Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Full Stack Javascript Development σε MEAN περιβάλλον** **Full Stack Javascript Development using MEAN stack**
Ονοματεπώνυμο Φοιτητή	**Στέφανος Βαρδάλος**
Πατρώνυμο	**Ηλίας**
Αριθμός Μητρώου	**ΜΠΣΠ/ 13012**
Επιβλέπων	**Κωνσταντίνος Πατσάκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Ιούνιος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Κωνσταντίνος Πατσάκης Επίκουρος Καθηγητής	Ευθύμιος Αλέπης Επίκουρος Καθηγητής	Γεώργιος Τσιχριτζής Καθηγητής

Περίληψη

Στόχος της εργασίας είναι η έρευνα και η μελέτη των καινούργιων εργαλείων αλλά και τεχνολογιών , μεθοδολογιών της γλώσσας προγραμματισμού Javascript. Αποτέλεσμα της εργασίας θα είναι η δημιουργία μίας εφαρμογής τύπου Project Management με τη χρήση των σημαντικότερων εργαλείων της μεθοδολογίας MEAN για να φανούν στην πράξη τα προτερήματα και τα μειονεκτήματα της συγκεκριμένης τεχνολογίας που κερδίζει συνέχεια χώρο στον κόσμο του Web Development.

Η εφαρμογή αυτή καθ' αυτή θα περιέχει όλες τις λειτουργίες εκείνες που παρέχονται και από τις αντίστοιχες εφαρμογές που κυκλοφορούν στο εμπόριο. Θα επιτρέπει την εγγραφή / σύνδεση του χρήστη τόσο με το email του όσο και με τους social λογαριασμούς του , θα έχει μία διεπαφή τύπου chat για την επικοινωνία με τους υπόλοιπους χρήστες και θα μπορεί να διαχειρίζεται τα projects / issues του.

Όλα τα παραπάνω θα υλοποιηθούν αποκλειστικά με την χρήση της γλώσσας JavaScript και των διαφόρων ειδών frameworks της. Εκτός αυτών που τελικά θα χρησιμοποιηθούν, θα γίνει αναφορά στα σημαντικότερα ανά τομέα frameworks καθώς και μια σύγκριση αυτών, ώστε να γίνουν πιο κατανοητές οι διαφορές τους αλλά και η σημασία τους. Όλα τα παραπάνω θα βοηθήσουν στην ανάλυση του τίτλου Full Stack Developer, όπως έχει καθιερωθεί, δηλαδή του προγραμματιστή που μπορεί να ασχοληθεί με όλο το Stack της εφαρμογής (Back-End , Front-End , Database).

Abstract

The purpose of this thesis is the research and study of all the new tools, technologies and methodologies of JavaScript programming language. The final outcome of this thesis will be the development of a Project Management application using the most important parts of the MEAN stack so that by the end of it, we will be able to understand the advantages and disadvantages of this particular stack.

The final application will have all the features a user can get from other similar commercial applications. He will be able to register or login with his email or using one of his social accounts, he will be able to chat with the other users of the application and most importantly, he will be able to manage his projects / issues.

All of the above will be implemented using only the JavaScript language and its various frameworks. Besides those that will be used, all the most important frameworks per category will be mentioned and finally compared, so that their differences will become more noticeable. Studying all of the above will make clearer the Full Stack Developer title, meaning the one that can participate with the same ability on the whole stack of the application (Back-End, Front-End, Database).

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

Κεφάλαιο 1ο

1. Εισαγωγή

Η ιδέα για την εργασία αυτή προέρχεται από το ιδιαίτερο ενδιαφέρον που προσφέρουν οι νέες τάσεις στον χώρο του προγραμματισμού στο Διαδίκτυο. Τα τελευταία χρόνια έχουν εμφανιστεί διάφορες γλώσσες προγραμματισμού που προσπαθούν, και κάποιες το καταφέρνουν, να αποκτήσουν κοινό στον χώρο των προγραμματιστών, είτε επειδή κάνουν πράγματα πιο εύκολα από άλλες γλώσσες, είτε επειδή ενσωματώνουν χαρακτηριστικά που λείπουν από άλλες, ή για άλλους λόγους. Η τάση αυτή, της γρήγορης δημιουργίας και προώθησης νέων γλωσσών (χαρακτηριστικά παραδείγματα η Ruby και η Python), έχει ξεκινήσει εδώ και μερικά χρόνια αλλά ακόμα δεν έχει κορυφωθεί, αφού δεν έχει κατασταλάξει ακόμα ο κόσμος σε κάποια από τις καινούργιες τεχνολογίες.

Παράλληλα με την τάση αυτή, παρατηρείται η ανάδειξη ενός καινούργιου όρου/ρόλου στον χώρο, αυτό του Full Stack Developer. Με τον όρο αυτό, χαρακτηρίζεται ο προγραμματιστής που δεν εξειδικεύεται αποκλειστικά σε έναν τομέα του development (Back-end , Front-end , Database) αλλά είναι ικανός να ασχοληθεί με όλους τους τομείς και συνήθως αναλαμβάνει να υλοποιήσει κομμάτια από όλη την ροή του προγράμματος.

Τα δύο παραπάνω στοιχεία δημιούργησαν ουσιαστικά την τεχνολογία MEAN. Το ακρώνυμο αυτό δημιουργείται από τις επιμέρους τεχνολογίες MongoDB – ExpressJS – AngularJS – NodeJS αλλά δεν αποτελείται αναγκαστικά από αυτές, όπως αντίστοιχα το άλλο μέχρι τώρα πολύ σύνηθες stack LAMP μπορούσε να περιλαμβάνει τεχνολογίες διαφορετικές από τις Apache – MySQL – PHP από τις οποίες αρχικά ξεκίνησε. Στο Mean Stack, η ιδιαιτερότητα και η μοναδικότητα που παρατηρείται είναι πως όλες οι τεχνολογίες που το απαρτίζουν είναι ουσιαστικά όλες frameworks της γλώσσας JavaScript. Από το Back-End κομμάτι (NodeJS , ExpressJS) στο Front-End (AngularJS) μέχρι και στη βάση δεδομένων (MongoDB) το stack αυτό περιλαμβάνει αποκλειστικά JavaScript, οπότε είναι πολύ πιο εύκολο για έναν γνώστη της συγκεκριμένης γλώσσας να ασχοληθεί και να επεκταθεί σε όλο το φάσμα του Development, δηλαδή να γίνει Full Stack Developer. Ταυτόχρονα , η JavaScript είναι μια γλώσσα που υπάρχει ήδη αρκετά χρόνια, έχει δοκιμαστεί εκτενώς και υπάρχουν ήδη πολλοί προγραμματιστές με ιδιαίτερη ευχέρεια σε αυτήν, οπότε και θα τους είναι σχετικά εύκολο να μεταπηδήσουν στις νέες αυτές τεχνολογίες.

1.1 Περιγραφή του υπό μελέτη προβλήματος

Με την ανάδειξη αυτών των τεχνολογιών δημιουργήθηκαν πάρα πολύ γρήγορα, πάρα πολλά και διαφορετικά frameworks για κάθε κομμάτι του stack. Κάθε ένα από αυτά προσφέρει κάποια θετικά και κάποια αρνητικά στοιχεία ενώ , όπως φαίνεται, ακόμα δεν υπάρχει κάποιο framework το οποίο να είναι καθολικά αποδεκτό άρα και η σίγουρη επιλογή για τον προγραμματιστή η τον project leader. Η ταχύτητα όμως με την οποία αυτά δημιουργούνται, , κάνει ιδιαιτέρως δύσκολη την ανακάλυψη και την αναγνώρισή τους, πόσο μάλλον τη μελέτη και χρήση τους. Χαρακτηριστικά μπορούν να αναφερθούν μερικά από αυτά παρακάτω :

1. Back-End
    a. NodeJS
    b. Express.js
    c. Hapi.js

d. RESTify
e. Total.js

2. Front-End
    a. AngularJS
    b. Ember
    c. Backbone
    d. React
    e. Knockout
3. Database
    a. MongoDB
    b. CouchDB
    c. Redis
    d. Cassandra
    e. Neo4j

1.2 Σκοπός και στόχοι της εργασίας

Στόχος της εργασίας είναι η ανασκόπηση μερικών εκ των σημαντικότερων frameworks του MEAN stack περιβάλλοντος , η σύγκριση και η μελέτη των ιδιαιτεροτήτων του καθενός από αυτά, ώστε να αποσαφηνιστεί ο ρόλος του καθενός και να γίνει ευκολότερη η επιλογή και η μετάβαση σε αυτές τις τεχνολογίες. Για τον λόγο αυτό, θα δημιουργηθεί μια εφαρμογή τύπου Project Management ώστε να φανεί πώς γίνεται με μία μόνο γλώσσα , την JavaScript , να δημιουργήσουμε μια ολοκληρωμένη εφαρμογή, αλλά και τα προτερήματα / μειονεκτήματα αυτής της επιλογής. Παράλληλα θα αναλυθούν και τα υπόλοιπα στάδια του Development, τα οποία πρέπει να γνωρίζει και να εκτελεί ένας Full Stack Developer, δηλαδή το build της εφαρμογής με χρήση αντίστοιχων εργαλείων (Gulp , Grunt) , το optimization (minify , concatenation , uglify , compression) , το testing (Mocha , Karma) και φυσικά το security (PassportJS , Lusca , Json Web Tokens).

Μετά την ανασκόπηση των σημαντικότερων εξ αυτών, και τη δημιουργία της εφαρμογής με τη χρήση κάποιων από αυτά, ο στόχος είναι να γίνει πια πιο εύκολη και σωστή η επιλογή της κατάλληλης τεχνολογίας για το εκάστοτε project.

1.3 Βασικοί ορισμοί

Ακολουθούν μερικές λέξεις κλειδιά μαζί με τους ορισμούς τους που θα χρησιμοποιηθούν κατά τη διάρκεια της διπλωματικής και θα πρέπει να είναι από πριν κατανοητές.

Software Stack – Είναι ένα πακέτο ξεχωριστών υποσυστημάτων που όλα μαζί δημιουργούν μια ολοκληρωμένη πλατφόρμα πάνω στην οποία μπορεί να λειτουργήσει αυτόνομα ένα σύστημα.

Back-End / Front-End Development – Είναι ο διαχωρισμός της φάσης της δημιουργίας μίας εφαρμογής ανάμεσα στο κομμάτι που θα τρέχει στον απομακρυσμένο server και σε αυτό που θα τρέχει στον υπολογιστή του χρήστη.

API – Είναι η Διασύνδεση Προγραμματισμού Εφαρμογών, ή αλλιώς διεπαφή, με την οποία ένα σύστημα μπορεί να επικοινωνήσει με ένα άλλο και να ζητήσει δεδομένα ή λειτουργίες.

Framework – Είναι ένα πρόγραμμα, ή σειρά προγραμμάτων που σκοπός τους είναι να προσφέρουν περισσότερες επιλογές και ευκολίες στη χρήση μιας γενικότερης πλατφόρμας.

Βιβλιοθήκη – Είναι μια συλλογή από εφόδια που έχουν φτιαχτεί για να κάνουν πιο εύκολη την διαδικασία του Development. Μοιάζει νοητικά με το Framework αλλά συνήθως είναι πολύ μικρότερου μήκους και δεν δημιουργούνται για συγκεκριμένες τεχνολογίες.

MVC – Είναι ένα μοντέλο αρχιτεκτονικής λογισμικού, σύμφωνα με το οποίο η εφαρμογή διαιρείται σε τρία ξεχωριστά κομμάτια, ώστε να είναι αποσυνδεδεμένη η παρουσίαση της πληροφορίας στον χρήστη από τη μορφή που θα βρίσκονται τα δεδομένα στο σύστημά μας.

DOM – Είναι μια ιεραρχική δομή παρουσίασης των στοιχείων που συνθέτουν μια ιστοσελίδα (και όχι μόνο).

NoSQL – Είναι νέα τεχνολογία βάσεων δεδομένων και το όνομά της προέρχεται από το Non SQL, θέλοντας να υποδείξουν την διαφορετικότητα αυτών από τις συμβατικές βάσεις δεδομένων.

Relational Model – Είναι μία μέθοδος διαχείρισης των δεδομένων. Οι βάσεις που το ακολουθούν χρησιμοποιούν πίνακες με δεδομένα και σχέσεις, που ενώνουν τους πίνακες αυτούς μεταξύ τους.

Database Schema – Είναι η λογική σχεδίαση του διαχωρισμού των δεδομένων σε πίνακες και της εύρεσης των συνδέσεων αυτών μεταξύ τους. Σύμφωνα με αυτό θα δημιουργηθεί και η βάση δεδομένων που θα γεμίσει με αυτά τα δεδομένα.

Horizontal / Vertical Scaling – Είναι μέθοδοι αναβάθμισης συστημάτων. Στην περίπτωση του Horizontal Scaling, προσθέτουμε στο ίδιο σύστημα καλύτερα χαρακτηριστικά (δυνατότερο επεξεργαστή, περισσότερη μνήμη RAM). Στην περίπτωση του Vertical Scaling, δημιουργούμε αντίγραφα του συστήματός μας και μεταφέρουμε μέρος του φόρτου εργασίας σε εκείνα.

JSON – Είναι μορφή αναπαράστασης δεδομένων και δημιουργήθηκε για την ευκολότερη μεταφορά αυτών στο Διαδίκτυο.

1.4 Παραδοτέα της εργασίας

Η παρούσα εργασία θα παραδοθεί στα εξής επιμέρους τμήματα :

1. Το έντυπο κείμενο της πτυχιακής εργασίας, το οποίο περιλαμβάνει την επισκόπηση , ανάλυση και μελέτη των τεχνολογιών που περιλαμβάνει το MEAN περιβάλλον προγραμματισμού, καθώς και τη διαδικασία δημιουργίας της εφαρμογής τύπου Project Management.
2. Ο κώδικας που αναπτύχθηκε για την δημιουργία της εφαρμογής Project Management, σε Back-end , Front-end και Database επίπεδο.

1.5 Δομή της εργασίας

Η παρούσα εργασία θα αναλυθεί σε επιμέρους τμήματα για να γίνει όσο το δυνατό περισσότερο κατανοητό κάθε κομμάτι αυτής. Στα πρώτα κομμάτια θα γίνει η περιγραφή και η ανάλυση των εκάστοτε τεχνολογιών μαζί με κομμάτια κώδικα που θα κάνουν πιο εύκολη την παρακολούθηση του κειμένου. Στα επόμενα κομμάτια , θα δούμε τον κώδικα και την διαδικασία υλοποίησης της εφαρμογής Project

Management. Τέλος, θα περιγραφούν οι προβληματισμοί και τα σχόλια γύρω από την εφαρμογή και τις τεχνολογίες που χρησιμοποιήθηκαν.

Κεφάλαιο 2ο

2. Επισκόπηση του MEAN Stack

Η ανάλυση που θα ακολουθήσει αφορά το MEAN Software Stack , δηλαδή το πακέτο προγραμμάτων που μπορούν να δημιουργήσουν την παραπάνω τεχνολογία. Το πακέτο αυτό περιλαμβάνει όλες τις πλευρές της διαδικασίας του προγραμματισμού μιας εφαρμογής και στα επόμενα κεφάλαια θα μελετηθεί η κάθε μία ξεχωριστά.

Συγκεκριμένα , στα επόμενα κεφάλαια θα αναλυθούν τα παρακάτω :

1. Back-End Programming
2. Front-End Side
3. Database Design and Implementation
4. Unit Testing
5. Software Optimization
6. Automation – Task Runners
7. Security
8. Project Management App Development

Σε κάθε ένα από τα επόμενα μέρη της εργασίας θα γίνει η ανάλυση ενός εκ των παραπάνω κομματιών του Development, με την επισκόπηση των επιμέρους εργαλείων που μπορούν να χρησιμοποιηθούν. Αφού αναλυθούν επιμέρους τα παραπάνω στοιχεία, θα ακολουθήσει η μελέτη της εφαρμογής όπου θα φανεί η πραγματική χρήση όλων όσων θα έχουν αναφερθεί μέχρι τότε. Με αυτό τον τρόπο θα γίνει πιο εύκολα κατανοητή η ροή της διαδικασίας του Development, μιας εφαρμογής γενικότερα και ειδικότερα με την χρήση όλων αυτών των τεχνολογιών.

2.1 Back-End Programming

Το Back-End, κομμάτι του προγραμματισμού, είναι αυτό που τρέχει στον διακομιστή (server) της εφαρμογής και δεν γίνεται αντιληπτό από τον τελικό χρήστη. Η JavaScript είναι μια γλώσσα που γράφτηκε αποκλειστικά για την χρήση της στο Front-End κομμάτι και μέχρι πριν λίγο καιρό η χρήση της στο Back-End όχι μόνο δεν ήταν δυνατή, αλλά έμοιαζε και παράλογη ως σκέψη.

Αυτό άλλαξε με την έλευση της πλατφόρμας Node.js . Το Node.js δημιουργήθηκε αρχικά από τον Ryan Dahl το 2009, ενώ είναι open-source project. Χρησιμοποιεί την V8 JavaScript engine της Google και διαφέρει σε φιλοσοφία από τους συμβατικούς servers αφού δεν στηρίζεται στην πολυνηματικότητα (multi-threaded), αλλά στην ασύγχρονη επικοινωνία και την εκμετάλλευση των callbacks της JavaScript. Το μοντέλο αυτό (event-callback) επιτρέπει την ευκολότερη δημιουργία εφαρμογών με πολύ μεγάλο κοινό χωρίς την ανάγκη διαχείρισης νημάτων. Παρακάτω φαίνεται πόσο εύκολα, με ελάχιστες γραμμές δημιουργείται ένας server με την Node.js.

```
var http = require('http');

http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World\n');
}).listen(8080);

console.log('Server started');
```

Το Node.js χρησιμοποιείται ήδη από πολλές μεγάλες εταιρείες, μεταξύ των οποίων οι Netflix , Uber , LinkedIn , Paypal. Πέρα από τις μεγάλες εταιρείες όμως, έχει και πολύ μεγάλη βάση χρηστών, οι οποίοι επεκτείνουν συνέχεια τις δυνατότητες του Node με τη δημιουργία επιπρόσθετων Frameworks που προσθέτουν πολλές και χρήσιμες λειτουργίες. Για τον λόγο αυτό άλλωστε δημιουργήθηκε και το npm, ένα εργαλείο τύπου package manager μέσω του οποίου με τη χρήση εντολών κονσόλας μπορεί ο χρήστης να κατεβάσει πακέτα – προγράμματα που έχουν δημιουργηθεί από άλλους χρήστες και να τα ενσωματώσει στην εφαρμογή του. Αυτά τα Frameworks στόχο έχουν να γίνει πιο εύκολη η δημιουργία της δρομολόγησης (routing) του server για να δημιουργηθεί τελικά μία διεπαφή API με την οποία θα επικοινωνεί η εφαρμογή με τον server. Οι σύγχρονες εφαρμογές , με τα καινούργια front-end εργαλεία που δημιουργούν όλη την εφαρμογή στον υπολογιστή του χρήστη, περιμένουν από τον server κυρίως την τροφοδότηση με δεδομένα τα οποία στο front-end θα εμφανιστούν αναλόγως. Έτσι γίνεται κατανοητό πως η σημαντικότερη δουλειά ενός server στο MEAN περιβάλλον είναι η παροχή ενός σωστού και γρήγορου API που θα επικοινωνεί με το front-end κομμάτι της εφαρμογής.

Εκτός από τη δημιουργία του API, τα frameworks, μπορούν να παρέχουν και άλλα εργαλεία για την ταχύτερη και σωστότερη λειτουργία του server , όπως σύνδεση με τη βάση δεδομένων, διαχείριση σφαλμάτων , βελτιστοποίηση προγράμματος (minify, compress) , συγχρονισμός σε πραγματικό χρόνο κατά την μεταφορά δεδομένων και αρχείων και άλλα πολλά. Το Node.js είναι σχεδόν μονόδρομος για το MEAN περιβάλλον , αλλά εδώ έρχεται η πρώτη επιλογή του προγραμματιστή, καθώς υπάρχουν πολλά και διαφορετικά web application frameworks με τα οποία μπορεί να συνδυάσει το Node. Στον παρακάτω πίνακα είναι μονάχα μερικά από αυτά, μαζί με το νούμερο της τελευταίας έκδοσής τους αλλά και τον αριθμό downloads που έχουν γίνει μέσω του εργαλείου npm, ενώ στη συνέχεια θα ακολουθήσει η ανάλυση μερικών εξ αυτών.

