ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Π.Μ.Σ. «ΕΦΑΡΜΟΣΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ»

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Εφαρμογών Πραγματικού χρόνου με το MEAN Stack

Αλέξανδρος Ντίτορας

Εισηγητής: Δρ Κωνσταντίνος Κουκουλέτσος, Καθηγητής

ΑΘΗΝΑ
ΜΑΡΤΙΟΣ 2018

(Κενό φύλλο)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Εφαρμογών Πραγματικού χρόνου με το MEAN Stack
Αλέξανδρος Ντίτορας
A.M. ais0062

Εισηγητής:

Δρ Κωνσταντίνος Κουκουλέτσος, Καθηγητής

Εξεταστική Επιτροπή:

Ημερομηνία εξέτασης 10/06/2018

(Κενό φύλλο)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος/η Ντίτορας Αλέξανδρος, του Θεοδώρου, με αριθμό μητρώου AIS0062 φοιτητής/τρια του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

(Κενό φύλλο)

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό της ανάπτυξης εφαρμογών πραγματικού χρόνου με το MEAN stack. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω.

Ακόμα θα ήθελα να ευχαριστήσω την οικογένειά μου που θα ήθελε να τελειώσω τις σπουδές μου σε λιγότερο από τρία χρόνια.

(Κενό φύλλο)

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία έχει σαν θέμα την ανάπτυξη εφαρμογών πραγματικού χρόνου κάνοντας χρήση των πιο σύγχρονων τεχνολογιών της αγοράς. Επιλέχθηκε το framework MEAN το οποίο συνδυάζει τις τεχνολογίες MongoDB, EXPRESS.JS, NODEJS, ANGULAR 5. Στην αρχή θα παρουσιαστεί συνοπτικά ο τρόπος εγκατάστασης και εφαρμογής των τεχνολογιών και κατόπιν θα αναπτυχθεί, με την χρήση των τεχνολογιών αυτών, μια ολοκληρωμένη εφαρμογή τύπου quiz.

ABSTRACT

The subject of this work is the development of a real time application employing some of the most modern tools and technologies in the market. The MEAN framework was selected for the application which includes MongoDB, EXPRESS.JS, ANGULAR 5, NODE JS. Firstly the installation and application of the framework and each component will be presented. Afterwards, using the selected framework, a complete quiz application will be developed.

Κατάλογος περιεχομένων

ΚΕΦΑΛΑΙΟ 1.....12

1.1 ΕΙΣΑΓΩΓΗ.....12

1.1.1 Περιγραφή του αντικειμένου της πτυχιακής εργασίας 12

1.1.2 Ιστορική αναδρομή 12

ΚΕΦΑΛΑΙΟ 2.....14

MongoDB και mlab.....14

2.1 Λίγα λόγια για τη MongoDB. 14

2.1.1 Εγκατάσταση του Mongo Server. 14

2.1.2 Δημιουργία λογαριασμού στο mlab.....14

ΚΕΦΑΛΑΙΟ 3.....20

Εισαγωγή και εγκατάσταση του NodeJS.....20

3.1 Ιστορική αναδρομή.....20

3.1.1 Εγκατάσταση του Node στα Windows 20

3.1.2 Event-Driven προγραμματισμός.....22

3.1.3 Closures.....23

3.2 Εισαγωγή στο Express.Js.....24

3.2.1 Εγκατάσταση του Express.Js.....24

3.2.2 Χρησιμοποιόντας Middleware στην εφαρμογή μας.....27

3.2.3 Χειρισμός Δρομολόγησης. 28

3.3 Εισαγωγή στο MongoDB με to Express 29

3.3.1 Εισαγωγή και εγκατάσταση του Mongoose 29

3.3.2 Σύνδεση Express και MongoDB 30

3.3.3 Ανάπτυξη του server στο Heroku 33

3.5 Περίληψη 38

ΚΕΦΑΛΑΙΟ 4.....39

Angular 5 και Ionic.....39

4.1 Εισαγωγή στην Angular 5 και στο Ionic 3.....39

4.1.1 Εγκατάσταση της Angular 5 και του Ionic 3.....39
4.1.2Δημιουργία εφαρμογής με το Ionic 40
4.1.3 Χαρακτηριστικά και απαιτήσεις της εφαρμογής 42
4.1.4 Οθόνες εφαρμογής.....42
ΚΕΦΑΛΑΙΟ 5.....45
Εγγραφή και σύνδεση του χρήστη.....45
5.1 Σύνοψη λειτουργιών οθόνης εγγραφής-εισόδου 45
5.1.1 Δημιουργία λογαριασμού στο Firebase.....45
5.1.2 Δημιουργία της οθόνης εγγραφής-εισόδου 48
5.1.3Δημιουργία εγγραφή χρήστη στην βάση μας.....54
5.2 Περίληψη.....65
ΚΕΦΑΛΑΙΟ 6.....66
Οθόνη “My profile”.....66
6.1 Λειτουργίες της οθόνης “My profile”.....66
6.1.1 Δημιουργία λειτουργίας δημιουργίας και ανανέωσης επιπέδου χρήστη.....66
6.1.2 Εύρεση τυχαίου αντιπάλου 70
6.2 ΠΕΡΙΛΗΨΗ.....74
ΚΕΦΑΛΑΙΟ 7.....75
Οθόνες της αρένας και ανοικτών αρένων.....75
7.1.1 Δημιουργία της αρένας στον server 75
7.1.2 Δημιουργία της οθόνης 79
7.2 Λειτουργίες της Οθόνης ανοικτών αρένων.....85
7.2.1 Δημιουργία λειτουργιών βραβείων στον server.....85
7.2.2 Δημιουργία της οθόνης 92
7.3 Περίληψη.....95
ΚΕΦΑΛΑΙΟ 8.....96
Επίλογος Συμπεράσματα.....96
ΒΙΒΛΙΟΓΡΑΦΙΑ.....97

ΚΕΦΑΛΑΙΟ 1

1.1 ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα μιλήσουμε συνοπτικά για τις τεχνολογίες που χρησιμοποιήθηκαν. Στα μετέπειτα κεφάλαια θα αναλύσουμε περισσότερο την κάθε μια από αυτές τις τεχνολογίες .

1.1.1 Περιγραφή του αντικειμένου της πτυχιακής εργασίας

Αντικείμενο της παρούσας πτυχιακής άσκησης η γνωριμία ενός προγραμματιστή με κάποιες από τις ποιο σύγχρονες τεχνολογίες για την ανάπτυξη εφαρμογών .Η κύρια γλώσσα που θα χρησιμοποιήσουμε είναι η Javascript. Το κομμάτι που αφορά το Front End (Angular) όπως και το Back End (NodeJs-Express.js) χρησιμοποιούν και τα δυο Javascript .

1.1.2 Ιστορική αναδρομή

Η γλώσσα προγραμματισμού JavaScript δημιουργήθηκε αρχικά από τον Brendan Eich της εταιρείας Netscape με την επωνυμία Mocha. Αργότερα, Mocha μετονομάστηκε σε LiveScript, και τελικά σε JavaScript, κυρίως επειδή η ανάπτυξή της επηρεάστηκε περισσότερο από τη γλώσσα προγραμματισμού Java.LiveScript ήταν το επίσημο όνομα της γλώσσας όταν για πρώτη φορά κυκλοφόρησε στην αγορά σε βήτα (beta) εκδόσεις με το πρόγραμμα περιήγησης στο Web, Netscape Navigator εκδοχή 2.0 τον Σεπτέμβριο του 1995. LiveScript μετονομάστηκε σε JavaScript σε μια κοινή ανακοίνωση με την εταιρεία Sun Microsystems στις 4 Δεκεμβρίου, 1995, όταν επεκτάθηκε στην έκδοση του προγράμματος περιήγησης στο Web, Netscape εκδοχή 2.0B3.

Η javaScript απέκτησε μεγάλη επιτυχία ως γλώσσα στην πλευρά του πελάτη (client-side) για εκτέλεση κώδικα σε ιστοσελίδες, και περιλήφθηκε σε διάφορα προγράμματα περιήγησης στο Web. Κατά συνέπεια, η εταιρεία Microsoft ονόμασε την εφάρμογή της σε JScript για να αποφύγει δύσκολα θέματα εμπορικών σημάτων. JScript πρόσθεσε νέους μεθόδους για να διορθώσει τα Y2K-προβλήματα στην JavaScript, οι οποίοι βασίστηκαν στην ,java.util.Datetάξη της Java. JScript περιλήφθηκε στο πρόγραμμα Internet Explorer εκδοχή 3.0, το οποίο κυκλοφόρησε τον Αύγουστο του 1996.

Τον Νοέμβριο του 1996, η Netscape ανακοίνωσε ότι είχε υποβάλει τη γλώσσα JavaScript στο Ecma International (*μια οργάνωση της τυποποίησης των γλωσσών προγραμματισμού*) για εξέταση ως βιομηχανικό πρότυπο, και στη συνέχεια το έργο είχε ως αποτέλεσμα την τυποποιημένη μορφή που ονομάζεται ECMAScript.[12]

Το MEAN Stack δημιουργήθηκε το 2013 από τον προγραμματιστή Valeri Karpov ο οποίος δούλευε στην MongoDB .Αποτελείται από τις τεχνολογίες MongoDB η οποία δημιουργήθηκε το 2009 και είναι μια NoSQL βάση δεδομένων. Για την διαχείριση του backend χρησιμοποιεί το NodeJS το οποίο είναι μια πλατφόρμα ανάπτυξη λογισμικού στην Javascript engine V8 της google. Το ExpressJS είναι ένα open source framework που χρησιμοποιείται μαζί με το Node Js και με αυτό φτιάχνουμε τον Server.Τέλος για το Front End χρησιμοποιούμε το framework Angular 5 της Google το οποίο βγήκε στην αγορά τον Οκτώμβρη του 2016.

ΚΕΦΑΛΑΙΟ 2

MongoDB και mlab

2.1 Λίγα λόγια για τη MongoDB.

Η **MongoDB** αποθηκεύει τα δεδομένα με ευέλικτο τρόπο που με την δομή JSON αρχείων που σημαίνει ότι τα πεδία μπορούν να διαφέρουν από αρχείο σε αρχείο και η δομή τους μπορεί να αλλάξει οποιαδήποτε στιγμή. Λόγο του ότι χρησιμοποιεί την τεχνική του replication αυτό σημαίνει ότι δημιουργούνται αντίγραφα server του εαυτού της προσφέρει υψηλή διαθεσιμότητα. Δεν θα μιλήσουμε παραπάνω για τις δυνατότητες και το τι μας προσφέρει η Mongo καθώς δεν είναι αυτός ο σκοπός της πτυχιακής αλλά θα προχωρήσουμε παρακάτω στο πως να κάνουμε εγκατάσταση και πως θα την χρησιμοποιήσουμε.

2.1.1 Εγκατάσταση του Mongo Server.

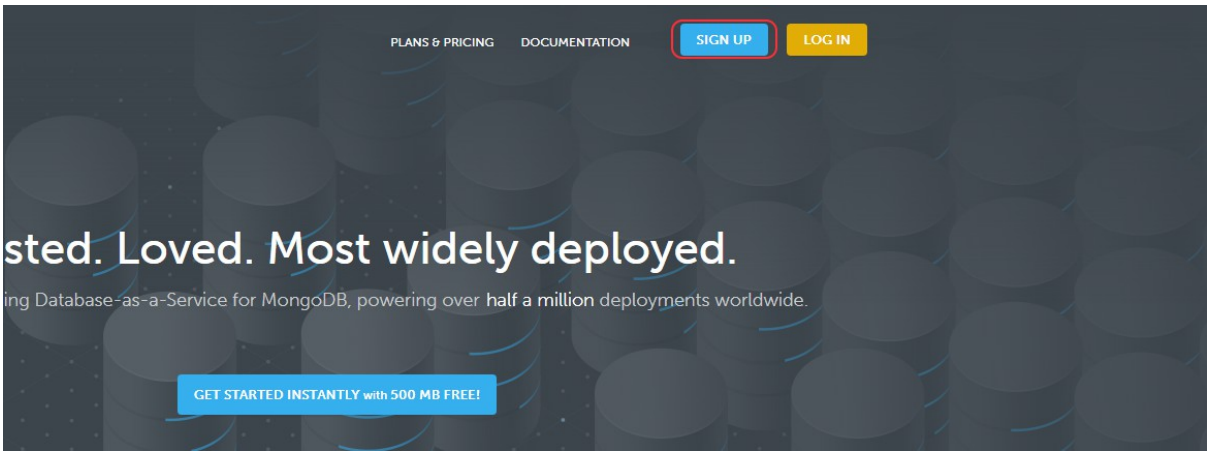
Για να μπορέσουμε να εγκαταστήσουμε θα χρειαστούμε έναν υπολογιστή που να τρέχει windows και να κατεβάσουμε το Mongo DB installer από το παρακάτω link MongoDB Installer και το εγκαθιστούμε.

Για να βεβαιωθούμε ότι η εγκατάσταση έγινε σωστά τρέχουμε το mongod.exe αρχείο που βρίσκεται στο C:\Program Files\MongoDB\Server\3.4\bin ο server μας ακούει στην πόρτα 27017.

2.1.2 Δημιουργία λογαριασμού στο mlab

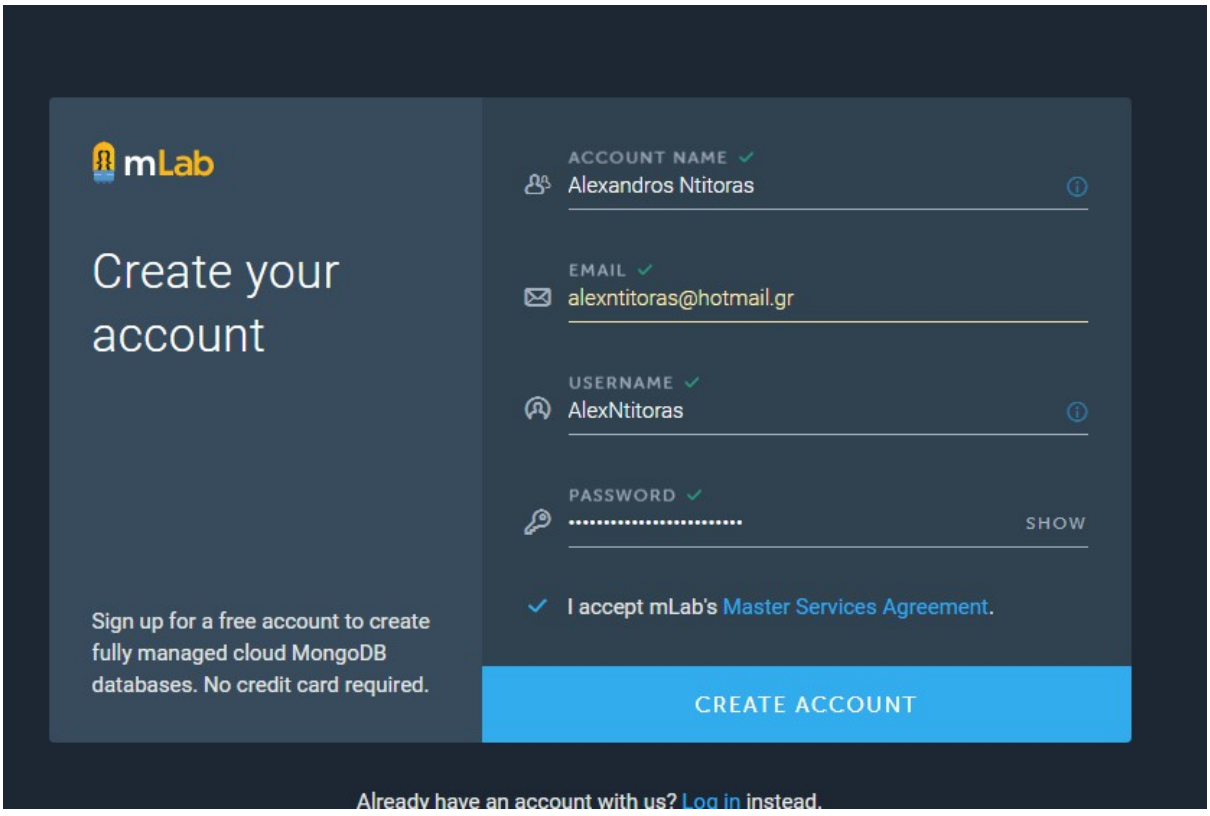
Στο προηγούμενο βήμα δημιουργήσαμε ένα τοπικό server στο windows υπολογιστή μας στον συγκεκριμένο server έχουμε πρόσβαση μόνο στον τοπικό μας δίκτυο τι όμως θα έπρεπε να κάνουμε αν θέλαμε η βάση μας να είναι ορατή από οποιοδήποτε σημείο στο internet?Σε αυτή την περίπτωση θα πρέπει να χρησιμοποιήσουμε κάποιον cloud server και συγκεκριμένα θα χρησιμοποιήσουμε το mlab που μας παρέχει δωρεάν 500 mb χώρου ο οποίος είναι υπεραρκετός για μικρού μεγέθους εφαρμογές σαν και τη δικιά μας. Η χρήση του mlab είναι πάρα

πολύ απλή το πρώτο πράγμα που πρέπει να κάνουμε είναι να μεταβούμε στο site του mlab "*mlab.com*". Στην πάνω δεξιά γωνία πατάμε το sign up όπως και φινετσάτε στην εικόνα 2.1.



Εικόνα 2.1: Sign up mlab

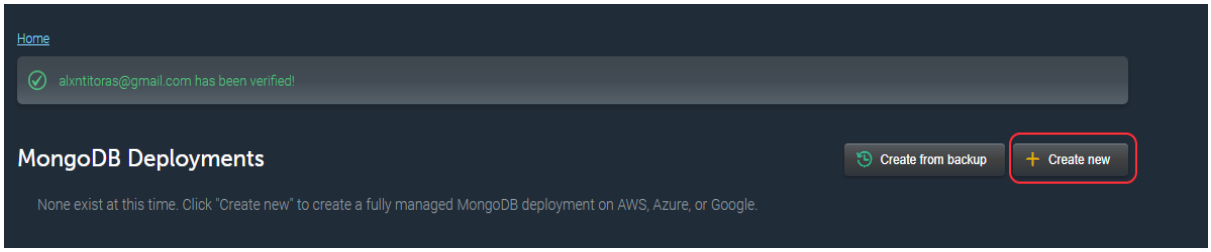
Στην συνέχεια συμπληρώνουμε τα στοιχεία μας και και αποδεχόμαστε τους όρους όπως και φαίνεται στην εικόνα 2.2 και πατάμε το κουμπί create account. Στο email μας θα λάβουμε ένα μήνυμα που θα επιβεβαιώσουμε την εγγραφή .



Εικόνα 2.2: Στοιχεία register

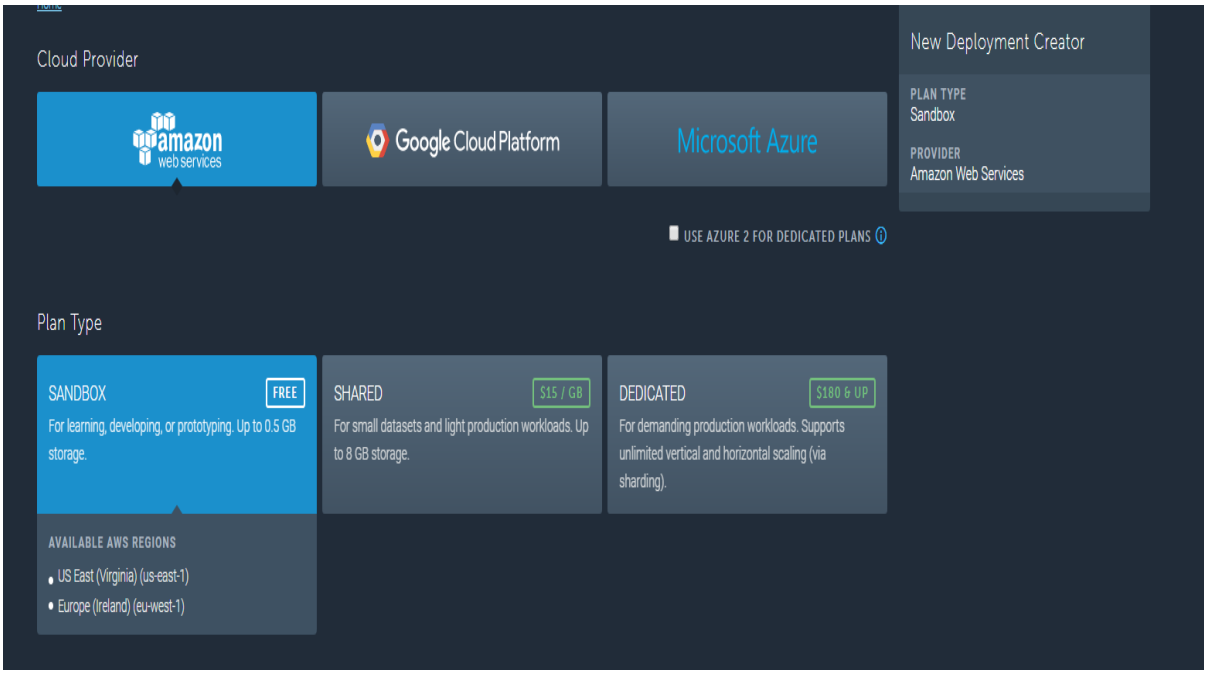
Μόλις πατήσουμε το create account και έχουμε συμπληρώσει σωστά τα στοιχεία μας θα ανακατευθύνουμε στην κεντρική οθόνη το mlab όπου και θα μπορέσουμε να

δημιουργήσουμε μια βάση δεδομένων πατώντας το κουμπί create new που βρίσκετε στα δεξιά μας όπως φαίνεται στην εικόνα 2.3.



