

Ιόνιο Πανεπιστήμιο

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



-Πτυχιακή Εργασία-

**Ανάπτυξη χρηματιστηριακής εφαρμογής web με τη χρήση των εργαλείων Bootstrap και
NodeJS**

Νικόλαος Παπαιοικονόμου

Επιβλέπων: -Στεφανιδάκης Μιχαήλ-

23 Φεβρουαρίου 2020

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Επίκουρος καθηγητής

Ιόνιο πανεπιστήμιο

Τριμελής επιτροπή

Μιχαήλ Στεφανιδάκης, Επίκουρος καθηγητής

Ιόνιο πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής καθηγητής

Ιόνιο πανεπιστήμιο

Δημήτριος Τσουμάκος, Αναπληρωτής καθηγητής

Ιόνιο πανεπιστήμιο

Στους γονείς μου Θανάση & Κατερίνα και στην αδερφή μου Ηλιάνα

ΠΕΡΙΛΗΨΗ

Η ανάπτυξη των διαδικτυακών εφαρμογών ήταν ραγδαία από τη στιγμή που παρουσιάστηκε ο παγκόσμιος ιστός. Παλιότερα οι προγραμματιστές έπρεπε να αναπτύξουν ξεχωριστές εφαρμογές για το κάθε λειτουργικό σύστημα και να τις εγκαταστήσουν τοπικά στους υπολογιστές τους για να μπορέσουν να τις χρησιμοποιήσουν. Αντίθετα, οι διαδικτυακές εφαρμογές είναι εύκολα προσβάσιμες από το πρόγραμμα περιήγησης ανεξάρτητα με πιο λειτουργικό σύστημα χρησιμοποιεί ο χρήστης. Οι διαδικτυακές εφαρμογές έχουν κάνει σημαντική πρόοδο σε σύγκριση με τις εφαρμογές που λειτουργούν τοπικά, από στατικές ιστοσελίδες σε μοντέρνες, διαδραστικές και έξυπνες εφαρμογές διαδικτύου.

Η ανάπτυξη μια διαδικτυακής εφαρμογής είναι ο συνδυασμός της ανάπτυξης του μπροστινού μέρους (front-end) και του οπίσθιου μέρους (back-end). Υπάρχουν πολλές προγραμματιστικές γλώσσες και πλαίσια που μπορούμε να διαλέξουμε για αναπτύξουμε μια διαδικτυακή εφαρμογή και ο μεγάλος αυτός όγκος εργαλείων είναι λογικό να δημιουργεί σύγχυση σε ένα αρχάριο προγραμματιστή. Για τον λόγο αυτό, στην παρούσα πτυχιακή θα αναλύσουμε κάποια από αυτά τα εργαλεία και πλαίσια που μπορούμε να χρησιμοποιήσουμε.

Το δεύτερο μισό της εργασίας έχει αφιερωθεί στην παρουσίαση μιας σειράς βημάτων για την ανάπτυξη μιας διαδικτυακής εφαρμογής με τη βοήθεια των εργαλείων όπως αναλύθηκαν στο πρώτο κομμάτι.. Μέσα από αυτή τη διαδικασία θα κατανοήσουμε τις βασικές αρχές του server, του client και της βάσης δεδομένων, αλλά και του τρόπου που επικοινωνούν μεταξύ τους. Για την κατασκευή θα χρησιμοποιήσουμε τα βασικά εργαλεία: HTML, CSS, JAVASCRIPT, NODEJS, BOOTSTRAP καθώς και πολλά άλλα που θα αναπτύξουμε παρακάτω.

Περιεχόμενα

Κεφάλαιο Α΄	1
Α.1 Εισαγωγή.....	1
Α.2 Σκοπός της εργασίας.....	1
Κεφάλαιο Β΄	2
Β.1 Εισαγωγή στο χρηματιστήριο.....	2
Β.2 Παγκόσμιος Ιστός και χρηματιστήριο.....	3
Κεφάλαιο Γ΄	4
Γ.1 JavaScript.....	4
Γ.1.1 Εξέλιξη της JavaScript τα τελευταία χρόνια.....	4
Γ.1.2 Εκδόχές της JavaScript.....	7
Γ.1.2.1 Πριν την ES6.....	8
Γ.1.2.2 ES6.....	9
Γ.1.2.3 Μετά την ES6.....	11
Γ.2 Back end.....	12
Γ.2.1 NodeJS & Express.....	12
Γ.2.1.1 Εισαγωγή στο NodeJS.....	12
Γ.2.1.2 Εισαγωγή στο Express.....	14
Γ.2.1.2.1 Απλότητα.....	15
Γ.2.1.2.2 Ευελιξία.....	15
Γ.2.1.2.3 Πλαίσιο Διαδικτυακών Εφαρμογών.....	15
Γ.2.1.2.4 Single-page Διαδικτυακές εφαρμογές.....	16
Γ.2.1.2.5 Multipage και Hybrid Διαδικτυακές Εφαρμογές.....	16
Γ.2.1.2.6 Μια σύντομη ιστορική αναδρομή για το Express.....	17
Γ.2.1.2.7 Η αναβάθμιση στο Express 4.0.....	17
Γ.2.1.2.8 Η αναβάθμιση στο Express 5.0.....	17
Γ.2.2 MongoDB & Mongoose.....	18
Γ.2.2.1 Εισαγωγή στο MongoDB.....	18
Γ.2.2.2 Εισαγωγή στο Mongoose.....	20
Γ.2.3 Εισαγωγή στο PassportJS.....	21
Γ.2.4 Εισαγωγή στο Socket.io.....	22
Γ.3 Front end.....	23
Γ.3.1 Εισαγωγή στην HTML.....	23
Γ.3.2 Εισαγωγή στην CSS.....	26
Γ.3.3 Εισαγωγή στο Bootstrap.....	27
Κεφάλαιο Δ΄	28
Δ.1 Σχεδίαση και ανάπτυξη εφαρμογής.....	28
Δ.1.1 Εισαγωγή στην εφαρμογή.....	28
Δ.1.2 Αρχιτεκτονική Εφαρμογής.....	29
Δ.1.2.1 Ορισμός Αρχιτεκτονικής Διαδικτυακών Εφαρμογών.....	30
Δ.1.2.1 Πως λειτουργεί.....	31
Δ.1.2.2 Συστατικά Διαδικτυακής Εφαρμογής.....	31
Δ.1.2.3 Αρχιτεκτονική της Διαδικτυακής Εφαρμογής Nstocks.....	32
Δ.1.3 Εγκατάσταση προγραμματιστικού περιβάλλοντος.....	35