Framework	Latest Release	Downloads τον τελευταίο μήνα
Express.js	4.13.3	3.885.760
Hapi.js	11.1.4	167.187
Restify	4.0.3	116.882
Koa.js	1.1.2	53.860
Sails.js	0.11.3	54.108
Total.js	1.9.5	17.651
Mojito	0.9.8	1.117
Meteor	0.5.2	1.063

Πίνακας 1. Web Application Frameworks

2.1.1. Express.js

Το Express είναι το framework που συναντάται στα περισσότερα projects, από τα πρώτα που δημιουργήθηκαν (το 2009, πολύ κοντά με τη δημιουργία του ίδιου του Node.js) και έτσι είναι αυτό που μπήκε και στο ακρωνύμιο MEAN. Χαρακτηρίζεται από την ταχύτητά του και τον ελάχιστο, μη-δογματικό χαρακτήρα του, αφού δεν επιβάλει στον προγραμματιστή ειδικές μεθόδους ή τακτικές. Φυσικά το μεγάλο του προτέρημα είναι η ηλικία του, καθώς μέσα από όλα αυτά τα χρόνια development έχει γίνει ένα πολύ σταθερό framework που χιλιάδες χρήστες χρησιμοποιούν για τις εφαρμογές τους. Λόγω του χαρακτήρα του είναι πολύ εύκολο να ξεκινήσει κάποιος με αυτό και να δημιουργήσει εύκολα μια εφαρμογή.

Στον αντίποδα, το μεγάλο του μειονέκτημα είναι αυτό για το οποίο διαφημίζεται. Η μη-δογματική μορφή του αφήνει το περιθώριο να γίνονται πολλά πράγματα με πολλούς διαφορετικούς τρόπους, πράγμα που στην αρχή θα βοηθήσει τον άπειρο προγραμματιστή αλλά σε μεγάλα projects που δουλεύουν ταυτόχρονα πολλά άτομα αυτό σίγουρα θα δημιουργήσει πολλά προβλήματα. Επίσης, σαν framework παρέχει τα ελάχιστα έτοιμα εργαλεία, πράγμα που δεν είναι απαραίτητα κακό αφού δίνει την ελευθερία στον χρήστη να δημιουργήσει τα δικά του, αλλά ταυτόχρονα αυτός είναι και ο ρόλος ενός σωστού framework. Παρακάτω φαίνεται πώς δημιουργείται ένας server Node.js ,με τη βοήθεια του Express αυτή τη φορά.

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

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});
```

2.1.2. Hapi.js

Το Hapi.js δημιουργήθηκε το 2011 και μπορεί να χαρακτηριστεί ως ιδιαίτερο framework , αφού η φιλοσοφία του είναι τέτοια που θέτει το Business Logic κομμάτι πάνω από τον ίδιο τον κώδικα, ώστε να δημιουργούνται εφαρμογές ή υπηρεσίες που να μπορούν εύκολα να ξαναχρησιμοποιηθούν σε άλλα προγράμματα. Παρέχει πάρα πολλά έτοιμα εργαλεία στον προγραμματιστή, όπως διαχείριση και καταγραφή σφαλμάτων , διαχείριση caching και αυθεντικοποίηση χρηστών. Το framework αυτό, στηρίζεται ήδη από μεγάλες εταιρείες λογισμικού, οι οποίες το έχουν ήδη χρησιμοποιήσει σε production περιβάλλον όπου και λειτουργεί χωρίς προβλήματα.

Το μειονέκτημά του Hapi είναι η πολύ μικρή βάση χρηστών του και το ελάχιστο υλικό που έχει δημιουργηθεί γι' αυτό στο διαδίκτυο. Σαν framework είναι ξεκάθαρο πως απευθύνεται σε μεγάλα κυρίως project και γι' αυτό δεν το έχουν ακολουθήσει πάρα πολύ χρήστες, αφού με την εγκατάστασή του

αποκτάται αυτόματα κώδικας για πάρα πολλά πράγματα που σε μικρά project είναι αναξιοποίητα. Παρακάτω φαίνεται πώς δημιουργείται ένας server Node.js, με τη βοήθεια του Hapi.js αυτή τη φορά.

```
const Hapi = require('hapi');

const server = new Hapi.Server();
server.connection({ port: 3000 });

server.start(() => {
    console.log('Server running at:', server.info.uri);
});
```

2.1.3. Restify

Το Restify είναι ένα πολύ μικρό framework το οποίο είναι υπεύθυνο για μία και μόνο δουλειά, τη σωστή δημιουργία της API διεπαφής του server. Όπως μαρτυράει και το όνομά του, αυτός είναι ο μοναδικός του σκοπός και δεν προσφέρει καθόλου επιπλέον λειτουργίες. Η δουλειά όμως που κάνει είναι και η σημαντικότερη σε έναν node server, οπότε έχει καταφέρει και έχει βρει μία βάση χρηστών. Αφού δεν προσφέρεται μέσα από το ίδιο το framework, τα άτομα τα οποία θα επιλέξουν να το χρησιμοποιήσουν είναι αναγκασμένα να πραγματοποιήσουν πολλές εργασίες από την αρχή, πράγμα το οποίο αποτελεί ανασταλτικό παράγοντα για τους αρχάριους προγραμματιστές. Παρακάτω φαίνεται πώς δημιουργείται ένας server Node.js, με τη βοήθεια του Restify αυτή τη φορά.

```
var restify = require('restify');

function respond(req, res, next) {
  res.send('hello ' + req.params.name);
  next();
}

var server = restify.createServer();
server.get('/hello/:name', respond);
server.head('/hello/:name', respond);

server.listen(8080, function() {
  console.log('%s listening at %s', server.name, server.url);
});
```

2.1.4. Koa.js

Το Koa είναι ένα σχετικά καινούργιο framework, δημιουργήθηκε μόλις το 2013 και μάλιστα από την ομάδα που ξεκίνησε το Express.js . Αυτό γίνεται εύκολα κατανοητό παρατηρώντας το συντακτικό των δύο framework. Έτσι, μπορεί εύκολα κάποιος με εμπειρία στο Express να κάνει την μετάβαση στο Koa. Το Koa όμως διαφέρει από το Express στη φιλοσοφία του. Το Express προσπαθεί να ακολουθήσει την Node, να την συμπληρώσει και απλά να της προσθέσει περισσότερα χαρακτηριστικά. Το Koa από την άλλη, προσπαθεί να διορθώσει πράγματα που δεν αρέσουν στην Node. Η κύρια διαφοροποίηση είναι πως ακολουθεί μια

τακτική με generators αντί για callbacks για να μπορεί να χειρίζεται πιο αποδοτικά τα σφάλματα. Επίσης, έχει λιγότερες λειτουργίες ενσωματωμένες στον κώδικά του, επιτρέποντας στον χρήστη είτε να φτιάξει τις δικές του μεθόδους για διάφορες λειτουργίες είτε να ενσωματώσει κώδικα τρίτων, επιλέγοντας ό,τι τον βολεύει καλύτερα ανά περίσταση. Παρακάτω φαίνεται πώς δημιουργείται ένας server Node.js, με την βοήθεια του Koa αυτή τη φορά.

```
var koa = require('koa');
var app = koa();

app.use(function *(){
  this.body = 'Hello World';
});

app.listen(3000);
```

2.1.5. Sails.js

Το Sails.js δημιουργήθηκε το 2012, και όπως αναφέρουν οι δημιουργοί του, φτιάχτηκε με γνώμονα το framework Rails της γλώσσας Ruby. Ακολουθεί τη γνωστή MVC αρχιτεκτονική, πράγμα που βοηθάει στη γρήγορη ανάπτυξη ενός project. Σε αντίθεση με το Express , το Sails είναι ένα αυστηρά opinionated framework, δηλαδή πολλά πράγματα πρέπει να γίνονται με κάποιον συγκεκριμένο τρόπο χωρίς να αφήνει την επιλογή στον χρήστη. Αυτό είναι πολύ καλό για κάποιον αρχάριο, ή για ένα μικρό – μεσαίο project στο οποίο πρέπει να γίνουν συγκεκριμένα πράγματα και πολύ γρήγορα, αλλά σε μεγάλα projects που οι ανάγκες μπορούν να αλλάξουν σε βάθος χρόνου, αυτές οι ιδιοτροπίες του framework μπορούν να δημιουργήσουν σοβαρά προβλήματα. Αντίστοιχα, το Sails έρχεται με πάρα πολλά ενσωματωμένα χαρακτηριστικά που θα βοηθήσουν αμέσως τον χρήστη για να φτιάξει μια εφαρμογή πολύ επαγγελματικού επιπέδου. Τέλος, επειδή ενσωματώνει τη βιβλιοθήκη Socket.io, η οποία βοηθά στην δημιουργία εφαρμογών όπου η επαφή με τους χρήστες γίνεται σε πραγματικό χρόνο, είναι ιδανικό framework για τη δημιουργία εφαρμογών όπως chat και multiplayer παιχνιδιών.

2.2 Front-End Programming

Το Front-End, κομμάτι του προγραμματισμού, είναι αυτό που τρέχει στο περιβάλλον του χρήστη, στον browser του δηλαδή. Τα προηγούμενα χρόνια αυτό περιελάμβανε τον προγραμματισμό σε HTML , CSS και JavaScript , όμως γρήγορα ξεκίνησαν να εμφανίζονται βιβλιοθήκες που βοηθούσαν στην εύκολη δημιουργία διαδραστικών προγραμμάτων (jQuery). Τώρα έχουν δημιουργηθεί πολλά, ολοκληρωμένα JavaScript Frameworks, που προσφέρουν τρομερές λειτουργίες στον χρήστη και πολλά εφόδια για τη δημιουργία πλούσιων σε χαρακτηριστικά εφαρμογών. Η χρήση ενός τέτοιου framework έχει πάρα πολλά προτερήματα, αφού δεν χρειάζεται κάθε φορά που ξεκινάει ένα project να δημιουργούνται τα πάντα από την αρχή, ο προγραμματιστής μπορεί να κάνει πράγματα γράφοντας πολύ λιγότερο κώδικα, γλιτώνει πολύτιμο χρόνο και μπορεί να βασιστεί πάνω σε κώδικα που έχει γραφτεί και δοκιμαστεί από πάρα πολύ κόσμο, άρα και αρκετά έμπιστο.

Πέρα από αυτά, τα σύγχρονα JavaScript Frameworks, είτε επιβάλλουν, είτε προωθούν την MVC αρχιτεκτονική. Το μοντέλο Model – View – Controller είναι ένας πολυχρησιμοποιημένος και αποδεδειγμένα επιτυχημένος τρόπος για να οργανώσεις σωστά ένα πρόγραμμα και με μία δομή τέτοια που οι αλλαγές, τροποποιήσεις, εξελίξεις που μπορεί να χρειαστούν να γίνουν στο μέλλον να είναι εύκολα υλοποιήσιμες.

Ειδικότερα, στο MEAN περιβάλλον, που ο server αποτελείται κυρίως από ένα REST API, η MVC αρχιτεκτονική είναι μονόδρομος.

Αυτό γίνεται καλύτερα κατανοητό ορίζοντας τις ανάγκες που υπάρχουν από το Front-End :

1. Λήψη / Αποστολή δεδομένων (μέσω κλήσεων στο API)
2. Παρουσίαση των δεδομένων (μέσω αλλαγών στη σελίδα που εμφανίζεται στον browser)
3. Αλληλεπίδραση με τον χρήστη (μέσω των χειριστηρίων που διαθέτει ο χρήστης ή αλλαγών των τιμών των μεταβλητών του προγράμματος)

Αντίστοιχα, μπορούμε να δούμε ότι οι παραπάνω ανάγκες αντιστοιχούν τέλεια στην αρχιτεκτονική MVC :

1. Model : Περιλαμβάνει όλη τη λογική των δεδομένων και της συσχέτισης με το API
2. View : Περιλαμβάνει όλη τη λογική της παρουσίασης των δεδομένων
3. Controller : Περιλαμβάνει όλη τη λογική της αλληλεπίδρασης μεταξύ του προγράμματος και του χρήστη

Ένα πρόβλημα που έχει δημιουργηθεί από την ξαφνική άνοδο αυτών των frameworks είναι το μεγάλο πλήθος αυτών και των διαφορών τους. Η επιλογή ανάμεσα σε αυτά δεν είναι ιδιαίτερα εύκολη, αφενός γιατί το καθένα έχει τα προτερήματά του και αφετέρου γιατί για το καθένα χρειάζεται κάποιο εύλογο χρονικό διάστημα, ώστε ο προγραμματιστής να είναι αρκετά οικείος με αυτό και τις ιδιαιτερότητές του. Επίσης, πέρα από το framework που θα επιλέξει ο χρήστης, στη συνέχεια μπορεί να ενσωματώσει και άλλες λειτουργίες ή βιβλιοθήκες που έχουν φτιαχτεί και θα τον βοηθήσουν στην υλοποίηση της εφαρμογής του. Γι' αυτή τη λειτουργία χρησιμοποιούμε εργαλεία τύπου package manager (όπως το npm που είδαμε προηγουμένως), με δημοφιλέστερο το bower, ενώ άλλα αντίστοιχα είναι τα Jam και Volo. Οπότε, στην επιλογή του εκάστοτε framework, πρέπει να συμπεριλάβουμε και την υποστήριξη που υπάρχει για αυτό από την κοινότητα και τις έτοιμες βιβλιοθήκες που θα μπορούμε να βρούμε.

Στον παρακάτω πίνακα είναι μερικά από τα JavaScript Frameworks που χρησιμοποιούνται περισσότερο αυτή τη στιγμή, μαζί με το νούμερο της τελευταίας έκδοσής τους αλλά και τον αριθμό stars στο αντίστοιχο αποθετήριο Github τους, νούμερο που φανερώνει την απήχηση που έχει το καθένα, ενώ στη συνέχεια θα ακολουθήσει η ανάλυση μερικών εξ αυτών.

Framework	Latest Release	Stars στο Github
AngularJS	1.4.8	45.675
React	0.14.3	34.051
Backbone.js	1.2.3	23.816
Ember.js	2.2.0	15.415
Polymer	1.2.3	13.790
Knockout	3.4.0	7.068
MarionetteJS	2.4.4	6.667
Cappuccino	0.9.8	2.108

Πίνακας 2. Front-End JavaScript Frameworks

2.2.1. AngularJS

Αν και η Angular υπάρχει αρκετά χρόνια (η δημιουργία της ξεκίνησε το 2009) η μεγάλη αποδοχή από τον κόσμο ξεκίνησε από τη στιγμή που η Google δέχτηκε να συνεχίσει την επέκτασή της από τον αρχικό δημιουργό της. Όπως φαίνεται και από τον παραπάνω πίνακα, είναι με διαφορά το πιο δημοφιλές framework και χρησιμοποιείται περισσότερο απ' όλα, ακόμα και εκτός του MEAN περιβάλλοντος. Λόγω αυτού, είναι πολύ πιο εύκολο να βρει κανείς σχετικό υλικό για αυτήν παρά για οποιοδήποτε άλλο framework.

Τα καλά στοιχεία της Angular είναι πολλά, ιδιαίτερο ενδιαφέρον όμως έχει το two-way-data-binding, κάτι που μέχρι πρόσφατα υπήρχε μόνο σε αυτήν (πρόσφατα το υιοθέτησε κατά κάποιον τρόπο και το React). Στην Angular τα Model και View μπορούν να είναι συγχρονισμένα, δηλαδή, μια μεταβλητή του μοντέλου μπορεί να αντιστοιχηθεί με ένα στοιχείο του View. Έτσι, είτε όταν ο χρήστης επεξεργαστεί κάτι στο View, είτε όταν από κάποια διαδικασία αλλάξει η εν λόγω τιμή στο μοντέλο, το DOM (Document Object Model , το αντικείμενο που περιλαμβάνει όλα τα στοιχεία που υπάρχουν στη σελίδα που εμφανίζεται στον χρήστη) θα αλλάξει ανάλογα για να εμφανιστεί η αλλαγή στον χρήστη. Αυτό είναι κάτι που μέχρι πρότινος χρειαζόταν πολλές έξτρα γραμμές κώδικα (με τη χρήση της jQuery για παράδειγμα) ενώ τώρα η Angular το κάνει αυτόματα. Στο χαρακτηριστικό παράδειγμα που ακολουθεί βλέπουμε τη δήλωση της μεταβλητής του μοντέλου (user.name) πάνω στο χειριστήριο input. Τη στιγμή που ο χρήστης θα ξεκινήσει να γράφει κάτι, αυτόματα θα του εμφανίζεται και στο κείμενο, ενώ αν αντίστοιχα η μεταβλητή ξαφνικά έπαιρνε κάποια τιμή χωρίς να την ελέγξει ο χρήστης (π.χ. μετά την ασύγχρονη επικοινωνία με τον server) πάλι το κείμενο θα ακολουθήσει την τελευταία τιμή της μεταβλητής.

```
<input ng-model="user.name" type="text" />
Hello {{user.name}}!
```

Άλλο σημαντικό κομμάτι της Angular είναι η δυνατότητά της να επεκτείνει τη γλώσσα HTML δίνοντας στον χρήστη τη δυνατότητα να φτιάξει δικά του HTML attributes. Αυτά τα ονομάζει Directives και προσθέτουν καινούργιες λειτουργίες και δυνατότητες στην εμφάνιση των δεδομένων της εφαρμογής και της λειτουργίας των Views. Χαρακτηριστικά Directives με τα οποία έρχεται απευθείας η Angular είναι τα ng-model και ng-repeat που χρησιμεύουν στο data-binding και στην εύκολη επανάληψη HTML στοιχείων αντίστοιχα.

```
<ul>
        <li ng-repeat="x in [1,2,3]">
          {{ x }}
        </li>
</ul>
```

Επίσης, ακολουθεί μια γερή δομή που περιλαμβάνει τις παρακάτω κατηγορίες :

1. Controllers
2. Directives
3. Factories
4. Filters
5. Services
6. Views

Το καθένα από τα παραπάνω έχει διακριτό ρόλο μέσα στην εφαρμογή και μπορούν να αναπτυχθούν ξεχωριστά έτσι ώστε να γίνουν πιο εύκολα αλλαγές όταν χρειαστούν.

Το μεγάλο αρνητικό της Angular, όσο και αν φαίνεται παράξενο, είναι το ταυτόχρονα ποιο σημαντικό της κομμάτι, το two-way-data-binding. Σαν τεχνική μπορεί να βοηθάει πάρα πολύ στη δημιουργία της εφαρμογής, αλλά αφενός είναι δύσκολο σαν λογική να ακολουθηθεί, αφετέρου αν δεν χρησιμοποιηθεί σωστά μπορεί να κάνει πολύ αργή την εφαρμογή. Κάτι αντίστοιχο ισχύει και για τα Directives της, που μπορούν να προσθέσουν μεγάλο βαθμό δυσκολίας στην περαιτέρω εξέλιξη της εφαρμογής.

Τα παραπάνω, μαζί με άλλα, δημιουργούν ένα ακόμα μειονέκτημα της Angular, και αυτό είναι η καμπύλη εκμάθησής της. Ο νέος στο συγκεκριμένο framework θα ενθουσιαστεί γρήγορα από την ευκολία που προσφέρει για να κάνεις γρήγορα, όμορφα πράγματα, όμως, όσο κανείς ασχολείται με σοβαρότερα project και την Angular ανακαλύπτει τις σοβαρές ιδιαιτερότητές της και τη δυσκολία με την οποία μπορεί κάποιος να τις μάθει όλες αυτές.

Τέλος, το μεγαλύτερο πρόβλημα της Angular δεν είναι άλλο από την ανάπτυξη της επόμενης έκδοσής της, Angular 2.0. Σύμφωνα με τους δημιουργούς της, η νέα έκδοση δεν θα έχει καμία σχέση με την προηγούμενη, καθώς δεν πρόκειται για εξέλιξή της αλλά για καινούργιο project. Αυτό σημαίνει πως ο χρόνος που θα διαθέσει κάποιος για να μάθει την Angular αυτή τη στιγμή, γρήγορα μπορεί να του φανεί άχρηστος, αφού θα πρέπει να μάθει την Angular 2.0, αν πιστέψουμε όσα έχουν αναφερθεί γι' αυτήν και τα προβλήματα της παλιάς που θα λύνει.

2.2.2. React

Το React είναι ένα από τα πιο καινούργια frameworks που όμως έχουν την πιο γρήγορα αναπτυσσόμενη κοινότητα. Δημιουργήθηκε μόλις το 2013 από την Facebook. Αυτή τη στιγμή το χρησιμοποιεί το ίδιο το Facebook για την εμφάνιση του User Interface του, το Instagram, το Flipboard και άλλα. Αυτό από μόνο του δίνει μια ιδέα γιατί το React είναι το κατάλληλο framework για τη δημιουργία μεγάλων και δυναμικών εφαρμογών.

Το React είναι πολύ διαφορετικό από τα υπόλοιπα Front-End Frameworks, γιατί χρησιμοποιεί μια τεχνολογία που ονομάζει Virtual DOM και λόγω αυτής είναι πολύ πιο γρήγορο από τα υπόλοιπα. Ουσιαστικά, το DOM, η σελίδα που εμφανίζεται στον χρήστη, δεν δημιουργείται μονάχα στο Front-End αλλά και στο Back-End, και το React αναλαμβάνει να βρει τις διαφορές και να τις εφαρμόσει καταλλήλως.

Ένα μεγάλο προτέρημα του React είναι η βασισμένη σε components προσέγγιση που ακολουθεί και επιτρέπει την επαναχρησιμοποίηση του ίδιου κώδικα ανάμεσα σε πολλά projects. Κάθε πράγμα που δημιουργεί ένας χρήστης, για παράδειγμα μια φόρμα login , μπορεί να γίνει ένα component που μετά θα ξαναχρησιμοποιηθεί σε άλλο project ή ακόμα και να μοιραστεί στο διαδίκτυο για να το χρησιμοποιήσουν άλλοι χρήστες. Ένα χαρακτηριστικό παράδειγμα αυτής της τεχνικής είναι το Material-UI της Google που δημιουργήθηκε χρησιμοποιώντας React Components. Παρακάτω φαίνεται ένα παράδειγμα δημιουργίας και χρήσης ενός Component, το οποίο εμφανίζει ένα από div.

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
ReactDOM.render(
  <CommentBox />,
  document.getElementById('content')
);
```

Το μειονέκτημα του React είναι η , τουλάχιστον αρχικά , πολυπλοκότητα των Components. Το γεγονός πως το UI δεν δημιουργείται από Templates αλλά από τα Components, τα οποία είναι JavaScript, είναι κάτι τελείως διαφορετικό και φαίνεται περίεργο. Επειδή όμως αυτό φέρνει πολλά προτερήματα μαζί του, αξίζει η ενασχόληση με αυτό.