Εικόνα 2.3: Δημιουργία νέας βάσης

Στην συνέχεια θα μεταβούμε στην οθόνη όπου θα πρέπει να επιλέξουμε ένα cloud provider και ένα plan type .Διαλέγουμε το amazon web services για cloud provider και το sand box για plan type όπως φαίνεται και στην εικόνα 2.4.

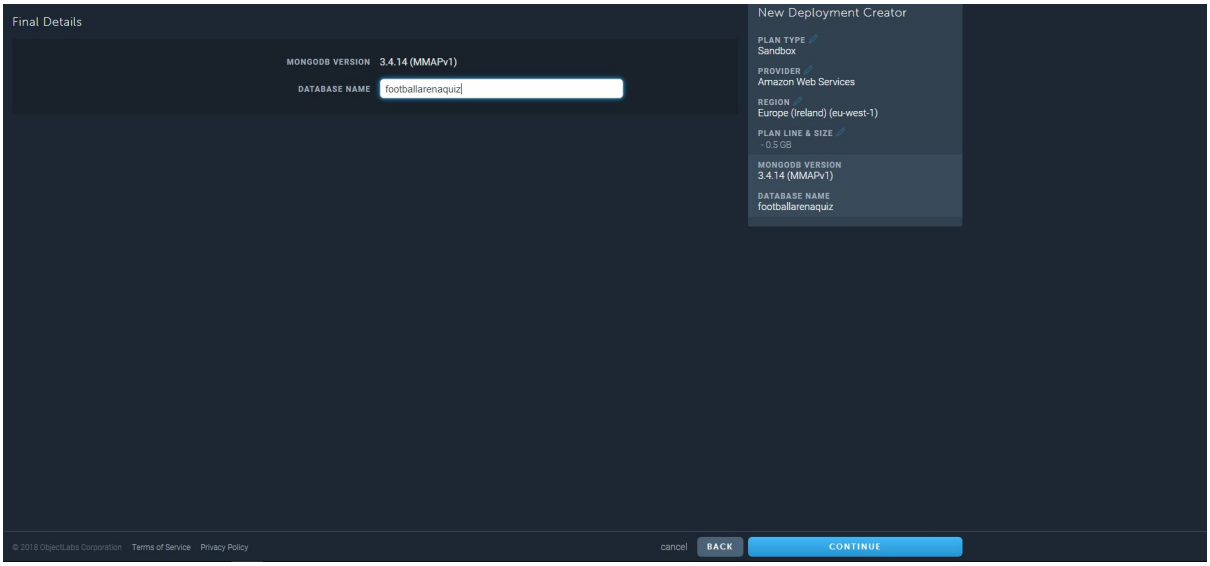


Εικόνα 2.4: Plan and Provider

Πατάμε continue και διαλέγουμε region Europe εικόνα 2.5 και πάλι πατάμε continue.

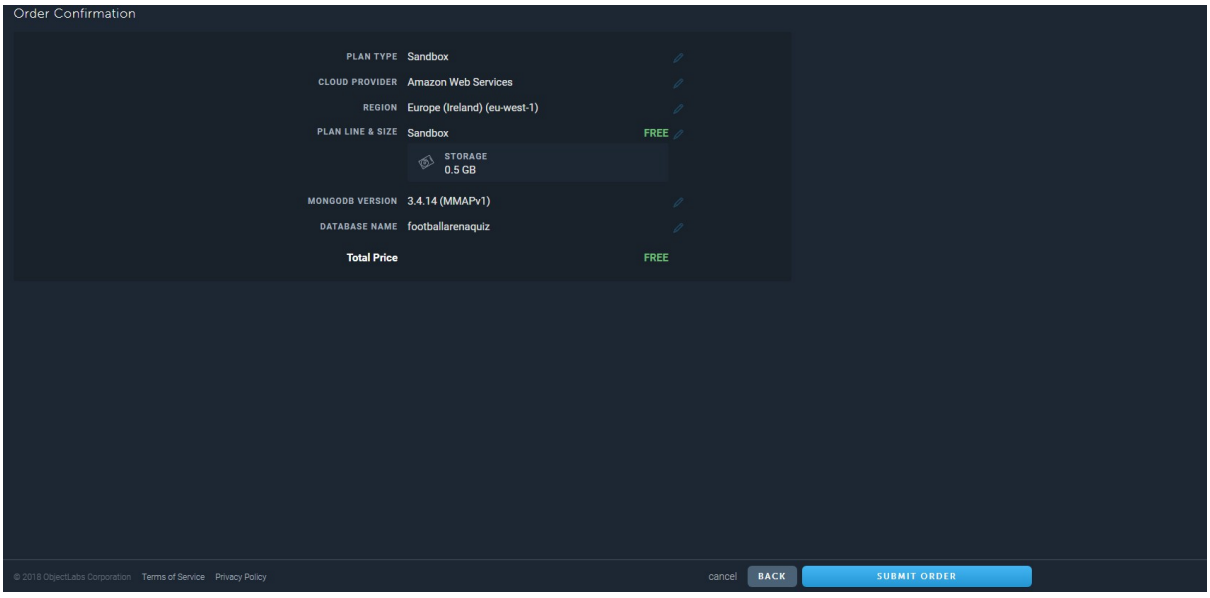
Εικόνα 2.5: region selection

Διαλέγουμε το όνομα της database που θέλουμε , το οποίο πρέπει να είναι μοναδικό και πατάμε continue.



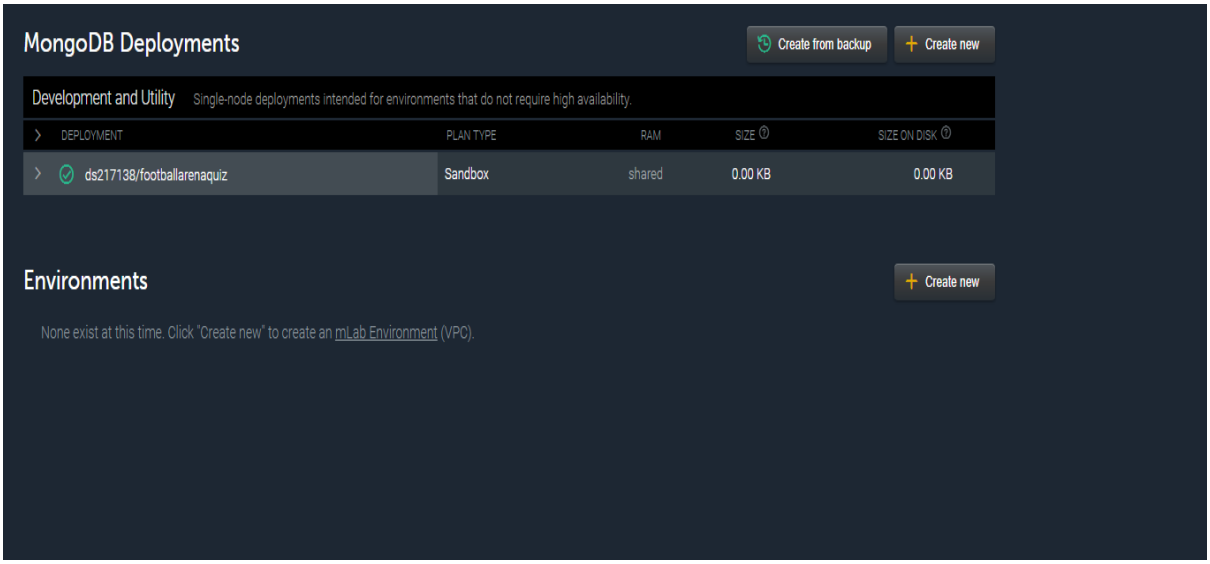
Εικόνα 2.6: database name

Τέλος βλέπουμε τα στοιχεία της βάσης μας και πατάμε submit order αν είναι όλα όπως τα θέλουμε εικόνα 2.7.



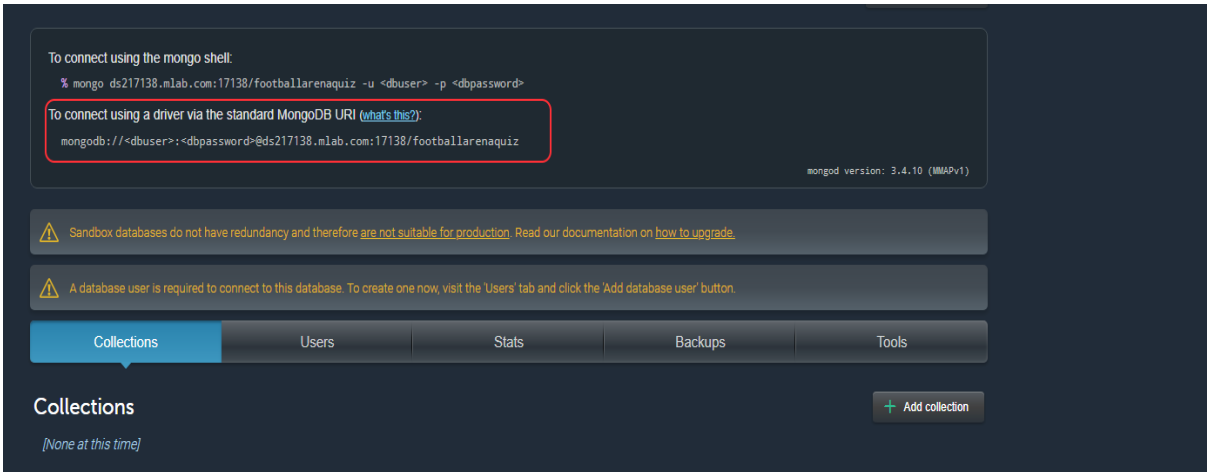
Εικόνα 2.7: Submit order

Αν όλα πήγαν καλά τότε θα πρέπει να βλέπουμε την βάση που δημιουργήσαμε στο home page του mlab εικόνα 2.8



Εικόνα *2.8:* mlab Database

Για να συνδέσουμε την βάση μας με τον backend server μας πρέπει να χρησιμοποιήσουμε το url που μας δίνει το mlab. Για να δούμε το url θα πρέπει να επιλέξουμε την βάση που φτιάξαμε στο προηγούμενο βήμα και αντιγράφουμε το url που βλέπουμε στην εικόνα 2.9 .Όπου <dbuser> βάζουμε το username που έχουμε στο mlab και όπου <dbpassword> τον κωδικό μας.



Εικόνα 2.9: mlab database url

Τέλος θα πρέπει να δημιουργήσουμε ένα user για την βάση μας , πατώντας την καρτέλα Users και την συνέχεια πατάμε το κουμπί add database user ,χρησιμοποιώντας τα στοιχεία που θέλουμε, θα πρέπει να χρησιμοποιηθεί κάτι που να θυμόμαστε γιατί θα χρησιμοποιηθούν αργότερα.Με τα παραπάνω βήματα ολοκληρώθηκε η εγγραφή και η δημιουργία βάσης δεδομένων στο mlab, επόμενο βήμα είναι η εγκατάσταση του nodeJs.

ΚΕΦΑΛΑΙΟ 3

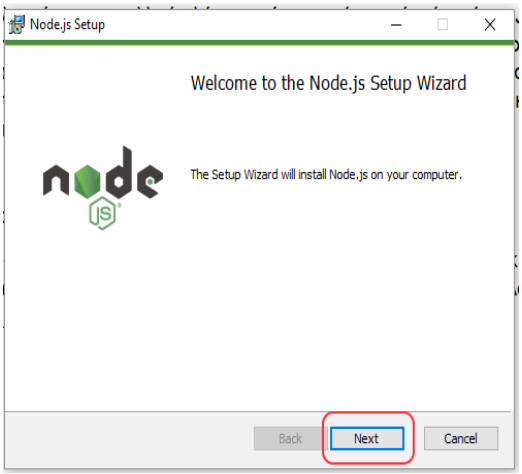
Εισαγωγή και εγκατάσταση του NodeJS

3.1 Ιστορική αναδρομή

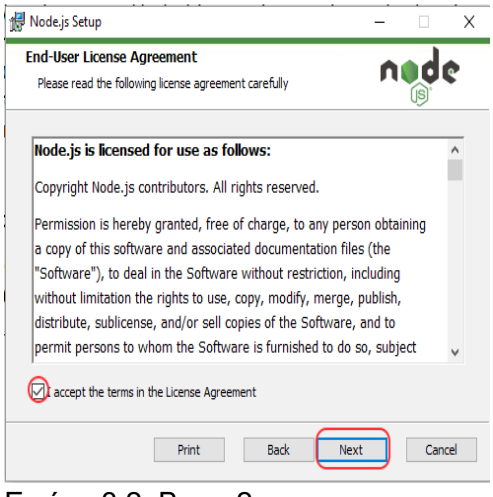
Το 2009 στο European JSConf ο Ryan Dahl παρουσίασε το project το οποίο δούλευε εκείνη την περίοδο. Το project χρησιμοποιούσε το Google v8 Javascript engine το event loop και ένα low-level I/O API .Χρησιμοποιούσε την απλότητα της Javascript για να δημιουργήσει event-driven server-side εφαρμογές. Το Node Js απέκτησε μεγάλη φήμη για πολλούς λόγους , ένας από αυτούς είναι ότι η Javascript είναι η ποίο χρησιμοποιούμενη γλώσσα. Άλλος λόγος είναι η απλότητα που παρέχει το Node όλα τα API που μας παρέχει είναι απλά και αν χρειαστεί να γράψουμε κάτι που είναι περισσότερο πολύπλοκο τότε μπορούμε να χρησιμοποιήσουμε κάποιο third-party library .

3.1.1 Εγκατάσταση του Node στα Windows .

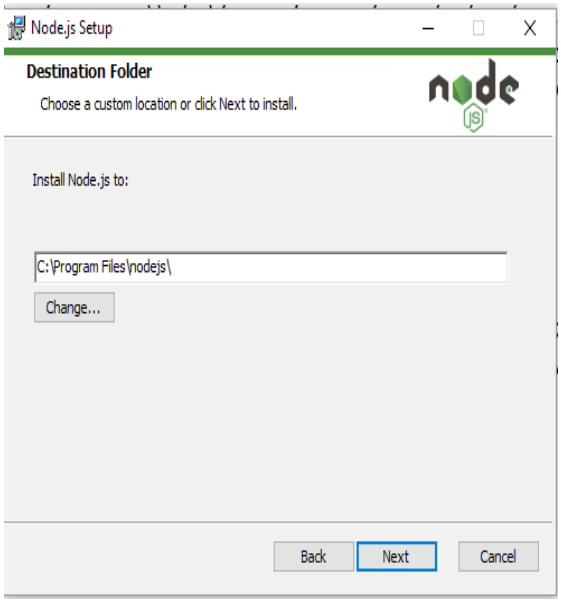
Το Node υποστηρίζετε από τα windows σχεδόν από όταν δημιουργήθηκε και ποιο συγκεκριμένα από την έκδοση 0.6.0 .Μπορούμε να το κατεβάσουμε εύκολα από το παρακάτω link *https://nodejs.org/en/download/*. Για να ολοκληρώσουμε την εγκατάσταση ακολουθάει τα παρακάτω βήματα στις εικόνες .



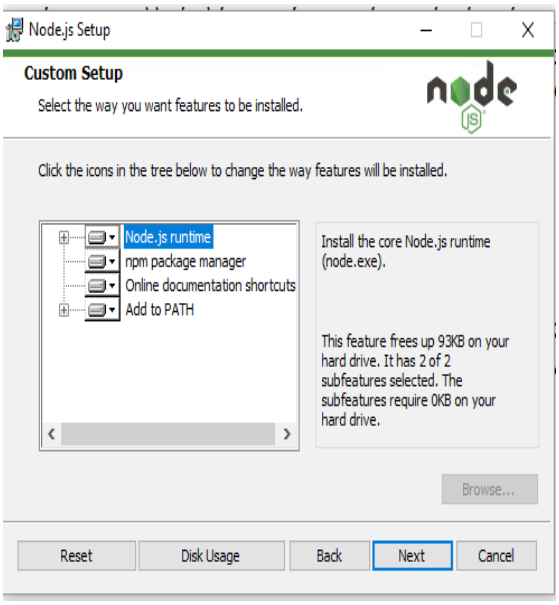
Εικόνα 3.1: Βημα 1



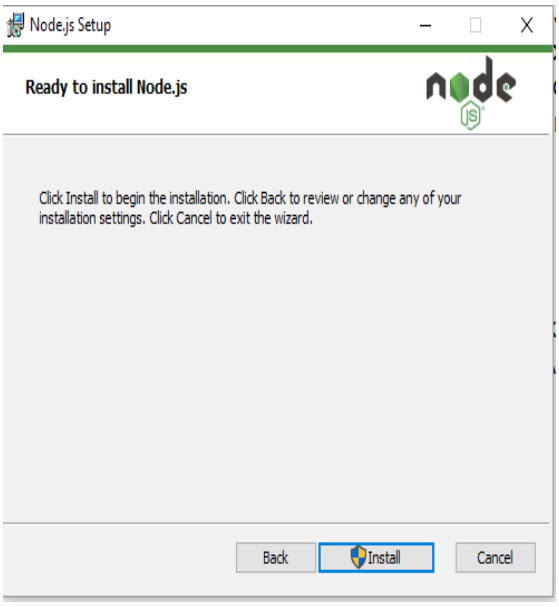
Εικόνα 3.2: Βημα 2



Εικόνα 3.3: Βημα 3

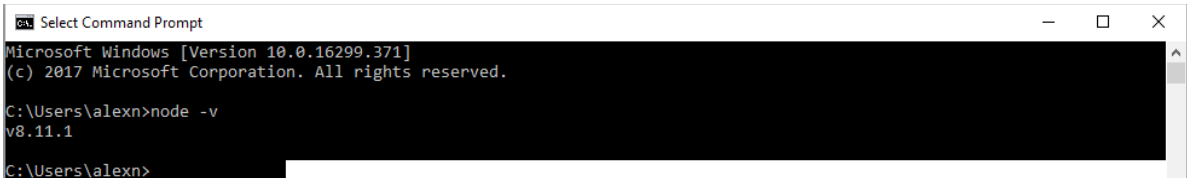


Εικόνα 3.4: Βημα 4



Εικόνα 3.5: Βημα 5

Αυτά τα 5 βήματα χρειάζονται μόνο για την εγκατάσταση του Node ,για να δούμε ότι όλα έχουν εγκατασταθεί σωστά ανοίγουμε το terminal και τρέχουμε την εντολή *node -v* με την οποία βλέπουμε την έκδοση του node που έχουμε στο σύστημα μας.



Εικόνα 3.6: node -v

Σημείωση οτί η έκδοση η δικία μου μπορεί να είναι διαφορετίκη απο την δικιά σας αναλόγως πότε κάνατε την εγκατάσταση.

3.1.2 Event-Driven προγραμματισμός

Στον Event-driven προγραμματισμό η ροή του προγράμματος εξαρτάται από τα events. Τα events χειρίζονται από τους event handler ή από κάποιο event callback. Ενα event callback είναι κάποια function η οποία καλείται όταν κάτι συγκεκριμένο συμβαίνει όπως ένα query είναι διαθέσιμο ή όταν πατηθεί κάποιο click.

Θεωρούμε ότι ένα query σε μια βάση πραγματοποιείτε με τον παρακάτω κώδικα :

```
result = query('SELECT * FROM posts WHERE id = 1');
do_something_with(result);
```

Χρησιμοποιώντας το παραπάνω query απαιτούμε από το thread ή το proccess να περιμένει μέχρι να ολοκληρωθεί η επεξεργασία του στην βάση .