Δ.1.3.1 Git.....	36
Δ.1.3.1.1 Εγκατάσταση Git.....	37
Δ.1.3.2 NodeJS & NPM.....	38
Δ.1.3.2.1 Εγκατάσταση NodeJS & NPM.....	38
Δ.1.3.3 GitHub.....	40
Δ.1.3.3.1 Αποθετήριο.....	40
Δ.1.3.4 Heroku.....	44
Δ.1.3.4.1 Σύνδεση με το Heroku.....	44
Δ.1.3.5 MongoDB.....	46
Δ.1.3.5.1 Σύνδεση με το MongoDB.....	47
Δ.1.3.6 Packages.....	52
Δ.1.3.6.1 Εγκατάσταση πακέτων.....	54
Δ.1.3 Αναλυτική παρουσίαση κώδικα.....	57
Δ.1.3.1 Δημιουργία server.....	57
Δ.1.3.2 Σύνδεση με Βάση Δεδομένων.....	59
Δ.1.3.3 Δημιουργία διαδρομών (routes).....	61
Δ.1.3.4 Εγγραφή χρήστη (Register).....	62
Δ.1.3.5 Πιστοποίηση χρήστη (Login).....	68
Δ.1.3.6 Διαχείριση δεδομένων API.....	74
Δ.1.3.7 Οπτικοποίηση δεδομένων (Chart.js).....	82
Δ.1.3.8 Κατασκευή σελίδας Stocks.ejs.....	87
Δ.1.3.8 Αποθήκευση Μετοχών στη Βάση Δεδομένων.....	98
Δ.1.3.9 Δημιουργία σελίδας userpage.....	101
Δ.1.3.10 Δημιουργία λίστας μετοχών.....	105
Δ.1.3.10 Ορισμός ανώτερου, κατώτερου ορίου τιμής μετοχών.....	111
Δ.1.3.11 Κατασκευή μπάρας πλοήγησης.....	121
Δ.1.3.8 Αλγόριθμοι συμπίεσης gzip και brotli.....	132
Κεφάλαιο Ε'	135
Ε.1 Στιγμιότυπα Διαδικτυακής Εφαρμογής.....	135
Ε.2 Μελλοντικές Βελτιώσεις.....	136
Ε.2 Συμπεράσματα.....	137
Βιβλιογραφία – Δικτυογραφία.....	137

Κεφάλαιο Α΄

A.1 Εισαγωγή

Η ανάπτυξη των διαδικτυακών εφαρμογών ήταν ραγδαία από τη στιγμή που παρουσιάστηκε ο παγκόσμιος ιστός. Παλαιότερα, οι προγραμματιστές έπρεπε να αναπτύξουν ξεχωριστές εφαρμογές για το κάθε λειτουργικό σύστημα και να τις εγκαταστήσουν τοπικά στους υπολογιστές τους προκειμένου να τις χρησιμοποιήσουν.

A.2 Σκοπός της εργασίας

Η παρακάτω πτυχιακή παρουσιάζει τα στάδια που πρέπει να ακολουθήσει κανείς για την ανάπτυξη μιας διαδικτυακής εφαρμογής χρηματιστηριακού περιεχομένου. Με τη χρήση κάποιων από τα πιο σύγχρονα εργαλεία ανάπτυξης frontend αλλά και backend θα παρουσιαστεί με απλό τρόπο, πως όλα αυτά μπορούν να συνδυαστούν αρμονικά για να δημιουργήσουμε μια πλήρως λειτουργική διαδικτυακή εφαρμογή.

Θα αναλυθούν λεπτομερώς τα εργαλεία που θα χρησιμοποιηθούν αλλά και γιατί είναι κατάλληλα για τη συγκεκριμένη εφαρμογή. Μέσα από αυτή τη πτυχιακή ο αναγνώστης θα μπορέσει να έχει μια πλήρη εικόνα για το πως να σχεδιάσει, να αναπτύξει και έπειτα να δημοσιεύσει τη διαδικτυακή του εφαρμογή στο παγκόσμιο ιστό.

Κεφάλαιο Β΄

B.1 Εισαγωγή στο χρηματιστήριο

Η παρούσα διαδικτυακή εφαρμογή φτιάχτηκε για να δίνει πληροφορίες στο χρήστη σχετικά με τις μετοχές που τον ενδιαφέρουν, αλλά και να τις παρακολουθεί στενά για σημαντικές αλλαγές, οπότε δεν θα μπορούσε να παραλειφθεί μια σύντομη αναφορά για το τι είναι το χρηματιστήριο.

Το χρηματιστήριο είναι μια συλλογή αγορών και συναλλαγών όπου γίνονται συνεχείς δραστηριότητες αγοράς, πώλησης και έκδοσης μετοχών δημοσίων εταιριών. Αυτές οι χρηματοοικονομικές δραστηριότητες πραγματοποιούνται μέσα από θεσμοθετημένες επίσημες ανταλλαγές ή έξω-χρηματιστηριακές αγορές (OTC) που λειτουργούν σύμφωνα με ένα καθορισμένο σύνολο κανονισμών. Τα κορυφαία χρηματιστήρια στις ΗΠΑ είναι το χρηματιστήριο της Νέας Υόρκης (NYSE), η Nasdaq και το Chicago Board Options Exchange (CBOE). Αυτά, μαζί με πολλά άλλα χρηματιστήρια (stock exchanges) που λειτουργούν στη χώρα, αποτελούν το χρηματιστήριο (stock market) των ΗΠΑ. Σε αυτό το σημείο είναι σημαντικό να αναφέρουμε ότι παρόλο που το stock market και το stock exchange χρησιμοποιούνται σαν να είναι το ίδιο, στη πραγματικότητα το δεύτερο είναι υποσύνολο του πρώτου.

Αν και ονομάζεται χρηματιστηριακή αγορά ή μετοχική αγορά και είναι κυρίως γνωστή για τη διαπραγμάτευση μετοχών, διαπραγματεύονται επίσης και άλλοι χρηματοοικονομικοί τίτλοι σε αυτήν, όπως χρηματιστηριακά κεφάλαια (ETF), εταιρικά ομόλογα και παράγωγα που βασίζονται σε μετοχές, εμπορεύματα, νομίσματα και ομόλογα στις χρηματιστηριακές αγορές.