2.2.3. Backbone

Το Backbone είναι από τα παλιότερα JavaScript Frameworks, κάποτε μάλιστα από τα δημοφιλέστερα, που τώρα όμως έχει πέσει πολύ στις προτιμήσεις των χρηστών. Δημιουργήθηκε το 2010 και χαρακτηρίζεται από το πολύ μικρό μέγεθος που προσθέτει στην εφαρμογή. Περιέχει ουσιαστικά τα απολύτως απαραίτητα για να επιβάλλει την MVC αρχιτεκτονική στην εφαρμογή. Η μινιμαλιστική προσέγγισή του έχει κερδίσει μερικούς έμπειρους χρήστες αλλά δημιουργεί αντίστοιχα προβλήματα στους πιο άπειρους, ενώ, για τα περισσότερα πράγματα πρέπει να βρει έξτρα βιβλιοθήκες ο χρήστης αν θέλει να χρησιμοποιήσει ένα σύγχρονο JavaScript Framework. Δεν είναι τυχαίο πως, το Backbone έχει χρησιμοποιηθεί για να φτιαχτούν με αυτό άλλα, πιο γεμάτα Frameworks.

2.2.4. Ember

Το Ember είναι άλλο ένα σχετικά καινούργιο Framework , δημιουργημένο από τον Yehuda Katz, ο οποίος ήταν στις δημιουργικές ομάδες των jQuery και Ruby on Rails. Ουσιαστικά, είναι το τρίτο Framework μαζί με τα Angular και React που προωθούν τη δημιουργία πλούσιων διαδραστικών εφαρμογών. Συγκεκριμένα, θα μπορούσε κανείς να πει ότι το Ember περιλαμβάνει τα καλύτερα κομμάτια των δύο άλλων, αφού ενσωματώνει το two-way-data-binding της Angular αλλά και τη δημιουργία του DOM από τον server, τεχνολογία που συναντάμε στο React.

Σαν Framework το Ember χαρακτηρίζεται βαθιά opinionated. Παρέχει ό,τι μπορεί να χρειαστεί ο προγραμματιστής για τη δημιουργία μιας εφαρμογής και περιμένει από αυτόν τα πάντα να γίνονται με αυτά τα ίδια εργαλεία. Όπως χαρακτηριστικά αναφέρουν οι ίδιοι στη σελίδα τους, όταν κάποιος χρησιμοποιεί το Ember, τα πάντα πρέπει να γίνονται “the Ember way”.

Αυτό μπορεί να χαρακτηριστεί και καλό και κακό. Το Ember μοιάζει να απευθύνεται σε μεγαλύτερα projects που η ταχύτητα και η σταθερότητα κατά τη διάρκεια του development είναι πάρα πολύ σημαντικά, όμως σε χρήστες που θέλουν να έχουν την ελευθερία να κάνουν κάποια πράγματα με τον τρόπο που

θέλουν αυτοί το Ember θα φανεί ιδιαίτερα δύστροπο. Παρακάτω φαίνεται η διαδικασία με την οποία το Ember ορίζει τους Routers που μεταφέρουν τα δεδομένα από το μοντέλο στο Template.

```
//routes.js
Router.map(function() {
  this.route('favorite-posts');
});

//favorite-posts.js
export default Ember.Route.extend({
  model() {
    return this.store.query('post', { favorite: true });
  }
});
//favorite-posts.hbs
<h1>Favorite Posts</h1>
{{#each model as |post|}}
  <p>{{post.body}}</p>
{{/each}}
```

Επίσης, όλο αυτό προσθέτει μεγάλο βαθμό δυσκολίας εκμάθησης του framework, ίσως μεγαλύτερο ακόμα και από την Angular, αφού τα πάντα θα πρέπει να μάθει να τα κάνει ο χρήστης με τις μεθόδους και τις διαδικασίες που ακολουθεί το συγκεκριμένο framework.

Τέλος, ένα χαρακτηριστικό του Ember, το οποίο πολλοί χρήστες το βλέπουν ως προτέρημα, είναι το γεγονός πως είναι το μόνο που δεν υποστηρίζεται από κάποια μεγάλη εταιρεία παρά μόνο από μια ομάδα προγραμματιστών που είναι αφοσιωμένοι στο ελεύθερο λογισμικό.

2.3 DataBase Design and Implementation

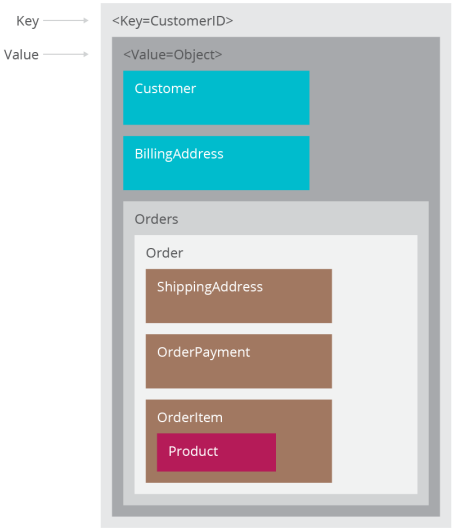
Η βάση δεδομένων είναι το σημείο που αποθηκεύονται όλα τα δεδομένα της εφαρμογής. Στο MEAN περιβάλλον, η τεχνολογία που προτιμάται είναι η NoSQL τύπου βάσεις δεδομένων. Αυτές διαφέρουν σημαντικά από τις παραδοσιακές Relational Databases (όπως η MySQL και η Postgres). Οι βάσεις αυτού του τύπου άρχισαν να δημιουργούνται το 2009 και σιγά σιγά έχουν αρχίσει να καθιερώνονται, ειδικά σε συγκεκριμένου τύπου εφαρμογές όπου θεωρούνται και ιδανικές. Οι NoSQL βάσεις μπορούν να χαρακτηριστούν από τα παρακάτω :

1. Δεν ακολουθούν το Relational Model
2. Είναι κατά κύριο λόγο βασισμένες σε ανοικτό λογισμικό
3. Δεν ακολουθούν προ-ορισμένο schema στα δεδομένα τους
4. Μπορούν να ανέβουν κλίμακα καθέτως αν χρειαστεί (horizontal scaling)

Οι NoSQL βάσεις προσφέρουν πολύ πιο απλή διαδικασία σχεδίασης της βάσης και πάρα πολύ πιο εύκολη ανάκτηση και αποθήκευση δεδομένων από την εφαρμογή, αφού άλλωστε αυτός ήταν ο λόγος που σχεδιάστηκαν. Οι βάσεις αυτές χωρίζονται σε κατηγορίες ανάλογα με τον τρόπο που αποθηκεύουν τα δεδομένα τους και είναι οι παρακάτω :

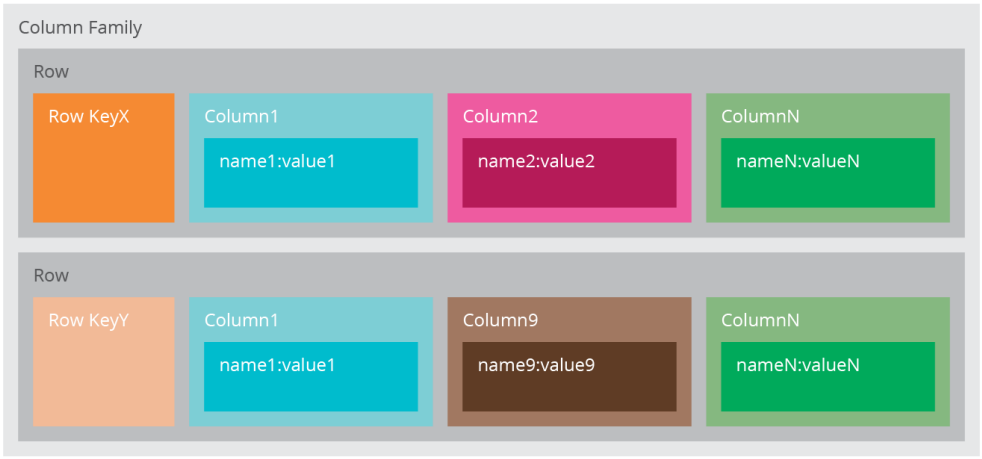
1. Key – Value Databases
2. Column Databases
3. Document Databases
4. Graph Databases

Οι βάσεις τύπου Key-Value είναι οι πιο απλές και όπως μαρτυράει το όνομά τους, αποτελούνται μονάχα από μία σχέση μεταξύ ενός κλειδιού και του προς αποθήκευση δεδομένου. Το δεδομένο που αποθηκεύεται είναι ουσιαστικά ένα blob (Binary Large Object) που η βάση δεδομένων δε γνωρίζει καθόλου τι περιέχει και αποτελεί ευθύνη της εφαρμογής να το διαχειριστεί αναλόγως. Αυτού του είδους οι βάσεις είναι πάρα πολύ γρήγορες και προτιμώνται στην αποθήκευση απλών δεδομένων ή δεδομένων που έχουν παραχθεί μετά από επεξεργασία.



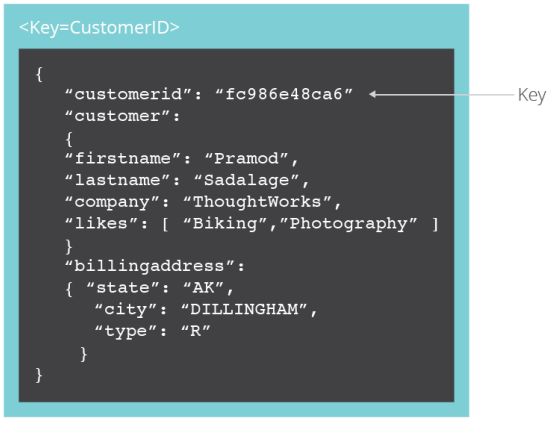
Εικόνα 1 Key-Value Database

Οι βάσεις τύπου Column θα μπορούσαν να χαρακτηριστούν ως το αντίστροφο των κανονικών πινάκων που γνωρίζουμε από τις Relational Databases. Εδώ, κάθε γραμμή ενός πίνακα περιλαμβάνει μια ομάδα από στήλες , οι οποίες με τη σειρά τους μπορούν να έχουν μέσα τους δεδομένα, αλλά οι γραμμές μεταξύ τους δεν έχουν τις ίδιες στήλες μέσα τους , άρα και διαφορετικού τύπου δεδομένα. Πολλές τέτοιες γραμμές μαζί δημιουργούν ένα Column Family, το οποίο θα μπορούσε νοητικά να αντιστοιχεί σε έναν κλασικό πίνακα και συνήθως περιέχει συναφή πληροφορία που προσπελαύνεται όλη μαζί.



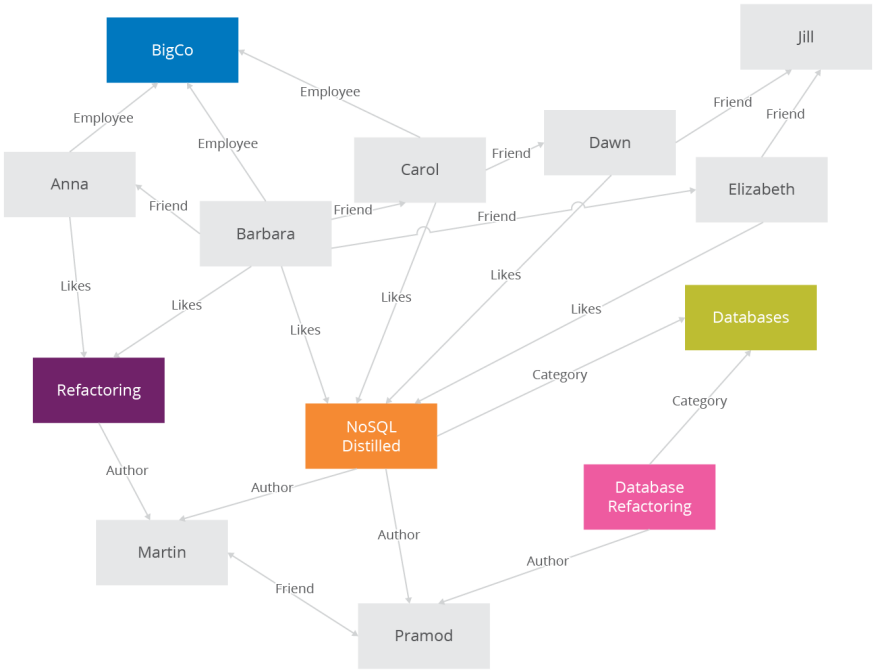
Εικόνα 2 Column Database

Οι βάσεις τύπου Document είναι οι πλέον διαδεδομένες όπως και οι πιο κατανοητές για τους προγραμματιστές, αφού η μορφή που ακολουθούν είναι ουσιαστικά αυτή του JSON. Δηλαδή, ακολουθούν την Key-Value λογική όπου όμως το Value μπορεί να είναι ταυτόχρονα Key ενός άλλου Value και ούτω καθεξής.



Εικόνα 3 Document Database

Οι βάσεις τύπου Graph είναι οι πιο περίπλοκες NoSQL βάσεις. Βασίζονται στη θεωρία γράφων και αποθηκεύουν δεδομένα που έχουν πολύ καλά ορισμένες σχέσεις μεταξύ τους. Αποτελούνται από κόμβους, οι οποίοι έχουν χαρακτηριστικά και σχέσεις, που ενώνουν τους κόμβους αυτούς μεταξύ τους.



Εικόνα 4 Graph Database

Τώρα που εξηγήθηκαν τα διαφορετικά είδη βάσεων NoSQL μπορούμε να δούμε μερικές από αυτές. Αν και σαν τεχνολογία είναι σχετικά νέα, έχουν ήδη δημιουργηθεί πάρα πολλές τέτοιες βάσεις, με σοβαρά προτερήματα αλλά και μειονεκτήματα, οπότε η επιλογή της κάθε μιας πρέπει να γίνεται με γνώμονα το project στο οποίο θα χρησιμοποιηθεί. Παρακάτω ακολουθεί ένας πίνακας με μερικές από τις επικρατέστερες τέτοιες βάσεις.

Database	Type	Latest Version	Stars στο Github
Redis	Key-Value	3.0.6	16.154
ElasticSearch	Document	2.0.2	14.219
RethinkDB	Document	2.2.2	11.640
MongoDB	Document	3.2.0	8.505
Cassandra	Column	3.1.1	2.507
CouchDB	Document	1.6.1	2.406
Riak	Key-Value	2.1.3	2.371
Neo4j	Graph	2.2.7	2.232
OrientDB	Graph	2.1.8	2.191
Aerospike	Key-Value	3.7.0.2	1.094

Πίνακας 3. NoSQL Databases

Στη συνέχεια θα ακολουθήσει η ανάλυση των σημαντικότερων εκ των παραπάνω, ενώ θα προσπαθήσουμε να βρούμε τα προτερήματα, τα μειονεκτήματα και τις καταλληλότερες χρήσεις της κάθε μίας.

2.3.1. Redis

Η Redis είναι η δημοφιλέστερη Key-Value Based βάση δεδομένων και είναι ιδιαίτερα γνωστή για την ταχύτητά της. Ουσιαστικά κρατάει όλα τα δεδομένα στη μνήμη της και κάνει περιοδική αποθήκευση σε δίσκο για την περίπτωση που χρειαστεί ανάκτηση μετά από επανεκκίνηση του server. Όλες οι διαδικασίες όμως (ανάγνωση, επεξεργασία, διαγραφή) γίνονται στα δεδομένα που υπάρχουν αποθηκευμένα στην προσωρινή μνήμη και άρα, μπορούν να υπάρχουν αναντιστοιχίες μεταξύ αυτών και των αποθηκευμενων στον δίσκο. Για να αποφευχθεί η απώλεια σημαντικών δεδομένων προτείνεται η χρήση μιας master / slave τοπολογίας. Με την παραπάνω τοπολογία, εκτός από τον πλεονασμό των δεδομένων μπορούμε να επιτύχουμε και καλύτερες ταχύτητες, αφού η Redis μπορεί να διαμοιράσει το φορτίο που δημιουργείται από την ανάγνωση των δεδομένων ανάμεσα στα slave συστήματα, διατηρώντας το master μονάχα για την εγγραφή. Όπως γίνεται κατανοητό, οι εφαρμογές για τις οποίες είναι κατάλληλη η συγκεκριμένη βάση είναι αυτές που περιλαμβάνουν δεδομένα πραγματικού χρόνου, όπως οι τιμές των μετοχών, πίνακες κατατάξεων ή εφαρμογές επικοινωνίας.

2.3.2. ElasticSearch

Η ElasticSearch είναι μια πολύ δημοφιλής Document Based βάση δεδομένων, η οποία είναι γνωστή για τις πολύ ανεπτυγμένες μεθόδους αναζήτησης που διαθέτει. Παρέχει ένα REST API μέσω του οποίου γίνεται η διεπαφή με τη βάση και όλες οι λειτουργίες του CRUD μπορούν να γίνουν μέσα από αυτό, όπως και η αναζήτηση. Τα δεδομένα αποθηκεύονται σε μορφή JSON η οποία μπορεί να διαφέρει από αντικείμενο σε αντικείμενο και η αναζήτηση μπορεί να γίνει με αντίστοιχο τρόπο, ορίζοντας τα αντίστοιχα πεδία JSON που επιθυμούμε. Προτιμάται σε projects των οποίων τα δεδομένα χρειάζονται γρήγορη αναζήτηση ή ταξινόμηση, όπως για παράδειγμα σε γεωγραφικά δεδομένα ή σε λίστες με επιλογές χρηστών.

2.3.3. RethinkDB

Η RethinkDB είναι άλλη μία Document Based βάση δεδομένων που χρησιμοποιεί επίσης μορφή JSON για τα δεδομένα της. Χαρακτηριστικό της είναι η γλώσσα με την οποία αλληλοεπιδρά κανείς με αυτήν. Η γλώσσα αυτή λέγεται ReQL και μοιάζει πολύ με την απλή JavaScript, πράγμα που την καθιστά ιδιαίτερα εύκολή. Αυτός είναι ο κύριος λόγος που πολλοί developers την επιλέγουν, κυρίως για μικρά projects, αφού μπορούν να την ενσωματώσουν στα προγράμματά τους πολύ γρήγορα.

2.3.4. MongoDB

Η MongoDB είναι η NoSQL βάση που έχει εισχωρήσει περισσότερο στην αγορά. Γι' αυτό, είναι αυτή που μπήκε και στο ακρωνύμιο MEAN. Είναι και αυτή τύπου Document Based και όπως μπορούμε να δούμε στην παρακάτω εικόνα, αυτή τη στιγμή είναι όχι μόνο η πιο διαδεδομένη NoSQL βάση άλλα και μία από τις πιο δημοφιλείς βάσεις γενικότερα.