Στα event driven συστήματα το query θα εκτελεστεί με τον παρακάτω κώδικα :

```
query_finished = function(result) {
do_something_with(result);
}
query('SELECT * FROM posts WHERE id = 1', query_finished);
```

Στον παραπάνω κώδικα στην μεταβλητή query_finished αποθηκεύουμε το τι θέλουμε να συμβεί και στην συνέχει το εισάγουμε σαν παράμετρο ,έτσι όταν το query ολοκληρωθεί θα καλεστεί και ο κώδικας μέσα στην μεταβλητή query finished .Με αυτόν τον τρόπο αντί απλά να επιστρέφουμε μια απλή τιμή ορίζουμε functions οι οποίες καλούνται αυτόματα από το σύστημα όταν κάποιο event συμβεί. Αυτό είναι ένα από τα κυριότερα και εξασφαλίζει ότι η η διεργασία δεν θα μπλοκάρει πουθενά όταν κάποια I/O διεργασία εκτελείτε, ως εκ τούτου μπορούμε να έχουμε ταυτόχρονα πολλές I/O διεργασίες που τρέχουν παράλληλα.

Το event loop συνοδεύεται στον event-driven προγραμματισμό .Αποτελείται από δύο κυρίως λειτουργίες που τρέχουν ατέρμονα η μία ελέγχει αν υπάρχουν events και η άλλη αν υπάρχει κάποιο που πρέπει να εκτελεστεί. Αν κάποιο event εκτελεστεί τότε βρίσκει ποιο είναι και εκτελεί το callback που πρέπει.

3.1.3 Closures

Οι Closures είναι function οι οποίες κληρονομούν μεταβλητές από το περιβάλλον στο οποίο περιβάλλονται. Αν εισάγουμε μια function callback ως παράμετρο σε μια άλλη function η οποία θα εκτελέσει κάποια I/O διεργασία, αυτή η callback function θα εκτελεστεί αργότερα και θα θυμάται όλες τις μεταβλητές από τον γονέα της. Στον παρακάτω κώδικα βλέπουμε ένα παράδειγμα closure :

```
var clickCount = 0;
$('button#mybutton').click(function() {
clickedCount ++;
```

```
alert('Clicked ' + clickCount + ' times.');
});
```

Στο παραπάνω παράδειγμα έχουμε ορίσει μια global μεταβλητή με το όνομα *clickCount* επίσης έχουμε ένα clickListener και έχουμε περάσει σαν παράμετρο μια function η οποία θα εκτελείτε κάθε φορά που κάνουμε click η function αυτή αυξάνει κατά ένα τον counter κάθε φορά που κάνουμε click, αυτή η function είναι ένα closure. Δεν είναι όμως 100% σωστό καλό θα ήταν να αποφεύγουμε να χρησιμοποιούμε global μεταβλητές το πρόβλημα αυτό λύνεται αν συμπεριλάβουμε τον κώδικα μέσα σε μια άλλη function έτσι ώστε να δημιουργήσουμε ένα επιπλέον closure.

```
(function() {
var clickCount = 0;
$('button#mybutton').click(function() {
clickCount ++;
alert('Clicked ' + clickCount + ' times.');
});
}());
```

3.2 Εισαγωγή στο Express.Js

Το express είναι ένα framework του NodeJS που μας βοηθάει να δημιουργήσουμε HTTP εφαρμογές. Βασίζεται στην Connect middleware engine. Εκτός από το να παρέχει τον μεσολαβητή για να απαντάει σε HTTP requests μας επιτρέπει και να εκτελούμε διάφορες εντολές ανάλογα με την μέθοδο και το URL. Επίσης μας επιτρέπει να καθιστά HTML σελίδες ανάλογα με τις παραμέτρους.

3.2.1 Εγκατάσταση του Express.Js

Για να εγκαταστήσουμε το express πρέπει να τρέξουμε την εντολή *npm install -g express@2.5.x* η οποία εγκαθιστά το express global στο σύστημα μας .

Τώρα απλα μπορούμε να μπούμε στην εφαρμογή που θέλουμε και να κάνουμε εγκατάσταση το express τρέχοντας την εντολή *express my_app*. Το αποτέλεσμα που θα πάρουμε θα είναι το παρακάτω :

create : my_app

create : my_app/package.json

create : my_app/app.js

create : my_app/public

create : my_app/public/javascripts

create : my_app/public/images

create : my_app/public/stylesheets

create : my_app/public/stylesheets/style.css

create : my_app/routes

create : my_app/routes/index.js

create : my_app/views

create : my_app/views/layout.jade

create : my_app/views/index.jade

Δημιουργήσαμε έναν φάκελο με το όνομα my_app και μέσα περιέχονται όλοι αυτοι οι φάκελοι που φαίνονται παραπάνω.

Το αρχείο package.json είναι ένα αρχείου μανιφέστου και αποτελείται από :

```
{
"name": "application-name"
, "version": "0.0.1"
, "private": true
```

```
, "dependencies": {
"express": "2.5.11"
, "jade": ">= 0.0.1"
}
}
```

Ο σκοπός του παραπάνω μανιφέστου είναι να περιέχει τις λίστες βιβλιοθηκών που το NPM χρησιμοποιεί για να κάνει εγκατάσταση τις βιβλιοθήκες .Ας κάνουμε εγκατάσταση αυτές τις βιβλιοθήκες εκτελώντας τις παρακάτω εντολές *npm install* .Με αυτές τις εντολές κάνουμε εγκατάσταση το module του express και την Jade. Όπως βλέπουμε ο φάκελος *public* περιέχει φακέλους για τα στατικά αρχεία τα οποία μπορεί να χρειαζόμαστε κατά κάποιον τρόπο είναι το βασική δομή για να ξεχωρίζουμε την τοποθεσία των css,εικόνων ή άλλων client-side javascript αρχείων .

Το αρχείο *app.js* περιέχει τον κώδικα αρχικοποίησης του server μας ο κώδικας του είναι κάπως έτσι :

```
/**
* Module dependencies.
*/
var express = require('express')
, routes = require('./routes');
var app = module.exports = express.createServer();
// Configuration
app.configure(function(){
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(__dirname + '/public'));
```

```
});
        app.configure('development', function(){
app.use(express.errorHandler({ dumpExceptions: true, showStack: true }));
});
app.configure('production', function(){
app.use(express.errorHandler());
});
// Routes
app.get('/', routes.index);
app.listen(3000, function(){
console.log("Express server listening on port %d in %s mode", app.address().port,
app.settings.env);
});
```

Αρχικοποιούμε τον server με την εντολή express.createServer().Ο server αυτός κληρονομεί από το Connect HTTP και αυτός με την σειρά του από τον Node HTTP server. Στην συνέχεια αρχικοποιούνται τα views και ορίζετε η view engine η οποία είναι η Jade.

3.2.2 Χρησιμοποιόντας Middleware στην εφαρμογή μας.

Συνεχίζοντας από την προηγούμενη ενότητα μπορούμε να δούμε στον κώδικα ότι ήδη έχε χρησιμοποιηθεί το middleware για την δημιουργία του server και αυτό το βλέπουμε από τον παρακάτω κώδικα :

```
app.configure(function(){
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(__dirname + '/public'));
```

});

Αυτός ο κώδικας είναι περιστοιχισμένος μέσα σε μια callback funtion .Μέσα στην callback χρησιμοποιούμε το body parser middleware το οποίο χρησιμεύει στο να κάνει parse τα ερωτήματα από τον client ανάλογα με το τη περιεχόμενο έχει το ερώτημα.

Στην συνέχεια το method ovveride middleware προστίθεται στην εφαρμογή. Ο σκοπός αυτού του middleware είναι να προσθέσει στην εφαρμογή μας έξτρα HTTP μεθόδους εκτός από το GET,POST όπως το DELETE και PUT.

Επόμενο είναι το router middleware , το οποίο διαχειρίζεται το που θα καταλήξουν τα κάθε request που έχουν οριστεί στον πίνακα δρομολόγησης ανάλογα με το URL ή την μέθοδο που χρησιμοποιήθηκε.

Τέλος το static.express middleware στήνει έναν server για στατικά αρχεία που είναι υπεύθυνος για τα στατικά αρχεία μέσα στον φάκελο public.

3.2.3 Χειρισμός Δρομολόγησης.

Στην γραμμή 32 του app.j βλέπουμε τον κώδικα *app.get('/',routes.index)* .Αν δεχτούμε κάποιο get request θα εκτελεστεί το ο χειριστής routes.index.

Αλλάζουμε τον κώδικα μέσα στο *routes/index.js* σε:

```
module.exports = function(app) {
app.get('/', function(req, res){
res.render('index', { title: 'Express' })
});
};
```

Αλλάζουμε την γραμμή 32 του app.js σε *require('./routes/index')(app)* .Φτιάχνουμε και τα καινούργια routes στον φάκελο *routes/users.js* :

```
module.exports = function(app) {
```

```
app.get('/users/:name', function(req, res){
res.render('users/profile', {title: 'User profile'});
});
};
```

Προσθέτουμε στο app.js και κάτω απο την γραμμή 32 *require('./routes/users')(app).*

Ορίσαμε δύο χειριστές δρομολόγησης για την εφαρμογή μας. Ο πρώτος ενεργοποιείται όταν σε ένα GET request έχουμε το endpoint /users. Σε αυτή την περίπτωση φορτώνετε το *users/index* μαζί με την παράμετρο του title.

Το επόμενο route είναι δυναμικό ταιριάζει οποιοδήποτε route ξεκινάει με /users και ακολουθείται από ένα string. Αυτο το string θα είναι διαθέσιμο από τον χειριστή δρομολόγησης στο *req.params.name* .

3.3 Εισαγωγή στο MongoDB με το Express .

Παραπάνω είδαμε πως μπορούμε να φτιάξουμε ένα MonogDB server και τοπικά αλλά και σε cloud. Εγκαταστήσαμε το express, δημιουργήσαμε τον http server. Το μόνο που μένει είναι πως θα συνδέσουμε τις δυο αυτές τεχνολογίες. Αυτό θα δούμε σε αυτό το κεφάλαιο γνωρίζοντας ακόμα δυο βιβλιοθήκες του nodeJs το Mongoose και το express generator.

3.3.1 Εισαγωγή και εγκατάσταση του Mongoose .

Η βιβλιοθήκη Mongoose μας επιτρέπει να επικοινωνούμε με την MongoDB βάση χρησιμοποιώντας μοντέλα αντικειμένων .Η εγκατάσταση του mongoose δεν διαφέρει σε τίποτα από κάθε άλλη εγκατάσταση βιβλιοθήκης στο NodeJs, εκτελούμε την εντολή *npm install mongoose* και το mongoose είναι έτοιμο για χρήση .

Το mongoose είναι ένα εργαλείο για την μοντελοποίηση δεδομένων που έχουν αλληλεπίδραση με την βάση δεδομένων. Ορίζουμε σχήματα αρχείων (document schemas). Κάθε σχήμα μπορεί να έχει τα δικά του πεδία και περιορισμούς όπως

ελάχιστη τιμή που μπορούμε να εισάγουμε ή αν κάποια τιμή είναι υποχρεωτική για να συμπληρωθεί.

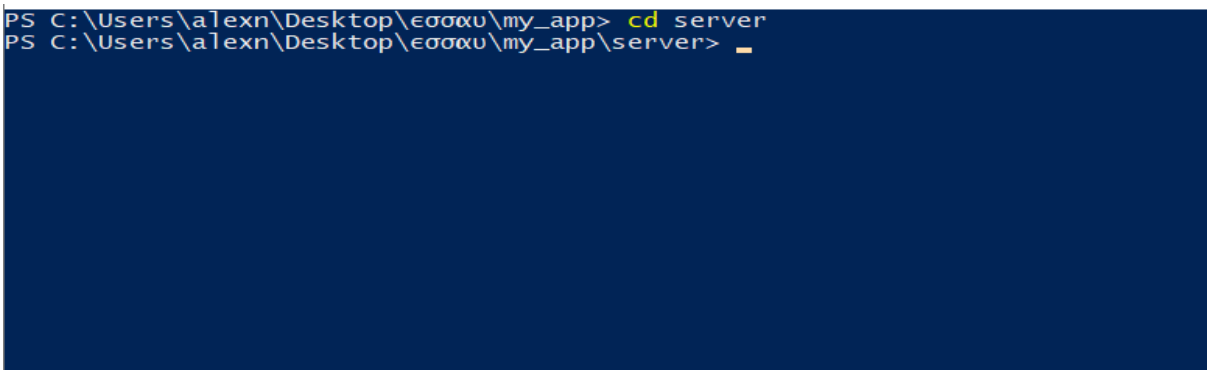
3.3.2 Σύνδεση Express και MongoDB .

Για λόγους ευκολίας και ταχύτητας θα εγκαταστήσουμε το express genarator η οποία είναι μια βιβλιοθήκη για να δημιουργούμαι ευκολότερα και γρηγορότερα έναν express server, εκτελούμε την εντολή *npm install express-generator -g* .Στην συνέχεια δημιουργούμε έναν καινούργιο φάκελο ο οποίος θα φιλοξενήσει το παράδειγμα μας με όνομα my_app. Εκτελούμε την κονσόλα των windows μέσα σε αυτό τον φάκελο και εκτελούμε την εντολή *express --view=pug server.* Το πρώτο όρισμα που έχουμε τοποθετήσει είναι για την μηχανή που θέλουμε να χρησιμοποιήσουμε έχει μικρή σημασία τη θα επιλέξουμε εφόσον αργότερα θα χρησιμοποιήσουμε την Angular 4 για το front end, το δεύτερο (server) είναι το όνομα του φακέλου που θέλουμε να εγκατασταθεί ο server.

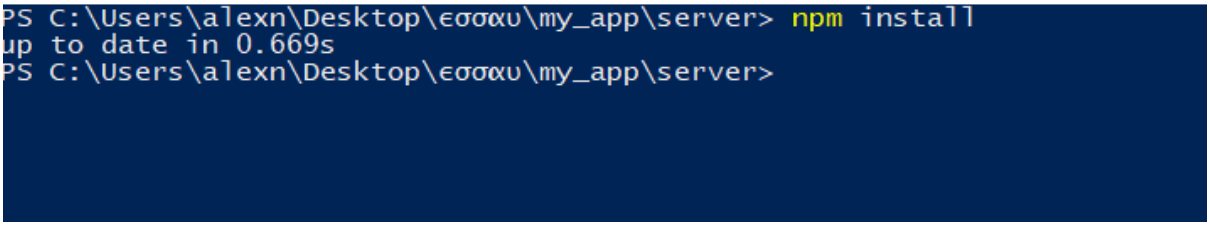
```
PS C:\Users\alexn\Desktop\εσσαυ\my_app> express --view=pug server

   create : server
   create : server/package.json
   create : server/app.js
   create : server/public
   create : server/routes
   create : server/routes/index.js
   create : server/routes/users.js
   create : server/views
   create : server/views/index.pug
   create : server/views/layout.pug
   create : server/views/error.pug
   create : server/bin
   create : server/bin/www
   create : server/public/javascripts
   create : server/public/images
   create : server/public/stylesheets
   create : server/public/stylesheets/style.css

   install dependencies:
     > cd server && npm install

   run the app:
     > SET DEBUG=server:* & npm start
```

Εικόνα 3.7: Δημιουργια server

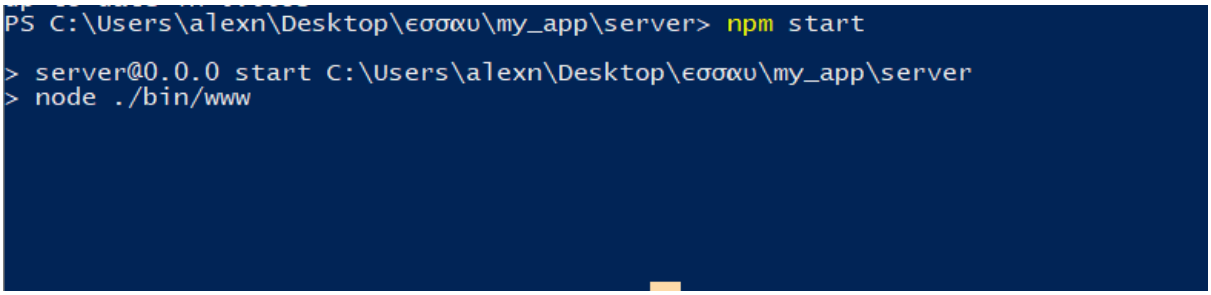
Εφόσον δημιουργήσαμε σωστά τον server μας θα πρέπει να βλέπουμε ότι και στην εικόνα 3.7. Κατευθυνόμαστε στον φάκελο server που δημιουργήσαμε και τρέχουμε την εντολή *npm install (*εικόνα 3.7,3.8). Εκτελούμε την εντολή *npm start* και ο server τόσο εύκολα ξεκινάει να τρέχει (εικόνα 3.9)!



Εικόνα 3.8: Αλλάγη φακέλου

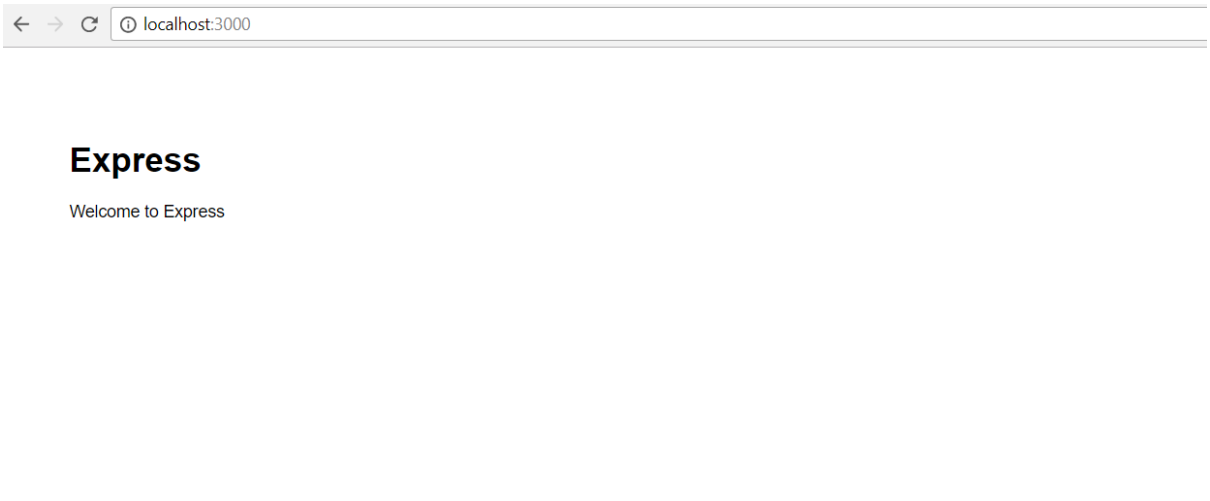


Εικόνα 3.9 Εγκατασταση βιβλιοθηκών



Εικόνα 3.10*:* εκκίνηση server

Για να διαπιστώσουμε ότι ο server τρέχει σωστά όντως ανοίγουμε οποιονδήποτε browser και κατευθυνόμαστε στην διεύθυνση http://localhost:3000/ .Θα πρέπει να έχουμε ένα αποτέλεσμα σαν την εικόνα 3.11 .



Εικόνα 3.11*:* localhost:3000

Επόμενο βήμα είναι να συνδέσουμε τον server που δημιουργήσαμε με το mongoDB και το mlab με την βοήθεια του mongoose. Ανοίγουμε το αρχείο app.js και προσθέτουμε τις εξής γραμμές κώδικα κάτω από την γραμμή *var app=exrpess();*

```
mongoose.connect('mongodb://<username>:<password>@<yourdatabaseurl>').the
n(
() => { console.log('You  logged in to your database!!!')},
err => { }
);
```

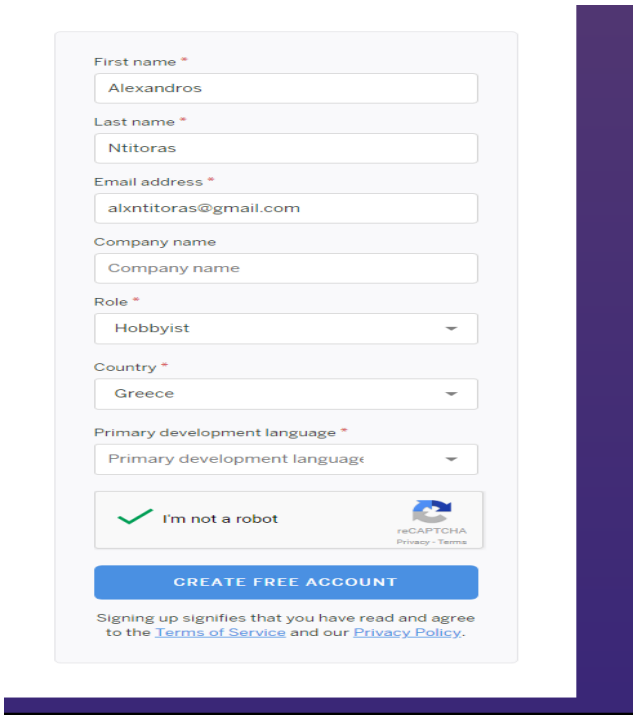
Όπου username είναι το username που είχαμε βάλει σε προηγούμενο κεφάλαιο στο mlab όπως και το password, το *yourdatabaseurl* είναι το url της βάσης που είχαμε δημιουργήσει. Για να ανανεωθεί ο κώδικας που αλλάξαμε στον server μας, θα πρέπει να επανεκκινήσουμε τον server πατώντας στην κονσόλα πάνω στην οποία τρέχει ο server το ctrl+c. Αφού το κάνουμε τότε ξαναπατάμε την εντολή npm start για να ξανά ξεκινήσει ο server μας με τις αλλαγές που κάναμε .Αν όλα πήγαν καλά θα πρέπει να δούμε στην κονσόλα το μήνυμα *“You logged in to your database!!!”* . Τόσο εύκολα φτιάξαμε τον server μας και συνδεθήκαμε στην cloud βάση μας! .

```
C:\Users\alexn\Desktop\εσσαυ\my_app\server>npm start

> server@0.0.0 start C:\Users\alexn\Desktop\εσσαυ\my_app\server
> node ./bin/www

You  logged on to your database!!!
```

Εικόνα 3.12: database connect

3.3.3 Ανάπτυξη του server στο Heroku .

