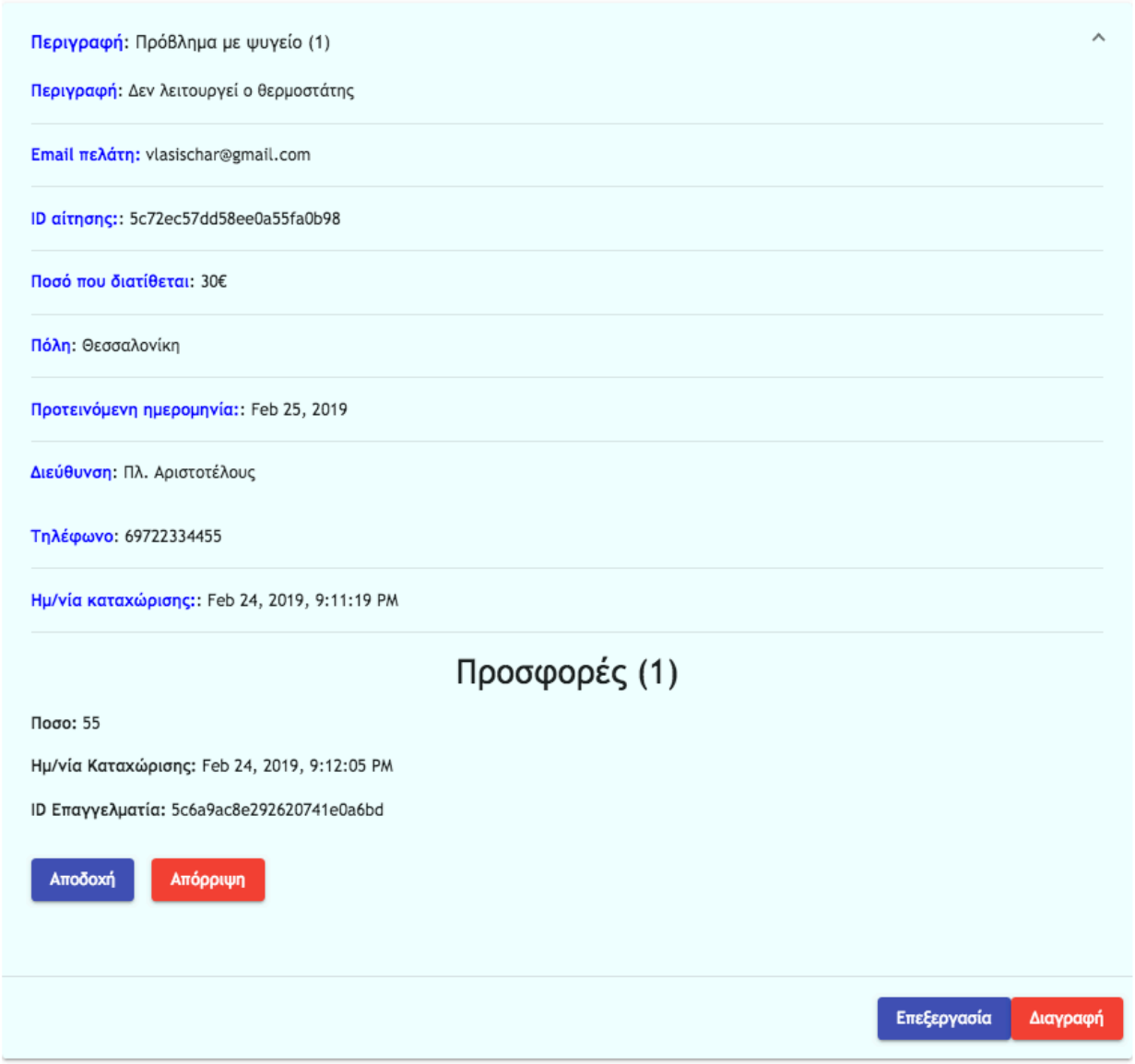
5.13) Λεπτομέρειες αίτησης για επαγγελματία (Μετά την προσφορά)