Rank Dec 2015	Rank Nov 2015	Rank Dec 2014	DBMS	Database Model	Score Dec 2015	Score Nov 2015	Score Dec 2014
1.	1.	1.	Oracle	Relational DBMS	1497.55	+16.61	+37.76
2.	2.	2.	MySQL	Relational DBMS	1298.54	+11.70	+29.96
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1123.16	+0.83	-76.89
4.	4.	↑ 5.	MongoDB	Document store	301.39	-3.22	+54.87
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	280.09	-5.60	+26.09
6.	6.	6.	DB2	Relational DBMS	196.13	-6.40	-14.13
7.	7.	7.	Microsoft Access	Relational DBMS	140.21	-0.75	+0.31
8.	8.	↑ 9.	Cassandra	Wide column store	130.84	-2.08	+36.78
9.	9.	↓ 8.	SQLite	Relational DBMS	100.85	-2.60	+6.15
10.	10.	10.	Redis	Key-value store	100.54	-1.87	+12.66
11.	11.	11.	SAP Adaptive Server	Relational DBMS	81.47	-2.24	-4.52
12.	12.	12.	Solr	Search engine	79.15	-0.63	+0.73
13.	↑ 14.	↑ 16.	Elasticsearch	Search engine	76.57	+1.79	+30.67
14.	↓ 13.	↓ 13.	Teradata	Relational DBMS	75.72	-1.37	+8.32
15.	↑ 16.	↑ 17.	Hive	Relational DBMS	55.27	+0.36	+18.90
16.	↓ 15.	↓ 15.	HBase	Wide column store	54.25	-2.21	+3.17
17.	17.	↓ 14.	FileMaker	Relational DBMS	50.12	-1.61	-2.10
18.	18.	↑ 20.	Splunk	Search engine	43.86	-0.76	+12.39
19.	19.	↑ 21.	SAP HANA	Relational DBMS	38.86	-0.76	+11.05
20.	20.	↓ 18.	Informix	Relational DBMS	36.40	-2.05	+1.28
21.	21.	↑ 23.	Neo4j	Graph DBMS	33.18	-0.86	+8.02
22.	22.	↓ 19.	Memcached	Key-value store	30.94	-1.46	-2.75

Εικόνα 5. Database Rankings σύμφωνα με το db-engines.com

Η MongoDB συγκεντρώνει τα καλύτερα στοιχεία του κόσμου των Relational Databases με την ευχρηστία των NoSQL βάσεων. Χρησιμοποιεί μία γλώσσα που δίνει μεγάλη ευκολία στη δημιουργία οποιοδήποτε ερωτήματος. Δημιουργεί δείκτες για γρήγορη αναζήτηση, ενώ επιτρέπει την επιλογή ανάμεσα σε πολύ γρήγορη λειτουργία ή μεγάλη συνοχή (consistency) δεδομένων. Ταυτόχρονα, τα δεδομένα τα αποθηκεύει σε μορφή BSON (Binary JSON) χωρίς επιβολή κάποιας μορφής schema. Τέλος, παρουσιάζει ιδιαίτερη ευκολία στην αναβάθμισή της για μεγαλύτερο φόρτο. Ουσιαστικά είναι μια βάση δεδομένων που μπορεί επάξια να αντικαταστήσει τις Relational Databases σε όποιο project τα δεδομένα δεν μπορούν να έχουν την αυστηρή δομή των προδιαγεγραμμένων πινάκων.

2.3.5. Cassandra

Η Cassandra είναι μία βάση που δημιουργήθηκε αρχικά από το Facebook. Είναι τύπου Column Based και το σημαντικό της χαρακτηριστικό είναι η ιδιαίτερη αρχιτεκτονική της που της επιτρέπει να είναι πάντα διαθέσιμη. Η Cassandra δεν χρησιμοποιεί κάποιο σύστημα τύπου master-slave όπως βλέπουμε στις υπόλοιπες βάσεις αλλά ουσιαστικά, δημιουργεί ένα δαχτυλίδι από κόμβους, οι οποίοι είναι όλοι ίσοι, δεν υπάρχει δηλαδή master node. Έτσι, πολύ εύκολα μπορεί το σύστημα να μεγαλώσει ανάλογα τις ανάγκες και να υποστηρίξει χιλιάδες χρήστες ταυτόχρονα. Χρησιμοποιείται κυρίως σε εφαρμογές που πρέπει να

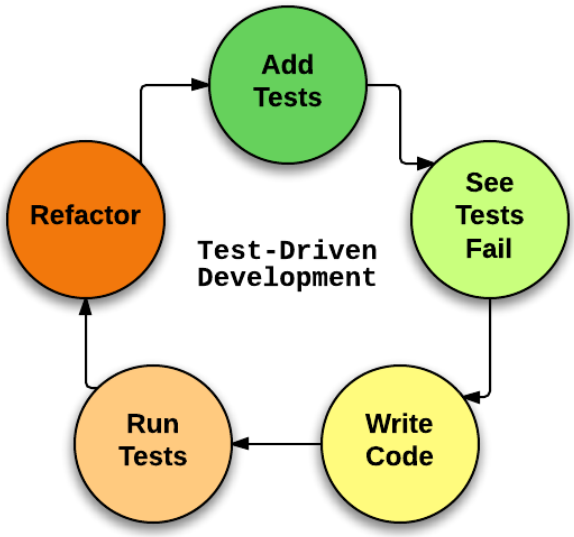
αποθηκευτεί πολύ μεγάλος όγκος δεδομένων, όπως σε μεγάλα συστήματα αισθητήρων που αποθηκεύουν δεδομένα σε αληθινό χρόνο.

2.3.6. Neo4j

Η Neo4j είναι η πιο διάσημη βάση τύπου Graph. Τα δεδομένα αποθηκεύονται σε μορφή γραφήματος, δηλαδή περιλαμβάνουν κόμβους, σχέσεις και τις ιδιότητες αυτών. Αυτή η μορφή είναι ιδιαίτερα χρήσιμη στην περίπτωση που τα δεδομένα μας έχουν πολλές συνδέσεις μεταξύ τους. Χαρακτηριστικό τέτοιο παράδειγμα είναι τα κοινωνικά δίκτυα, με τους χρήστες τους και τις σχέσεις μεταξύ αυτών. Τέτοιου είδους δεδομένα θα μπορούσαν να αποθηκευτούν σε Relational Databases αλλά θα ήταν πολύ πιο δύσκολη και αργή η προσπέλασή τους, αφού θα χρειάζονταν διάφορα JOINS για να ανακτήσουμε ό,τι χρειαζόμαστε, ενώ με τις Graph Based βάσεις, η απαιτούμενη πληροφορία είναι πολύ πιο μαζεμένη και εύκολα ανακτήσιμη. Τέλος, προσφέρει μία πολύ απλή γλώσσα που κάνει πολύ εύκολη τη χρήση της. Εκτός από το παράδειγμα των social media, βάσεις τέτοιου τύπου μπορούν να χρησιμεύσουν, μεταξύ άλλων, σε αποθήκευση και αναζήτηση μέσα σε χάρτες και τοπολογίες δικτύου.

2.4 Unit Testing

Όσο πιο δύσκολα και περίπλοκα συστήματα προσπαθούμε να φτιάξουμε, τόσο πιο σημαντικό είναι να ακολουθούμε κάποια διαδικασία Unit Testing. Δηλαδή, τον κώδικα που γράφουμε να τον ελέγχουμε με διάφορα test όχι μόνο για τη σωστή σύνταξή του αλλά και για το αν επιφέρει το κατάλληλο αποτέλεσμα με διάφορες εισόδους ή αν δουλεύει σωστά σε συνδυασμό με άλλα κομμάτια κώδικα. Το κομμάτι αυτό είναι πολύ σημαντικό στη διαδικασία του Software Development και δεν θα έπρεπε να παραμελείται. Έχουν δημιουργηθεί και πολλές μεθοδολογίες (TDD – Test Driven Development , BDD – Behavior Driven Development , FDD – Feature Driven Development) οι οποίες το ενσωματώνουν με διαφορετικούς τρόπους το καθένα, ανάλογα το σημείο στο οποίο θεωρεί η κάθε μία ότι είναι το σημαντικότερο.



Εικόνα 6. TDD Workflow

Άσχετα με τις παραπάνω μεθοδολογίες, το Unit Testing μπορεί ο καθένας να το ενσωματώσει στη ροή εργασίας του και ειδικά στο MEAN περιβάλλον, όπου για την JavaScript έχουν δημιουργηθεί πολλά και χρήσιμα εργαλεία σχετικά με αυτό. Παρακάτω ακολουθεί μια λίστα με μερικά μονάχα από αυτά.

1. Karma
2. Protractor
3. Mocha
4. Jasmine
5. QUnit
6. CasperJS
7. PhantomJS

Ο σκοπός όλων των παραπάνω είναι ο προγραμματιστής να μπορεί να γράψει μια, συνήθως, πολύ μικρή διαδικασία που όταν τρέξει θα ελέγξει το εκάστοτε σημείο του κώδικα για το αποτέλεσμά του. Κατά τη ροή του Development πρέπει να γραφτούν πολλά τέτοια tests και να εκτελούνται πολύ συχνά, ώστε αν δημιουργηθούν σφάλματα να αναγνωριστούν και να διορθωθούν άμεσα. Παρακάτω φαίνεται ένα απλό test γραμμένο με το Framework Mocha, το οποίο ελέγχει τη σωστή δημιουργία ενός νέου χρήστη στην εφαρμογή, αν έχει δηλαδή username.

```
describe('Creating a new User',function(){
var user;

before(function(done){
  User.create({ username: 'test' , function(err,u){
    user = u;
    done();
  });
});

it('should have a username',function(){
  user.should.have.property('username','test');
});
});
```

Οι διαφορές μεταξύ τους είναι μικρές αλλά σημαντικές, για αυτό ο κάθε χρήστης επιλέγει σύμφωνα με τις προσωπικές του προτεραιότητες αυτό που θέλει να χρησιμοποιήσει. Η μεγάλη διαφορά τους που εμφανίζεται κατευθείαν είναι το συντακτικό τους, αφού το καθένα χρησιμοποιεί διαφορετικούς τρόπους για να ορίσει τα test του. Παρακάτω φαίνεται άλλο ένα παράδειγμα όπου με το Framework Jasmine ελέγχουμε αν ένας πίνακας έχει όσα στοιχεία χρειαζόμαστε, δηλαδή τέσσερα.

```
describe("Our data array", function() {
  it("has four items", function() {
    expect(ourDataArray.Questions.length).toBe(4);
  });
});
```

Μετά, το καθένα έρχεται είτε με πολλά, είτε με λιγότερα χαρακτηριστικά που μπορούν να βοηθήσουν πολύ τον χρήστη ανά περίσταση, όπως υποστήριξη ασύγχρονων κλήσεων ή ελευθερία επιλογής διαφορετικών βιβλιοθηκών για ελέγχους. Φυσικά, τα Frameworks αυτά υποστηρίζουν αντίστοιχα και τα Front-End frameworks που αναλύσαμε σε προηγούμενο κεφάλαιο. Στο παρακάτω παράδειγμα βλέπουμε ένα test γραμμένο με το εργαλείο Karma όπου ελέγχει ένα απλό service της AngularJS αν μπορεί να δημιουργηθεί και να θέσει τιμή σε μία μεταβλητή του, στο παράδειγμα το όνομα του χρήστη.

```
describe('Person', function () {

  var Person;
  beforeEach(module('myApp'));
  beforeEach(inject(function (_Person_) {
    Person = _Person_;
  }));

  describe('Constructor', function () {

    it('assigns a name', function () {
      expect(new Person('Ben')).to.have.property('name', 'Ben');
    });

  });

});
```

2.5 Software Optimization

Μετά από την ολοκλήρωση του προγράμματος, η διαδικασία του Development δε σταματάει, αφού υπάρχει το κομμάτι του Optimization, κατά το οποίο προσπαθούμε να κάνουμε το πρόγραμμα να τρέχει όσο πιο γρήγορα γίνεται για να μην προκαλεί δυσφορία στον χρήστη. Γι' αυτό, υπάρχουν διάφορες διαδικασίες και μέθοδοι, τα πράγματα που πια όμως θεωρούνται δεδομένα σε ένα σωστό workflow είναι τα παρακάτω.

1. Concatenation των αρχείων JavaScript
2. Minify των αρχείων JavaScript
3. Minify των αρχείων Stylesheet
4. Χρησιμοποίηση Cache μνήμης για τα Templates
5. Συμπίεση δεδομένων κατά την μεταφορά από το Back-End στο Front-End
6. Χρησιμοποίηση Cache μνήμης για το στατικό περιεχόμενο του Back-End

Συγκεκριμένα, όταν ο χρήστης προσπαθήσει να εισέλθει στην εφαρμογή μας, θα γίνουν διάφορα HTTP Requests για να λάβει στον υπολογιστή του όλα τα αρχεία που χρειάζεται το Front-End κομμάτι της για να δουλέψει. Ένα πρώτο βήμα προς τη βελτιστοποίηση του προγράμματός μας είναι το concatenation αυτών των αρχείων, δηλαδή η δημιουργία ενός μονάχα αρχείου που θα περιέχει όλα τα υπόλοιπα. Αυτό θα δημιουργήσει ουσιαστικά ένα πάρα πολύ μεγάλο αρχείο JavaScript, το οποίο δεν θα έχει διαφορά στο

μέγεθος με όλα τα προηγούμενα μαζί, θα προσθέσει όμως ιδιαίτερη ταχύτητα αφού θα χρειαστεί μονάχα ένα HTTP Request για να το κατεβάσει από τον server μας.

Επίσης, τα αρχεία JavaScript αλλά και τα αρχεία Stylesheet, είτε είναι CSS είτε LESS είτε SASS, είναι γραμμένα με τρόπο που να μπορεί ένας άνθρωπος να τα διαβάσει. Έχουν κενά, δόμηση κώδικα και οι εντολές χωρίζονται ανά γραμμή. Αυτό θεωρείται αναγκαίο κατά τη διάρκεια του Development, αλλά δεν χρειάζεται στο αρχείο που θα διαβάσει το browser, αφού ο υπολογιστής θα διαβάσει το ίδιο αρχείο είτε έχει κενά είτε όχι. Γι' αυτό, τα αρχεία αυτά μπορούμε να τα περάσουμε από τη διαδικασία του minify, δηλαδή να αφαιρεθούν όλα τα κενά και ουσιαστικά να γίνει ολόκληρο το αρχείο μια τεράστια γραμμή μόνο. Αυτή η μέθοδο μειώνει αισθητά το μέγεθος του τελικού αρχείου και αντίστοιχα τον χρόνο που χρειάζεται για να φορτώσει η εφαρμογή. Το πρόβλημα με αυτή τη διαδικασία είναι ότι μπορεί να δημιουργήσει προβλήματα στην εκτέλεση του κώδικα, οπότε πρέπει να ελέγχεται πάντα η χρήση της.

Κάτι αντίστοιχο μπορούμε να κάνουμε και για τα Templates της εφαρμογής, τα html αρχεία της δηλαδή. Ειδικά στην περίπτωση της AngularJS, μπορούμε να τα μετατρέψουμε και αυτά σε JavaScript και να φορτώσουμε στη σελίδα μας μαζί όλον τον υπόλοιπο κώδικα και, η Angular θα ξέρει μετά να φορτώσει το κατάλληλο Template αναλόγως το View το οποίο πρέπει να εμφανιστεί στον χρήστη. Με αυτό κερδίζουμε αισθητά και σε μέγεθος αρχείου και σε χρόνο εκτέλεσης.

Τέλος, εφαρμογές optimization μπορούν να γίνουν και στο Back-End. Ανάλογα το Framework που έχουμε επιλέξει, μπορούμε να ενεργοποιήσουμε κάποιου είδους συμπίεση, συνήθως gzip, ώστε όλα τα δεδομένα που μεταφέρονται προς το Front-End να έρχονται με το μικρότερο δυνατό μέγεθος. Μαζί, μπορούμε να μεταφέρουμε στην Cache μνήμη του server μας το στατικό περιεχόμενο που αυτός χειρίζεται, δηλαδή τα αρχεία που θα μεταφέρει στο Front-End, άρα τα JavaScript, Stylesheet αρχεία και τις όποιες φωτογραφίες.

Φυσικά υπάρχουν και άλλα βήματα που μπορεί να ακολουθήσει κανείς για την περαιτέρω βελτιστοποίηση της εφαρμογής, ανάλογα πάντα το μέγεθος αυτής και των αναγκών της. Για παράδειγμα θα μπορούσε να χρησιμοποιήσει κάποιο εργαλείο για την βελτιστοποίηση των εικόνων που χρησιμοποιεί στην εφαρμογή, καθώς οι φωτογραφίες συνήθως έχουν μεγάλο μέγεθος το οποίο μπορεί να μειωθεί αισθητά χωρίς να χαθεί ποιότητα.

Στο MEAN περιβάλλον και στην JavaScript γενικότερα υπάρχουν πάρα πολλά εργαλεία που μπορεί ο καθένας να χρησιμοποιήσει για τις παραπάνω λειτουργίες. Αυτά τα εγκαθιστούμε μέσα από το εργαλείο npm που είδαμε σε προηγούμενο κεφάλαιο και τα εκτελούμε είτε μόνα τους, συνήθως από κονσόλα εντολών, είτε σε συνδυασμό με κάποιο εργαλείο τύπου Task Runner, τα οποία θα αναλύσουμε στο επόμενο κεφάλαιο.

2.6 Automation – Task Runner

Όπως έγινε κατανοητό από τα προηγούμενα κεφάλαια, η ανάπτυξη μιας εφαρμογής στο MEAN περιβάλλον περιλαμβάνει πολλά κομμάτια και διαφορετικά εργαλεία. Γι' αυτό, έχουν δημιουργηθεί διάφορα εργαλεία τύπου Task Runner ή αλλιώς Build Tools όπως αλλιώς λέγονται, με σκοπό να αυτοματοποιούν μερικές ενέργειες που επαναλαμβάνονται. Αυτές τις εργασίες τις συναντήσαμε ήδη στα παραπάνω κεφάλαια και περιλαμβάνουν, μεταξύ άλλων τα παρακάτω.

1. Concatenation / Minification αρχείων
2. CSS preprocessing
3. Optimization εικόνων
4. Εκτέλεση των Unit Tests

5. Linting , δηλαδή έλεγχος / ανάλυση του κώδικα για πιθανά σφάλματα

Για την εγκατάσταση αυτών των εργαλείων χρησιμοποιούμε ξανά το εργαλείο npm που έχουμε ξαναδεί και παρακάτω ακολουθεί ένας πίνακας με μερικά από αυτά μαζί με το νούμερο τις τελευταίας τους έκδοσης, τον αριθμό εγκαταστάσεων μέσω του εργαλείου npm αλλά και τα stars στα αντίστοιχα αποθετήρια GitHub τους.

Task Runner	Latest Version	Downloads τον τελευταίο μήνα	Stars στο GitHub
Gulp	3.9.0	1.393.428	18.501
Grunt	0.4.5	1.269.888	10.363
Broccoli	0.16.9	134.507	2.604
Brunch	2.1.1	26.206	4.879
Jake	8.0.12	11.205	1.196
Mimosa	2.3.32	3.000	507

Πίνακας 4. JavaScript Task Runners

Όπως γίνεται εύκολα αντιληπτό, το Gulp και το Grunt είναι τα δύο εργαλεία που χρησιμοποιούνται περισσότερο και μάλιστα με μεγάλη διαφορά από τα υπόλοιπα. Επίσης, είναι τα δύο εργαλεία για τα οποία έχουν γραφτεί και τα περισσότερα plugins, άρα και αυτά που παρέχουν τις περισσότερες δυνατότητες. Μεταξύ τους διαφέρουν σημαντικά στον τρόπο με τον οποίο δημιουργούν τα Tasks τα οποία θα τρέχουν και είναι θέμα προτίμησης του χρήστη ποιο θα επιλέξει.