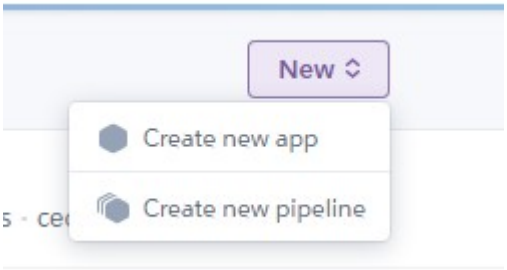
Ο server μας αυτή τη στιγμή τρέχει τοπικά και δεν είναι διαθέσιμος στους χρήστες έξω από το τοπικό μας δίκτυο. Για να λυθεί αυτό το πρόβλημα θα πρέπει να αναπτυχθεί σε ένα cloud περιβάλλον. Η λύση είναι να χρησιμοποιήσουμε το Heroku για να αναπτύξουμε την εφαρμογή μας. Το Heroku μας παρέχει ένα μικρό χώρο δωρεάν έτσι ώστε να το δοκιμάσουμε, ο χώρος αυτός είναι αρκετός για να αναπτύξουμε την εφαρμογή μας. Πρώτο βήμα είναι να επισκεφτούμε το site του Heroku και να κάνουμε εγγραφή όπως βλέπουμε στην εικόνα 3.13.



Εικόνα 3.13: heroku sign up

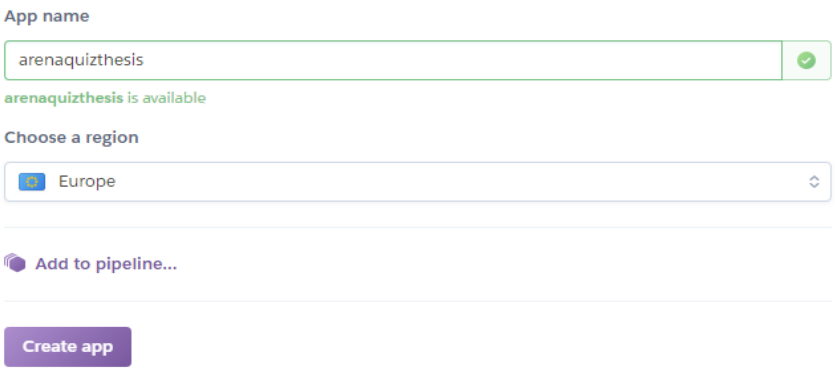
Εφόσον η εγγραφή έγινε σωστά τότε θα πρέπει να έχουμε ανακατευθυνθεί στην σελίδα *https://dashboard.heroku.com/apps* από αυτή την σελίδα μπορούμε να

δημιουργήσουμε μια νέα εφαρμογή Heroku η οποία θα φιλοξενήσει τον NodeJs Server μας. Πατάμε το κουμπί new το οποίο βρίσκεται στην πάνω δεξιά γωνία της οθόνης και στην συνέχεια πατάμε το Create new app εικόνα 3.14 .



Εικόνα 3.14: new

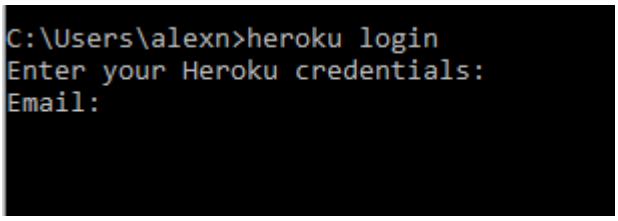
Διαλέγουμε το όνομα της εφαρμογής που θέλουμε π.χ arenaquizthesis διαλέγουμε region Europe και στην συνέχεια πατάμε create app. Αν όλα πήγανε καλά τότε ο χώρος που θα φιλοξενήσει την εφαρμογή μόλις δημιουργήθηκε. Επόμενο βήμα είναι να ανεβάσουμε τον server μας στο Heroku .



Εικόνα 3.15: λεπτομέρειες εφαρμογής

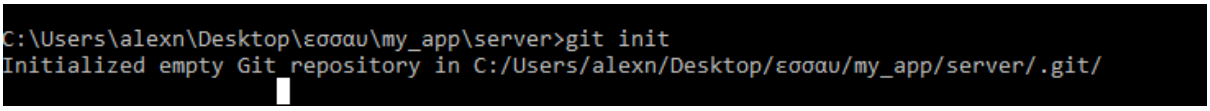
Το Heroku μας δίνει πολλές επιλογές για τον τρόπο που θέλουμε να αναπτύξουμε την εφαρμογή , εμείς θα επιλέξουμε την μέθοδο του Heroku git. Πρώτο βήμα ειναι

να κατεβάσουμε και να εγκαταστήσουμε το Heroku cli από το παρακάτω link https://devcenter.heroku.com/articles/heroku-command-line. Αφού το εγκαταστήσουμε τότε θα πρέπει να πραγματοποιήσουμε είσοδο στο heroku μέσω της κονσόλας των windows στο προσωπικό μας υπολογιστή με την εντολή *heroku login* εικόνα 3.16 .



Εικόνα 3.16: Heroku login

Αν όλα πάνε καλά τότε θα δούμε ένα μήνυμα Logged in as “your email”. Επόμενο βήμα είναι να κατευθυνθούμε μέσω της κονσόλας των windows στον φάκελο που είναι ο nodeJs server μας και να τρέξουμε την εντολή *git init* εικόνα 3.17 .



Εικόνα 3.17: Git init

Στην συνέχεια τρέχουμε την εντολή για να γίνει η σύνδεση του sever μας με τον χώρο που έχουμε δημιουργήσει , η εντολή είναι *heroku git:remote -a arenaquizthesis* όπου arenaquizthesis το όνομα της δικιά σας εφαρμογής εικόνα 3.18.

Εικόνα 3.18: Heroku git:remote -a arenaquizthesis

Μέσα στον root του server μας προσθέτουμε ένα αρχείο με το όνομα. gitignore και μέσα προσθέτουμε την γραμμή κώδικα *node_modules/* έτσι ώστε όταν γίνει το push στο heroku να αγνοηθεί ο φάκελος *node_modules* ,αυτό το κάνουμε γιατί όταν ανεβάζουμε τα αρχεία μας στο heroku αυτό κάνει εγκατάσταση του server μέσω του

package.json οπότε έτσι και αλλιώς τα αρχεία *node_modules* δημιουργούνται εκ νέου. Εκτελούμε τις εντολές *git add* , *git commit -am "first commit"* , *git push heroku master* εικόνες *3.19,3.20,3.21* . Ο server είναι έτοιμος και προσβάσιμος από οποιοδήποτε σημείο. Για να δούμε την εφαρμογή μας πατάμε το κουμπί open app που βρίσκεται στην δεξιά γωνία , πατώντας το βλέπουμε ακριβώς ότι και στο local host με την διαφορά οτί πλέον μπορούμε να δεχτούμε request από παντού.

```
C:\Users\alexn\Desktop\εσσαυ\my_app\server>git add .
warning: LF will be replaced by CRLF in app.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in bin/www.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package-lock.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/stylesheets/style.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in routes/index.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in routes/users.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in views/error.pug.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in views/index.pug.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in views/layout.pug.
The file will have its original line endings in your working directory.
```

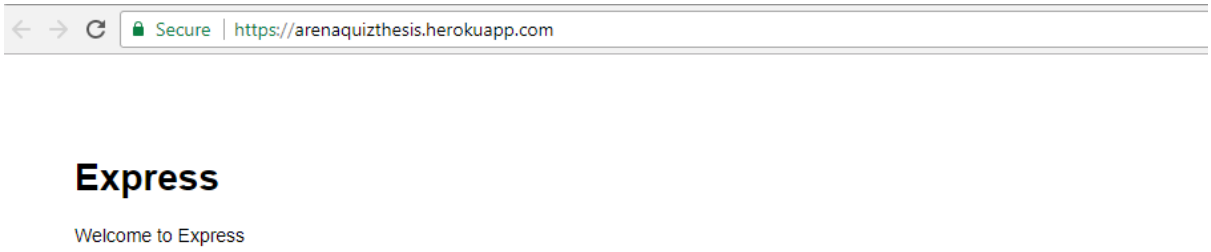
Εικόνα 3.19: Git add

```
C:\Users\alexn\Desktop\εσσαυ\my_app\server>git commit -am "first commit"
[master (root-commit) 6983744] first commit
warning: LF will be replaced by CRLF in app.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in bin/www.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package-lock.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/stylesheets/style.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in routes/index.js.
```

Εικόνα 3.20: Git commit -am "first commit"

```
C:\Users\alexn\Desktop\εσσαυ\my_app\server>git push heroku master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 299 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:        NPM_CONFIG_LOGLEVEL=error
remote:        NODE_VERBOSE=false
remote:        NODE_ENV=production
remote:        NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:        engines.node (package.json):  unspecified
remote:        engines.npm (package.json):   unspecified (use default)
remote:
remote:        Resolving node version 8.x...
remote:        Downloading and installing node 8.11.1...
remote:        Using default npm version: 5.6.0
remote:
remote: -----> Restoring cache
```

Εικόνα 3.21: Git push heroku master



Εικόνα 3.22: heroku express online

3.5 Περίληψη

Ο event-driven προγραμματισμός κάνει την ζωή των προγραμματιστών ευκολότερη κάνοντας της ασύγχρονες function ευκολότερες να διαχειριστούν και δίνει το πλεονέκτημα ότι δεν χρειάζονται πολλαπλά threads ή διεργασίες όταν θέλουμε να επεκτείνουμε τον κώδικα.

Στο κεφάλαιο αυτό κάναμε μια μικρή στο εισαγωγή στο expressJS ,είδαμε πώς μπορούμε να το εγκαταστήσουμε σε ένα project, πως να δημιουργήσουμε το server και πως αυτός ο server διαχειρίζεται τα στατικά αρχεία και τα routes. Στο επόμενο κεφάλαιο θα δούμε πως συνδέσουμε το MongoDB με τον express server .

Δημιουργήσαμε ένα express server με την βοήθεια του express generator ,στην συνέχεια με την βοήθεια του mongoose συνδέσαμε τον server με την βάση δεδομένων που είχαμε δημιουργήσει στο mlab και ανεβάσαμε τον server στον cloud πάροχο Heroku έτσι ώστε η εφαρμογή να είναι προσβάσιμη και εκτός του τοπικού δικτύου. Στο επόμενο κεφάλαιο θα δούμε τα βασικά για το front end framework που θα χρησιμοποιήσουμε του Ionic 3.

ΚΕΦΑΛΑΙΟ 4

Angular 5 και Ionic

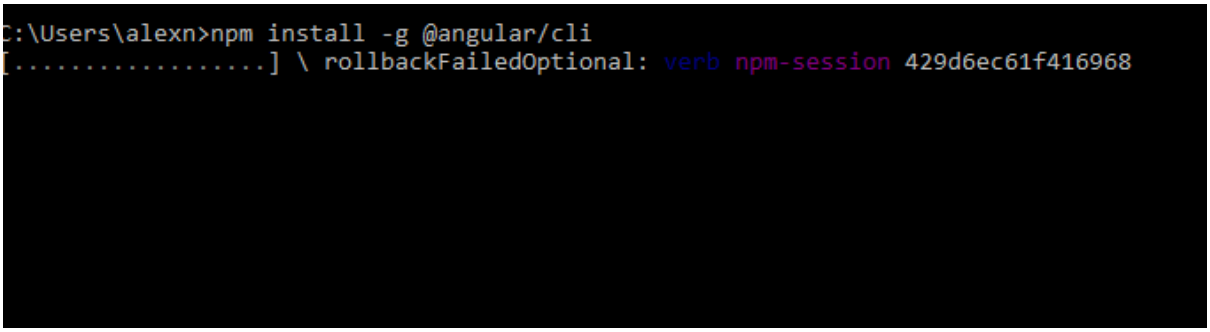
4.1 Εισαγωγή στην Angular 5 και στο Ionic 3

Η **Angular 5** αποτελεί ένα javascript open source framework η γλώσσα που χρησιμοποιεί είναι η typescript η οποία θα μπορούσαμε να πούμε ότι είναι μια ποιό εξελιγμένη javascript με την διαφορά ότι έχει types. Το framework **Angular 5** αποτελεί εξέλιξη του **AngularJS** αν και αυτά τα δυο μεταξύ τους δεν έχουν καμία απολύτως σχέση ως τον τρόπο που γράφουμε τον κώδικα . Οι προγραμματιστές της Google η οποία και έχει δημιουργήσει το framework της **Angular** αποφάσισαν να αλλάξουν τόσο πολύ την **Angular 5** για το λόγο ότι η AngularJS υστερούσε σε σύγκριση με άλλα δημοφιλή framework όπως το React.

Το **Ionic 3** είναι ένα framework για να δημιουργούμε υβριδικές εφαρμογές με τον όρο υβριδικές εφαρμογές εννοούμε τις εφαρμογές που μπορούν να τρέξουν ταυτόχρονα σε πολλές πλατφόρμες που έχουν παραχθεί από τον ίδιο κώδικα .Για παράδειγμα όταν γράφουμε μια εφαρμογή στο **Ionic** τότε μπορούμε να επιλέξουμε αν θέλουμε η εφαρμογή αυτή να τρέχει στο Android στο IOS ή ακόμα και σαν κανονική web app σε κάποιο περιηγητή .Για να χρησιμοποιήσουμε το **Ionic** θα πρέπει να έχουμε γνώση της Angular 5 καθώς αυτή είναι η γλώσσα που υποστηρίζει .Το Ionic χρησιμοποιεί το framework **Cordova** της **Apache** το οποίο μας παρέχει βιβλιοθήκες με τις οποίες μπορούμε να επικοινωνήσουμε με το hardware του κινητού δηλαδή μπορούμε να έχουμε πρόσβαση στους αισθητήρες ή στην κάμερα του κινητού .

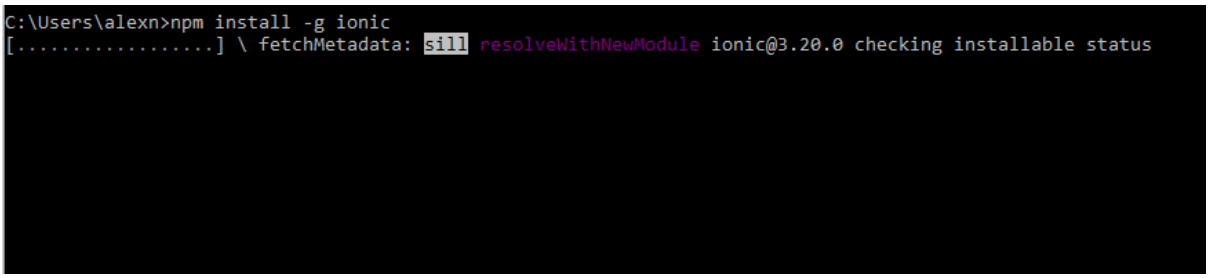
4.1.1 Εγκατάσταση της Angular 5 και του Ionic 3

Για την εγκατάσταση της Angular θα πρέπει να τρέξουμε την εντολή *npm install -g @angular/cli* στη κονσόλα των windows , παρατηρούμε την παράμετρο -g η όποια σημαίνει ότι η βιβλιοθήκη θα εγκατασταθεί global και θα έχουμε πρόσβαση σε αυτή από παντού στο σύστημα μας.



Εικόνα 4.1: Npm instal angular cli

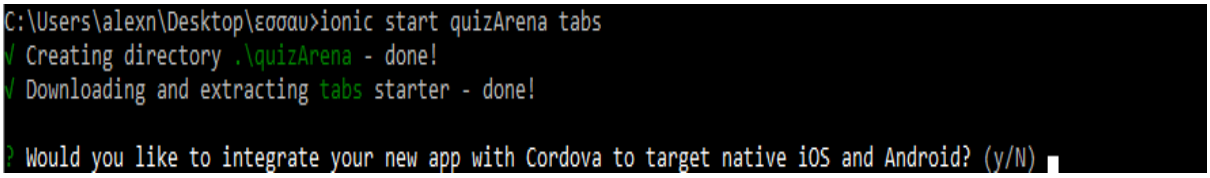
Στην συνέχεια κάνουμε εγκατάσταση την βιβλιοθήκη του Ionic εκτελώντας την εντολή *npm install -g ionic* από την κονσόλα των widows.



Εικόνα 4.2: Npm install -g ionic

4.1.2Δημιουργία εφαρμογής με το Ionic .

Για την δημιουργία βασικού project εκτελούμε την εντολή *Ionic start quizArena* στην κονσόλα των windows. Στην ερώτηση “Would you like to intergrate your new app wih cordova to target native Ios and android?” πατάμε N και περιμένουμε να γίνει η εγκατάσταση.