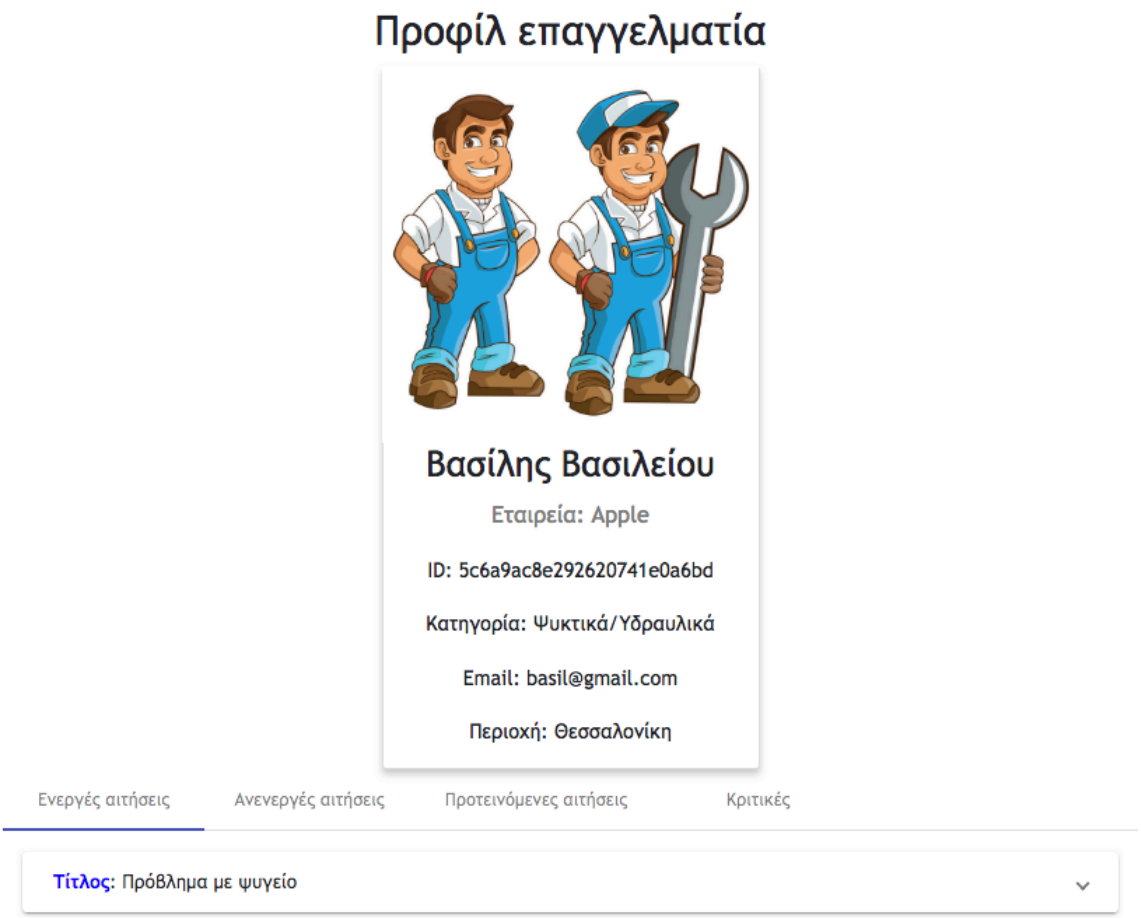
5.14) Λεπτομέρειες προσφοράς για επαγγελματία (Πριν την προσφορά)