Και τα δύο εργαλεία χρησιμοποιούν ένα αρχείο (gulpfile.js και gruntfile.js αντίστοιχα) στο οποίο ο χρήστης γράφει τις ενέργειες που θα πρέπει να γίνονται από το αντίστοιχο πρόγραμμα. Το Grunt ακολουθεί μια τεχνική που θυμίζει configuration file και μοιάζει πιο δύσκολο να κατανοηθεί αρχικά από τον απλό χρήστη. Παρακάτω ακολουθεί ένα παράδειγμα ενός τέτοιου αρχείου, ο σκοπός του οποίου είναι να τρέξει το εργαλείο uglify.js (ένα από τα εργαλεία που χρησιμοποιούνται για να γίνει minify στα αρχεία) και να του περάσει δυναμικά κάποια παράμετρο στις ρυθμίσεις του.

```
// Project configuration.
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  uglify: {
    options: {
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-
dd") %> */\n'
    },
    build: {
      src: 'src/<%= pkg.name %>.js',
      dest: 'build/<%= pkg.name %>.min.js'
    }
  }
});
```

Αντίθετα, στο Gulp το αρχείο gulpfile περιέχει κανονικό κώδικα JavaScript. Χρησιμοποιεί μια μεθοδολογία ροών όπου η μία λειτουργία περνάει το αποτέλεσμά της στην επόμενη. Για παράδειγμα, μπορούν να εκτελεστούν σειριακά οι λειτουργίες του concatenation και του minify και η δεύτερη θα πάρει αυτόματα το αρχείο που δημιουργήθηκε από την πρώτη. Αυτή η τεχνική δημιουργεί στο τέλος ένα αρχείο πολύ πιο ευανάγνωστο και εύκολα παραμετροποιήσιμο. Παρακάτω ακολουθεί ένα παράδειγμα από ένα αρχείο που εκτελεί μια ροή από λειτουργίες για να ενώσει και να μικρύνει τα αρχεία css.

```
var gulp = require('gulp');
var sass = require('gulp-sass');
var csso = require('gulp-csso');
var concat = require('gulp-concat');
var plumber = require('gulp-plumber');

gulp.task('compress', function() {
  gulp.src([
        'public/stylesheets/main.css',
        'public/stylesheets/secondary.css',
        'public/stylesheets/other.css'
    ])
    .pipe(plumber())
    .pipe(sass())
    .pipe(concat('conc_main.css'))
    .pipe(csso())
    .pipe(gulp.dest('public/stylesheets'));
});

gulp.task('default', ['compress' ]);
```

Φυσικά η επιλογή ανάμεσα στα δύο παραμένει στον προγραμματιστή, αφού και τα δύο έχουν εξίσου μεγάλες κοινότητες, οπότε είναι εύκολο να βρει κανείς το κατάλληλο plugin για τη λειτουργία που χρειάζεται.

2.7 Security

Η ασφάλεια είναι το κομμάτι του Development που όλοι συμφωνούν ότι είναι πολύ σημαντικό αλλά λίγοι ασχολούνται όπως θα έπρεπε με αυτό. Τα τελευταία χρόνια παρόλα αυτά, ο τομέας της ασφάλειας έχει αρχίσει να παίρνει την προσοχή που του χρειάζεται. Στην NodeJS η ασφάλεια δεν έχει ανελιχθεί ακόμα ως πρωτεύον ζήτημα, λόγω του νέου της τεχνολογίας, αλλά όσο αυτή κερδίσει χώρο στην αγορά, τόσο θα εξελίσσεται ο εν λόγω τομέας τομέα. Παρακάτω θα δούμε μερικές από τις τακτικές που πρέπει να ακολουθούν οι χρήστες τόσο στο Back-End κομμάτι όσο και στο Front-End.

Αρχικά, το μεγάλο προτέρημα της Node είναι το εργαλείο npm που όπως έχουμε δει προσφέρει πρόσβαση σε χιλιάδες άλλες βιβλιοθήκες και εργαλεία που βοηθούν απίστευτα τη διαδικασία του Development. Ο χρυσός κανόνας χρήσης του npm είναι να γνωρίζεις τι κατεβάζεις καθώς, αυτές οι βιβλιοθήκες μπορούν να έχουν δημιουργηθεί από τον οποιονδήποτε. Αυτό σημαίνει πως μία τέτοια βιβλιοθήκη μπορεί ταυτόχρονα να κάνει πολλά άλλα πράγματα στο Back-End περιβάλλον, ή κάτι σίγουρα

πιο πιθανό, να έχει κάποια ευπάθεια η οποία να επιτρέπει σε χρήστες να δημιουργήσουν πρόβλημα στην εφαρμογή μας. Γι' αυτό, θα πρέπει να είμαστε πολύ σίγουροι για τον κώδικα τρίτων ατόμων που ενσωματώνουμε στις εφαρμογές μας. Ως βοηθητικό αυτού του προβλήματος δημιουργήθηκε το Node Security Project, μέσα από το οποίο ελέγχονται όσα περισσότερα από τα εργαλεία που υπάρχουν στο npm γίνεται, ώστε να βρίσκονται οι ευπάθειές τους και να λύνονται. Μέσω αυτού έχουν βρεθεί και λυθεί τέτοια προβλήματα σε μεγάλα εργαλεία που είδαμε και παραπάνω όπως το Hapi και το Express.

Σε συνέχεια του παραπάνω προβλήματος, δεν πρέπει να αμελούμε την αναβάθμιση αυτών των βιβλιοθηκών. Πολλές φορές όταν η εφαρμογή μας δουλεύει σωστά με μία συγκεκριμένη έκδοση μιας βιβλιοθήκης δεν μπαίνουμε στην διαδικασία αναβάθμισής της, γιατί μπορεί να χρειαστούν αλλαγές για να δουλέψει πάλι σωστά. Αυτό είναι λάθος γιατί οι αναβαθμίσεις των βιβλιοθηκών εκτός από νέα χαρακτηριστικά συχνά φέρνουν και διορθώσεις σε ευπάθειες των μέχρι τότε εκδόσεων.

Μετά, δεν πρέπει να ξεχνάμε πως η Node παραμένει να είναι JavaScript, οπότε και όλες τις ευπάθειες που αυτή έχει στο Front-End, η Node τις περνάει στο Back-End, όπου και μπορούν να δημιουργήσουν πολύ μεγαλύτερα προβλήματα. Για παράδειγμα, η χρήση της eval μπορεί να προκαλέσει σοβαρά προβλήματα και θα πρέπει να αποφεύγεται, όπως γίνεται δηλαδή και στην JavaScript μέχρι τώρα.

Αντίστοιχα, όπως η JavaScript στον browser όταν βρει κάποιο error σταματάει να δουλεύει, έτσι και στη Node, όταν βρεθεί κάποιο error τότε ο server θα πέσει. Έτσι, αν κάποιος βρει έναν τρόπο για να κάνει την JavaScript στο Back-End να βρει error, για παράδειγμα στέλνοντας λάθος δεδομένα, θα μπορεί να ρίχνει συνέχεια τον server και ουσιαστικά να κάνει επιτυχημένες DOS (Denial of Service) επιθέσεις. Γι' αυτό, πρέπει είτε να προσπαθούμε να προλάβουμε τα σφάλματα αυτά, πράγμα δύσκολο, είτε να ελέγχουμε συνέχεια την κατάσταση του server ώστε με μια αυτοματοποιημένη διαδικασία να ξαναλειτουργεί. Ένα εργαλείο που κάνει ακριβώς αυτή τη δουλειά είναι το forever.

Τέλος, υπάρχουν κάποια ακόμα πράγματα που μπορούμε να κάνουμε για να αποφύγουμε ένα συγκεκριμένο κομμάτι επιθέσεων. Οι επιθέσεις τύπου CSRF (Cross-Site Request Forgery) προσπαθούν να εκμεταλλευτούν την υπάρχουσα σύνδεση του χρήστη με κάποια άλλη εφαρμογή και την ύπαρξη του ενεργού cookie αυτού στον browser. Μέσω αυτού, μπορεί να γίνει κλίση της άλλης εφαρμογής χωρίς ο χρήστης να το καταλάβει. Ο τρόπος για να αποφευχθεί αυτό είναι απλός. Η εφαρμογή στο Back-End δημιουργεί ένα τυχαίο κλειδί και το στέλνει στο Front-End. Από εκεί και πέρα, κάθε επικοινωνία μεταξύ Back-End και Front-End θα πρέπει να περιλαμβάνει το συγκεκριμένο κλειδί. Αν κάποια άλλη εφαρμογή προσπαθήσει να καλέσει το Back-End της δικιάς μας, δεν θα γνωρίζει το συγκεκριμένο κλειδί άρα δεν θα μπορέσει να αποκτήσει πρόσβαση.

Το CSP (Content Security Policy) είναι ουσιαστικά μια λίστα από επιτρεπτές πηγές για κώδικα JavaScript, CSS, φωτογραφίες και άλλα για την εφαρμογή μας. Δηλαδή, δημιουργώντας αυτή την λίστα, ένα αρχείο JavaScript που προέρχεται από εξωτερική πηγή, αν αυτή η πηγή δεν είναι δηλωμένη τότε δεν θα τρέξει ο κώδικάς του. Αντίστοιχα, αν μια φωτογραφία πρέπει να φορτωθεί από άλλον server, ο οποίος δεν έχει δηλωθεί σε αυτή τη λίστα, δεν θα εμφανιστεί μπροστά στην εφαρμογή. Με αυτό το εργαλείο αποκτά μεγάλη ασφάλεια η εφαρμογή, αφού μπορούμε να ελέγξουμε ακριβώς από ποιες πηγές θα δεχόμαστε να τρέχει κώδικας στην εφαρμογή μας.

Δύο άλλες σημαντικές ρυθμίσεις είναι το HTTPS , για ασφαλή σύνδεση και κρυπτογράφηση των δεδομένων που μεταφέρονται, και την επιλογή X-Frame, με την οποία μπορούμε να αποτρέψουμε άλλες εφαρμογές να εμφανίζουν – ενσωματώνουν τη δικιά μας εφαρμογή μέσα σε iframes.

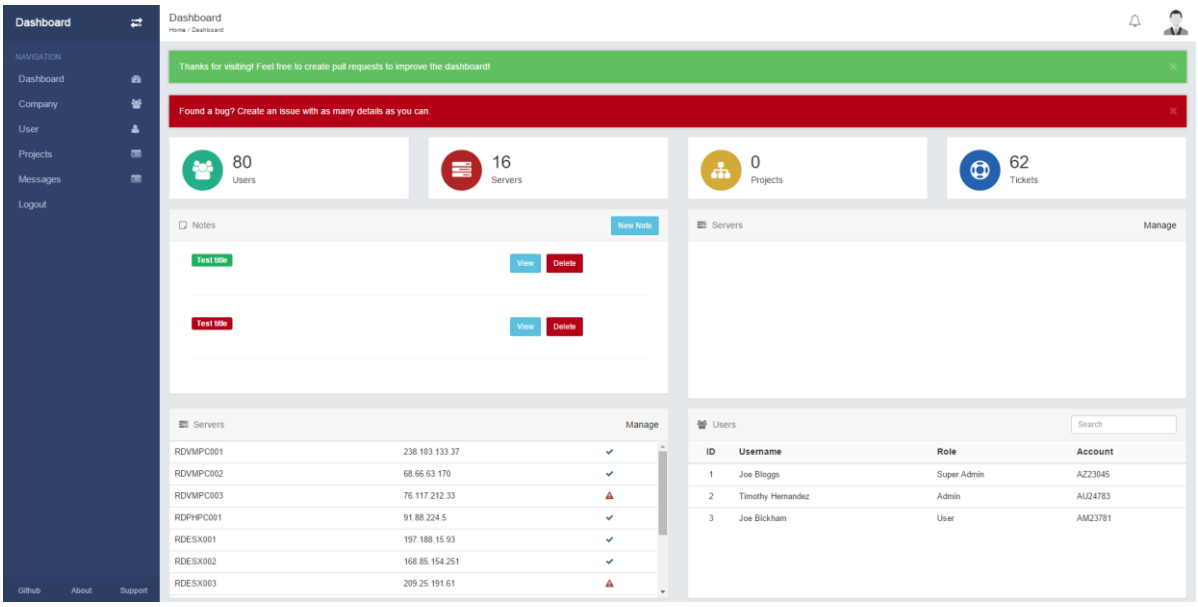
Αυτά είναι μερικά από τα βήματα που πρέπει να ακολουθούμε στο Back-End κομμάτι. Στο Front-End κομμάτι το μεγάλο πρόβλημα είναι οι επιθέσεις τύπου XSS (Cross Site Scripting). Οι επιθέσεις αυτές προσπαθούν να προσθέσουν κακόβουλο κώδικα σε κομμάτια της σελίδας που θα δουν αργότερα άλλοι χρήστες. Η λύση σε αυτό το πρόβλημα είναι ο συνεχής έλεγχος των εισαγωγών του χρήστη και η αποφυγή των επικίνδυνων κομματιών του. Αυτό πρέπει να γίνεται, τόσο στις εισαγωγές του χρήστη όσο και στα

δεδομένα που έρχονται στο Front-End από APIs, είτε από το Back-End της ίδιας της εφαρμογής είτε από κάποιο εξωτερικό.

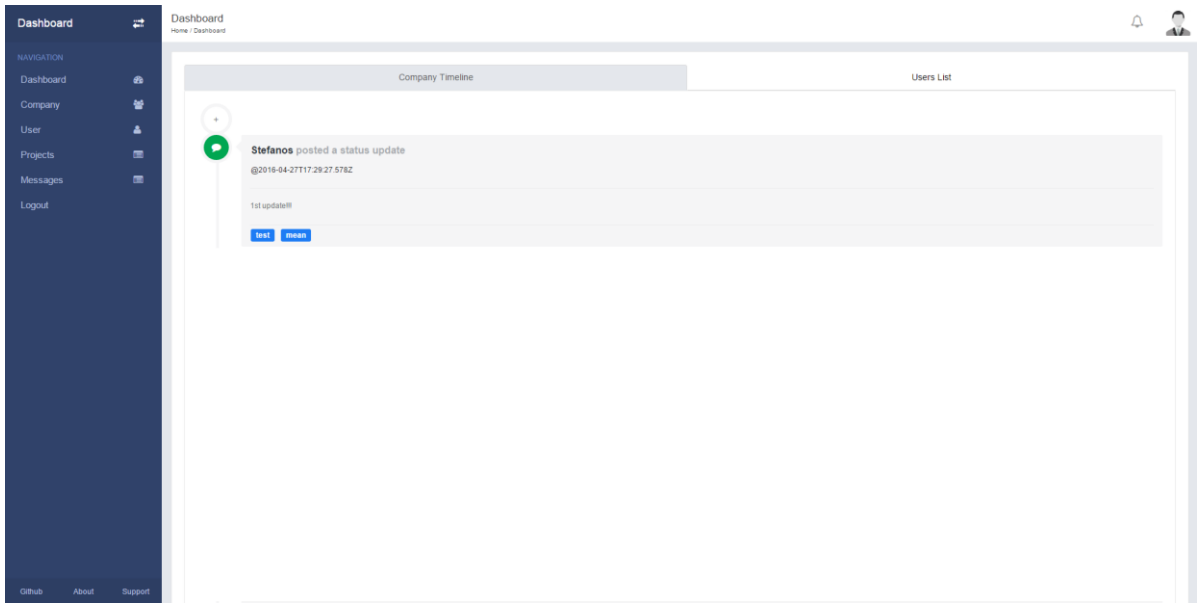
2.8 Project Management App Development

Για να γίνουν όλα τα παραπάνω πιο κατανοητά, θα δημιουργήσουμε μια εφαρμογή χρησιμοποιώντας όλες τις τεχνολογίες του MEAN περιβάλλοντος. Αυτή η εφαρμογή θα είναι ένα project management εργαλείο που θα προσφέρει διάφορες λειτουργίες στον χρήστη για την εύκολη διαχείριση των project και των ατόμων της ομάδας του. Παρακάτω κάνουμε μια σύνοψη των λειτουργιών που θα υλοποιηθούν, πριν προχωρήσουμε στην τεχνική ανάλυση.

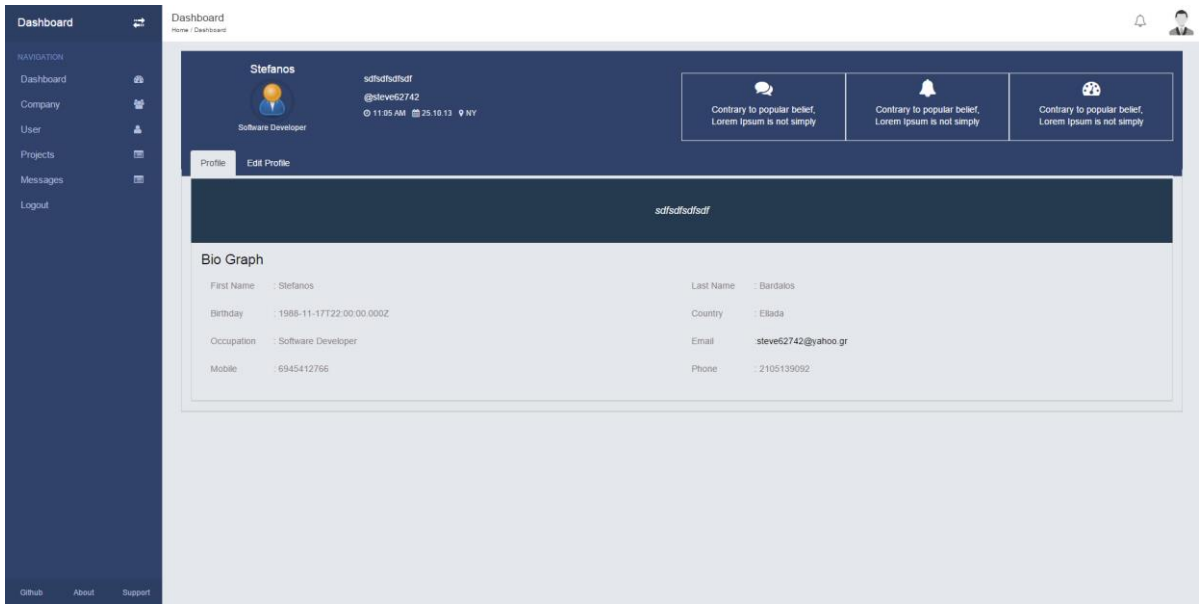
Αρχικά, ο χρήστης θα μπορεί να συνδεθεί στην εφαρμογή είτε με το email του είτε με κάποιο social λογαριασμό του. Μετά από την επιτυχή αυθεντικοποίηση του χρήστη, αυτός θα έχει πρόσβαση στην κεντρική σελίδα της εφαρμογής από την οποία θα βλέπει μια σύνοψη των πληροφοριών που διαθέτει και θα μπορεί να περιηγηθεί στα επιμέρους κομμάτια της.



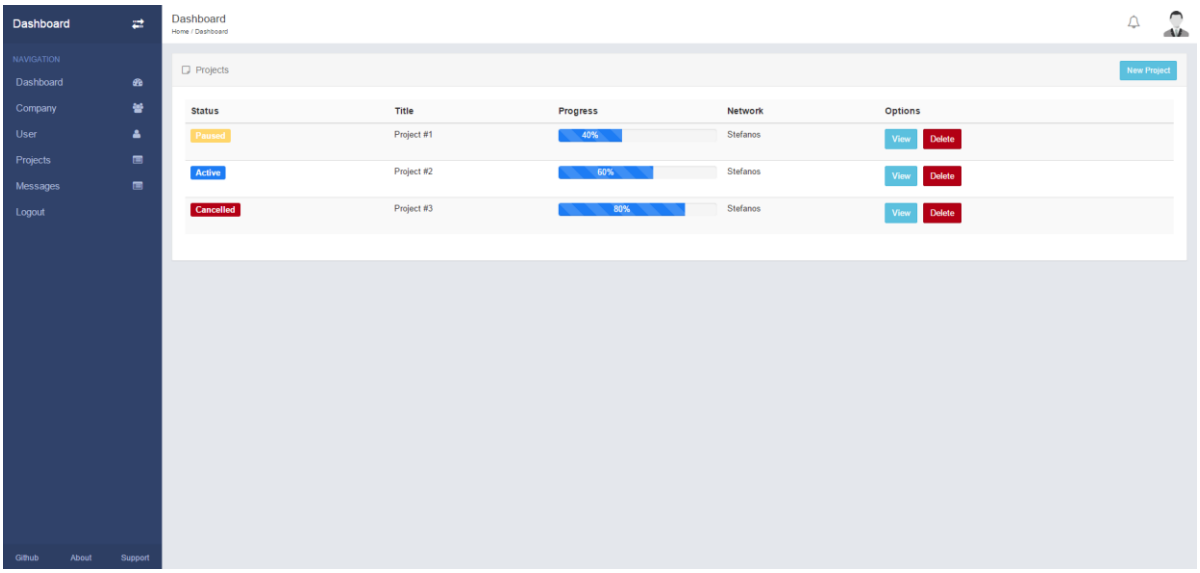
Η επιλογή Company θα τον μεταφέρει σε μία σελίδα με ένα χρονικό timeline στο οποίο εμφανίζονται μηνύματα από τα μέλη της εφαρμογής καθώς και μία λίστα με όλα αυτά ώστε να μπορεί ο καθένας να βρει πληροφορίες γι' αυτούς αλλά και να επικοινωνήσει μαζί τους.