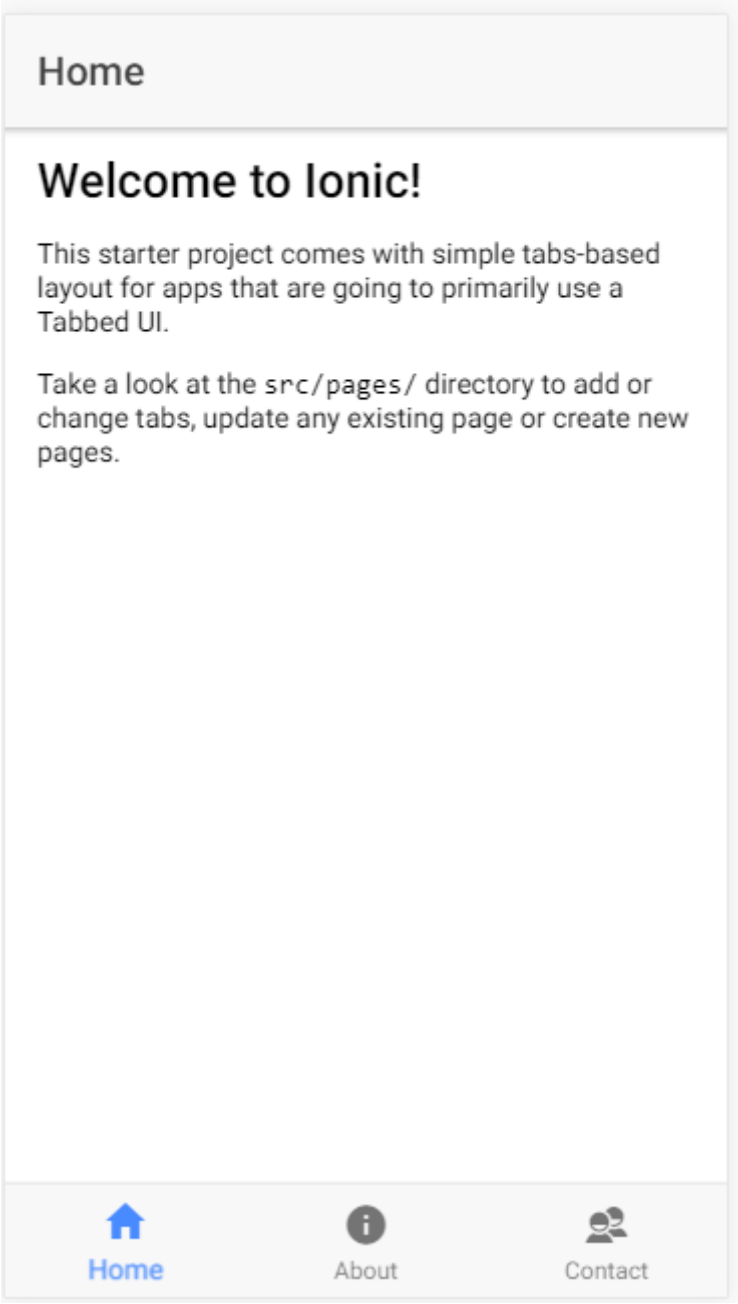


Εικόνα 4.3: Ionic start quizArena tabs

Κατευθυνόμαστε στον φάκελο που μόλις δημιουργήθηκε και εκτελούμε την εντολή *ionic serve* και κατευθυνόμαστε στην διεύθυνση http://localhost:8100 και τόσα απλά δημιουργήσαμε μια βασική εφαρμογή Ionic.

```
C:\Users\alexn\Desktop\εσσαυ\quizArena>ionic serve
Starting app-scripts server: --address 0.0.0.0 --port 8100 --livereload-port 35729 --dev-logger-port 53703 --nobrowser -

Ctrl+C to cancel
[16:23:38]  watch started ...
[16:23:38]  build dev started ...
[16:23:38]  clean started ...
[16:23:38]  clean finished in 4 ms
[16:23:39]  copy started ...
[16:23:39]  deeplinks started ...
[16:23:39]  deeplinks finished in 20 ms
[16:23:39]  transpile started ...
[16:23:43]  transpile finished in 4.24 s
[16:23:43]  preprocess started ...
[16:23:43]  preprocess finished in 1 ms
[16:23:43]  webpack started ...
[16:23:43]  copy finished in 4.49 s
```

Εικόνα 4.4: Ionic serve



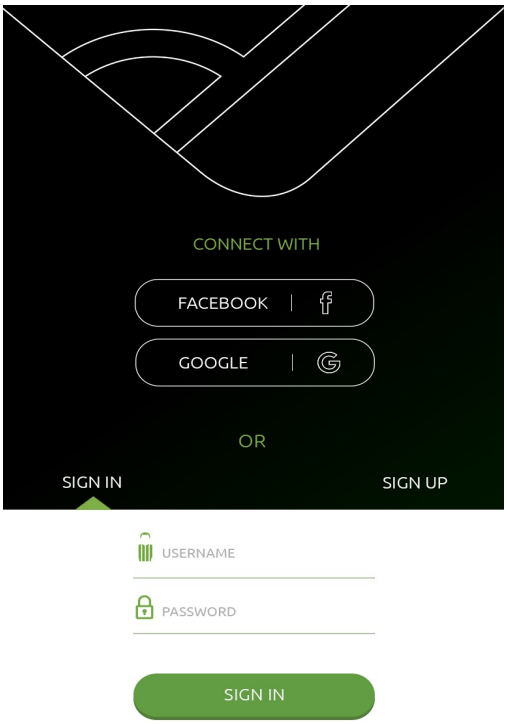
Εικόνα 4.5: localhost:8100

4.1.3 Χαρακτηριστικά και απαιτήσεις της εφαρμογής .

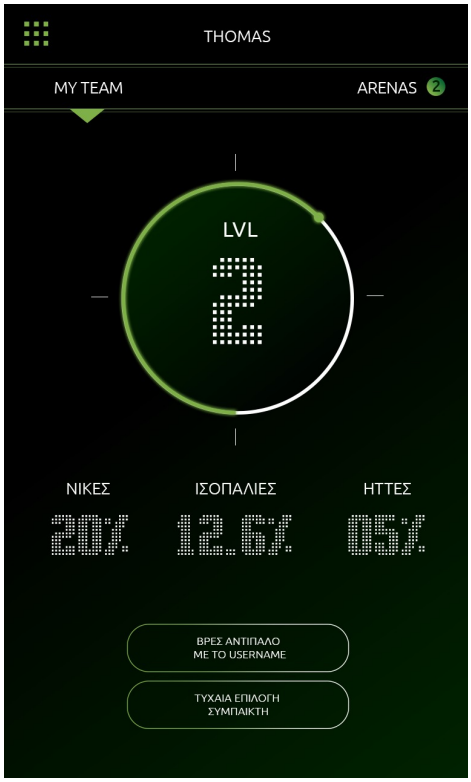
Η εφαρμογή που θα αναπτύξουμε είναι ένα ποδοσφαιρικό κουίζ πραγματικού χρόνου , όπου οι χρήστες μας θα πραγματοποιούν είσοδο στην εφαρμογή με το προσωπικό του email και τον κωδικό πρόσβασης τους .Ο κάθε χρήστης θα έχει την δυνατότητα να παίξει αντίπαλος με κάποιον τυχαίο χρήστη ή με κάποιον φίλου του εισάγοντας το username του και να εισέλθουν σε μια ποδοσφαιρική αρένα όπου θα αγωνιστούν σε 10 ποδοσφαιρικές ερωτήσεις. Η κάθε ερώτηση έχει 4 πιθανές απαντήσεις που μόνο η μία είναι σωστή. Ο κάθε χρήστης όσο απαντάει σωστά τις ερωτήσεις συνεχίζει κανονικά το παιχνίδι. Ο κάθε παίκτης έχει να επιλέξει ανάμεσα από δύο τύπου βοηθειών σε περίπτωση που δεν ξέρει ποια είναι η σωστή απάντηση ,μια βοήθεια η οποία προσθέτει στον χρόνο μας άλλα 5 δευτερόλεπτα και άλλη μια βοήθεια στην οποία εξαφανίζονται οι δύο από τις 3 λάθος απαντήσεις ,οι βοήθεια ανανεώνονται κάθε μισή ώρα. Στην πρώτη λάθος απάντηση ή αν δεν δώσει απάντηση μέσα σε χρόνο 30 δευτερολέπτων η αρένα τερματίζεται και ο χρήστης θα πρέπει να περιμένει και τον αντίπαλο του να τελειώσει τη αρένα που παίζει .Όταν και οι δύο χρήστες ολοκληρώσουν τις αρένες τους τότε μπορούν να παραλάβουν το βραβείο τους και να δουν το αποτέλεσμα του αγώνα. Ο κάθε χρήστης όσο βρίσκεται online ενημερώνετε με εσωτερική ενημέρωση ότι έχει να παραλάβει κάποιο βραβείο ή ότι ένας χρήστης τον προσκάλεσε σε αρένα αν ο χρήστης δεν είναι online τότε πάλι ενημερώνετε μέσω της τεχνολογίας push notification. Το βραβείο του αγώνα περιλαμβάνει πόντους εμπειρίας που βοηθάνε στο να ανέβει επίπεδο ο χρήστης και πόντους κατάταξης οι οποίοι ανεβάζουν θέση τον χρήστη στην γενική κατάταξη του παιχνιδιού. Οι πόντοι υπολογίζονται ανάλογα με το πόσες ερωτήσεις απάντησε σωστά ο κάθε παίκτης αλλά και σε τι ταχύτητα της απάντησε .

4.1.4 Οθόνες εφαρμογής

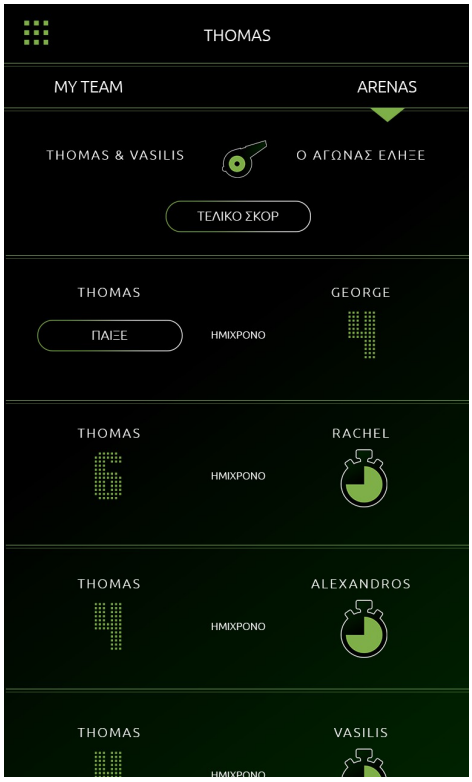
Η εφαρμογή μας θα αποτελείται από τέσσερις βασικές οθόνες ένα μενού και ένα popup πλαίσιο. Η πρώτη οθόνη είναι η οθόνη εγγραφής-εισόδου (εικόνα 4.1). Η επόμενη οθόνη είναι η οθόνη του προφίλ του κάθε χρήστη και θα έχει κάποια βασικά στατιστικά και το επίπεδο του (εικόνα 4.2). Η τρίτη οθόνη είναι η οθόνη με τις αρένες που έχει ανοιχτές ο χρήστης (εικόνα 4.3). Η τελευταία βασική οθόνη είναι η οθόνη της αρένας οπού ο χρήστης θα απαντάει τις ερωτήσεις του (εικόνα 4.4).



Εικόνα 4.1: signup-register



Εικόνα 4.2: my profile



Εικόνα 4.3: arenas

Εικόνα 4.4: arena

4.2 Περίληψη

Σε αυτό το κεφάλαιο κάναμε εγκατάσταση τη Angular 5 και του Ionic 3 και δημιουργήσαμε μια απλή εφαρμογή για το Ionic. Σε αυτό το κεφάλαιο ολοκληρώνουμε την εισαγωγή στις βασικές τεχνολογίες που θα χρησιμοποιήσουμε και στο επόμενο κεφάλαιο θα ξεκινήσουμε να δημιουργούμε το παιχνίδι quiz χρησιμοποιώντας όλες τις παραπάνω τεχνολογίες. Είδαμε μια μικρή παρουσίαση στο τι θα χαρακτηριστικά και τι απαιτήσεις θα έχουν οι οθόνες επίσης είδαμε και το εικαστικό κομμάτι της κάθε οθόνης. Στο επόμενο κεφάλαιο θα αρχίσουμε να χτίζουμε σιγά σιγά την εφαρμογή μας ξεκινώντας από την οθόνη εγγραφής του χρήστη .

ΚΕΦΑΛΑΙΟ 5

Εγγραφή και σύνδεση του χρήστη.

5.1 Σύνοψη λειτουργιών οθόνης εγγραφής-εισόδου

Ο χρήστης για να εισέλθει στην εφαρμογή θα πρέπει να δημιουργήσει ένα λογαριασμό. Για να γίνει αυτό το μόνο που χρειάζεται είναι ένα email και ένα κωδικό .

Οι χρήστες μας για αρχή αποθηκεύονται σε έναν authentication και πιό συγκεκριμένα αυτόν του **Firebase.** Το **Firebase** είναι υπηρεσία της Google και εκτός από τον authentication server ,μας παρέχει και άλλες υπηρεσίες όπως αυτή του Push Notification η analytics και πολλές άλλες .

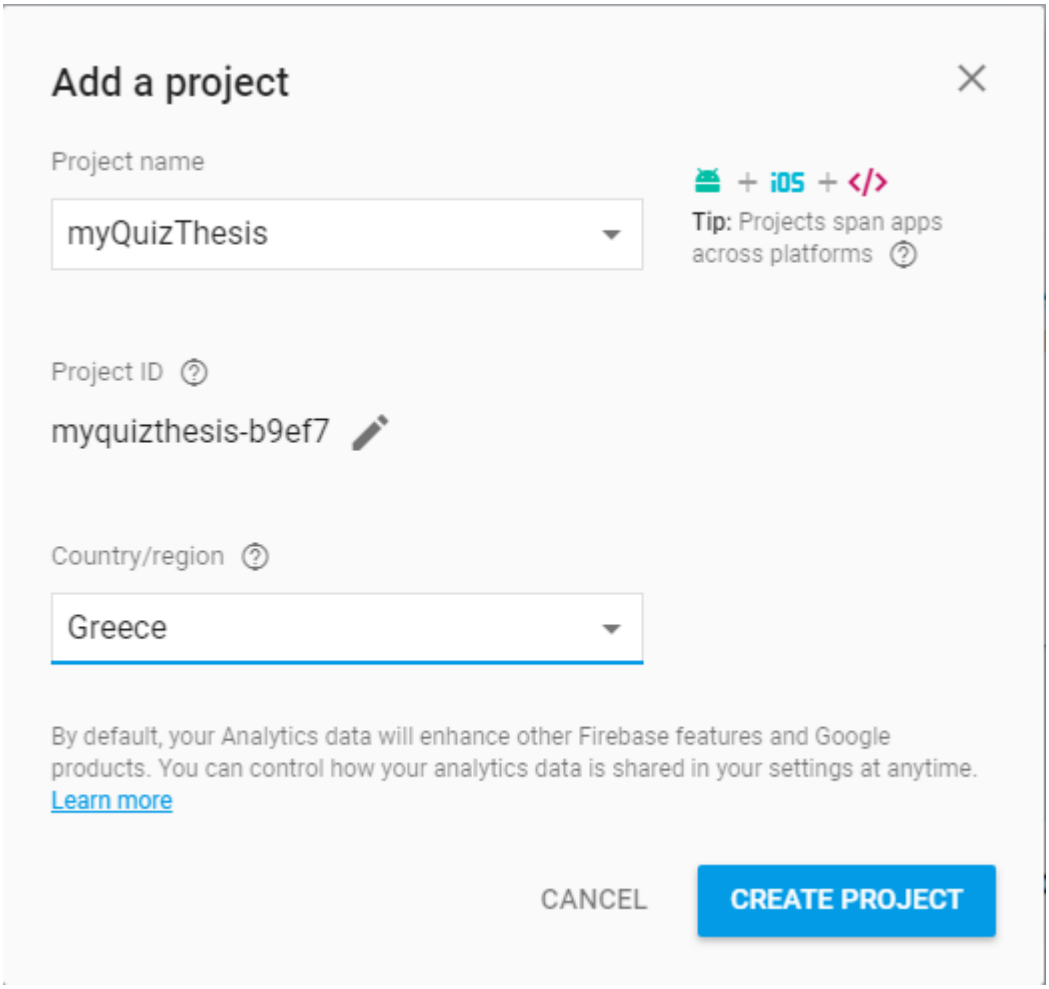
Όταν ο χρήστης θα πατάει το κουμπί sign up τότε θα στέλνονται στο **Firebase** τα στοιχεία του χρήστη και θα ελέγχετε αν τα στοιχεία του χρήστη υπάρχουν ήδη στην βάση ,αν δεν υπάρχουν τότε θα δημιουργείται στην βάση μια καινούργια εγγραφή με τα στοιχεία του χρήστη. Στην συνέχεια θα χρειαστεί ο χρήστης να εισάγει το όνομα που θα θέλει να εμφανίζεται στους άλλους χρήστες , το οποίο θα αποθηκευτεί στην δικιά μας βάση στο mlab.

Γενικά το **Firebase** θα το χρησιμοποιούμε μόνο για να πιστοποιούμε τον χρήστη ότι έχει δικαίωμα να χρησιμοποιεί την εφαρμογή και τον server μας. Για όλες τις άλλες πληροφορίες που θέλουμε είτε να αποθηκεύσουμε ή να ζητήσουμε θα χρησιμοποιούμε την βάση στο mlab.

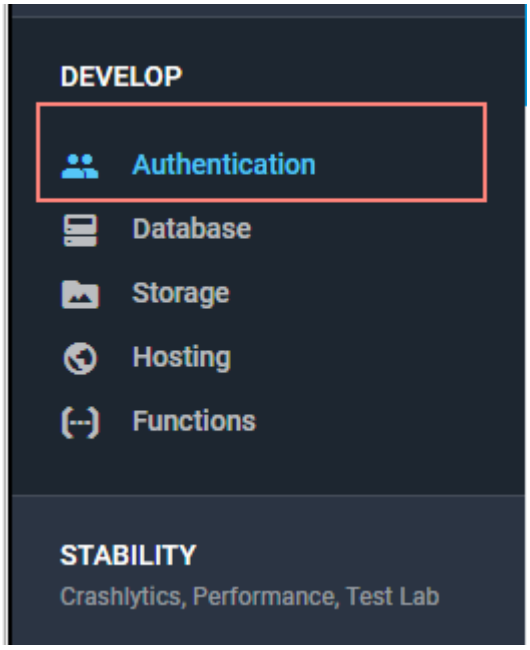
5.1.1 Δημιουργία λογαριασμού στο Firebase

Η δημιουργία λογαριασμού στο Firebase είναι απλή διαδικασία απλά χρειάζεται να έχουμε λογαριασμό στην Google. Κατευθυνόμαστε στην σελίδα **https://console.firebase.google.com/u/0/** και πατάμε το κουμπί “add project”,

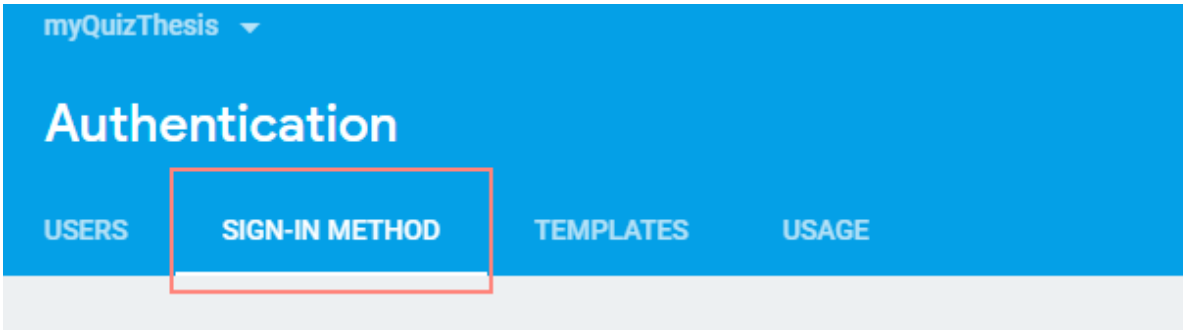
στην οθόνη εμφανίζεται ένα pop up που μας ζητάει κάποια πληροφορίες για την εφαρμογή στο πλαίσιο Project name εισάγουμε το όνομα που θέλουμε για την εφαρμογή μας και στο Region διαλέγουμε Greece και στην συνέχεια πατάμε το κουμπί create project. Στην συνέχεια είμαστε μέσα στην εφαρμογή που δημιουργήσαμε και πατάμε στο μενού στα δεξιά μας το κουμπί που “Authentication (εικόνα 5.2), διαλέγουμε την καρτέλα SIGN-IN-METHOD (εικόνα 5.3) και επιλέγουμε στους Provider το email/password πατάμε τον διακόπτη και γίνεται enable (εικόνα 5.4). Τέλος πατάμε το κουμπί “WEB SETUP” και κάνουμε αντιγραφή τον κώδικα που εμφανίζεται και τον αποθηκεύουμε σε ένα αρχείο γιατί θα τον χρειαστούμε αργότερα .



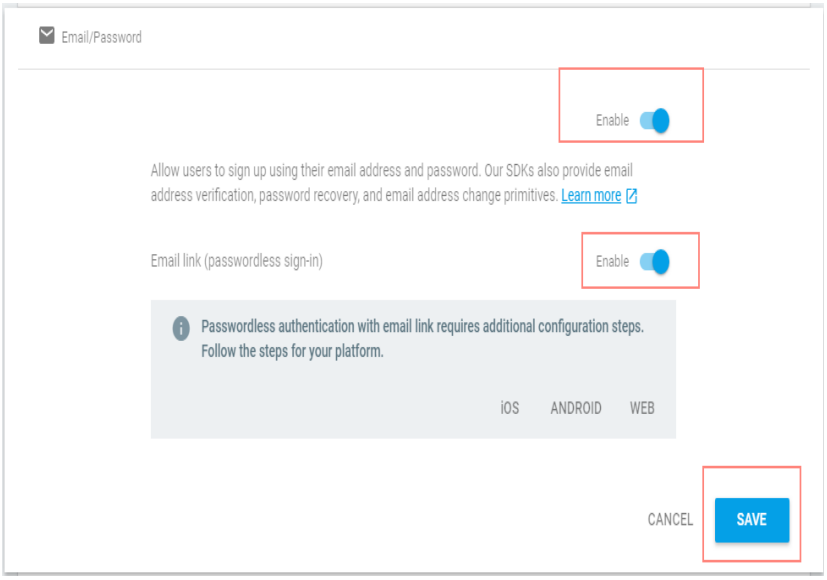
Εικόνα 5.1: firebase app details



Εικόνα 5.2: authentication



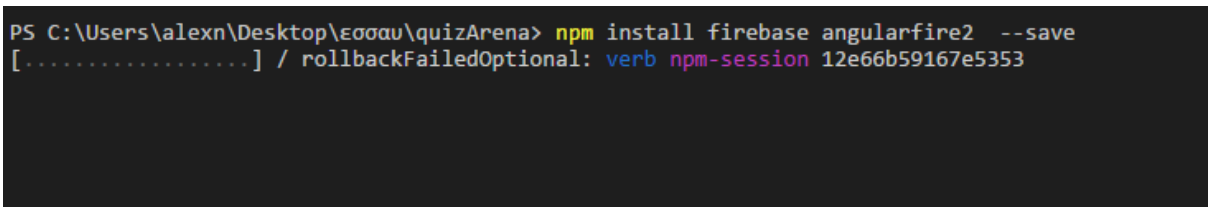
Εικόνα 5.3: sign-in-method



Εικόνα 5.4: enable email/password

5.1.2 Δημιουργία της οθόνης εγγραφής-εισόδου

Αν δεν έχουμε ήδη ανοιχτή την εφαρμογή που δημιουργήσαμε σε προηγούμενο βήμα, την ανοίγουμε και τρέχουμε την εντολή *ionic serve* για να τρέξει η εφαρμογή μας. Πρώτα πρέπει να εγκαταστήσουμε τις βιβλιοθήκες AngularFire2 και Firebase με την εντολή *npm install firebase angularfire2 --save.* Στην συνέχεια μέσα στο φάκελο src δημιουργούμε το αρχείο config.ts και μέσα κάνουμε επικόλληση τον κώδικα πού είχαμε αποθηκεύσει από το web setup όπως φαίνεται στον παρακάτω κώδικα.