B.2 Παγκόσμιος Ιστός και χρηματιστήριο

Ο παγκόσμιος ιστός ήταν μία από τις πιο επαναστατικές τεχνολογίες που δημιούργησε ποτέ ο άνθρωπος, αλλάζοντας ριζικά τη ζωή του. Επηρέασε σημαντικά το τρόπο με τον οποίο οι καταναλωτές ακούνε μουσική, παρακολουθούν ταινίες, αγοράζουν και πωλούν προϊόντα και επικοινωνούν. Είχε επίσης εξαιρετικά ευεργετικό αντίκτυπο στις επενδύσεις, ειδικά στους ιδιώτες επενδυτές.

Η ευρεία διαθεσιμότητα πληροφοριών ήταν το μεγαλύτερο όφελος που παρείχε το διαδίκτυο στις επενδύσεις. Πριν το διαδίκτυο ο επενδυτής έπρεπε να πάει στη βιβλιοθήκη να διαβάσει τη χρηματοοικονομική βιβλιογραφία και να ερευνήσει εταιρίες, μετοχές, ομόλογα και αμοιβαία κεφάλαια. Πλέον το μόνο που έχει να κάνει είναι απλά να μπει στο πρόγραμμα περιήγησης και να πλοηγηθεί σε μια από τις πολλές διαδικτυακές εφαρμογές για να ενημερωθεί για ότι θέλει. Με τη χρήση διαδικτυακών εφαρμογών ή και απλών εφαρμογών ο επενδυτής μπορεί εύκολα και πολύ γρήγορα να βρει μια ηλεκτρονική αναφορά της εταιρίας που τον ενδιαφέρει, μπορεί να κατεβάσει μεγάλα οικονομικά έγγραφα και να αναζητήσει βασικές λέξεις, θέμα ή και συγκεκριμένες οικονομικές καταστάσεις. Ένα άλλο μεγάλο πλεονέκτημα είναι η ταχύτητα που μπορεί να μάθει ένας επενδυτής τις αλλαγές στη τιμή μιας μετοχής και αντίστοιχα να αγοράσει και να πουλήσει. Υπάρχουν πολλές εφαρμογές στο διαδίκτυο που προσφέρουν όλα τα απαραίτητα εργαλεία σε ένα επενδυτή για να μπορέσει να είναι όσο πιο ευέλικτος και αποδοτικός γίνεται.

Εν κατακλείδι, το διαδίκτυο έχει δώσει σημαντική δύναμη στα χέρια των ατόμων, και αυτό έχει μια βαθιά επίδραση στο τρόπο με τον οποίο ο επενδυτής αποκτά οικονομικές πληροφορίες. Εξίσου σημαντικό είναι ότι έχει μειώσει σημαντικά το κόστος για τους περισσότερους συμμετέχοντες στη χρηματοοικονομική αγορά, μειώνοντας σημαντικά τα τέλη. Ειδικότερα, οι επενδυτές λιανικής είχαν δραματική μείωση των επιτοκίων προμηθειών που πληρώνουν σε κινητές αξίες.

Κεφάλαιο Γ΄

Γ.1 JavaScript

Γ.1.1 Εξέλιξη της JavaScript τα τελευταία χρόνια

Προτού αναλυθούν λεπτομερώς τα εργαλεία που χρησιμοποιήθηκαν σε αυτή τη πτυχιακή δεν θα μπορούσε να παραληφθεί μια αναφορά στην εξέλιξη της βασικής γλώσσας που χρησιμοποιήθηκε για την ανάπτυξη αυτής της διαδικτυακής εφαρμογής (web application). Η εξέλιξη της JavaScript είναι πραγματικά με το μέρος μας. Από τις αρχές αυτής της γλώσσας όπου χρησιμοποιούταν σαν client side scripting γλώσσα, όχι μόνο έχει γίνει πανταχού παρούσα στην πλευρά του πελάτη (client side), αλλά επεκτάθηκε και ως γλώσσα διακομιστή (server side) χάρις στο Node.

Η φιλοδοξία να χρησιμοποιείται μια γλώσσα μόνο, πλέον είναι πιο ξεκάθαρη από ποτέ με αποτέλεσμα να διευκολύνει το προγραμματιστή γιατί δεν χρειάζεται να γνωρίζει παραπάνω γλώσσες προγραμματισμού παρά μόνο τη JavaScript. Πιο συγκριμένα δεν χρειάζεται πλέον να αλλάζει ο προγραμματιστής μεταξύ διαφορετικών γλωσσών όπως από JavaScript σε PHP, C #, Ruby ή Python (ή οποιαδήποτε άλλη γλώσσα διακομιστή). Επιπλέον, η JavaScript παρέχει τη δυνατότητα στους προγραμματιστές του client side να κάνουν το άλμα στο προγραμματισμό του server side. Αυτό δεν σημαίνει όμως ότι, για να μπορέσει κάποιος ο οποίος θέλει να ασχοληθεί με την ανάπτυξη server side εφαρμογών, αρκεί να γνωρίζει μόνο τη JavaScript. Υπάρχουν ακόμα πολλά κομμάτια που ένας νέος προγραμματιστής πρέπει να καλύψει για να μπορέσει να ασχοληθεί και με αυτό το κομμάτι.

Με τη Javascript τουλάχιστον αυτό δεν θα είναι εμπόδιο. Ένας μηχανικός λογισμικού μπορεί να καταλάβει ότι πολλές γλώσσες προγραμματισμού, πλαίσια (frameworks), και API ήταν κατά

καιρούς στη μόδα και χρησιμοποιούνταν από το ευρύτερο κοινό. Κάποιες στο πέρασμα του χρόνου χάθηκαν άλλες αντικαταστάθηκαν από καινούργιες τεχνολογίες και άλλες υπάρχουν ακόμα αναλλοίωτες στο χρόνο. Αυτό από τη μία δημιουργεί μια αίσθηση υπερηφάνειας καθώς όσο ασχολούμαστε με περισσότερες γλώσσες και εργαλεία τόσο διευρύνουμε τις γνώσεις μας, από την άλλη όμως δημιουργεί μια κούραση γιατί πρέπει συνέχεια να προσαρμοζόμαστε στα νέα δεδομένα. Κάποιες φορές θέλουμε να κάνουμε κάτι χωρίς να χρειάζεται να μάθουμε μια νέα τεχνολογία ή δεξιότητες που δεν έχουμε.