Η επιλογή User θα τον μεταφέρει στο προσωπικό του προφίλ, στο οποίο φαίνονται διάφορα στοιχεία για τον ίδιο τον χρήστη, καθώς και μία σελίδα με ρυθμίσεις ώστε να μπορεί να το εμπλουτίσει με τις πληροφορίες που θέλει να φαίνονται στους υπόλοιπους χρήστες όταν τον αναζητούν.

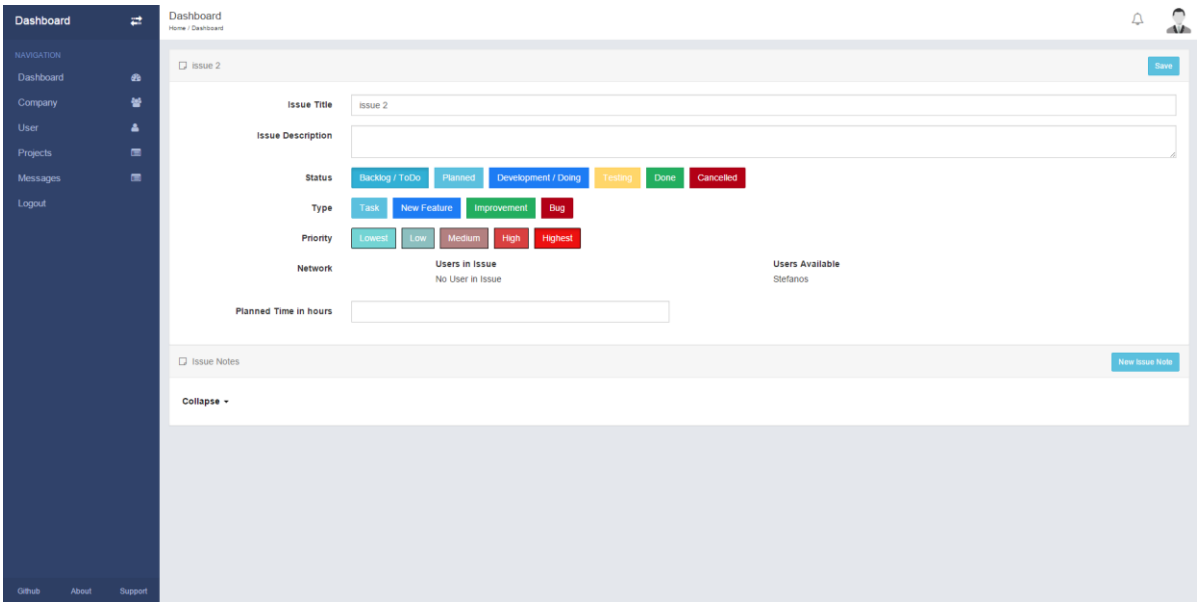


Η επιλογή Projects θα τον μεταφέρει στην σελίδα με τα projects που διαχειρίζεται ο χρήστης, και στην οποία εκτελείται κύριο μέρος της λειτουργικότητας της εφαρμογής. Εδώ ο χρήστης μπορεί να δει τα υπάρχοντα projects αλλά και να δημιουργήσει καινούργια.



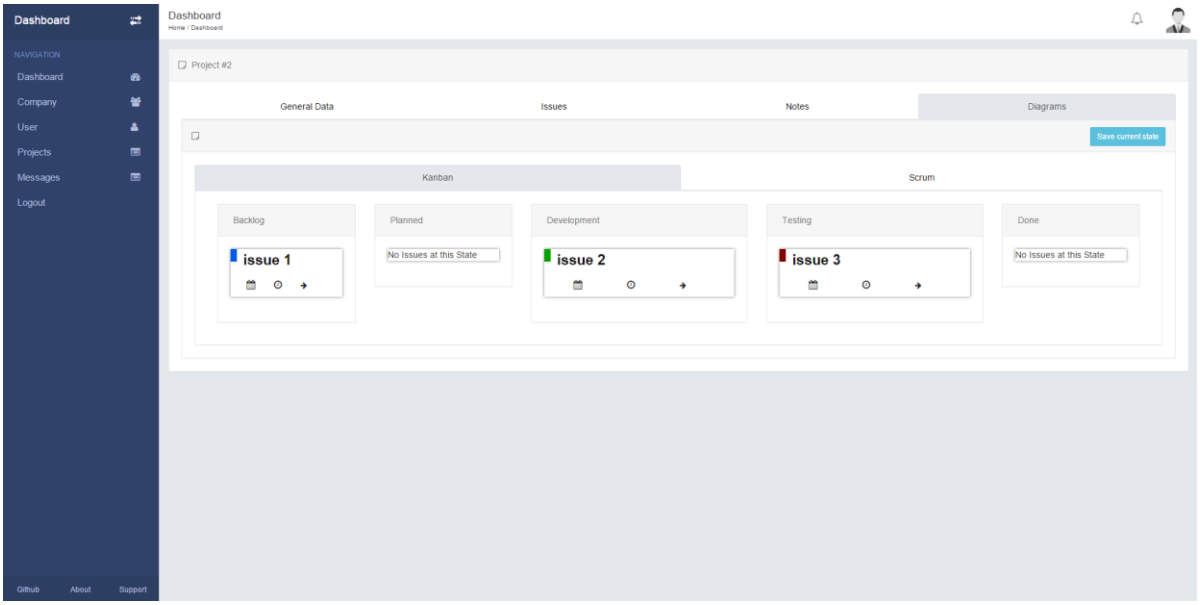
Με την επιλογή ενός από αυτά, ο χρήστης έρχεται μπροστά σε τέσσερις καρτέλες που περιλαμβάνουν τις σημαντικότερες λειτουργίες της εφαρμογής. Αρχικά ο χρήστης μπορεί να επεξεργαστεί τις ιδιότητες του συγκεκριμένου project, αλλάζοντας χαρακτηριστικά όπως την κατάστασή του, αλλά και να προσθέσει άλλους χρήστες που θα δουλέψουν πάνω σε αυτό.

Στην συνέχεια, ο χρήστης μπορεί να δει αλλά και να δημιουργήσει καινούργια issues και, επιλέγοντας κάποιο από αυτά να επεξεργαστεί αντίστοιχα τα δικά του χαρακτηριστικά, όπως τον τύπο του ή την προτεραιότητα την οποία θα πρέπει να έχει, αλλά και να το αναθέσει σε συγκεκριμένους χρήστες για να το επιλύσουν.



Μετά, μπορεί να δει και να δημιουργήσει σημειώσεις σχετικές με το συγκεκριμένο project τις οποίες θα μπορούν να δουν οι υπόλοιποι χρήστες που σχετίζονται με αυτό. Τέλος, μπορεί να χρησιμοποιήσει τα διαδραστικά διαγράμματα τύπου Kanban και Scrum, τα οποία γεμίζουν με τα issues που έχει φτιάξει ο

χρήστης και σχεδιάζονται ανάλογα με τις ιδιότητες αυτών. Με την χρήση drag-and-drop τεχνικής ο χρήστης μπορεί να ανακατανείμει τα issues όπως αυτός νομίζει, ανάλογα πάντα την τεχνική την οποία ακολουθεί η κάθε ομάδα.



Τέλος, με την επιλογή Messages, ο χρήστης μεταφέρεται σε μία σελίδα που του δίνει την δυνατότητα μιας εφαρμογής chat. Μπορεί να δημιουργήσει συνομιλίες με τους υπόλοιπους χρήστες της εφαρμογής και να ανταλλάξει μηνύματα με αυτούς.

Για τη δημιουργία των παραπάνω θα χρησιμοποιηθούν η NodeJS μαζί με το Express Framework για το Back-End, η AngularJS για το Front-End και η MongoDB για βάση δεδομένων. Φυσικά μαζί με αυτά θα χρησιμοποιηθούν και πολλές άλλες βιβλιοθήκες τις οποίες θα δούμε στη συνέχεια. Η δομή αυτού του κεφαλαίου, για να γίνει πιο κατανοητό, θα ακολουθήσει τα σημαντικά χαρακτηριστικά της εφαρμογής, άσχετα αν προέρχονται από το Back-End ή το Front-End, αφού και στην πραγματικότητα τα δύο κομμάτια δημιουργούνται παράλληλα.

Παρακάτω βλέπουμε τη δομή που θα ακολουθήσουμε στο πρόγραμμά μας για να έχουμε όσο καλύτερα γίνεται χωρισμένο τον κώδικά μας. Τα ονόματα των φακέλων είναι αρκετά επεξηγηματικά, οι φάκελοι στο root του προγράμματος αντιστοιχούν στην NodeJS (controllers , models , config), ο φάκελος public περιέχει όλο το στατικό περιεχόμενο που χρειάζεται η εφαρμογή όπως φωτογραφίες και γραμματοσειρές ενώ, ο φάκελος app περιλαμβάνει όλο το Front-End κομμάτι, δηλαδή τα controllers, directives, services, views της AngularJS.

```
|   bower.json
|   gulpfile.js
|   package.json
|   server.js
|
+---app
|   |   app.js
|   |   index.html
|   |
|   +---controllers
|   +---directives
|   +---services
|   \---views
+---config
+---controllers
+---models
\---public
    +---fonts
    +---img
    \---stylesheets
```

Στο αρχείο bower.json θα προσθέσουμε όλες τις βιβλιοθήκες που θέλουμε να κατέβουν με το εργαλείο bower, στο package.json όλες τις βιβλιοθήκες που θέλουμε να κατέβουν με το εργαλείο npm και στο gulpfile.js τις αυτοματοποιημένες εργασίες που θέλουμε να τρέχει το gulp. Το server.js είναι το βασικό αρχείο που θα περιλαμβάνει τον κώδικα της NodeJS και αντίστοιχα το app.js το αρχείο που θα περιλαμβάνει τον κώδικα της AngularJS.

2.8.1. Δημιουργία server και REST API

Ξεκινάμε από την δημιουργία του node server μας. Αρχικά πρέπει να χρησιμοποιήσουμε το npm για να προσθέσουμε κάποια βασικά εργαλεία στο project μας. Παρακάτω φαίνεται η δομή ενός τέτοιου αρχείου με μερικά μόνο εργαλεία από όσα τελικά θα χρειαστούμε. Χαρακτηριστικά μπορούμε να παρατηρήσουμε τα Express, grunt, bower και gulp, τα οποία αναφέραμε σε προηγούμενα κεφάλαια.

```
{
  "name": "Project_Management",
  "version": "0.0.1",
  "dependencies": {
    "async": "^1.4.2",
    "bcryptjs": "^2.2.1",
    "body-parser": "^1.13.3",
    "bower": "*",
    "compression": "^1.5.2",
    "cookie-parser": "^1.3.0",
    "express": "^4.13.1",
    "express-session": "^1.11.3",
    "grunt": "*",
    "install": "^0.1.8",
    "mongoose": "^4.1.0",
    "npm": "^3.3.8",
    "passport": "*",
    "passport-local": "^1.0.0",
    "passport-facebook": "~1.0.2",
    "passport-google-oauth": "~0.1.5",
    "request": "^2.61.0",
    "lusca": "1.3.0",
    "gulp": "^3.9.0",
  }
}
```

Μετά την εγκατάσταση των απαραίτητων εργαλείων μπορούμε να ξεκινήσουμε την δημιουργία του server μας. Παρακάτω βλέπουμε μια αρχική μορφή του server.js που εκτελώντας την κατάλληλη εντολή από την κονσόλα (node server.js) θα έχουμε έναν λειτουργικό node server μαζί το Framework Express.

```
var express = require('express');
var bodyParser = require('body-parser');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.set('port', process.env.PORT || 3000);
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(cookieParser());
app.use(session({ secret: 'session secret key' }));

app.listen(app.get('port'), function() {
  console.log('Express server listening on port ' + app.get('port'));
});
```

Στην αρχή του αρχείου δηλώνουμε όσες βιβλιοθήκες θέλουμε να έχουμε διαθέσιμες στο Back-End μας με τη χρήση της require. Αυτές τις βιβλιοθήκες τις έχουμε πρώτα κατεβάσει με το npm και τώρα τις φορτώνουμε στο πρόγραμμά μας. Μετά ξεκινάμε την εφαρμογή Express (var app = express()) και χρησιμοποιούμε αντίστοιχα τις βιβλιοθήκες που θέλουμε. Εδώ χρησιμοποιούμε τις body-parser (ώστε να μπορεί να διαβάσει το Express τα request που έρχονται προς τον server σε μία json μορφή) , cookie-parser (ώστε να μπορεί να χειριστεί το cookie) και τη session (για τη δημιουργία session μεταξύ του χρήστη και της εφαρμογής). Αυτά είναι όσα χρειαζόμαστε για να δημιουργήσουμε έναν αρχικό server και από εδώ και πέρα, όποια άλλη βιβλιοθήκη θελήσουμε να προσθέσουμε στη συνέχεια, θα γίνεται με τον ίδιο τρόπο. Δηλαδή, θα προστίθεται αρχικά στο πρόγραμμα μέσω της require και μετά θα χρησιμοποιείται στην express με κάποια εντολή τύπου app.use(). Στη συνέχεια, ο server μας πρέπει να έχει κάποια σημεία με τα οποία το Front-End να μπορεί επικοινωνήσει. Παρακάτω βλέπουμε πώς ορίζουμε μερικά REST API endpoints.

```
app.get('/api/projects', function(req, res, next) {
  console.log('GET Request at /api/projects endpoint');
});

app.post('/api/projects/add', function(req, res, next) {
  console.log('POST Request at /api/projects/add endpoint');
});
```

Το πρώτο θα ενεργοποιηθεί όταν γίνει ένα GET request στη διεύθυνση localhost:3000/api/projects ενώ το δεύτερο όταν γίνει ένα PUT request στη διεύθυνση localhost:3000/api/projects/add. Έτσι θα δημιουργήσουμε πολλά σημεία διεπαφής ανάλογα με τις ανάγκες του Front-End, προσθέτοντας απλά στη συνέχεια και άλλες τέτοιες γραμμές. Φυσικά, θα θέλουμε να γίνεται κάτι περισσότερο από ένα απλό console.log. Εδώ έρχεται ο διαχωρισμός του κώδικα ανάλογα με τη λειτουργία του σε διαφορετικά αρχεία και φακέλους για να είναι πιο κατανοητός και εξελίξιμος. Για παράδειγμα, ότι διεργασία έχει να κάνει με τα

projects θα πρέπει να μπει σε ένα αρχείο. Αυτό το αρχείο θα είναι μέσα στον φάκελο controllers (για παράδειγμα , projects-ctrl.js) και θα περιλαμβάνει μεθόδους για κάθε λειτουργία που απαιτούν τα projects. Παρακάτω ακολουθεί ένα παράδειγμα, λογική συνέχεια του προηγούμενου.

```
var projectCtrl = require('express').Router();
projectCtrl.getAll = function(req, res) {
        //Fetch all projects from Database
});
projectCtrl.add = function(req, res) {
        //Add new project to Database
};
module.exports = projectCtrl;
```

Το μόνο που χρειάζεται τώρα είναι να προσθέσουμε το καινούργιο αρχείο με χρήση της require, όπως κάναμε και για τις βιβλιοθήκες, και θα μπορούμε να καλέσουμε τις αντίστοιχες συναρτήσεις στο server.js αντί για απλά console.logs. Ακόμα οι συναρτήσεις αυτές δεν κάνουν κάτι, γιατί δεν έχουμε συνδέσει την εφαρμογή μας με κάποια βάση δεδομένων, πράγμα που θα δούμε στο επόμενο κομμάτι της εργασίας.

2.8.2. Βάση δεδομένων, σχεδίαση μοντέλων και σύνδεση με τον server

Όπως αναφέραμε παραπάνω, θα χρησιμοποιήσουμε για βάση δεδομένων την MongoDB. Η Mongo είναι Document Based βάση και επιτρέπει την αποθήκευση ακανόνιστων δεδομένων. Μαζί με την Mongo, θα χρησιμοποιήσουμε και ένα πολύ γνωστό Framework γι' αυτήν, το Mongoose, το οποίο επιτρέπει τη δημιουργία μοντέλων ή σχημάτων για τα δεδομένα που θέλουμε να αποθηκεύσουμε ώστε να έχουμε μεγαλύτερο έλεγχο ως προς αυτά, ενώ προσθέτει διάφορα άλλα χρήσιμα χαρακτηριστικά όπως μία μέθοδο τύπου JOIN μεταξύ διαφορετικών δεδομένων, πράγμα που στη συμβατική Mongo δε γίνεται. Όπως είπαμε και παραπάνω, το Mongoose πρέπει να δηλωθεί στο package.json για να κατέβει από το npm και να προστεθεί με την require στο server.js. Μετά είμαστε έτοιμοι να χρησιμοποιήσουμε τις εντολές του Mongoose για να επικοινωνήσουμε με τη βάση μας.

Αρχικά θα χρησιμοποιήσουμε το Mongoose για να φτιάξουμε μοντέλα για τα δεδομένα μας. Αυτό θα το κάνουμε σε καινούργιο αρχείο, στον φάκελο models (για παράδειγμα, projects.js). Παρακάτω φαίνεται ένα παράδειγμα για το μοντέλο των δεδομένων που θέλουμε να υπάρχουν στη βάση μας για τα projects. Ουσιαστικά είναι ένα JSON το οποίο δηλώνει το όνομα και το είδος κάθε πεδίου που πρέπει να υπάρχει μέσα σε κάθε στοιχείο που αποθηκεύεται στο collection projects της βάσης.

```
var mongoose = require('mongoose');
var ProjectsSchema = new mongoose.Schema(
  {
    status : Number ,
    id : String ,
    title : String ,
    milestones : [
        {
            title : String ,
            value : Number ,
            done : Boolean
        }
    ],
    notes : [
        {
            content : String ,
            time : Date ,
            user : {
                type : mongoose.Schema.ObjectId ,
                ref : 'User'
            }
        }
    ]
  }
);
ProjectsSchema.methods.comparePassword = function(candidatePassword,
cb) {
  bcrypt.compare(candidatePassword, this.password, function(err,
isMatch) {
    if (err) return cb(err);
    cb(null, isMatch);
  });
};
var Project = mongoose.model('Project', ProjectsSchema);
module.exports = Project;
```

Όπως βλέπουμε , μπορούμε να ορίσουμε διάφορους τύπους δεδομένων, όπως String , Number ή Boolean. Σημαντικό είναι ότι μπορούμε να ορίσουμε και δεδομένα τύπου ObjectId, το οποίο είναι ένα μοναδικό κλειδί που προσθέτει η Mongoose σε κάθε εισαγωγή στη βάση και μπορεί να χρησιμεύσει για κλειδί αναφοράς. Ουσιαστικά παραπάνω, κάθε αντικείμενο note πρέπει να περιλαμβάνει τα content, time και user, όπου αντίστοιχα είναι το κείμενο , η ώρα και ο χρήστης στον οποίο αντιστοιχεί το συγκεκριμένο note. Αυτό θα μας βοηθήσει αργότερα ώστε να μπορούμε να ανακτήσουμε για παράδειγμα τα projects στα οποία έχει αφήσει note κάποιος συγκεκριμένος χρήστης, ενώ με χρήση της Mongoose θα μπορούμε

ζητώντας τα projects να πάρουμε μαζί και τα στοιχεία του χρήστη (τύπου JOIN μεταξύ πινάκων). Ένα άλλο προτέρημα της σχεδίασης των μοντέλων της βάσης και της χρήσης της Mongoose είναι πως μπορούμε να φτιάξουμε συναρτήσεις που να αφορούν συγκεκριμένα δεδομένα και να τις ορίσουμε να τρέχουν αυτόματα είτε πριν είτε μετά την εκτέλεση κάποιας εντολής στη βάση (για παράδειγμα, η κρυπτογράφηση του κωδικού του χρήστη πριν την εισαγωγή του στη βάση δεδομένων).