Εικόνα 5.5: npm install firebase angularfire2

```
export const firebaseConfig = {

config: {

apiKey: "AIzaSyCHu5W-Y7htiQz2R4sJvI97tpdVAh0a8JY",
authDomain: "myquizthesis.firebaseapp.com",
databaseURL: "https://myquizthesis.firebaseio.com",
projectId: "myquizthesis",
storageBucket: "myquizthesis.appspot.com",
messagingSenderId: "713681366979"
}
}
```

Στην συνέχεια κατευθυνόμαστε στον φάκελο app/app.module.ts και προσθέτουμε στον πίνακα *imports* το module του AngularFire και προσθέτουμε και τα imports .

```
import { firebaseConfig } from "../config";

import { AngularFireModule } from 'angularfire2'; .

Imports: [

BrowserModule,
```

```
IonicModule.forRoot(MyApp),
AngularFireModule.initializeApp(firebaseConfig),
],
```

Προσθέτουμε και στον πίνακα των providers και το import ,

```
import { AngularFireAuth } from 'angularfire2/auth';

providers: [
StatusBar,
SplashScreen,
{provide: ErrorHandler, useClass: IonicErrorHandler},
AngularFireAuth
]
```

Στην συνέχεια δημιουργούμε την σελίδα sign-up στην οποία θα γίνεται η εγγραφή του χρήστη .Εκτελούμε την εντολή *ionic g page auth* στον φάκελο pages μπορούμε να δούμε τον φάκελο που δημιουργήσαμε. Για να μπορέσουμε να χρησιμοποιήσουμε την οθόνη που μόλις δημιουργήσαμε θα πρέπει να την προσθέσουμε στον πίνακα declarations στον app.module.ts .

Και στη συνέχεια θα πρέπει στο φάκελο app.component.ts να κάνουμε τις εξής αλλαγές .

```
rootPage:any = 'AuthPage';
```

Ανοίγουμε το locahost:8100 και βλέπουμε ότι η πρώτη οθόνη είναι η signup οθόνη.

Στο αρχείο auth.ts προσθέτουμε δυο μεταβλητές για το email και τον κωδικό .

```
import { IonicPage, NavController, NavParams, AlertController, LoadingController }
from 'ionic-angular';

email: string = '';
password: string = '';

constructor(

public alertCtrl: AlertController,
public loadingCtrl: LoadingController,
private firebaseService: FirebaseProvider,



) {
}

signUp() {
this.showLoader();
this.firebaseService.signUpWithEmailPassword(this.email, this.password).
then(res => {
this.presentAlert('You succefully sign up!);
}, error => {
this.presentAlert(error.message);
this.loading.dismiss();
});;
}

presentAlert(message) {
let alert = this.alertCtrl.create({
title: message,
buttons: ['Dismiss']
});
alert.present();
}
```

```
showLoader() {
this.loading = this.loadingCtrl.create({
content: 'Authenticating...'
});

this.loading.present();

}


        signIn() {
this.showLoader();
this.firebaseService.signUpWithEmailPassword(this.email, this.password).
then(res => {
console.log(res);
this.presentAlert('You succesfully sign in!');
}, error => {
console.log(error);
this.presentAlert(error.message);
this.loading.dismiss();
});;
}
```

Ας δούμε μια μια τη κάθε γραμμή κώδικα που προσθέσαμε ,μέσα στον constructor προσθέσαμε το alertctrl για να δείχνουμε ότι μήνυμα θέλουμε στον χρήστη μέσω pop up μηνύματος ,το loadingCtrl είναι ένα loader για να δείχνουμε ότι κάποια διεργασία εκτελείτε, η κλάση firebaseServiece θα περιέχει όλες τις μεθόδους που έχουν να κάνουμε με το firebase όπως το sign in ή το sign up θα δημιουγήσουμε την κλάση αυτή στο επόμενο βήμα. Προσθέσαμε την μέθοδο signUp(),signIn() όπου μέσω αυτών των μεθόδων θα εκτελείται η εγγραφή και η σύνδεση στο firebase .

Εκτελούμε την εντολή ionic g provider firebase , στον φάκελο src/providers έχει δημιουγηθέι το αρχείο firebase/firebase.ts. Προσθέτουμε τον παρακάτω κώδικα.

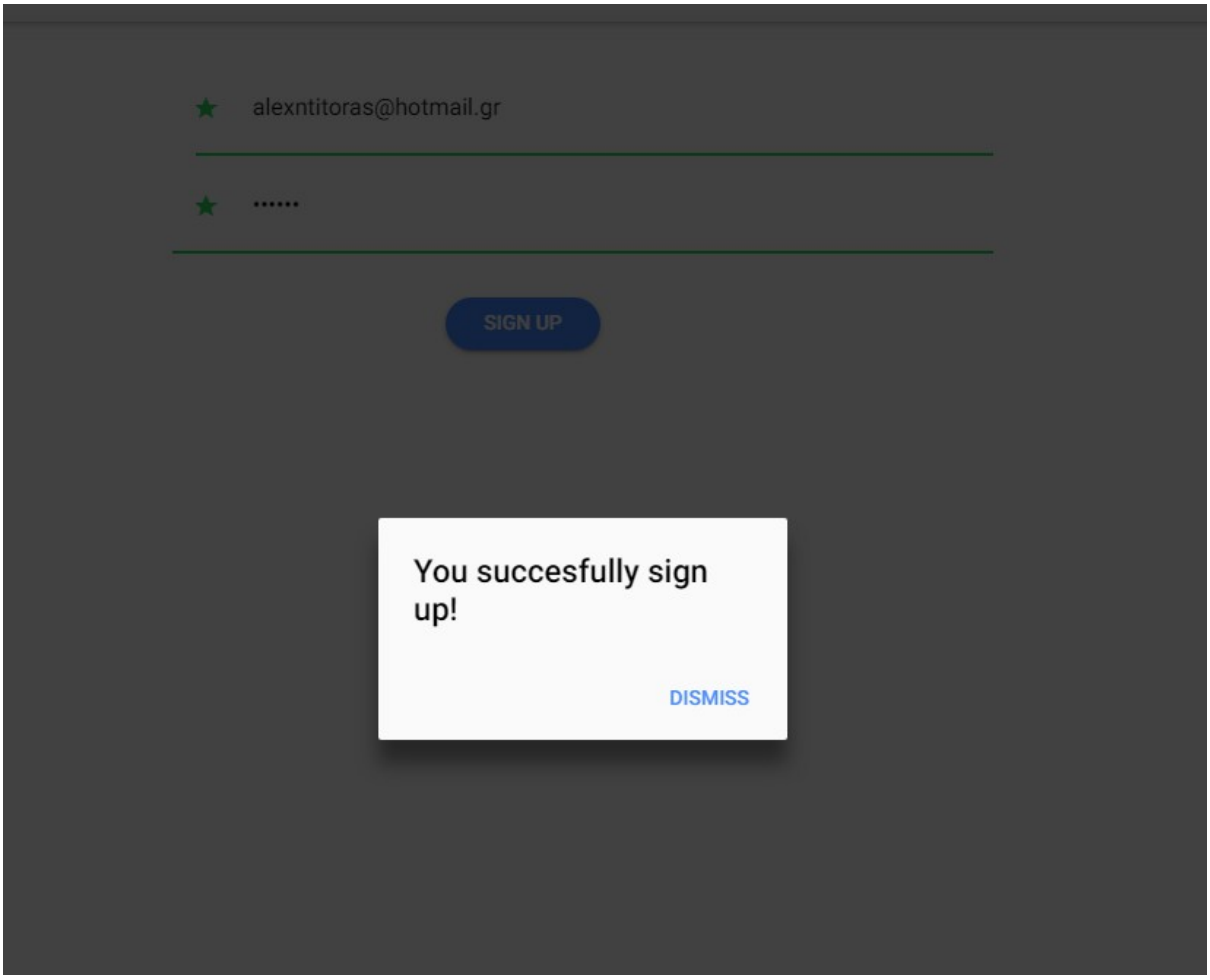
```
import { AngularFireAuth } from "angularfire2/auth";

signUpWithEmailPassword(email, password) {
return this.afAuth.auth.
createUserWithEmailAndPassword(email, password)
}

signInWithEmailPassword(email, password) {
return this.afAuth.auth.
signInWithEmailAndPassword(email, password)
}

getToken() {
let obsevable = new Observable((observer: any) => {
this.afAuth.auth.currentUser.getToken(false).then((idToken) => {
observer.next(idToken);
})
});
return obsevable;
}
```

Στο αρχείο auth-up.html προσθέτουμε τον παρακάτω κώδικα .

```
<ion-content padding>

<ion-row justify-content-center text-center class="sigin-section">
<ion-col col-12>

<ion-row justify-content-center text-center>
```

```
<ion-col col-7>
<ion-list>
<ion-item>
<ion-label>
<ion-icon color="secondary" name="star" item-start></ion-icon>
</ion-label>
<ion-input [(ngModel)]="email" placeholder="email"></ion-input>
</ion-item>
<ion-item>
<ion-label>
<ion-icon color="secondary" name="star" item-start></ion-icon>
</ion-label>
<ion-input [(ngModel)]="password" placeholder="password" type="password"></ion-
input>
</ion-item>
</ion-list>
</ion-col>
</ion-row>

<ion-row>
<ion-col col-7 offset-2>
<button ion-button round (click)="signUp()">SIGN UP</button>
<button ion-button round (click)="signIn()">SIGN IN</button>


</ion-col>
</ion-row>


</ion-col>
</ion-row>
</ion-content>
```

Κατευθυνόμαστε στο *localhost:8100* και εισάγουμε ένα email και ένα κωδικό και πατάμε το signUp αν όλα πήγανε καλά θα πρέπει να δούμε το παρακάτω μήνυμα



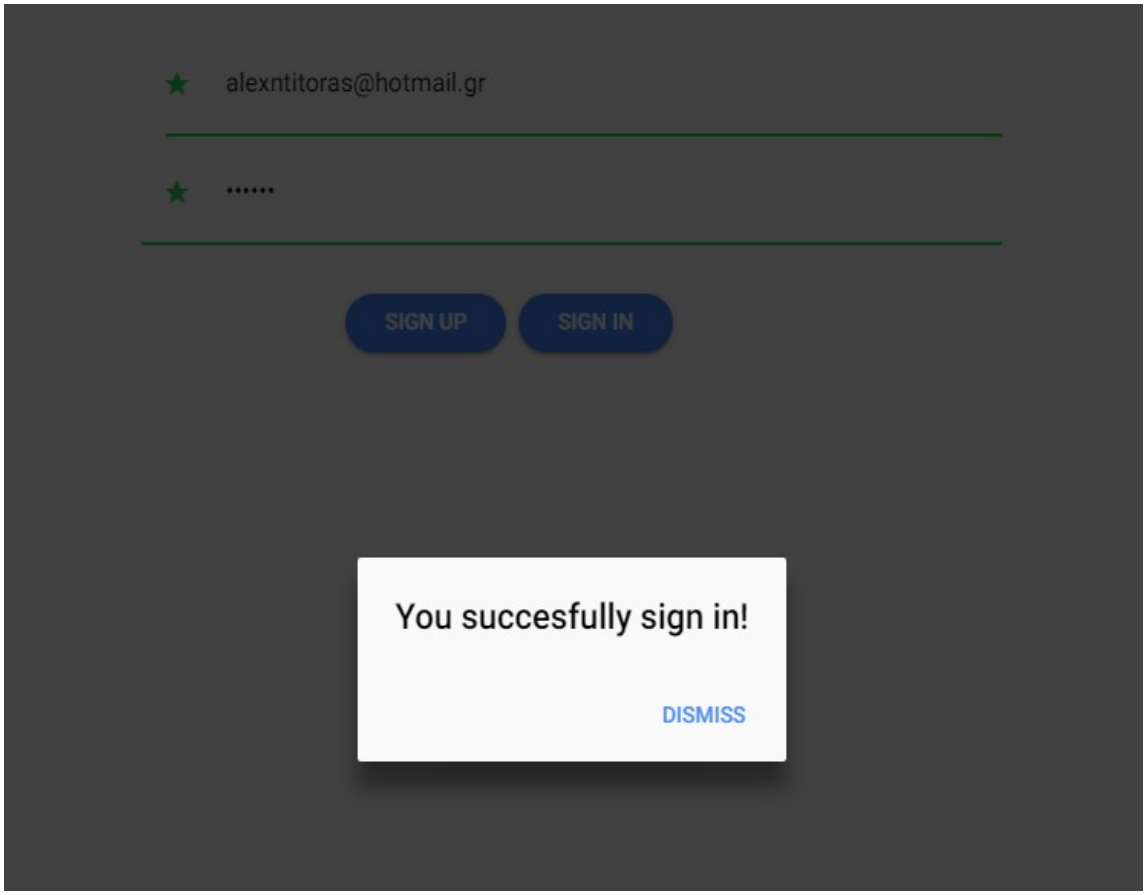
Εικόνα 5.6: sign up success

Δοκιμάζουμε να κάνουμε και σύνδεση εισάγοντας τα στοιχεία μας και πατώντας το κουμπί sign in .

5.1.3Δημιουργία εγγραφή χρήστη στην βάση μας

Ανοίγουμε τον φάκελο που έχουμε τον server μας και δημιουργούμε τον φάκελο models και προσθέτουμε το αρχείο user.js με τον παρακάτω κώδικα.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema
```



Εικόνα 5.7: sign in success

```
var UserSchema = new mongoose.Schema({
 firebaseId: {
  type: String,
  required: true,
  unique: true
 },
 username: {
  type: String,
  required: true,
  trim: true,
  index: true,
  unique: true,
  uniqueCaseInsensitive: true,
  maxlength: [10,'Username must be less than 10 characters']

 },
```

```
  arenas: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Arena'
    }
  ],
  statistics: {
    type: Schema.Types.ObjectId,
    ref: 'Stats'
  },
  hints:{
    type: Schema.Types.ObjectId,
    ref:'Hints'
  },
  deviceToken: [{
    type: Schema.Types.ObjectId,
    ref: 'DeviceTokens'
  }],
  awards:[{
    type: Schema.Types.ObjectId,
    ref: 'Awards'}],

})


module.exports = mongoose.model('Users', UserSchema);
```

Δημιουργήσαμε το μοντέλο πληροφοριών του χρήστη , στο οποίο αποθηκεύουμε πληροφορίες όπως το firebaseId για να μπορούμε να βρίσκουμε τον χρήστη , το username, έναν πίνακα με τις αρένες που έχει ανοιχτές ο χρήστης τα βραβεία που μπορεί να παραλάβει και το token της συσκευής που έχει κάνει login.

Δημιουργούμε τον φάκελο controllers και μέσα του τον φάκελο user.js ,όπου και θα καταλήγουν τα request που θα αφορούν τους χρήστες, όπως η δημιουργία χρήστη .

```
var User = require('../models/user');
exports.userCreate = function (req, res, next) {
  var userId = req.body.firebase_id
  var userName = req.body.username


  try {

    User.findOne({username: userName})
      .exec(function (error, user) {
        if (error) {
          console.log(error);
          return res.status(500).json({
            message: 'Unexpected error please login again...'
          })
        }
        if (user) {
          return res.status(400).json({
            message: 'UserName already exists please try another one'
          })
        } else {
          var user = new User({
            firebaseId: userId,
            username: userName
          })
          user.save(function (err, result) {
            try {
              if (err) {
                console.log(err);
```

```
                }
                return res.status(200).json({
                  message: 'User created',
                  user_id: result._id,
                  username: result.userName
                })
              } catch (err) {
                console.log(err);

                return res.status(500).json({
                  message: 'Unexpected error please login again...'
                })

              }

            })

        }
      })
    } catch (err) {
      return res.status(500).json({
        message: 'Unexpected error please login again...'
      })
    }
  }
```

Για να δημιουργηθεί ο χρήστης θα πρέπει να στείλουμε το firebaseId καθώς και ένα username.

Επόμενο βήμα είναι να δημιουργήσουμε τα routes στο nodeJs server. Κατευθυνόμαστε στον φάκελο routes και δημιουργούμε τον αρχείο firebaseAuth.js με τον παρακάτω κώδικα :

```
var express = require('express'),
router = express.Router()
```

```
var userController = require('../controllers/user');
var middleware = require('../controllers/firebaseMiddleware')
router.post('/createUser', userController.userCreate);
router.get('/protected', middleware, function (req, res) {
  return res.status(200).json(
    {
      content: 'Success',
      user_id:req.body.uid
    }
  )

})

module.exports = router;
```

Στο app.js προσθέτουμε :

```
var firebaseRoutes=require('./routes/firebaseAuth');
app.use('/api/firebase',firebaseRoutes);
```

Ολα τα request που θα λαμβάνουμε με κατάληξη api/firebase θα διαχειρίζονται απο τα routes και τους controllers που μόλις δημιουργήσαμε.

Για να μπορέσουμε να δημιουργήσουμε τον χρήστη στην βάση μας πρέπει να πιστοποιήσουμε τον χρήστη, αυτό γίνεται με την με τον admin sdk του firebase. Για να το εγκαταστήσουμε τρέχουμε την εντολή *npm install firebase-admin –save.* Στον φάκελο controllers προσθέτουμε το αρχείο firebaseMiddleware.js και τον κώδικα :

```
var admin = require("firebase-admin");

  module.exports =function (req,res,next) {
    var token=req.get('authorization');
    admin.auth().verifyIdToken(token)
```

```
      .then(function(decodedToken) {
        var uid = decodedToken.uid;
        req.body.uid=uid;
        next();
      }).catch(function(error) {
      return res.status(401).json({
        message:'Not Authenticated'
      });
    });

}
```

Για να ολοκληρώσουμε την διαδικασία δημιουργίας εγγραφής χρήστη γυρνάμε στην εφαρμογή στο ionic και προσθέτουμε τον εξής κώδικα στο αρχείο firebase.ts :