Η JavaScript μπορεί να φαινόταν στην αρχή ένας απίθανος πρωταθλητής και μπορεί παλιότερα να υπήρχε η προκατάληψη ότι η JavaScript ήταν μια απλή γλώσσα έτσι ώστε ένας ερασιτέχνης να μπορέσει να φτιάξει κάτι απλό. Η αλήθεια είναι πως, όντως η JavaScript έριξε το πήχη για τους ερασιτέχνες, και αυτός ήταν ο λόγος που δεν βοήθησε τη φήμη της. Είναι ατυχές που οι άνθρωποι υποφέρουν από αυτή τη προκατάληψη, η οποία εμποδίζει πολλούς στο να ανακαλύψουν τελικά πόσο ισχυρή, ευέλικτη και κομψή είναι. Πολλοί άνθρωποι τώρα αρχίζουν να λαμβάνουν σοβαρά υπόψη τους τη JavaScript, παρόλο που η γλώσσα όπως ξέρουμε δημιουργήθηκε γύρω στο 1996 (αν και πολλά από τα πιο ελκυστικά της χαρακτηριστικά προστέθηκαν το 2005). Το 2009, αφότου πέρασαν τόσα χρόνια οι άνθρωποι άρχισαν να αντιλαμβάνονται τη δύναμη και την εκφραστικότητα της σαν μια browser scripting γλώσσα.

Ο Ryan Dahl είδε τις προοπτικές της JavaScript και τότε δημιουργήθηκε το Node. Αυτή ήταν μια ιστορική μέρα για το χώρο της διαδικτυακής τεχνολογίας. Ο David Heinemeier Hansson ο οποίος δημιούργησε τη Ruby on Rails πήρε κάποιες έξυπνες ιδέες από την ακαδημαϊκή του καριέρα, στην επιστήμη υπολογιστών, τα συνδύασε με κάποιες νέες ιδέες, και έδειξε στο κόσμο ένα πιο γρήγορο τρόπο να φτιάχνεις διαδικτυακές ιστοσελίδες και διαδικτυακές εφαρμογές.

Η Microsoft, σε μια γενναία προσπάθεια να γίνει σχετική στην εποχή του διαδικτύου, έκανε εκπληκτικά πράγματα με το .NET και έμαθε όχι μόνο από τη Ruby και την JavaScript, αλλά και από τα λάθη της Java, καθώς δανείστηκε πολλά στοιχεία από τις αίθουσες της ακαδημαϊκής κοινότητας. Είναι μια συναρπαστική στιγμή να συμμετέχει κανείς στη τεχνολογία διαδικτύου. Παντού υπάρχουν εκπληκτικές ιδέες (ή εκπληκτικές παλιότερες ιδέες). Το πνεύμα της καινοτομίας και του ενθουσιασμού είναι μεγαλύτερο από ότι τα προηγούμενα χρόνια.

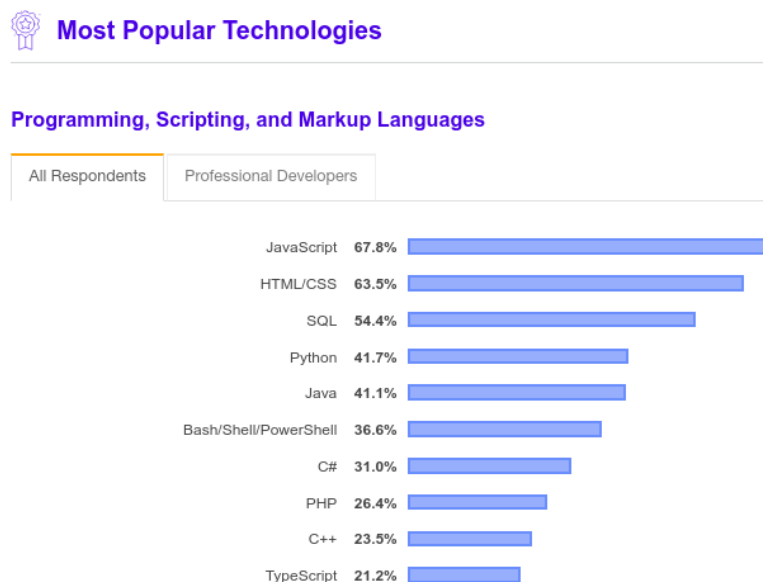
Μετά από αυτή τη σύντομη ιστορική αναδρομή θα αναλυθεί τώρα, πιο συγκεκριμένα η γλώσσα JavaScript. Η JavaScript είναι μια δυναμική,αντικειμενοστραφής γλώσσα προγραμματισμού με συναρτήσεις πρώτης κατηγορίας που βασίζονται σε αντικείμενα που μπορούν να επιστρέψουν μια τιμή μεταβιβάζοντας μια ίδια συνάρτηση σε άλλες συναρτήσεις ως όρισμα.

Η JavaScript δημιουργήθηκε για πρώτη φορά το 1995 από τον Brendan Eich σε μια εταιρία που ονομάζεται Netspace για να χρησιμοποιείται σαν scripting language για τη Java. Η JavaScript όπως αναφέρθηκε και πιο πάνω εξελίχθηκε και μεταπήδησε και σε άλλους κλάδους όπως στο server side, στις βάσεις δεδομένων αλλά και στη ρομποτική. Η κοινή χρήση κώδικα και η επαναχρησιμοποίηση του συνέβαλαν σημαντικά στη μείωση του αριθμού των γραμμών κώδικα μέχρι και 40% με τη κατανομή των τμημάτων κώδικα ή ακόμη και τη κοινή χρήση προτύπων,βιβλιοθηκών και μοντέλων.