Τώρα, μένει να το προσθέσουμε στην εφαρμογή μας, συγκεκριμένα στο αρχείο που αντιστοιχεί στον controller των projects, αφού εκεί θα γίνουν και οι λειτουργίες με τη βάση δεδομένων. Το συντακτικό που ακολουθεί το Mongoose για να πάρει ή να αποθηκεύσει δεδομένα στη βάση είναι επαρκές και επιτρέπει πολύ εύκολα να βρούμε ότι χρειαζόμαστε μέσα σε αυτήν. Παρακάτω ακολουθούν οι αντίστοιχες συναρτήσεις για τα projects, μαζί με τις αλλαγές που θα γίνουν στο server.js ώστε να φανεί η συνολική δουλειά που έγινε για να δουλέψουν κανονικά οι δύο αυτές διεπαφές του REST API.

```
//server.js
var mongoose = require('mongoose');

var projectCtrl = require('./controllers/projects-ctrl.js')

mongoose.connect('mongodb://db_username:db_password@db_url:db_port/db
_name');

app.get('/api/projects', function(req, res, next) {
  projectCtrl.getAll(req,res);
});

app.post('/api/projects/add', function(req, res, next) {
  projectCtrl.add(req,res);
});
```

```
//projects-ctrl.js
var User = require('../models/users.js');
var Project = require('../models/projects.js');
var projectCtrl = require('express').Router();
var randomstring = require("randomstring");

projectCtrl.getAll = function(req, res) {
  Project.find({}).populate('notes.user network issues.notes.user
issues.createdBy issues.network'  ).exec(function(err, items) {
      res.send(items);
});

projectCtrl.add = function(req, res) {
  var newproject = new Project({
    title: req.body.title,
    description: req.body.description,
    status: req.body.status,
    network: req.body.network,
    id: randomstring.generate(8)
  });
  newproject.save(function(err) {
    if (err) return next(err);
    res.sendStatus(200);
  });
};
```

Στη συνάρτηση getAll χρησιμοποιούμε την find() η οποία θα μπορούσε να αντιστοιχηθεί με την SELECT της SQL και στο αποτέλεσμά της καλούμε την populate, η οποία κάνει κάτι αντίστοιχο με το JOIN και ενώνει τα δεδομένα του project με τα στοιχεία των users, όπως είδαμε και στο μοντέλο τους, ενώ στο τέλος τα στέλνει με το response στο Front-End ή σε όποιον έκανε την κλήση του API. Αντίστοιχα η συνάρτηση add δημιουργεί ένα καινούργιο αντικείμενο τύπου Project, όπως το ορίσαμε εμείς στο μοντέλο και του περνάει τα στοιχεία του request που ήρθαν από το POST που έγινε στο API. Τέλος, το αποθηκεύει στη βάση και επιστρέφει ανάλογο σήμα σύμφωνα με την επιτυχία ή μη της αποθήκευσης.

Αντίστοιχη δουλειά θα πρέπει να γίνει για όλα τα δεδομένα που θέλουμε να αποθηκεύουμε στη βάση μας. Δηλαδή, να ορίσουμε τα μοντέλα τους, να δημιουργήσουμε τα endpoints για αυτά στο API μας και, να φτιάξουμε controllers που να περιέχουν της συναρτήσεις που θα εκτελούν τις αντίστοιχες εργασίες. Για την εφαρμογή που δημιουργούμε για παράδειγμα, θα πρέπει να δημιουργηθούν μοντέλα για τα δεδομένα των χρηστών, των projects , των σημειώσεων , των υπενθυμίσεων , των μηνυμάτων επικοινωνίας και των μηνυμάτων ενημέρωσης. Γι' αυτά θα πρέπει να υπάρχουν οι αντίστοιχοι controllers με διάφορες συναρτήσεις για το καθένα και φυσικά τα endpoints του API για να μπορεί να υπάρξει επικοινωνία με το Front-End.

2.8.3. Δημιουργία AngularJS App

Συνέχεια έχει το Front-End κομμάτι της εφαρμογής. Ξεκινάμε όπως και με το Back-End, προσθέτοντας στο project μας τις βιβλιοθήκες και τα εργαλεία που θα χρειαστούμε. Αυτή τη φορά, θα χρησιμοποιήσουμε το εργαλείο bower και το αρχείο bower.json. Εκεί θα δηλώσουμε τις βιβλιοθήκες που θέλουμε να εγκαταστήσει το εργαλείο bower, σε μία μορφή JSON η οποία φαίνεται και στο παρακάτω παράδειγμα, όπου εισάγουμε μερικές μόνο από τις βιβλιοθήκες που χρειαζόμαστε για την εφαρμογή μας. Στη συνέχεια, όποτε χρειαζόμαστε κάποια καινούργια, μπορούμε να γυρίσουμε σε αυτό το αρχείο και να την προσθέσουμε.

```
{
  "name": "Project_Management",
  "version": "0.0.1",
  "dependencies": {
    "angular": null,
    "angular-resource": null,
    "angular-cookies": null,
    "angular-sanitize": null,
    "angular-animate": null,
    "angular-touch": null,
    "angular-messages": null,
    "angular-strap": null,
    "angular-route": null,
    "angular-ui-router": null,
    "angular-chart.js": null,
    "angular-bootstrap": null,
    "angular-filter": null,
    "bootstrap": null,
    "font-awesome": null,
    "sortable.js": null,
    "angular-smart-table": "~2.1.4",
    "angular-scroll-glue": "~2.0.6"
  },
  "appPath": "app"
}
```

Χαρακτηριστικά βλέπουμε ότι, όπως και στο npm, μπορούμε να δηλώσουμε συγκεκριμένες εκδόσεις αυτών των βιβλιοθηκών ή, να το αφήσουμε να εγκαταστήσει κατευθείαν την τελευταία που υπάρχει. Το bower θα εγκαταστήσει όλες τις παραπάνω βιβλιοθήκες και θα μπορούμε να ξεκινήσουμε τη δημιουργία της AngularJS εφαρμογής μας.

Για να χρησιμοποιήσουμε την AngularJS χρησιμοποιήσαμε αρχικά δύο αρχεία, το index.html που θα περιέχει την αρχική μας σελίδα, η οποία συνήθως έχει μόνο όλα τα scripts και stylesheets που χρειαζόμαστε, και το app.js από το οποίο θα ελέγχουμε μέσω της Angular την ροή της εφαρμογής. Μια τυπική δομή για το index.html είναι η παρακάτω. Αυτό που χρειάζεται να παρατηρήσουμε είναι πως στην αρχή της html ορίζουμε το όνομα της Angular εφαρμογής μας (ng-app=”MyApp”). Ύστερα, μέσα στο head

φορτώνουμε όλα τα stylesheet αρχεία των βιβλιοθηκών που έχουμε, ενώ μετά, φορτώνουμε διαδοχικά, την Angular, τις υπόλοιπες βιβλιοθήκες που έχουμε και μετά, τον δικό μας κώδικα όπου αντίστοιχα τον έχουμε χωρίσει σε κατάλληλα χωρισμένα αρχεία και υποφακέλους.

```
<!doctype html>
<html ng-app="MyApp">
<head>
  <base href="/">
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <title>Project Management</title>
  <link rel="icon" type="/public/image/png" href="favicon.png" >
  <link rel="stylesheet" type="text/css"
href="/bower_components/*/*.css" >
  <link rel="stylesheet" type="text/css"
href="/public/stylesheets/*.css" >
</head>

<body>
  <div ui-view></div>

  <script src="/bower_components/angular/angular.js"></script>
  <script src="/bower_components/*.js"></script>
  <script src="app.js"></script>
  <script src="controllers/*.js"></script>
  <script src="services/*.js"></script>
  <script src="directives/*.js"></script>

</body>
</html>
```

Αντίστοιχα, στο αρχείο app.js πρέπει να δημιουργήσουμε την Angular εφαρμογή μας. Μία απλή μορφή αυτού του αρχείου ακολουθεί.

```
angular.module('MyApp', ['ngCookies', 'ngResource', 'ngMessages',
'ngRoute', 'ui.router','ui.bootstrap'])
  .config(['$stateProvider', '$urlRouterProvider',
function($stateProvider, $urlRouterProvider ) {

  });
```

Εδώ ξεκινάμε την εφαρμογή μας με το ίδιο όνομα που έχουμε ορίσει στην html σελίδα και μαζί φορτώνουμε έναν πίνακα που περιέχει όσες βιβλιοθήκες έχουμε εγκαταστήσει από το bower και θα

χρησιμοποιήσουμε στην εφαρμογή μας. Ακολουθεί η συνάρτηση config όπου εκεί θα ορίσουμε διάφορα πράγματα για τον χειρισμό της εφαρμογής μας.

Πριν το κάνουμε αυτό, πρέπει να ξεκαθαρίσουμε τα διαφορετικού είδους εργαλεία της Angular που θα χρησιμοποιήσουμε για να γίνει κατανοητός και ο διαχωρισμός σε αρχεία και φακέλους. Ο controller περιέχει, όπως και στο Back-End, όλες τις μεθόδους με τις οποίες γίνονται αλλαγές τόσο στα δεδομένα όσο και στο View, σε αυτό που εμφανίζεται δηλαδή στον χρήστη. Τα directives είναι στοιχεία, κομμάτια κώδικα τα οποία δημιουργούν κάτι στο Front-End το οποίο μπορούμε να χρησιμοποιήσουμε ξανά και ξανά, όπως για παράδειγμα, μία φόρμα ή ένα messagebox ή, στην περίπτωσή μας, ένα widget-box. Τα services είναι περίπου ο τρόπος με τον οποίο ορίζει το μοντέλο (Model στην MVC αρχιτεκτονική) η Angular. Εκεί κρατάμε τα δεδομένα της εφαρμογής τα οποία θέλουμε να μπορούν να προσπελάσουν όλοι οι controllers. Τέλος, τα views είναι αρχεία html που περιλαμβάνουν το template που, μαζί με τα δεδομένα που θα ενώσει ο controller, θα εμφανιστεί στην οθόνη του χρήστη.

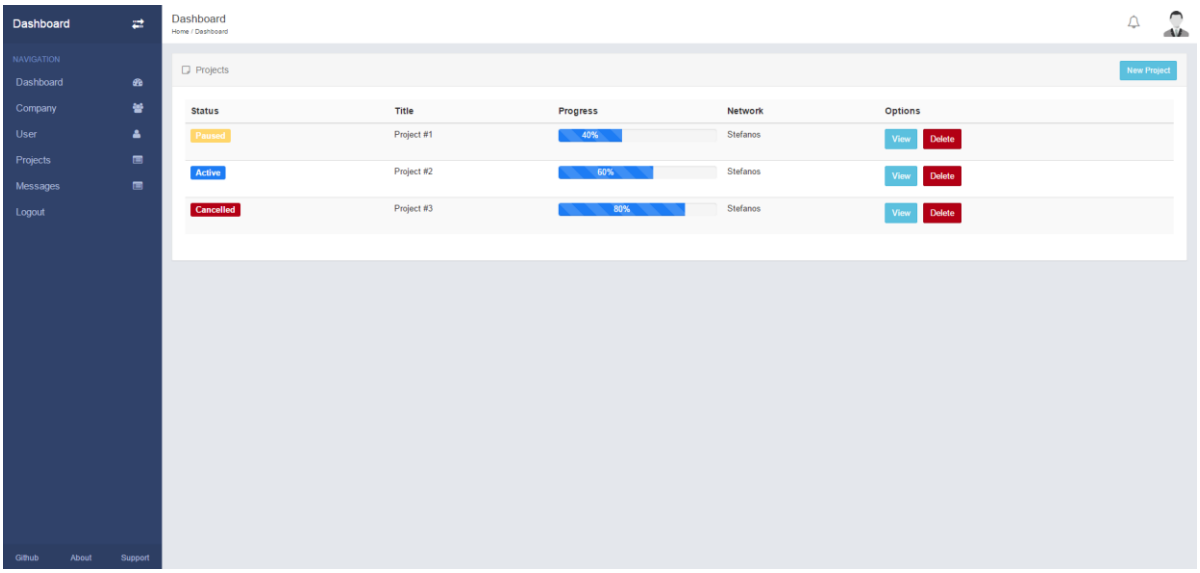
Τώρα, η σύνδεση των παραπάνω μπορεί να γίνει με δύο εργαλεία της Angular, είτε χρησιμοποιώντας states (καταστάσεις) είτε χρησιμοποιώντας routes (διευθύνσεις). Στην πρώτη περίπτωση, ορίζουμε καταστάσεις της εφαρμογής, που δεν εξαρτώνται από τη διεύθυνση στην οποία βρίσκεται ο χρήστης και προσφέρει μια μεγαλύτερη ευκολία στον ορισμό εμφωλευμένων καταστάσεων. Αντίθετα, στη δεύτερη περίπτωση ορίζουμε διευθύνσεις στις οποίες ορίζουμε τη λογική που θα τους αντιστοιχεί. Στην εφαρμογή μας επιλέξαμε την πρώτη μέθοδο, γι ‘αυτό και στο app.js προσθέσαμε τον $stateProvider. Παρακάτω φαίνεται ο κώδικας που πρέπει να προσθέσουμε μέσα στην config για να ορίσουμε μερικές καταστάσεις.

```
$stateProvider
  .state('home', {
    url: '/home',
    templateUrl: 'views/dashboard.html',
    controller: 'MasterCtrl',
    data: {
      requireLogin: true
    }
  })
  .state('login', {
      url: '/login',
      templateUrl: 'views/login.html',
      controller: 'LoginCtrl',
      data: {
        requireLogin: false
      }
  })
  .state('projects',{
      templateUrl:'views/projects.html',
      url:'/projects',
      controller: 'ProjectCtrl',
      data: {
         requireLogin: true
      }
  })
```

Όπως βλέπουμε, ορίζουμε τον Controller ο οποίος θα ελέγχει τη ροή της εφαρμογής σε εκείνη την κατάσταση, το View το οποίο θα εμφανιστεί μπροστά, το url που θέλουμε να εμφανιστεί στην μπάρα διεύθυνσης του browser και τέλος, προαιρετικά κάποια δεδομένα που θα μπορούσαν να μας χρησιμεύσουν στην αλλαγή κατάστασης. Στη δικιά μας περίπτωση, χρησιμοποιούμε μια μεταβλητή για να ξεχωρίζουμε τις καταστάσεις που θα μπορεί να βρεθεί ένας χρήστης ανάλογα αν έχει αυθεντικοποιηθεί προηγουμένως ή όχι. Μετά από αυτό, μπορούμε να προχωρήσουμε στη δημιουργία ενός Controller και ενός View για να γίνει πιο κατανοητή η σύνδεση όλων αυτών.

2.8.4. Δημιουργία Controller, View και Service

Θα ακολουθήσουμε το παράδειγμα του Project, όπως κάναμε και για το Back-End. Πρέπει να δημιουργήσουμε ένα Service που θα μεταφέρει τα δεδομένα μας, ένα View, δηλαδή ένα template για τα Projects και έναν Controller όπου θα αναλάβει τη σύνδεση των δυο. Παρακάτω βλέπουμε το πραγματικό αποτέλεσμα όπως δημιουργήθηκε για την εφαρμογή.



Εικόνα 7. Projects State

Θα ξεκινήσουμε από το Service. Παρακάτω ακολουθεί ένα παράδειγμα που περιλαμβάνει ένα κομμάτι του πραγματικού Service που φτιάχτηκε για την εφαρμογή.

```
angular.module('MyApp')
  .factory('ProjectsService',  function() {
      var ProjectsService = {};
      var projects = [];
      ProjectsService.setData = function(data) { projects = data; }
      ProjectsService.getItem = function(index) {
        return projects[index]; }
      ProjectsService.getItemId = function(id) {
              for (var i =0 ; i < projects.length ; i++){
                if (projects[i].id == id){
                  return projects[i];
                }
              }
              return false;
       }
      ProjectsService.addItem = function(item) {projects.push(item);}
      ProjectsService.removeItem = function(item) {
        projects.splice(projects.indexOf(item), 1) }
      ProjectsService.removeItemIndex = function(index) {
        projects.splice(index, 1) }
      ProjectsService.removeItemId = function(id) {
        for (var i =0 ; i < projects.length ; i++){
          if (projects[i].id == id){
            projects.splice(i,1);
            return true;
          }
        }
        return false;
       }
      ProjectsService.size = function() { return projects.length; }
      ProjectsService.getAll = function() {return projects; }
      return ProjectsService;
  });
```

Όπως βλέπουμε, ο ρόλος του Service είναι να κρατάει τα δεδομένα και να τα δίνει στον Controller μέσα από διάφορες συναρτήσεις ανάλογα την ανάγκη κάθε φορά. Στο παράδειγμα βλέπουμε συναρτήσεις που επιστρέφουν ή διαγράφουν όλα ή συγκεκριμένα projects.

Προφανώς θα χρειαστεί να γράψουμε πολλές ακόμα συναρτήσεις ανάλογα με το τι θέλουμε να κάνουμε, όπως για παράδειγμα να μας επιστρέφει μόνο τα projects που είναι ακόμη ενεργά ή αυτά που έχουν βαθμό ολοκλήρωσης πάνω από 50%.

Στη συνέχεια μπορούμε να δούμε το View με το οποίο θα εμφανιστούν τα δεδομένα μας. Παρακάτω φαίνεται ο κώδικας για τη δημιουργία του πίνακα με τα projects. Εδώ βλέπουμε τις μεγάλες δυνατότητες που προσφέρει η AngularJS με τα δικά της Directives όπως το ng-repeat, που για κάθε project που έχουμε

στα δεδομένα μας θα δημιουργήσει άλλη μία γραμμή στον πίνακα. Αντίστοιχα, με το ng-if εμφανίζουμε ή όχι δυναμικά στοιχεία στο View. Χαρακτηριστικά μπορούμε να δούμε το ui-sref το οποίο ορίζεται στο link View της κάθε γραμμής, το οποίο θα μας μεταφέρει στην αντίστοιχη κατάσταση με τα κατάλληλα δεδομένα ώστε να αλλάξει το View στο αντίστοιχο template.

```
    <tr ng-repeat="project in projects">
        <td>
            <span class="label label-primary" ng-if='project.status
=== 1'>Active</span>
            <span class="label label-danger" ng-if='project.status
=== 0'>Cancelled</span>
            <span class="label label-warning" ng-if='project.status
=== 2'>Paused</span>
        </td>
        <td>{{project.title}}
        </td>
        <td>
            <uib-progressbar class="progress-striped"
value="project.progress" >{{project.progress}}%</uib-progressbar>
        </td>
        <td>
              <ul >
                <li ng-repeat="person in project.network"
style="display: inline;">
                  <span>{{person.name}}</span>
                </li>
              </ul>
        </td>
        <td>
                <a class="btn btn-sm btn-info" ui-
sref="project({projectId :  project.id })" >View</a>
                <button class="btn btn-sm btn-danger" ng-
click="launchDeleteModal('project' , project)">Delete</button>
        </td>
    </tr>
```

Τώρα, μένει να φτιάξουμε και τον Controller, ο οποίος θα ενώσει τα δύο παραπάνω στοιχεία για να εμφανιστεί τελικά η κατάσταση της εφαρμογής όπως την είδαμε στην παραπάνω φωτογραφία.