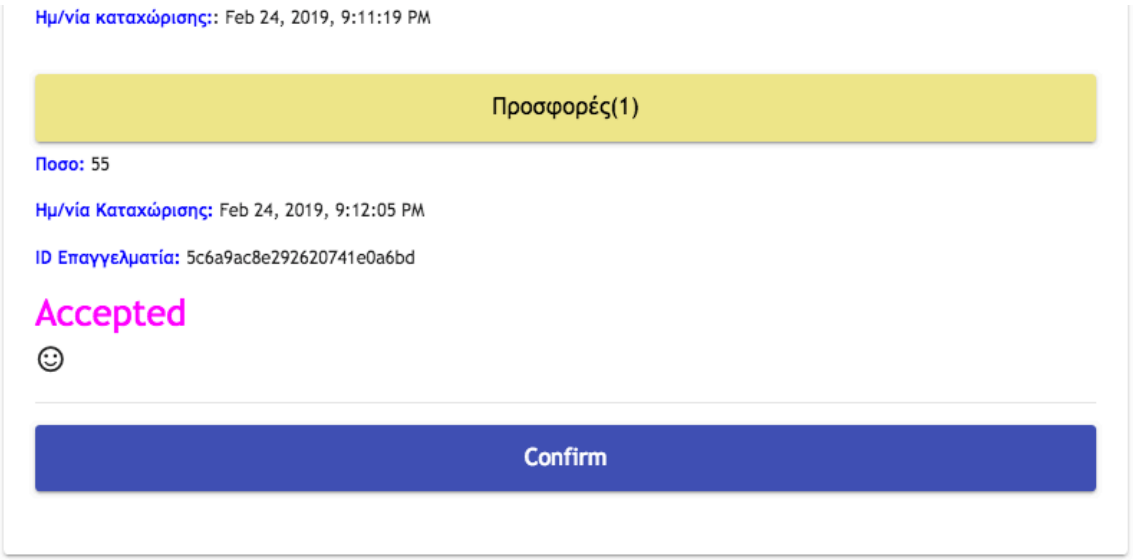
5.15) Λεπτομέρειες αίτησης για πελάτη (Μετά την προσφορά)



5.16) Ενεργές αιτήσεις για πελάτη (Μετά την προσφορά)

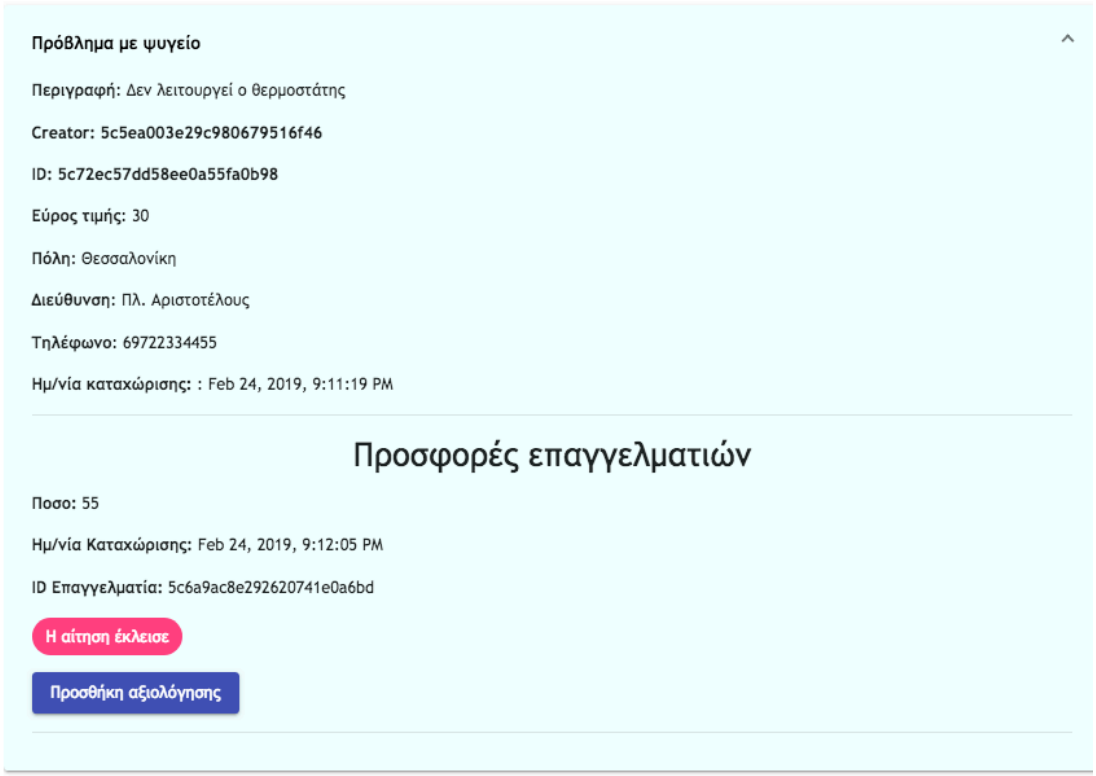


5.17) Λεπτομέρειες προσφοράς για επαγγελματία (Μετά την αποδοχή προσφοράς απο πελάτη)