```
checkAuthentication() {
return new Promise((resolve, reject) => {
this.checkSub = this.afAuth.authState.subscribe((user: firebase.User) => {
if (!user) {
reject('User not logged in');
}
else {
this.afAuth.auth.currentUser.getToken(true).then((idToken) => {
this.token = idToken;
let headers = new Headers();
headers.append('Authorization', this.token);
this.http.get(myGlobals.host + 'firebase/protected', { headers: headers })
.map((response: Response) => response.json())
.catch((error: Response) => {
return Observable.throw(error.json())
})
.subscribe(res => {
```

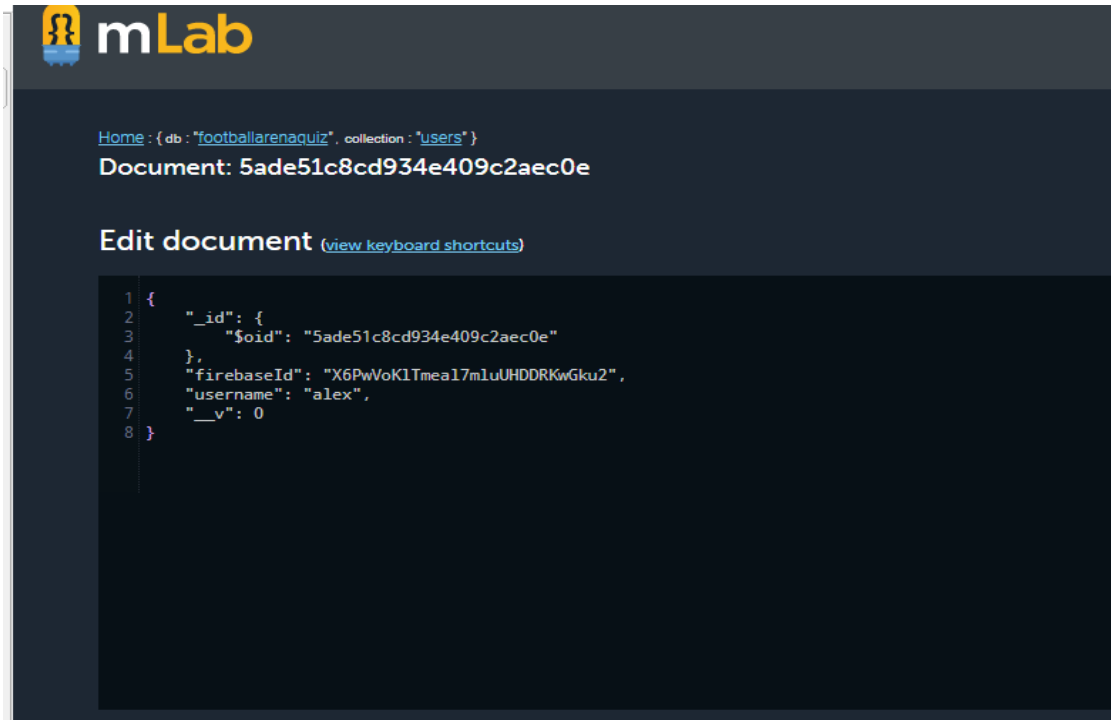
```
this.firebaseUserId = res.user_id;
resolve(res);
}, (err) => {
reject(err);
});
}).catch(function (error) {
reject(error);
});
}
});
});
}


createUser(username) {
const body = JSON.stringify({ firebase_id: this.firebaseUserId, username:
username });
let headers = new Headers();
headers.append('Content-Type', 'application/json');
headers.append('Authorization', this.token);
return this.http.post(myGlobals.host + 'firebase/createUser', body, { headers:
headers })
.map((response: Response) => response.json())
.debounceTime(1000)
.distinctUntilChanged()
.catch((error: Response) => {
return Observable.throw(error.json())
});
}
```

Στο αρχείο auth.ts διαμορφώνουμε την μέθοδο sign-up :

```
signUp() {
this.showLoader();
```

```
this.firebaseService.signUpWithEmailPassword(this.email, this.password).
then(res => {
this.firebaseService.checkAuthentication().then(data                    =>
{ this.firebaseService.createUser('alex').subscribe() });
this.presentAlert('You succesfully sign up!');
this.loading.dismiss();
}, error => {
console.log(error);
this.presentAlert(error.message);
this.loading.dismiss();
});;
}
```

Τώρα πατώντας το κουμπί sign up δημιουργείτε και η εγγραφή του χρήστη στην βάση μας όπως βλέπουμε και στην παρακάτω εικόνα :



Εικόνα 5.8: record mlab

Εφόσον ολοκληρώσαμε την εγγραφή και είσοδο στην εφαρμογή τώρα πρέπει να βλέπουμε ότι ο χρήστης είναι online και να μπορούμε του στέλνουμε δεδομένα σε

πραγματικό χρόνο χωρίς να τα ζητήσει. Για αυτό τον λόγο θα χρησιμοποιήσουμε την βιβλιοθήκη *socket io* . Τρέχουμε στην κονσόλα του server μας την παρακάτω εντολή *npm install socket.io –save* .Στην συνέχεια κατευθυνόμαστε στον φάκελο bin/www και προσθέτουμε τον παρακάτω κώδικα στην γραμμή 23 :

```
var io = app.io;
io.attach(server,{'pingInterval': 2000, 'pingTimeout': 5000});
```

και στο αρχείο app.js :

```
var socket_io=require("socket.io");
var io=socket_io();
app.io=io;
var sockets=require('./sockets/socket')(io);
```

Στον root folder προσθέτουμε τον φάκελο sockets και μέσα δημιουργούμε το αρχείο socket.js με τον εξής κώδικα :

```
module.exports = function (io) {

  var mainGameNsp=io.of('/mainGame');

  mainGameNsp.on('connection', function (socket) {
    console.log('user connected',);
  })
}
```

Δημιουργήσαμε έναν server-socket όπου κάθε χρήστης θα συνδέετε σε αυτόν μόλις κάνει εγγραφή και θα μένει μέσα σε αυτόν για όση ώρα είναι online , όσο βρίσκετε μέσα σε αυτό θα δέχεται δεδομένα χωρίς να έχει κάνει κάποιο ερώτημα π,χ αν κάποιος τον προκαλέσει ή δεχτεί κάποιο βραβείο θα ενημερώνετε σε πραγματικό χρόνο .

Στην ionic εφαρμογής μας πρέπει να κάνουμε και εκεί εγκατάσταση τον socket-client ο οποίος θα συνδέετε στον server μας και θα στέλνει τα μηνύματα στον server.

Εκτελούμε τις εντολές *npm install socket.io-client* για να εγκαταστήσουμε την βιβλιοθήκη και *ionic g provider socket* για να δημιουργήσουμε το αρχείο από όπου θα διαχειριζόμαστε τα sockets. *Στο αρχείο socket.ts ο κώδικας διαμορφώνετε ως εξής :*

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import * as io from "socket.io-client";

@Injectable()
export class SocketProvider {
private socket: any;

constructor(public http: HttpClient) {
console.log('Hello SocketProvider Provider');
}

connect() {
this.socket = io.connect('http://localhost:3000/mainGame', {
transports: ['websocket', 'polling', 'flashsocket'],
reconnectionAttempts: 10,
reconnectionDelayMax: 5000,
reconnectionDelay: 1000,
query:
{
message:'HELLO FROM IONIC'
}
});

}

}
```

και στο αρχείο auth.ts προσθέτουμε τον κώδικα :

```
signIn() {
this.showLoader();
this.firebaseService.signInWithEmailPassword(this.email, this.password).
then(res => {
this.socket.connect();

this.presentAlert('You succesfully sign in!');
this.loading.dismiss();
}, error => {
console.log(error);
this.presentAlert(error.message);
this.loading.dismiss();
});;
}
```

Πατώντας το κουμπί sign in βλέπουμε στον server μας ότι ο χρήστης συνδέθηκε!

```
C:\Users\alexn\Desktop\essay\my_app\server>npm start

> server@0.0.0 start C:\Users\alexn\Desktop\essay\my_app\server
> node ./bin/www

You  logged on to your database!!!
user connected
```

Εικόνα 5.9: user connected

5.2 Περίληψη

Δημιουργήσαμε την εγγραφή και την είσοδο του χρήστη ,είδαμε πως να συνδεθούμε στο firebase αλλά και πως να φτιάξουμε και να συνδεθούμε σε ένα server η πρώτη οθόνη ολοκληρώθηκε και θα συνεχίσουμε στις επόμενες για το πως να παίρνουμε στατιστικά και πως να προσκαλούμε ένα παίκτη σε arena .

ΚΕΦΑΛΑΙΟ 6

Οθόνη “My profile”

6.1 Λειτουργίες της οθόνης “My profile”

Είναι η πρώτη Οθόνη που βλέπει ο χρήστης όταν κάνει σύνδεση στην εφαρμογή , εδώ ο χρήστης μπορεί να δει το επίπεδό του, τις νίκες ισοπαλίες και τις ήττες που έχει. Υπάρχουν επίσης δύο κουμπιά τα οποία τα μπορούμε να βρούμε τυχαίο αντίπαλο ή να προσκαλέσουμε ένα χρήστη με το username του. Στην πάνω δεξιά γωνία υπάρχει ένα “menu burger κουμπί “ το οποίο μας δίνει πρόσβαση σε ποιο αναλυτικά στατιστικά αλλά και στην γενική κατάταξη των χρηστών.

6.1.1 Δημιουργία λειτουργίας δημιουργίας και ανανέωσης επιπέδου χρήστη.

Στον φάκελο controllers στον σέρβερ μας δημιουργούμε το αρχείο levelup.js με τον παρακάτω κώδικα :

```
module.exports=function (level,currentExperience) {
  var levelArray=[{level:1,nextLevel:300},
    {level:2,nextLevel:600},
    {level:3,nextLevel:900},
    {level:4,nextLevel:1200},
    {level:5,nextLevel:1500},
    {level:6,nextLevel:1800},
    {level:7,nextLevel:2100},
    {level:9,nextLevel:3000},
    {level:10,nextLevel:8000},
    {level:11,nextLevel:16000},
```

```
        {level:12,nextLevel:24000},
        {level:13,nextLevel:32000},
        {level:14,nextLevel:40000}]
      var levelInfo={level:level,currentExperience:currentExperience};
      if (levelInfo.currentExperience>levelArray[level-1].nextLevel){
         levelInfo.level++;
                              levelInfo.currentExperience=levelInfo.currentExperience-
levelArray[level-1].nextLevel;
      }
      return levelInfo;
}
```

Η παραπάνω μέθοδος δέχεται δύο ορίσματα το ένα είναι το τρέχον επίπεδο του χρήστη το άλλο είναι πόσους πόντους έχει ο χρήστης. Έχουμε δημιουργήσει ένα πίνακα με τα επίπεδα που μπορεί να φτάσει ο χρήστης και τους πόσους πόντους πρέπει να κερδίσει για να για να φτάσει στο επίπεδο αυτό. Το level του χρήστη αποθηκεύεται στον πίνακα statistics τον οποίο θα δημιουργήσουμε μέσα στο αρχείο models και θα έχει το παρακάτω κώδικα :

```
    var mongoose = require('mongoose')
var Schema = mongoose.Schema

var schema = new Schema({
  user: {type: Schema.Types.ObjectId, ref: 'Users'},
  firebase_id: {type: String, unique: true},
  level: {type: Number, default: 1},
```

```
  currentExp: {type: Number, default: 0},
  wins: {type: Number, default: 0},
  loses: {type: Number, default: 0},
  draws: {type: Number, default: 0},
  rating:{type:Number,default:0},
  previousRanking:{type:Number,default:0},
  ranking:{type:Number,default:0},
  winningStreak: {
    currentStreak:{type:Number,default:0},
    longestStreak:{type:Number,default:0}
  },
  losingStreak:{
    currentStreak:{type:Number,default:0},
    longestStreak:{type:Number,default:0}
  },
  drawStreak:{
    currentStreak:{type:Number,default:0},
    longestStreak:{type:Number,default:0}
  },
  rightQuestionsNumber:{type:Number,default:0}
})

module.exports = mongoose.model('Stats', schema)
```

Στον πίνακα αυτόν αποθηκεύουμε όλα τα στατιστικά του χρήστη που θα χρησιμοποιήσουμε όπως νίκες ισοπαλίες, ήττες, συνεχόμενες νίκες και άλλα .Κάθε φορά που ο χρήστης θα παίρνει κάποιο βραβείο ο πίνακας των στατιστικών θα ανανεώνετε αυτόματα και ο χρήστης θα ενημερώνετε μέσω των sockets όπως φαίνεται στον παρακάτω κώδικα :

```
module.exports=function (io,connectedUserList) {
    io.on('getStats',function (req) {
        console.log('get stasts');
        Statistics.findOne({user:req.userId}).exec(function (err,result) {
            if (err) {
                throw err;
            }
            if(connectedUserList!=null) {
                connectedUserList.emit('loadStats', {
                    obj: result,
                    message:'Loaded succesfully'
                });
            }
        });


    });

};
```

Η μέθοδος αυτή παίρνει δύο ορίσματα το ένα είναι το io για να μπορούμε να χρησιμοποιήσουμε τις μεθόδους του socket io και το connecteduserList που είναι το id του χρήστη στον οποίο θέλουμε να στείλουμε τα δεδομένα.

6.1.2 Εύρεση τυχαίου αντιπάλου .

Για να βρούμε τυχαία κάποιο χρήστη δημιουργούμε μέσα στον φάκελο controllers το αρχείο findRandom.js με τον παρακάτω κώδικα :

```
var User = require('../models/user')
exports.findRandomPlayer=function (req,res,next,userId) {
  var userId = req.body.userId
  var filter
  var arrayId = []
  User.findById(userId)
    .populate('arenas')
    .exec(function (err, result) {
      if (err) {
        return res.status(500).json({
          title: 'Error',
          message: 'Could not find user',
          status: '500'
        })
      }

      if (result) {
        for (var i = 0; i < result.arenas.length; i++) {
```

```
    if(result.arenas[i].user!=userId){
      arrayId.push(result.arenas[i].user);
    }
    if(result.arenas[i].invite!=userId){
      arrayId.push(result.arenas[i].invite);
    }
  }
  arrayId.push(userId);
  User.count({_id:{$nin:arrayId}}).exec(function (err,count) {
    var random = Math.floor(Math.random() * count);
    User.findOne({_id:{$nin:arrayId}})
      .populate('statistics')
      .populate('history')
      .skip(random)
      .exec(function (err,randomUser) {
        if (randomUser){
          return res.status(200).json({
            message: 'User Found',
            username: randomUser.username,
            inviteId: randomUser._id,
            statistics: randomUser.statistics
          })
        }
        else {
```

```
                return res.status(500).json({
                  title: 'Error',
                  message: 'Sorry could not find user please try again....',
                  status: '500'
                })
              }
            })
          })
        }
      })
    }
```

Ο χρήστης με το που πατήσει το κουμπί εύρεση τυχαίου αντιπάλου τότε ,καλείται ο παραπάνω κώδικας που ψάχνει να βρει έναν αντίπαλο με το οποίο δεν έχει ήδη ανοικτή αρένα. Όταν τον βρει εμφανίζει στον χρήστη τα αποτελέσματα από τα μεταξύ τους προηγούμενα παιχνίδια .

Στην ionic εφαρμογή δημιουργούμε την σελίδα με όνομα *myprofile και* τον *provider user*. Προσθέτουμε τον παρακάτω κώδικα στο myprofile.ts:

```
findRandom() {

this.showLoader();
this.startPageService.findRandomUser()
.subscribe((result: UserFound) => {
this.loading.dismiss();
this.choosePlayer(result);
}, err => {
this.loading.dismiss();
console.log(err);
});
```

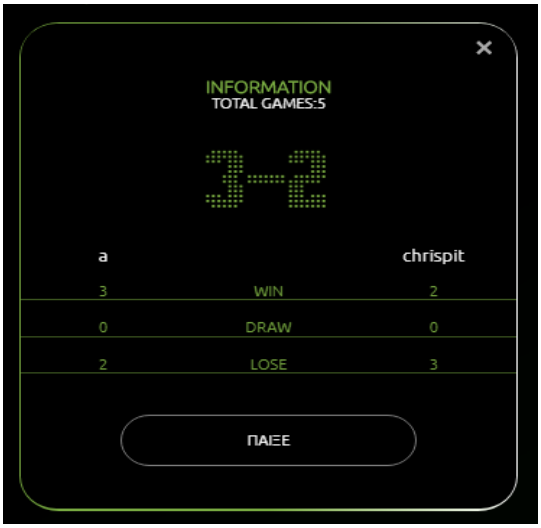
```
}

 choosePlayer(userFound = undefined) {
let modal = this.modalCtrl.create(this.findPlayerPage, { userFound: userFound });
modal.present();
}
```

και μέσα στο user.ts provider το εξής :

```
findRandomUser() {

return new Observable((observer: any) => {
this.firebasaService.getToken()
.subscribe((token: any) => {
let body = { userId: this.firebasaService.userId };
let headers = new Headers();
headers.append('Content-Type', 'application/json');
headers.append('Authorization', token);
return this.http.post(myGlobals.host + 'users/findRandom', body, { headers: headers })
.map((response: Response) => {
let stats = response.json().statistics;
let statsModel = new Stats(stats.level, stats.currentExp, stats.wins, stats.loses, stats.draws)
let transFormed: UserFound = new UserFound(response.json().message, response.json().username, response.json().inviteId, stats);
return transFormed;
})
.subscribe(data=>{
observer.next(data);
},err=>{
observer.error(err.json())
```

```
})
})
})

}
```

Μόλις ο χρήστης πατήσει το κουμπί παίξε με τυχαίο αντίπαλο τότε καλείται η μέθοδος *findRandom()* η οποία στην συνέχεια καλεί την μέθοδο findRandomUser() με το ο που βρίσκετε μέσα στο user provider. Η οποία με την σειρά της καλεί την μέθοδο findRandomPlayer() στον server μας. Ο server επιστρέφει το αποτέλεσμα το οποίο εμφανίζεται σε έναν modal όπως φαίνεται στην εικόνα 9.1. Αν ο χρήστης πατήσει το κουμπί παίξε τότε δημιουργείται νέα αρένα και ο χρήστης κατευθύνετε μέσα σε αυτή .



Εικόνα 6.1: Random user

6.2 ΠΕΡΙΛΗΨΗ

Είδαμε πώς ο χρήστης λαμβάνει τα στατιστικά και το τα δεδομένα για το επίπεδο που βρίσκετε. Φτιάξαμε τις μεθόδους για το πώς ο χρήστης μπορεί να βρει κάποιον αντίπαλο στο επόμενο κεφάλαιο θα δημιουργήσουμε την αρένα με τις ερωτήσεις

ΚΕΦΑΛΑΙΟ 7

Οθόνες της αρένας και ανοικτών αρένων

7.1 Λειτουργίες της Οθόνης αρένα

Στην οθόνη αυτή ο χρήστης θα έχει τι δυνατότητα να απαντήσει σε 10 ερωτήσεις. Η αρένα πριν δημιουργηθεί θα ψάξει στην βάση για να βρει 10 τυχαίες ερωτήσεις, οι οποίες είναι χωρισμένες σε 5 διαφορετικά επίπεδα δυσκολίας, το επίπεδο αυξάνετε αναλόγως σε ποίον αριθμό ερώτησης βρισκόμαστε .Όσο μεγαλύτερος ο αριθμός τόσο δυσκολότερη. Κάθε απάντηση θα έχει 4 πιθανές απαντήσεις, η κάθε απάντηση θα είναι ένα button μόλις ο χρήστης το πατήσει τότε αυτό θα κοκκινίζει ή θα πρασινίζει ανάλογα με το αν είναι σωστή η όχι. Η οθόνη έχει ένα χρονομετρητή ο οποίος μόλις περάσουν 30 δευτερόλεπτα θα τελειώνει την αρένα, ο μετρητής θα μηδενίζετε σε κάθε σωστή απάντηση και στην επόμενη θα ξεκινάει να μετράει ξανά. Μόλις τελειώσουν και οι δύο χρήστες το παιχνίδι τους θα δημιουργήστε το βραβείο και ο χρήστης θα παίρνει bonus εμπειρίας ανάλογα το αποτέλεσμα του παιχνιδιού και πόντους κατάταξης ανάλογα πόσες σωστές απαντήσεις έδωσε και σε τι χρόνο. Την δημιουργία του βραβείου καθώς και τον τρόπο παραλαβής θα το δούμε στο επόμενο κεφάλαιο .

7.1.1 Δημιουργία της αρένας στον server .

Για την δημιουργία της αρένας θα χρειαστούμε να δημιουργήσουμε δύο καινούργιους πίνακες στην βάση μας. Ο ένας θα είναι ο πίνακας που θα αποθηκεύουμε τις ερωτήσεις του παιχνιδιού. Ο δεύτερο πίνακας θα είναι ο πίνακας που θα αποθηκεύονται οι αρένες μας. Ο πίνακας της αρένας θα έχει τα παρακάτω πεδία: *user* το οποίο θα αποθηκεύει το id του χρήστη που δημιούργησε την αρένα,*invite* αποθηκεύει το id του χρήστη που καλέστηκε,*questions* αποθηκεύονται οι 10 ερωτήσεις, *user_played* αποθηκεύεται το αν ο χρήστης έχει παίξει*, awardPlayerOne* αποθηκεύει αν ο χρήστης πήρε το βραβείο του*, invite_played* αποθηκεύεται το αν ο χρήστης που καλέστηκε έχει παίξει και το πεδίο *questionsAnswered* στο οποίο αποθηκεύονται ο αριθμός των ερωτήσεων που απάντησε ο κάθε χρήστης .