Παρακάτω βλέπουμε μία αρχική μορφή του Controller μας, όπου αρχικά διαβάζουμε από το Service των projects που δημιουργήσαμε προηγουμένως (και των User αντίστοιχα) καθώς και δύο ασύγχρονα GET Requests τα οποία επικοινωνούν με το REST API που φτιάξαμε για να φέρουν από τη βάση τα δεδομένα που χρειάζεται η εφαρμογή μας.

```
angular.module('MyApp')
    .controller('ProjectCtrl', ['$scope' , '$http' , '$state' ,
'$filter' , '$stateParams' ,  'ProjectsService' , 'MessageBox' ,
'UsersService' ,'FormModal'  , ProjectCtrl]);

function ProjectCtrl($scope , $http , $state , filter , $stateParams
, ProjectsService , MessageBox , UsersService , FormModal) {

    $scope.projects = ProjectsService.getAll();
    $scope.allusers = UsersService.getAll();

    $http({method: 'GET', url: '/api/projects'}).
    success(function(data, status, headers, config) {
        console.log('todos: ', data );
        ProjectsService.setData(data);
        $scope.projects = ProjectsService.getAll();
    }).
    error(function(data, status, headers, config) {
        console.log('Oops and error', data);
    });

    $http({method: 'GET', url: '/api/users'}).
    success(function(data, status, headers, config) {
        console.log('todos: ', data );
        UsersService.setUsers(data);
        $scope.users = UsersService.getAll();
    }).
    error(function(data, status, headers, config) {
        console.log('Oops and error', data);
    });
}
```

Φυσικά αυτό δε φτάνει, πρέπει να προσθέσουμε και τις συναρτήσεις που θα κάνουν τις διεργασίες που θέλουμε. Για παράδειγμα, μία συνάρτηση η οποία θα δημιουργεί ένα καινούργιο project και θα το αποθηκεύει και στη βάση ή μία που να αλλάζει τα χαρακτηριστικά ενός project και να περνάει τις αλλαγές και στη βάση δεδομένων. Αυτές οι συναρτήσεις θα καλούνται από τον χρήστη μέσα από κουμπιά του template που του εμφανίζονται. Παρακάτω βλέπουμε τις δύο αυτές συναρτήσεις.

```
$scope.saveNewProject = function(  title, desc, type , network ) {
    new_project = {};
    new_project.status = parseInt(type);
    new_project.description = desc;
    new_project.title = title;
    new_project.network = [];
    for (var i in network){
        if (network[i]){
            new_project.network.push ( {name:i} ) ;
        }
    }
    new_project.progress =0;
    $http({method: 'POST', data : new_project , url:
'/api/projects/add'}).
    success(function(data, status, headers, config) {
      ProjectsService.addItem(new_project);
    }).
    error(function(data, status, headers, config) {
      console.log('Oops and error', data);
    });
    $state.go('projects');
};
$scope.editProject = function(    ) {
    post_data = {
        curProject : $scope.project._id ,
        editoptions : {
            title : $scope.project.title ,
            description : $scope.project.description ,
            status : $scope.project.status ,
            milestones : $scope.project.milestones ,
            network : $scope.project.network
        }
    };
    $http({method: 'POST', data : post_data , url:
'/api/projects/edit'}).
    success(function(data, status, headers, config) {
    }).
    error(function(data, status, headers, config) {
      console.log('Oops and error', data);
    });
};
```

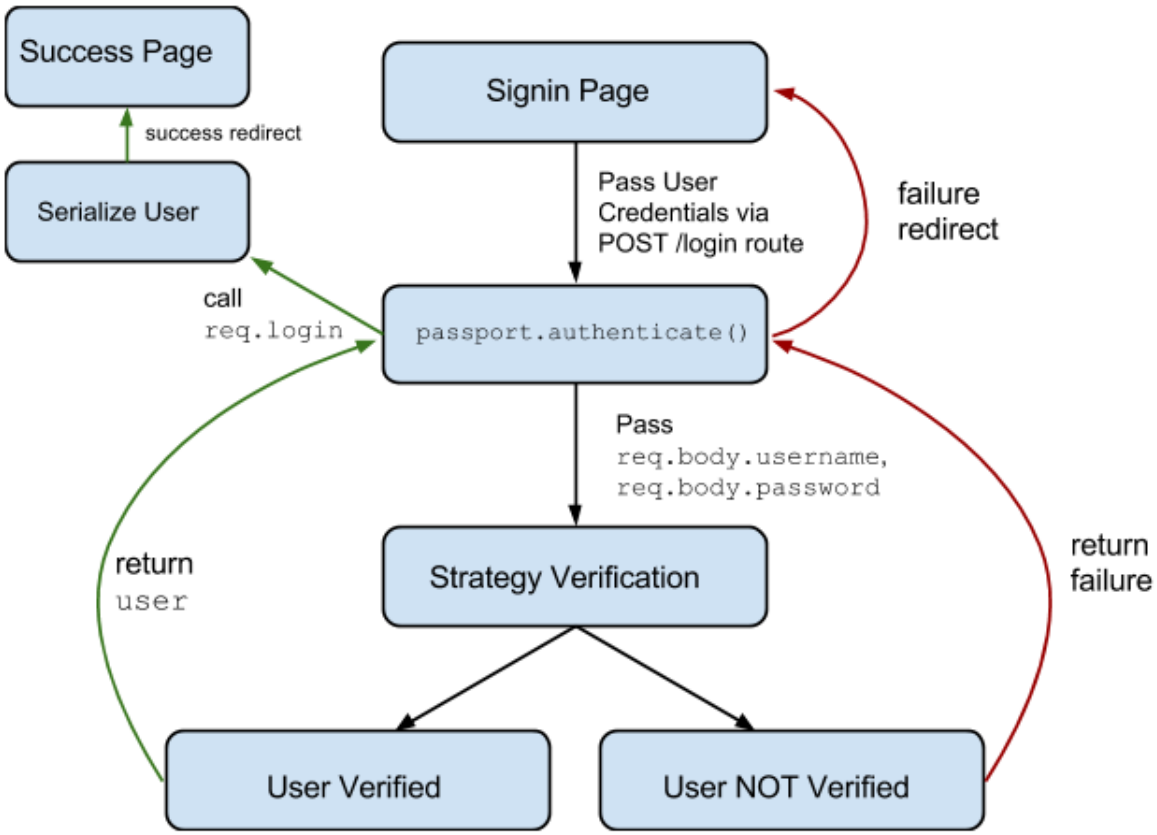
Με τα παραπάνω παραδείγματα πρέπει να έχει γίνει κατανοητή η ροή του Development στην AngularJS. Για όσα νέα χαρακτηριστικά θέλουμε να προσθέσουμε για τα projects, θα πρέπει να δημιουργήσουμε καινούργιες συναρτήσεις που να εκτελούν τις αντίστοιχες λειτουργίες.

Η παραπάνω δομή πρέπει να ακολουθηθεί και για τις υπόλοιπες καταστάσεις και τύπους δεδομένων της εφαρμογής μας, δηλαδή για παράδειγμα, πρέπει να δημιουργηθεί ένα Service που να περιέχει τα στοιχεία του χρήστη , ένα View που να περιέχει το template για τη σελίδα profile του χρήστη και ένας Controller, που να ενώνει τα δύο και να επικοινωνεί με το Back-End για τις όποιες αλλαγές στη βάση δεδομένων.

2.8.5. Αυθεντικοποίηση χρηστών

Στη συνέχεια, θα ασχοληθούμε με το κομμάτι της αυθεντικοποίησης των χρηστών της εφαρμογής. Ο χρήστης έχει τρεις επιλογές για να συνδεθεί με την εφαρμογή, είτε να δημιουργήσει δικό του λογαριασμό, δίνοντας στον λογαριασμό email του και έναν κωδικό πρόσβασης , είτε μέσα από τα account που διατηρεί στο facebook και google+. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε τη βιβλιοθήκη Passport.js . Η συγκεκριμένη βιβλιοθήκη επιτρέπει την εύκολη αυθεντικοποίηση των χρηστών και γι' αυτό χρησιμοποιείται πολύ σε εφαρμογές node, ειδικά όταν χρησιμοποιούν και το Express.js.

Το μεγάλο προτέρημα της Passport είναι πως δίνει την δυνατότητα στον χρήστη να ενσωματώσει λειτουργίες φτιαγμένες από άλλους χρήστες, τις οποίες ονομάζει strategies. Αυτή τη στιγμή έχουν δημιουργηθεί πάνω από 300 τέτοιες λειτουργίες που επιτρέπουν τη σύνδεση χρηστών με πάρα πολλούς τρόπους είτε χρησιμοποιώντας λογαριασμούς γνωστών δικτύων (instagram , dropbox , github) είτε ακολουθώντας γνωστές τεχνικές (Kerberos 5 , JSON Web Token). Φυσικά, ο καθένας μπορεί να γράψει και τη δικιά του λειτουργία αλλά οι ήδη δημοσιευμένες έχουν ενσωματωθεί και ελεγχθεί από πάρα πολλές εφαρμογές. Στη δικιά μας εφαρμογή θα χρησιμοποιήσουμε τρεις τέτοιες λειτουργίες, την passport-local , που χρησιμοποιεί τη βάση δεδομένων της εφαρμογής μας, και τις passport-facebook και passport-google-oauth για τα δύο social media αντίστοιχα. Παρακάτω βλέπουμε ένα διάγραμμα με την ολοκληρωμένη ροή που ακολουθεί η passport για τη διαδικασία της αυθεντικοποίησης.



Εικόνα 8. Passport.js Workflow

Τα βήματα για να ενσωματώσουμε την παραπάνω λειτουργία στην εφαρμογή μας είναι τρία. Αρχικά πρέπει να εγκαταστήσουμε τα απαραίτητα πακέτα μέσω του εργαλείου npm, άρα να προσθέσουμε τις απαραίτητες βιβλιοθήκες στο αρχείο package.json ώστε αυτό να τις κατεβάσει. Ύστερα, όπως κάναμε και με τις υπόλοιπες βιβλιοθήκες, πρέπει να την προσθέσουμε στο πρόγραμμά μας με τη χρήση της require και να την χρησιμοποιήσουμε με την use. Παρακάτω φαίνονται οι ελάχιστες γραμμές που πρέπει να προστεθούν στο αρχείο server.js για να ενσωματωθεί η Passport.js .

```
var passport = require('passport');
app.use(passport.initialize());
app.use(passport.session());
```

Το δεύτερο βήμα είναι η κατάλληλη ρύθμιση της Passport ανάλογα με τον τρόπο αυθεντικοποίησης που θέλουμε να χρησιμοποιήσουμε. Αυτό θα μπορούσε να γίνει είτε στο αρχείο server.js είτε σε ξεχωριστό, αν θέλουμε να κρατήσουμε όλη την λογική αυτής της διαδικασίας μαζεμένη για να γίνει πιο ευανάγνωστη. Παρακάτω φαίνεται ο κώδικας που χρειάζεται να γράψουμε για να δουλέψει η αυθεντικοποίηση με τη χρήση της δικιάς μας βάσης δεδομένων.

```
var LocalStrategy = require('passport-local').Strategy;
passport.serializeUser(function(user, done) {
  done(null, user._id);
});
passport.deserializeUser(function(id, done) {
  User.findById(id, function(err, user) {
    done(err, user);
  });
});
passport.use(new LocalStrategy({ usernameField: 'email' },
function(email, password, done) {
  User.findOne({ email: email }, function(err, user) {
    if (err) return done(err);
    if (!user) return done(null, false);
    user.comparePassword(password, function(err, isMatch) {
      if (err) return done(err);
      if (isMatch) return done(null, user);
      return done(null, false);
    });
  });
}));
```

Η συνάρτηση που χρησιμοποιεί την LocalStrategy παίρνει τα email και password που έχει στείλει ο χρήστης κατά τη διάρκεια του login. Ψάχνει στη βάση δεδομένων (εδώ έχουμε δημιουργήσει και το μοντέλο User, αντίστοιχο του Project που αναλύσαμε προηγουμένως) και αν βρει χρήστη με το email αυτό, τότε ελέγχει αν ταιριάζει ο αποθηκευμένος κωδικός με αυτός που έδωσε ο χρήστης. Σε κάθε περίπτωση καλείται η συνάρτηση done, η οποία επιστρέφει στην passport το αποτέλεσμα της διαδικασίας, δηλαδή είτε τα στοιχεία του χρήστη σε περίπτωση επιτυχής σύνδεσης, είτε το κατάλληλο σφάλμα.

Οι άλλες δύο συναρτήσεις, serializeUser και deserializeUser, χρησιμοποιούνται από την Passport για τη σωστή λειτουργία του session. Η serializeUser αποθηκεύει μέσα στο cookie του χρήστη ότι θέλουμε, στη συγκεκριμένη περίπτωση id του. Αντίστοιχα, η deserializeUser, διαβάζει το cookie όταν έρχεται κάποιο request, κάνει αναζήτηση στη βάση για να βρει τον χρήστη με το συγκεκριμένο id και, τον αποθηκεύει στο request για να προχωρήσει για τη λειτουργία στην οποία αντιστοιχεί.

Τέλος, το τρίτο βήμα που πρέπει να γίνει, είναι να δημιουργήσουμε το αντίστοιχο endpoint στο REST API μας για να μπορεί να επικοινωνήσει με αυτό η εφαρμογή μας. Παρακάτω βλέπουμε τις γραμμές κώδικα που πρέπει να προστεθούν στο αρχείο server.js .

```
app.post('/login',
  passport.authenticate('local', { successRedirect: '/',
                                   failureRedirect: '/login'
                                 })
);
```

Το παραπάνω σημαίνει πως όταν η εφαρμογή μας κάνει POST κλήση στον server μας, στην διεύθυνση /login, θα ενεργοποιηθεί η Passport, θα εκτελέσει τον κώδικα που γράψαμε παραπάνω για την local

αυθεντικοποίηση και , σε περίπτωση επιτυχίας ο χρήστης θα μεταφερθεί στην αρχική σελίδα της εφαρμογής, αλλιώς θα επιστρέψει στην /login.

Όπως βλέπουμε, η Passport διευκολύνει ιδιαιτέρως την όλη διαδικασία ενώ παρουσιάζει μεγάλη ευελιξία, αφού μπορεί να χρησιμοποιηθεί σε οποιαδήποτε εφαρμογή ανεξαρτήτως του σχεδιασμού της βάσης που έχει επιλέξει ο καθένας.

2.8.6. Βελτιστοποίηση Εφαρμογής

Στο τέλος, αφού έχουμε ολοκληρώσει την εφαρμογή μας και όλες οι λειτουργίες της δουλεύουν, μένει να κάνουμε κάποια βήματα για τη βελτιστοποίησή της για την καλύτερη εμπειρία του χρήστη. Όπως είδαμε και σε προηγούμενο κεφάλαιο, ο σκοπός είναι να μειώσουμε το μέγεθος της μεταφερόμενης πληροφορίας αλλά και τις πολλές συνδέσεις. Θα χρησιμοποιήσουμε το εργαλείο gulp με το οποίο θα χρησιμοποιήσουμε διάφορες άλλες λειτουργίες τις οποίες όμως θα κατεβάσουμε με το εργαλείο npm όπως έχουμε κάνει μέχρι τώρα.

Συγκεκριμένα, θα χρησιμοποιήσουμε τα csso , uglify , concat και angular-templatecache. Στο αρχείο gulpfile.js θα δημιουργήσουμε τρία διαφορετικά tasks τα οποία θα χρησιμοποιούν τα παραπάνω εργαλεία για να βελτιστοποιήσουν το css μας, την JavaScript αλλά και τα html view της εφαρμογής μας. Το concat , θα ενώσει όλα τα αρχεία που θα του ζητήσουμε σε ένα μοναδικό, τα csso και uglify θα κάνουν minify σε css και JavaScript κώδικα αντίστοιχα. Το angular-templatecache, θα μετατρέψει τα templates της εφαρμογής μας σε κώδικα για να τρέχουν πολύ πιο γρήγορα στην εφαρμογή μας. Παρακάτω, βλέπουμε το task που έχει δημιουργηθεί για τα αρχεία JavaScript. Με αντίστοιχο τρόπο δημιουργούνται και τα υπόλοιπα tasks.

```
gulp.task('compress', function() {
  gulp.src([
    'bower_components/angular/angular.js',
    'bower_components/angular-ui-router/release/angular-ui-
router.js',
    'bower_components/angular-route/angular-route.js',
    'bower_components/angular-messages/angular-messages.js',
    'bower_components/angular-resource/angular-resource.js',
    'bower_components/angular-cookies/angular-cookies.js',
    'bower_components/oclazyload/dist/ocLazyLoad.min.js',
    'bower_components/angular-animate/angular-animate.js',
    'bower_components/angular-bootstrap/ui-bootstrap.js',
    'bower_components/angular-bootstrap/ui-bootstrap-tpls.js',
    'bower_components/angular-smart-table/dist/smart-table.js',
    'bower_components/sortable.js/Sortable.js',
    'bower_components/sortable.js/ng-sortable.js',
    'bower_components/angular-filter/dist/angular-filter.js',
    'bower_components/angular-scroll-glue/src/scrollglue.js',
    'app/app.js',
    'app/controllers/*.js',
    'app/services/*.js',
    'app/directives/*.js'
  ])
    .pipe(concat('app.min.js'))
    .pipe(uglify({ mangle: false }))
    .pipe(gulp.dest('app'));
});
```

Το παραπάνω παίρνει όλα τα ξεχωριστά JavaScript αρχεία και δημιουργεί ένα μοναδικό. Τώρα πρέπει και στην index.html, να αφαιρέσουμε όλες τις εισαγωγές των ξεχωριστών αρχείων και να εισάγουμε μόνο το τελικό app.min.js. Παρακάτω, βλέπουμε συγκριτικά το κέρδος που είχαμε από την παραπάνω διαδικασία.

	Αριθμός αρχείων	Αρχικό συνολικό μέγεθος	Τελικό μέγεθος
JavaScript	46	2,12MB	836KB
CSS	11	191KB	7.40KB
Templates	22	138KB	146KB
Σύνολο	77	2.45MB	989.40KB

Πίνακας 5. Optimization Results

Όπως φαίνεται, η διαφορά που πετυχαίνουμε είναι πολύ μεγάλη και σημαντική για την εμπειρία του χρήστη. Αντί για 77 διαφορετικά HTTP Requests, θα γίνουν μόλις 3, ενώ τα 2,45 MB που θα έπρεπε να κατεβάσει ο χρήστης πριν μπορέσει να χειριστεί την εφαρμογή μας έγιναν μόλις 989,40KB.

Κεφάλαιο 3ο

3. Συμπεράσματα

Η εργασία ασχολήθηκε με τις νέες τεχνολογίες στον χώρο του προγραμματισμού στο διαδίκτυο και συγκεκριμένα στις εξελίξεις που υπάρχουν γύρω από τη γλώσσα JavaScript και το MEAN περιβάλλον. Στόχος της ήταν η ανάδειξη των τεχνολογιών που το περιβάλλουν και η αναγνώριση των σημαντικότερων εξ αυτών.

Αρχικά έγινε η ανάλυση των διαφορετικών Frameworks που έχουν δημιουργηθεί και των αναγκών που αυτά καλύπτουν. Αναφέρθηκαν εργαλεία από κάθε στάδιο του προγραμματισμού μίας τέτοιας εφαρμογής καθώς και διάφορα άλλα εργαλεία που συνήθως χρησιμοποιούνται μαζί με αυτά και κάνουν την διαδικασία του Development πιο εύκολη.

Μετά δημιουργήθηκε μία εφαρμογή χρησιμοποιώντας αποκλειστικά τεχνολογίες από αυτές που αναφέρθηκαν στην παραπάνω ανάλυση. Με αυτό, φάνηκε η πρακτική χρήση όλων αυτών που αναλύθηκαν καθώς και τα πραγματικά πλεονεκτήματα τα οποία αυτά προσφέρουν.

Ένας γενικός χαρακτηρισμός για το περιβάλλον MEAN είναι ότι πρόκειται για μία πολύ ενδιαφέρουσα τεχνολογία που θα βοηθούσε πάρα πολύ σε συγκεκριμένου είδους εφαρμογές. Οι δυνατότητες που προσφέρουν τα καινούργια αυτά εργαλεία κάνουν τη δουλειά του προγραμματιστή πολύ πιο εύκολη και τη ροή της εργασίας πολύ πιο σωστή.

Στον αντίποδα, έγινε κατανοητό πως ακόμα αυτές οι τεχνολογίες δεν έχουν ωριμάσει αρκετά και πως έρχονται ακόμα σημαντικές αλλαγές στον χώρο. Οι επιλογές που έχει ο προγραμματιστής είναι πάρα πολλές και αυτό, δίνει μεγάλη ελευθερία στον έμπειρο να επιλέξει τις καταλληλότερες για κάθε περίπτωση αλλά, αντίθετα προκαλεί σύγχυση και προβληματισμό στον αρχάριο.

Το στοιχείο όμως που φάνηκε περισσότερο από όλα ειδικά κατά τη διάρκεια της δημιουργίας της εφαρμογής, είναι η χρήση μίας και μοναδικής γλώσσας, σε όλες τις πλευρές του προγραμματισμού. Η χρήση της, ουσιαστικά, JavaScript τόσο στο Back-End, στο Front-End όσο και στην Database κάνει τον προγραμματισμό πολύ πιο εύκολα κατανοητό αλλά και την όλη διαδικασία πιο εύκολα συντονίσιμη, αφού όσοι θα δουλεύουν πάνω σε ένα τέτοιο έργο θα μπορούν να καταλάβουν όλες τις εκφάνσεις του, χωρίς αναγκαστικά να έχουν δουλέψει πάνω σε όλες αυτές.

Από όλα τα παραπάνω, φαίνεται πως ο χώρος του MEAN θα συνεχίσει να αναπτύσσεται και πως όποιος θέλει να ακολουθήσει τις εξελίξεις, θα πρέπει να ασχοληθεί μαζί του.

Κεφάλαιο 4ο

4. Βιβλιογραφικές Πηγές

1. **Amos Q. Haviv**. MEAN Web Development. Packt Publishing, 2014
2. **Brad Dayley**. Node.js, MongoDB and AngularJS Web Development: The Definitive Guide to Building JavaScript-Based Web Applications from Server to Frontend. Addison Wesley, 2014
3. **Eric Redmond , Jim R. Wilson**. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Pragmatic Bookshelf, 2012
4. **Jeff Dickey**. Write Modern Web Apps with the Mean Stack: Mongo, Express, AngularJS, and Node.js (Develop and Design). Peachpit Press, 2014
5. **Kristina Chodorow**. MongoDB: The Definitive Guide. O'Reilly Media, 2013
6. **Mike Cantelon , Marc Harter , TJ Holowaychuk , Nathan Rajlich**. Node.js in Action. Manning Publications, 2013
7. **Nicholas Cloud , Tim Ambler**. JavaScript Frameworks for Modern Web Dev. Apress, 2015
8. **Pramod J. Sadalage , Martin Fowler**. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison Wesley, 2012
9. **Simon Holmes**. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2015