Η JavaScript κάνει τη συντήρηση του κώδικα μια πολύ εύκολη και οικονομική διαδικασία. Επιπλέον η χρήση μόνο μιας γλώσσας συμβάλλει στην καλύτερη αποτελεσματικότητα και συνεννόηση μεταξύ των ομάδων ανάπτυξης αλλά και στη μείωση του χάσματος μεταξύ αυτών,αφού όλες οι ομάδες που εργάζονται πάνω σε μια εφαρμογή έχουν καλύτερη κατανόηση του τι κάνει ο καθένας. Επιπλέον, οι προγραμματιστές δεν χρειάζονται να ανησυχούν σχετικά με συντακτικές διαφορές όπως με τις διαφορετικές γλώσσες που χρησιμοποιούνται σε μια εφαρμογή. Μάλιστα η JavaScript αυξάνει την απόδοση αλλά και τη ταχύτητα μιας εφαρμογής και αυτός ήταν ο κύριος λόγος που η PayPal μια από τις πιο γνωστές εταιρίες αποφάσισε να μετακινηθεί από τη Java στη χρήση του NodeJS ως πλατφόρμα διακομιστή (server) για την εφαρμογή τους και μάλιστα δημοσίευσε μια αναφορά σχετικά με το τρόπο με τον οποίο η JavaScript και NodeJS τους βοήθησαν να αυξήσουν την απόδοση τους αλλά και πόσο σημαντικά μειώθηκαν οι γραμμές κώδικα τους. Σύμφωνα με την αναφορά, η εφαρμογή που δημιουργήθηκε με JavaScript έγινε δύο φορές πιο γρήγορη από αυτή που είχε φτιαχτεί με τη Java αλλά και με τη βοήθεια λιγότερων ανθρώπων.

Συγκεκριμένα δήλωσαν ότι η εφαρμογή που δημιουργήθηκε με τη γλώσσα JavaScript αποτελούνταν από 33% λιγότερες γραμμές κώδικα και με 40% λιγότερα αρχεία παρόλο που και οι δύο εφαρμογές αποτελούνταν από τις ίδιες λειτουργίες. Ομοίως, ο αριθμός αιτημάτων ανά

δευτερόλεπτο διπλασιάστηκε και ο μέσος χρόνος απόκρισης ελαχιστοποιήθηκε κατά 35%. Παρόλα τα πλεονεκτήματα της πλήρους στοίβας ανάπτυξης με JavaScript (Fullstack JavaScript development), υπάρχουν και μερικά μειονεκτήματα, τα οποία όμως δεν εμποδίζουν τη γλώσσα από το να θεωρείται μια από τις πιο αγαπητές που υπάρχουν αυτή τη στιγμή. Όπως φαίνεται και από την ετήσια επισκόπηση που κάνει το Stack Overflow:



Εικόνα 1: Stack Overflow Developer Survey 2019

Γ.1.2 Εκδοχές της JavaScript

Όπως έχει αναφερθεί πιο πάνω, η JavaScript σαν γλώσσα δημιουργήθηκε το 1996 από την εταιρία Netspace και την ECMA (International European Association for Standardizing Information and Communication), η οποία καθορίζει τα πρότυπα στο τομέα της πληροφορικής και των ICT. Η ECMA είναι αυτή που συνεργάζεται διεθνώς για να κάνει τη JavaScript μια σύγχρονη και σχετική με τα τεχνολογικά πρότυπα γλώσσα. Έκτοτε, διάφορες εκδοχές ECMAScript έχουν δημιουργηθεί, από τις οποίες η EcmaScript 6 (ES6) είναι η σημαντικότερη από αυτές. Μετά από τη μεγαλύτερη τους δημοσίευση που έγινε το 2015, η ECMA δεσμεύτηκε να εκδίδει καινούργιες εκδόσεις κάθε χρόνο.

Ως εκ τούτου, η ECMA άλλαξε το όνομα για το ECMAScript από αριθμό έκδοσης σε αριθμό χρονολογίας έκδοσης. Στο παρακάτω πίνακα παρουσιάζονται λεπτομερώς οι εκδόσεις της JavaScript που βγήκαν καθώς και η ημερομηνίες στις οποίες εκδόθηκαν. Μετά το 2011, οι επόμενες εκδόσεις της JavaScript, άλλαξαν το όνομα τους και χρησιμοποιούσαν την ημερομηνία που κυκλοφόρησαν.

Πίνακας 1: Εκδόσεις ECMA και οι ημερομηνίες έκδοσης τους

Έκδοση	Όνομα	Ημερομηνία Έκδοσης
ES1	ES1	Ιούνιος 1997
ES2	ES2	Ιούνιος 1998
ES3	ES3	Δεκέμβριος 1999
ES4	ES4	Ιούλιος 2008
ES5	ES5	Δεκέμβριος 2009
ES5.1	ES5.1	Ιούνιος 2011
ES6	ES 2015	Ιούνιος 2015
ES7	ES 2016	Ιούνιος 2016
ES8	ES 2017	Ιούνιος 2017
ES9	ES 2018	Ιούνιος 2018
ES10	ES 2019	Ιούνιος 2019

Γ.1.2.1 Πριν την ES6

Η πρώτη έκδοση της JavaScript κυκλοφόρησε τον Ιούνιο του 1997 από την ECMA με το όνομα ECMAScript 1. Έπειτα, τον Ιούνιο του 1998 κυκλοφόρησε η καινούργια έκδοση ECMAScript 2 η οποία περιείχε τις δευτερεύουσες αλλαγές ώστε να ακολουθήσει το ISO πρότυπο για τη JavaScript. Ωστόσο στην τρίτη έκδοση ECMAScript 3 η οποία εκδόθηκε το 1999 παρουσιάστηκε ένας αριθμός νέων χαρακτηριστικών. Τα χαρακτηριστικά αυτά ήταν πολύ σημαντικά για τα εκάστοτε δεδομένα όπου μερικά από αυτά ήταν: η επανάληψη do-while και οι λειτουργίες χειρισμού συμβολοσειρών, όπως η συναρμολόγηση, η αντικατάσταση, η αντιστοίχιση, η διάσπαση και η σύνδεση χρησιμοποιώντας κανονικές εκφράσεις (regular expressions). Ο χειρισμός σφαλμάτων με το try/catch και πολλές άλλες δευτερεύουσες αλλαγές, σε συνδυασμό με τα παραπάνω χαρακτηριστικά οδήγησαν στην αναβάθμιση της γλώσσας.