Δημιουργούμε το αρχείο arenas.js μέσα στον φάκελο models με το παρακάτω κώδικα:

```
var schema = new Schema({
  user: {type: Schema.Types.ObjectId, ref: 'Users'},
  invite: {type: Schema.Types.ObjectId, ref: 'Users'},

  questions: [{type: Schema.Types.ObjectId, ref: 'Question'}],
  user_played: {type: Boolean, default: false},
  invite_played: {type: Boolean, default: false},
  awardPlayerOne: {type: Boolean, default: false},
  awardPlayerTwo: {type: Boolean, default: false},
  questionsAnswered: {
    user: {
      questionNumber: {type: Schema.Types.ObjectId, ref: 'ArenaQuestion'}
    },
    invite: {
      questionNumber: {type: Schema.Types.ObjectId, ref: 'ArenaQuestion'}
    }
  }

})
```

Δημιουργούμε μέσα στο φάκελο models και τον πίνακα των questions με όνομα questions.js :

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
```

```
var schema = new Schema({
   question:{type: String},
   optionA: {type: String},
   optionB: {type: String},
   optionC: {type: String},
   optionD: {type: String},
   answer: {type: String},
   level:{type:Number ,default:0}
});

module.exports = mongoose.model('Question', schema);
```

Στην συνέχεια πρέπει να δημιουργήσουμε την μέθοδο που θα δημιουργείται η αρένα. Το request από την μεριά του ionic θα πρέπει να περιέχει τα id των δύο χρηστών τα οποία θα αποθηκευτούν στο μοντέλο της αρένας που φτιάξαμε παραπάνω. Στην συνέχεια θα αποθηκευτούν 10 τυχαίες ερώτησης και η αρένα θα φτιαχτεί. Δημιουργούμε το αρχείο arenas μέσα στον φάκελο controllers μαζί με το παρακάτω κώδικα :

```
exports.createArena = function (req, res, next) {
  if(req.body.userId==req.body.inviteId){
    return res.status(500).json({
      title: 'Error',
      message: 'Error you cant play with your self',
      status: '500'
    })
  }
```

```
try {
  User.findById(req.body.userId, function (err, user) {
    if (err) {
      return res.status(500).json({
        title: 'Error',
        message: 'An error has occured....',
        status: '500'
      })
    }
    User.findById(req.body.inviteId, function (err, userInvite) {

      if (err) {
        return res.status(500).json({
          title: 'Error',
          message: 'An error has occured....',
          status: '500'
        })
      }

      var level1 = {level: 1}
      var level2 = {level: 2}
      var level3 = {level: 3}
      var level4 = {level: 4}
      var level5 = {level: 5}
```

```
var questionsArray = []
Questions.findRandom(level1).limit(1).exec(function (err, level1) {
 if (err) {
  return res.status(500).json({
   title: 'An error occured',
   error: err
  })
 }
```

7.1.2 Δημιουργία της οθόνης

Εκτελούμε την εντολή *ionic g page match* για να δημιουργηθεί η σελίδα στο ionic. Η πρώτη λειτουργία είναι να ζητήσουμε από τον server τις ερωτήσεις για την συγκεκριμένη αρένα. Μόλις ο ο server απαντήσει τότε η η κάθε πιθανή απάντηση της πρώτης ερώτησης θα ενσωματωθεί σε ένα από τα τέσσερα κουμπιά της οθόνης και θα ξεκινήσει το χρονόμετρο .Όλες οι υπόλοιπες ερωτήσεις θα αποθηκευτούν σε έναν πίνακα και κάθε φόρα που θα απαντάμε σωστά τότε ο πίνακας θα μας δίνει την επόμενη ερώτηση. Όταν ο χρήστης θα πατήσει ένα από τα κουμπιά των ερωτήσεων τότε θα εκτελείται μια συνάρτηση η οποία θα στέλνει στο server την απάντηση που διάλεξε ο χρήστης, το δευτερόλεπτο που απαντήθηκε και αν απάντησε σωστά η όχι. Σε περίπτωση λάθους τότε θα τελειώνει η αρένα και θα στέλνετε μέσω των socket ενημέρωση στον αντίπαλο χρήστη για το τέλος της αρένας. Κάποιες από τις μεθόδους που χρησιμοποιούμε είναι οι εξής.

```
getQuestions() {
this.showLoader();
let arenaInfo: ArenaCorrect = new ArenaCorrect(this.userId,
this.arena.arenaId);
```

```
        this.questionServie.getQuestions(arenaInfo)
        .subscribe((questions: Question[]) => {
        /*          this.questionServie.initAnswers(true,          this.arena.arenaId,
        this.userId).subscribe();
        */

        this.arenaQuestions = questions;
        this.loading.dismiss();
        this.timer();
        }, err => {
        this.appCtrl.getRootNav().setRoot('TabsPage', { index: 1 });
        this.loading.dismiss();
        console.log(err.error);

        })
}

        checkQuestion(chosenAnswer, currentQuestion: Question, buttonNumber) {
        if (chosenAnswer === currentQuestion.answer) {
        this.buttonDisabled = true;
        let questionAnswer = new AnsweredQuestion(currentQuestion.questionId,
        true, this.realTime);
        let questionAns = new ArenaAnsweredQuestion(this.arena.arenaId,
        this.userId, questionAnswer);
        this.rightButtons[buttonNumber] = true;
        this.initButtons[buttonNumber] = false;
        this.questionServie.saveAnsweredQuestion(questionAns)
        .subscribe(
        data => {
        this.time = 100;
        this.subscription.unsubscribe();
        setTimeout(() => {
        if (this.index > 9) {
        this.playerLost();
```

```
}
this.realTime = 30;
this.nextQuestion();
}, 1200);
},
error => console.error(error),

);

} else {
this.subscription.unsubscribe();
this.buttonDisabled = true;
this.findRightQuestion(currentQuestion);
this.wrongButtons[buttonNumber] = true;
this.initButtons[buttonNumber] = false;

setTimeout(() => {
this.playerLost();
}, 1200);
}
}

findRightQuestion(currentQuestion: Question) {

if (currentQuestion.answer == currentQuestion.optiona) {
this.rightButtons[0] = true;
this.initButtons[0] = false;
} else if (currentQuestion.answer == currentQuestion.optionb) {
this.rightButtons[1] = true;
this.initButtons[1] = false;
} else if (currentQuestion.answer == currentQuestion.optionc) {
this.rightButtons[2] = true;
this.initButtons[2] = false;
}
```

```
else if (currentQuestion.answer == currentQuestion.optiond) {
this.rightButtons[3] = true;
this.initButtons[3] = false;
}



}
```

Στον αρχείο match.html γράφουμε τον παρκάτω κώδικα

```
<div class="answerRow">

<ion-row justify-content-center text-center class="rowAnswerContainer">
<ion-col >
<button
(click)="checkQuestion(arenaQuestions[index]?.optiona,arenaQuestions[inde
x],0)"                                          [disabled]="buttonDisabled"
[ngClass]="{'initializeButton':initButtons[0],'wrongButton':wrongButtons[0],'righ
tButton':rightButtons[0],'hideButton':hideButtons[0]}"
ion-button round outline [disabled]="buttonDisabled">
<span><div class="word-container">
<p>{{arenaQuestions[index]?.optiona}} </p>
</div>
</span>
</button>
</ion-col>
</ion-row>

<ion-row justify-content-center text-center class="rowAnswerContainer">
<ion-col>
<button                                                  autocapitalize="none"
(click)="checkQuestion(arenaQuestions[index]?.optionb,arenaQuestions[inde
x],1)"
```

```
[ngClass]="{'initializeButton':initButtons[1],'wrongButton':wrongButtons[1],'righ
tButton':rightButtons[1],'hideButton':hideButtons[1]}"
ion-button round outline [disabled]="buttonDisabled">
<span><div class="word-container">
<p>{{arenaQuestions[index]?.optionb}} </p>
</div>
</span>
</button>
</ion-col>
</ion-row>

<ion-row justify-content-center text-center class="rowAnswerContainer">
<ion-col>
<button
(click)="checkQuestion(arenaQuestions[index]?.optionc,arenaQuestions[inde
x],2)"
[ngClass]="{'initializeButton':initButtons[2],'wrongButton':wrongButtons[2],'righ
tButton':rightButtons[2],'hideButton':hideButtons[2]}"
ion-button round outline [disabled]="buttonDisabled">
<span><div class="word-container">
<p>{{arenaQuestions[index]?.optionc}} </p>
</div>
</span>
</button>
</ion-col>
</ion-row>

<ion-row justify-content-center text-center class="rowAnswerContainer">
<ion-col>
<button
(click)="checkQuestion(arenaQuestions[index]?.optiond,arenaQuestions[inde
x],3)"
[ngClass]="{'initializeButton':initButtons[3],'wrongButton':wrongButtons[3],'righ
tButton':rightButtons[3],'hideButton':hideButtons[3]}"
```

```
ion-button round outline [disabled]="buttonDisabled">
<span><div class="word-container">
<p>{{arenaQuestions[index]?.optiond}} </p>
</div>
</span>
</button>
</ion-col>
</ion-row>
</div>
```

Εικόνα 7.1: Αρένα

7.2 Λειτουργίες της Οθόνης ανοικτών αρένων

Η οθόνη αυτή είναι η ποιο απλή οθόνη της εφαρμογής, στην οθόνη αυτή θα φαίνονται οι αρένες που έχει ανοικτές ο κάθε χρήστης. Ο χρήστης από αυτή την οθόνη θα μπορεί να παραλάβει το βραβείο του και θα μπορεί να παίξει κάποια αρένα που τον έχουν προσκαλέσει .

7.2.1 Δημιουργία λειτουργιών βραβείων στον server

Δημιουργούμε στον φάκελο models το αρχείο με όνομα awards.js.Δημιουργούμε τα πεδία: winner όπου θα βρίσκετε το βραβείο του νικητή και θα έχει τα πεδία points, experince,received.correctAnswer.loser το βραβείο του ηττημένου που θα έχει τα ίδια πεδία με τον winner.draw θα είναι το βραβείο και των δύο user με παρόμοια πεδία όπως του winner. Ο κώδικας φαίνεται παρακάτω :

```
var schema = new Schema({
 arenaId: String,
 draw:{type:Boolean,default:false},
 awards: {
  winner: {
   userId: String,
   points: Number,
   experience: Number,
   correctAnswers:Number,
   received: {type: Boolean, default: false}
  },
  loser: {
   userId: String,
   points: Number,
```

```
  experience: Number,
  correctAnswers:Number,
  received: {type: Boolean, default: false}
 },
 draw: {
  receivedP1: {
   userId: {
    type: String
   },
   received: {
    type: Boolean,
    default: false
   },
   points: {
    type: Number
   },
   experience: Number,
   correctAnswers:Number
  },
  receivedP2: {
   userId: {
    type: String
   },
   received: {
```

```
            type: Boolean,
            default: false
          },
          points: {
            type: Number
          },
          experience: Number,
          correctAnswers:Number
        }
      }
    }
  })
```

Στον φάκελο controllers δημιουργούμε το αρχείο awardCreate.js.Εδώ θα έχουμε την μέθοδο που θα δημιουργείτε το βραβείο μας. Η μέθοδος αυτή θα κάνει μια σύγκριση των αποτελεσμάτων των δύο χρηστών και θα αποφασίζει ποιος είναι ο νικητής, θα υπολογίζει τους πόντους που θα παραλάβει ο κάθε χρήστης από το βραβείο και τέλος θα το αποθηκεύει στον πίνακα που δημιουργήσαμε προηγουμένως. Ο κώδικας συνοπτικά φαίνεται παρακάτω :

```
ArenaQuestions.findOne().where({$and: [{arenaId: arenaId}, {userId: {$ne: userId}}]})
          .populate('userId', 'userName')
          .exec(function (err, answerCountB) {
            console.log('anwseer count b:',answerCountB)
            if (err) {
              console.log(err)
            }
```

```
            else if (answerCountB == null) {
              userTwoLenth = 0;
              bonusPlayerB=0;
            } else {
              try {
                userTwoLenth = answerCountB.questionAnswer.length;
                for (var i = 0; i < answerCountB.questionAnswer.length; i++) {
                  if (answerCountB.questionAnswer[i].time > 25) {
                    bonusPlayerB = bonusPlayerB + 0.05;
                  }
                  else if (answerCountB.questionAnswer[i].time > 20 &&
answerCountB.questionAnswer[i].time < 25) {
                    bonusPlayerB = bonusPlayerB + 0.03

                  } else {
                    bonusPlayerB = bonusPlayerB + 0;
                  }
                  pointsB = 10 + pointsB;

                }
              }catch (err){
                reject(err);
              }

            }
```

```
try {
  if (userOneLength > userTwoLenth) {
    pointsA=pointsA+50;
    pointsA=(pointsA*bonusPlayerA)+pointsA;

    pointsB=pointsB+10;
    pointsB=(pointsB*bonusPlayerB)+pointsB;


    try {
     var awards = new Awards({
       arenaId: arenaId,
       awards: {
         winner: {
           userId: userId,
           points: pointsA,
           experience: 140,
           correctAnswers: userOneLength
        }, loser: {
           userId: inviteId,
           points: pointsB,
           experience: 40,
           correctAnswers: userTwoLenth
```

```
        }
      }
    })
    awards.save(function (err,result) {
      if(err){
        reject(err);
      }else{
        resolve(arenaId);
      }

    })

  } catch (err) {
    console.log(err);
  }
```

Για να συλλέξουμε το βραβείο δημιουργούμε μέσα στον φάκελο controllers το αρχείο awards.js . Στο αρχείο αυτό θα προσθέτονται στα στατιστικά του χρήστη τα ποσοστά που περιέχει μ το βραβείο συνοπτικά ο κώδικας είναι ο εξής :

```
Awards.findOne().where({arenaId: arenaId})
  .exec(function (err, result) {
    if (err) {
      return res.status(500).json({
        message: 'Unexpected Error'
```

```
        })
      }
      try {
        if (result.awards.winner.userId == userId) {
          try {
            if (result.awards.winner.received != false) {
              res.status(500).json({
                where: 'At awards',
                title: 'Error',
                message: 'You already received that award....',
                status: '500',
                error: err
              })

            } else {

statisticAwardController.statisticsAddedWinner(req,res,next,userId,result,arenaId).then(function (message) {
                return res.status(200).json({
                  message: message.message
                })
                }
              ).catch(function (err) {
                return res.status(500).json({
                  where: err.where,
```

```
            message: err.message,
            error: err.error
          })
          }
        )
      }
    } catch (err) {
      return res.status(500).json({
        where: 'Awards',
        message: 'Unexpected error',
        error: err
      })
    }
```

7.2.2 Δημιουργία της οθόνης

Δημιουργούμε την οθόνη εκτελώντας την εντολή *ionic g page myArenas* .Στην οθόνη αυτή θα βλέπουμε μια λίστα με τις ανοικτές αρένες αν υπάρχουν. Αν κάποια αρένα από αυτές δεν την έχουμε παίξει θα φαίνεται πάνω στην οθόνη ότι μπορούμε να παίξουμε απλά πατώντας πάνω της. Αν είναι αρένα που έχουμε παίξει τότε η αρένα αυτή θα βρίσκετε σε αναμονή μέχρι να παίξει και ο αντίπαλος. Η τρίτη κατάσταση της αρένας είναι όταν και οι δύο χρήστες έχουνε παίξει και η αρένα θα εμφανίζει ένα μήνυμα για να παραλάβουμε το βραβείο μας. Συνοπτικά οι συναρτήσεις για να παίξουμε την αρένα και να παραλάβουμε το βραβείο αντίστοιχα :

```
playMatch(arena: Arenas) {
if (arena.user_played == true && arena.userId == this.userId ||
arena.invite_played == true && arena.inviteId == this.userId) {
console.log('you already played');
```

```
        } else {
        this.appCtrl.getRootNav().push('MatchPage', { arena: arena });

        }

}

        getReward() {

        let modal = this.modalCtrl.create('ShowRewardPage', { arenaId:
        this.arena.arenaId, userId: this.userId }, { enableBackdropDismiss: false });
        modal.onDidDismiss(data => {
        if (data != null) {
        this.tabs.removeArena(data.arenaId);
        }
        });
        modal.present();

        }
```

Παρακάτω βλέπουμε την δημιουργία μια αρένας στο αρχείο .html:

```
        <div *ngIf="arena.userId==userId && arena.user_played==false||
        arena.inviteId==userId && arena.invite_played==false"
        class="arenaContainer">

        <ion-row class="rowContainer">
        <ion-col text-center col-4 class="userCol">
        <ion-row class="userNameRow">
        <ion-col class="userNameContainer">
        <p>
        {{username}}

        </p>
```

```
</ion-col>
</ion-row>

<ion-row class="buttonRow">
<ion-col>
<button class="btn" (click)="playMatch(arena)">
<span><div class="word-container">
<p>ΠΑΙΞΕ</p>
</div>
</span>
</button>
</ion-col>
</ion-row>
</ion-col>

<ion-col text-center col-4 class="halftimeCol">
<ion-row class="halfTime1row">
<ion-col style="color:black">
<p>
.
</p>
</ion-col>
</ion-row>
<ion-row class="halftimeRow">
<ion-col class="halfTimeContainer">
<p>
ΗΜΙΧΡΟΝΟ
</p>
</ion-col>
</ion-row>
</ion-col>

<ion-col text-center col-4 class="opponentCol">
<ion-row class="userName">
```

```
<ion-col class="userNameContainer">

<p> Ειναι η σειρα σου να παιξεις με τον {{arena.userName}} </p>
</ion-col>
</ion-row>
<ion-row class="userName">
<ion-col class="correctNumber">
<p>
?
</p>
</ion-col>
</ion-row>
</ion-col>

</ion-row>
</div>
```

7.3 Περίληψη

Δημιουργήσαμε την οθόνη της αρένας και τις λειτουργίες της ,ο χρήστης πλέον μπορεί να παίξει και να απαντήσει τις ερωτήσεις. Στο επόμενο κεφάλαιο θα φτιάξουμε την οθόνη όπου ο χρήστης θα μπορεί να δει τις αρένες που έχει ανοιχτές καθώς και να παραλάβει τα βραβεία του .

Δημιουργήσαμε και την τελευταία οθόνη της εφαρμογής και είδαμε κάποιες από τις λειτουργίες τις. Το quiz είναι έτοιμο και ο χρήστης μπορεί να παίξει κανονικά με ένα αντίπαλό .

ΚΕΦΑΛΑΙΟ 8

Επίλογος Συμπεράσματα

Η τεχνολογία για τα εργαλεία για την ανάπτυξη εφαρμογών εξελίσσεται με ταχείς ρυθμούς, καθημερινά δημιουργούνται καινούργια framework και γλώσσες προγραμματισμού που κάνουν την δουλειά του προγραμματιστή ευκολότερη. Η γλώσσα javascript την οποία χρησιμοποιήσαμε για την ανάπτυξη της εφαρμογής είναι η δημοφιλέστερη γλώσσα ανάμεσα στους προγραμματιστές και είναι η βάση για τα περισσότερα frontend frameworks.

Πλέον ένας προγραμματιστής πολύ δύσκολα θα χρησιμοποιήσει σκέτη javascript χωρίς την βοήθεια κάποιου framework για να αναπτύξει μια εφαρμογή. Το framework της Angular που χρησιμοποιήσαμε είναι ένα από τα δημοφιλέστερα framework για την ανάπτυξη διαδικτυακών εφαρμογών. Με την βοήθεια της Angular δημιουργήσαμε μια εφαρμογή η οποία είναι διαθέσιμη σχεδόν σε όλες τις πλατφόρμες και με αυτό τον τρόπο εξοικονομούμε χρόνο συντηρώντας μόνο ένα πηγαίο κώδικα τουλάχιστον για τον front end κομμάτι. Ομοίως και για το backend χρησιμοποιήσαμε το NodeJs χωρίς να χρειαστούμε να μάθουμε κάποια άλλη γλώσσα από την javascript.

Δημιουργήσαμε ένα quiz που μπορούμε να αγωνιστούμε εναντίων άλλων χρηστών. Σε κάποια μελλοντική ενημέρωση θα μπορούσαμε να αλλάξουμε την εγγραφή των χρηστών στην εφαρμογή και να χρησιμοποιήσουμε αυτή που μας παρέχει η το google play services οπού η εγγραφή γίνετε μόνο το email της google που είμαστε συνδεδεμένοι και έτσι ο χρήστης δεν θα χρειάζεται να θυμάται τον κωδικό του και το όνομα χρήστη.

Ακόμα μια αλλαγή θα ήταν η βελτίωση του κώδικα εύρεσης τυχαίου αντιπάλου , έτσι ώστε να είναι ποιο δίκαιος και να βρίσκει αντιπάλους που να είναι περίπου στο ίδιο επίπεδο με εμάς .

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] <<Mongo db docs>>https://docs.mongodb.com/?_ga=2.68943897.946141296.1524817933-576424928.1523566652

[2] <<Quick start guide mongolab>> https://docs.mlab.com/

[3] <<Node Js docs>> https://nodejs.org/en/docs/

[4] Pedro Texeira, Proffessional Node.js:Building Javascript Based Scalable Software(1η έκδοση) 2012

[5] <<ExpressJS api reference>> https://expressjs.com/en/4x/api.html

[6] <<Angular docs>> https://angular.io/docs

[7] <<Ionic docs>> https://ionicframework.com/docs/

[8] <<Firebase docs>> https://firebase.google.com/docs (documenation του firebase)

[9] <<Heroku docs>> https://devcenter.heroku.com/

[10] Code craft, Angular 5: From Theory To Practice: Build the web applications of tomorrow using the new Angular web framework from Google,Amazon,2017