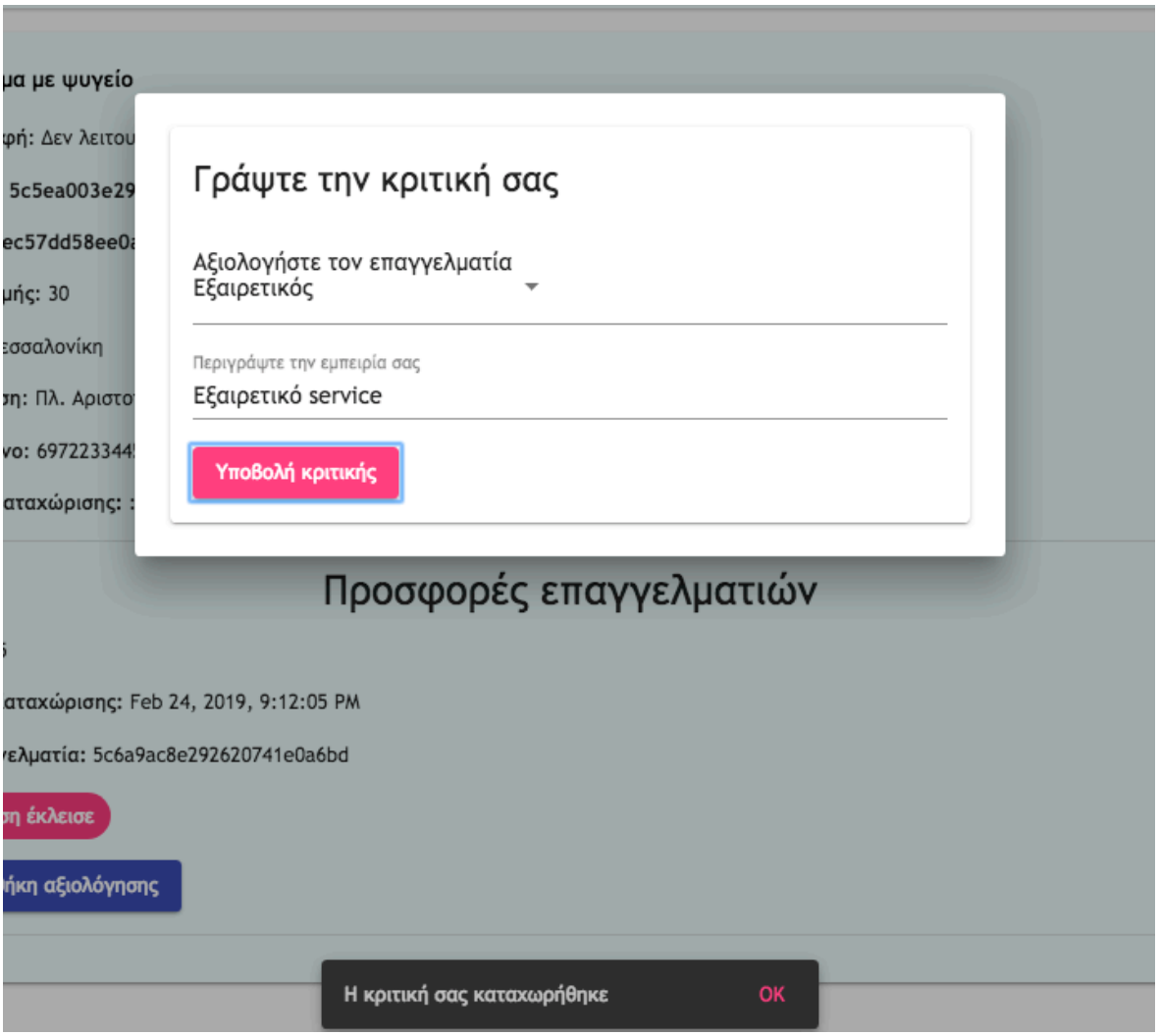


5.18) Μεταφορά της αίτησης στην καρτέλα Ανενεργές αιτήσεις

5.19) Λεπτομέρειες αίτησης αφού την έχει κλείσει ο επαγγελματίας



5.20) Υποβολή αξιολόγησης από τον πελάτη



Η παρούσα εφαρμογή μπορεί να βελτιωθεί και να επεκταθεί. Κάποια χαρακτηριστικά που θα μπορούσαν να προστεθούν μελλοντικά είναι:

- Προσθήκη αλγορίθμων μηχανική μάθησης (Machine Learning) οι οποίοι θα δημιουργούν προσωποποιημένες προτάσεις για τον χρήστη, σύμφωνα με τι συνήθειες του.
- Προσθήκη συστήματος ειδοποιήσεων (Notification system) με την χρήση sockets.
- Δημιουργία ενός σύγχρονου και ελκυστικού UI (User Interface)
- Δημιουργία dashboard για τον administrator
- Ανάκτηση του κωδικού από τον χρήστη μέσω mail
- Προσθήκη εικόνας στην αίτηση του πελάτη

ΠΑΡΑΠΟΜΠΕΣ

[1] *Web development- Wikipedia*
https://en.wikipedia.org/wiki/Web_development

[2] *Stackoverflow 2018 survey- Stackoverflow*
https://insights.stackoverflow.com/survey/2018/

[3] *Stackoverflow 2018 survey- Stackoverflow*
https://insights.stackoverflow.com/survey/2018/

[4] *PWA vs Hybrid App vs Native- Priyesh Patel*
https://blog.bitsrc.io/4-ways-to-build-your-mobile-app-make-the-right-choice-efe079c7c817

[5] *The MEAN stack, the big picture- Maximilian Schwarzmüller*
https://www.udemy.com/angular-2-and-nodejs-the-practical-guide/learn/v4/overview

[6] *Sharding- MongoDB Documentation*
https://docs.mongodb.com/manual/sharding/

[7] *SQL vs NoSQL*
https://sql-vs-nosql.blogspot.com/2013/11/indexes-comparison-mongodb-vs-mssqlserver.html