Είναι σημαντικό να αναφερθεί ότι η έκδοση ECMAScript 4 δεν εκδόθηκε ποτέ. Περιλάμβανε διάφορα νέα χαρακτηριστικά όπου μερικά από αυτά ήταν οι τάξεις, οι διεπαφές, τα getters και τα setters καθώς και πολλά άλλα. Ο πρωταρχικός σκοπός αυτής της έκδοσης ήταν να διαμορφωθεί η γλώσσα κατάλληλα ώστε να έχει την ικανότητα να μπορεί να αναπτυχθεί με αυτήν, λογισμικό μεγάλης κλίμακας. Ωστόσο λόγω της διαφωνία μεταξύ των δυο ομάδων που συνεργάστηκαν για την αναβάθμιση της γλώσσας, το ES4 δεν εκδόθηκε ποτέ. Αντ'αυτού κατέληξαν στην έκδοση μια κλιμακωτής ενημέρωσης, την ECMAScript 5.

Η πέμπτη έκδοση του ECMAScript δημοσιεύτηκε το 2009. Κάποια από τα χαρακτηριστικά της ES5 ήταν η υποστήριξη αυστηρού τρόπου λειτουργίας που βοηθούσε να γράφει ο προγραμματιστής καθαρότερο κώδικα, επίσης περιλάμβανε διάφορες μεθόδους αναζήτησης και χειρισμού πινάκων όπως το map, το reduce, το filter, το indexOf και το forEach καθώς και το JSON. Το JSON χρησιμοποιείται για την αποθήκευση και τη μεταφορά δεδομένων ως ζεύγη όνομα/ τιμή το οποίο μάλιστα είναι η πιο δημοφιλής μορφή αποθήκευσης δεδομένων μέχρι σήμερα.

Γ.1.2.2 ES6

Το ES6, που όπως αναφέραμε και πιο πάνω ήταν και γνωστό σαν ECMAScript 2015, είναι το όνομα που χρησιμοποιήθηκε για τη έκδοση της JavaScript από τότε που ο όρος JavaScript έγινε εμπορικό

σήμα της Oracle. Η ES6, η οποία όπως αναφέραμε εκδόθηκε το 2015, είναι η σημαντικότερη αναβάθμιση μετά την έκδοση το 1999. Το ES6 κυκλοφόρησε με ένα νέο στόχο να υποστηρίζει τη σύνταξη σύνθετων εφαρμογών, βιβλιοθηκών καθώς και γεννήτριες κώδικα. Σαν αποτέλεσμα είχε η καινούργια έκδοση ES6 να φέρει νέα χαρακτηριστικά και λειτουργίες όπως τα modules, τις κλάσεις και τις arrow functions.

Παρόλο που η ES6 εκδόθηκε το 2015 τα περισσότερα προγράμματα περιήγησης δεν μπορούσαν να την υποστηρίξουν. Έτσι με τη χρήση transpilers μεταφερόταν ο κώδικας της JavaScript από την έκδοση ES6 στην ES5. Το Babel αλλά και το Google Traceur ήταν τα πιο δημοφιλή transpiler που χρησιμοποιήθηκαν από το ευρύτερο κοινό. Παρόλα αυτά πλέον τα προγράμματα περιήγησης υποστηρίζουν την ES6 οπότε η χρήση transpilers πλέον δεν έχει νόημα.

Παρακάτω θα συζητήσουμε μερικά από τα νέα χαρακτηριστικά που εμπλουτίστηκαν στη JavaScript μετά την έκδοση της ES6. Με τη έκδοση αυτή προστέθηκαν διάφορες λέξεις κλειδιά όπως η let και η const οι οποίες ορίζουν μια μεταβλητή. Παλιότερα υπήρχε μόνο η var. Ας εξηγήσουμε το let και το const μιας και είναι πολύ σημαντικές λειτουργίες. Η let μεταβλητή μπορεί να οριστεί μέσα σε μια συνάρτηση αλλά δεν είναι δυνατή η πρόσβαση της έξω από αυτή τη συνάρτηση. Αντίθετα με το let, η const μεταβλητή δεν μπορεί δηλωθεί ξανά με το ίδιο όνομα. Οι μεταβλητές που δηλώνονται με το const δεν είναι αμετάβλητες όμως, οι τιμές τους μπορούν να αλλάξουν εκτελώντας λειτουργίες της JavaScript αν αυτές είναι αντικείμενα.

Ας μιλήσουμε τώρα για τις arrow functions οι οποίες είναι ένα εξίσου σημαντικό χαρακτηριστικό το οποίο προστέθηκε με την ES6. Οι arrow functions είναι σύντομες κλήσεις συναρτήσεων. Παραλείπει την ανάγκη γραφής της ονομασίας μια συνάρτησης. Επίσης δεν δημιουργεί το δικό της περιβάλλον αλλά το κληρονομεί από τη λειτουργία στην οποία είναι μέσα. Αυτό βοηθά στην επίλυση του προβλήματος με το this της JavaScript. Ομοίως, οι προεπιλεγμένες τιμές μπορούν να μεταβιβαστούν σε μια συνάρτηση όταν οι παράμετροι δεν παρέχονται στη κλήση της συνάρτησης, αρχικοποιώντας την παράμετρο με μια προεπιλεγμένη τιμή. Αυτή η λειτουργία υπάρχει ήδη στη PHP και στη Java.

Γ.1.2.3 Μετά την ES6

Η ES6 εκδόθηκε μετά από 4 χρόνια και υπήρχαν πολλές νέες προσθήκες στη γλώσσα. Παρόλα αυτά η επιτροπή της ECMA κατάλαβε ότι δεν μπορεί να δημοσιεύει μια τέτοια έκδοση μέσα σε ένα τόσο μεγάλο χρονικό διάστημα οπότε αποφάσισαν να εκδίδουν κάθε χρόνο και από μια νεότερη έκδοση με όχι τόσο σημαντικές αλλαγές για τη γλώσσα.

Το 2016 λοιπόν εμφανίζεται η ES7 με τρεις αλλαγές από αυτή της 2016 έκδοσης της ECMAScript. Αρχικά, προστέθηκε ένας νέος χειριστής με τη σύνταξη (**), ο οποίος είναι μια συντόμευση για τη λήψη της εκθετικής τιμής. Εκτελεί ακριβώς την ίδια λειτουργία με το Math.pow αλλά με πολύ λιγότερο κώδικα. Δεύτερον, εισάχθηκε η εντολή Array.prototype.includes η οποία επιστρέφει μια Boolean τιμή όταν ελέγχει αν ένα στοιχείο υπάρχει ή όχι στο πίνακα. Τέλος το function-scoped strict mode άλλαξε. Η χρήση των use strict σε συναρτήσεις με αποδομημένες (destructured) παραμέτρους ή συναρτήσεις με προεπιλεγμένες παραμέτρους προκαλεί συντακτικά σφάλματα αφού οι τιμές των προεπιλεγμένων παραμέτρων μπορούν επίσης να είναι και αυτές συναρτήσεις.

Μετά την ES7 εκδόθηκε η νέα έκδοση ES8 η οποία είναι γνωστή ως ECMAScript 2017. Περιλάμβανε τις λειτουργίες async, την κοινή μνήμη (shared memory) και τα ατομικά (atomics), τα trailing commas σε λίστες και κλήσεις παραμέτρων σε συναρτήσεις, επίσης πρόσθεσε το string padding και τις μεθόδους όπως object.values, object.entries, string.prototype.padStart, string.prototype.padEnd και object.getOwnPropertyDescriptors.

Η ES9 ή ECMAScript 2018 εκδόθηκε τον Ιούνιο του 2018. Υποστηρίζει την ασύγχρονη (async) επανάληψη ενός iterable object όπως πίνακες (arrays) και σειρές γραμμάτων (strings). Πλέον οι υποσχέσεις (promises) και οι generator συναρτήσεις μπορούν να επαναληφθούν χρησιμοποιώντας τον ασύγχρονο iterator. Τέλος πρόσθεσε τις νέες λειτουργίες όπως το dotAll flag, look-behind assertions, Unicode capture groups και named capture groups. Επιπρόσθετα η ES9 πρόσθεσε τη μέθοδο .finally() στις υποσχέσεις (promises). Αυτή είναι μια callback συνάρτηση η οποία εκτελείται κάθε φορά που η υπόσχεση είτε επιλύεται είτε απορρίπτεται.

Η τελευταία έκδοση που έχει βγει μέχρι και τώρα που συγγράφεται αυτή η πτυχιακή είναι η ES10 που ονομάζεται ECMAScript 2019 και η οποία εκδόθηκε τον Ιούνιο του 2019. Τα χαρακτηριστικά που πρόσθεσε στη γλώσσα Javascript είναι το `Array.prototype.flat` και το `Array.prototype.flatMap`, καθώς και τις αλλαγές στο `Array.sort` και `Object.fromEntries`.

Γ.2 Back end

Γ.2.1 NodeJS & Express

Γ.2.1.1 Εισαγωγή στο NodeJS

Ας μιλήσουμε τώρα για ένα βασικό εργαλείο που χρησιμοποιήθηκε για τη δημιουργία αυτής της διαδικτυακής εφαρμογής, το NodeJS. Κατά κάποιο τρόπο το Node έχει πολλά κοινά με τους άλλους γνωστούς διακομιστές ιστού (web servers), όπως αυτός της Microsoft Internet Information Services (IIS) ή το Apache. Το ενδιαφέρον όμως που μπορούμε να παρατηρήσουμε είναι το πως διαφέρει από αυτούς, οπότε ας ξεκινήσουμε αναλύοντας γιατί διαφέρει και γιατί αυτό είναι κάτι θετικό. Η προσέγγιση του Node στους διακομιστές δικτύου (web servers) είναι μηδαμινή. Σε αντίθεση με το IIS ή το Apache, όπου κάποιος πρέπει να ξοδέψει πολύ χρόνο για να καταφέρει να μάθει καλά αυτά τα εργαλεία, το NodeJS είναι πολύ εύκολο να το χειριστεί κάποιος. Αυτό δεν σημαίνει ότι ο συντονισμός των Node servers για μέγιστη απόδοση σε μια ρύθμιση παραγωγής είναι μια ασήμαντη υπόθεση, αλλά ότι οι επιλογές διαμόρφωσης είναι απλούστερες και πιο άμεσες. Μια ακόμα σημαντική διαφορά μεταξύ του Node και των άλλων κλασικών διακομιστών δικτύου (web servers) είναι ότι το Node είναι single threaded. Στην αρχή αυτό φαίνεται σαν να κάνουμε ένα βήμα πίσω αλλά αποδεικνύεται ότι είναι πάρα πολύ έξυπνο. Το single threading απλοποιεί το τρόπο που φτιάχνει κανείς μια διαδικτυακή εφαρμογή (web application) και όταν χρειάζεται η αποδοτικότητα μιας multithreaded εφαρμογής, μπορούν να εκτελεστούν περισσότερα στιγμιότυπα του Node, αποκτώντας με επιτυχία τα προνόμια απόδοσης που έχει το multithreading.

Επίσης με τη αυξανόμενη δημοτικότητα του cloud computing και με τον χειρισμό των servers σαν γενικά εμπορεύματα, αυτή η προσέγγιση είναι πολύ λογική. Τα ISS και Apache είναι όντως πολύ ισχυρά εργαλεία, και έχουν σχεδιαστεί για να εκμεταλλεύονται όλη την απόδοση που παρέχουν πλέον τα ισχυρά hardwares. Αυτό δημιουργεί και προβλήματα, γιατί τα εργαλεία αυτά απαιτούν σημαντική τεχνογνωσία για να επιτευχθεί αυτή η απόδοση.

Με κριτήριο τον τρόπο με τον οποίο οι Node διαδικτυακές εφαρμογές (web apps) φτιάχνονται έχουν πιο πολλά κοινά με τη PHP, τη Ruby από ότι με τη .NET ή τη Java. Ενώ η μηχανή Javascript που χρησιμοποιεί το Node (Google's V8) όντως μεταγλωττίζει τη Javascript σε κώδικα μητρικής μηχανής όπως η C ή C++, το κάνει διαφανώς, οπότε από τη πλευρά του χρήστη, συμπεριφέρεται σαν μια καθαρά ερμηνευμένη γλώσσα. Μη έχοντας ξεχωριστό βήμα μεταγλώττισης μειώνονται οι ενοχλήσεις συντήρησης και ανάπτυξης. Αυτό πρέπει μόνο να γίνεται είναι η ενημέρωση του αρχείου Javascript, και οι αλλαγές- θα είναι αυτόματα διαθέσιμες.