[8] *Embedded documents, MongoDB- Pablo Godel*
https://www.slideshare.net/pgodel/symfony2-and-mongodb

[9] *Data Binding- AngularJS*
https://docs.angularjs.org/guide/databinding

[10] *Why NodeJS is asynchronous?- The Web Stop*
https://thewebstop.blogspot.com/2017/09/why-nodejs-is-asynchronous.html

[11] *What is REST API? in plain English- <? php*
https://phpenthusiast.com/blog/what-is-rest-api

[12] *JWT decode/ encode- JWT.io*
https://jwt.io/

[1] *SPA Authentication- Maximilian Schwarzmüller*
https://www.udemy.com/angular-2-and-nodejs-the-practical-guide/learn/v4/overview

ΒΙΒΛΙΟΓΡΑΦΙΑ

Angular & NodeJS - The MEAN Stack Guide- Maximilian Schwarzmüller
https://www.udemy.com/angular-2-and-nodejs-the-practical-guide/learn/v4/overview

Advanced CSS and Sass: Flexbox, Grid, Animations and More!- Jonas Schmedtmann
https://www.udemy.com/advanced-css-and-sass/learn/v4/overview

Top 8 Web Development Trends 2019- Maximilian Schwarzmüller
https://www.youtube.com/watch?v=VLm3Y7Odb74&t=1304s

Angular official documentation
https://angular.io/docs

Stackoverflow Developer Survey Results 2018
https://insights.stackoverflow.com/survey/2018/#technology

Definition of web development
https://www.techopedia.com/definition/23889/web-development

A Brief History of JavaScript
https://auth0.com/blog/a-brief-history-of-javascript/

History of web development
https://interactivepython.org/runestone/static/webfundamentals/WWW/history.html

w3schools- How to- Profile card
https://www.w3schools.com/howto/howto_css_profile_card.asp