Ένα ακόμα σημαντικό πλεονέκτημα των Node εφαρμογών είναι ότι το NodeJS είναι μια απίστευτα ανεξάρτητη πλατφόρμα. Δεν είναι η πρώτη ή η μόνη τεχνολογία ανεξάρτητου διακομιστή, αλλά η ανεξαρτησία πλατφόρμας είναι πολύ περισσότερο του φάσματος από ότι δυαδική πρόταση. Για παράδειγμα, μπορούμε να τρέξουμε μια .NET εφαρμογή σε ένα Linux διακομιστή (server) χάρις στο Mono, αλλά είναι μια επώδυνη διαδικασία. Ομοίως, μπορούμε να τρέξουμε PHP εφαρμογές σε Windows διακομιστή, αλλά δεν είναι γενικότερα εύκολο να εγκατασταθεί σε Linux μηχανή. Το Node, από την άλλη, είναι ένα εργαλείο το οποίο μπορείς να εγκαταστήσεις σε οποιοδήποτε λειτουργικό σύστημα (Windows, OS X και Linux) και επιτρέπει την εύκολη συνεργασία. Μεταξύ των ομάδων σχεδιασμού ιστοσελίδων, είναι λογικό να δουλεύει ο καθένας σε διαφορετικά λειτουργικά συστήματα.

Ορισμένες πλατφόρμες, όπως το .NET, παρουσιάζει προκλήσεις για τους front-end προγραμματιστές και σχεδιαστές, όπου συχνά χρησιμοποιούν Mac , το οποίο έχει τεράστιο αντίκτυπο στη συνεργασία και την αποτελεσματικότητα. Η ιδέα του να μπορείς να δημιουργήσεις ένα λειτουργικό διακομιστή (server) σε ένα λειτουργικό σύστημα μέσα σε μερικά λεπτά (ή ακόμα και σε δευτερόλεπτα) είναι μια ιδέα που έγινε πραγματικότητα με τη βοήθεια του Node.

Το Node, φυσικά, βρίσκεται στη καρδιά της στοίβας. Είναι το λογισμικό που επιτρέπει τη Javascript να τρέχει στο διακομιστή (server) , αποσυνδεδεμένο από ένα πρόγραμμα περιήγησης, το οποίο με τη σειρά του επιτρέπει πλαίσια (frameworks) να γραφτούν στη Javascript όπως το Express και να μπορούν να χρησιμοποιηθούν. Ένα άλλο σημαντικό στοιχείο είναι η βάση δεδομένων η οποία καλύπτεται σε βάθος παρακάτω. Όλες εκτός από τις απλούστερες εφαρμογές θα χρειαστούν μια βάση δεδομένων, και υπάρχουν βάσεις δεδομένων που είναι περισσότερο στη βάση του Node οικοσυστήματος παρά κάποια άλλα. Δεν είναι έκπληξη το γεγονός ότι οι διεπαφές βάσης δεδομένων είναι διαθέσιμες για όλες τις μεγάλες σχεσιακές βάσεις δεδομένων (MySQL, MariaDB, PostgreSQL, Oracle, SQL Server). Δεν θα ήταν σωστό να παραμελήσουμε αυτά τα καθιερωμένα μεγαθήρια. Ωστόσο, η εμφάνιση της ανάπτυξης με Node έχει δημιουργήσει μια νέα προσέγγιση στην αποθήκευση βάσεων δεδομένων, τις λεγόμενες NoSQL βάσεις δεδομένων. Δεν είναι πάντα χρήσιμο να ορίζουμε κάτι ως κάτι που δεν είναι, οπότε θα προσθέσουμε ότι αυτές οι NoSQL βάσεις δεδομένων μπορεί να ονομαστούν πιο σωστά document databases ή key/value pair databases.

Τα άλλα δυο συστατικά που είναι συνήθως απαραίτητα για ανάπτυξη εφαρμογών δικτύου (web app development) είναι ο διακομιστής βάσης δεδομένων (database server) και ένας μηχανισμός που επεξεργάζεται (μια μηχανή που παρέχει ότι παρέχει η PHP, JSP ή Razor φυσικά. Η δυνατότητα των οποίων είναι να συνδυάζουν άψογα το αποτέλεσμα του κώδικα και της σήμανσης). Αυτό που συνδέει όλες τις τεχνολογίες μαζί είναι η Javascript, έτσι σε μια προσπάθεια να είμαστε πιο περιεκτικοί θα αναφερόμαστε στη Javascript stack. Για τους σκοπούς αυτής της πτυχιακής, θα χρησιμοποιήσουμε τα Node, Express, και MongoDB.

Γ.2.1.2 Εισαγωγή στο Express

Η διαδικτυακή σελίδα του Express περιγράφει το Express σαν ένα απλό και ευέλικτο πλαίσιο (framework) για τις NodeJS διαδικτυακές εφαρμογές (web applications), το οποίο παρέχει ένα ισχυρό σύνολο από λειτουργίες για την κατασκευή ενιαίων ή πολλαπλών σελίδων και υβριδικών διαδικτυακών εφαρμογών (web applications). Τι σημαίνει αυτό θα το αναλύσουμε παρακάτω.

Γ.2.1.2.1 Απλότητα

Η απλότητα είναι μια από τις πιο ελκυστικές πτυχές του Express. Πολλές φορές, οι προγραμματιστές πλαισίων (frameworks developers) ξεχνούν ότι το “λιγότερο είναι καλύτερο”. Η φιλοσοφία του Express παρέχει το απλούστερο στρώμα (layer) μεταξύ του μυαλού μας και του διακομιστή (server). Αυτό βέβαια δεν σημαίνει ότι μας περιορίζει, αλλά αντιθέτως μας επιτρέπει να εκφράσουμε πλήρως τις ιδέες μας.

Γ.2.1.2.2 Ευελιξία

Μια άλλη βασική πτυχή της φιλοσοφίας του Express είναι η επεκτασιμότητα. Το Express όπως προαναφέρθηκε παρέχει ένα απλό πλαίσιο (framework), όπου μπορούν να προστεθούν και να αφαιρεθούν λειτουργίες ανάλογα με τις ανάγκες του προγραμματιστή. Πολλά πλαίσια (frameworks) παρέχουν πολλές λειτουργίες, αφήνοντας το χρήστη με ένα φουσκωμένο, μυστήριο και περίπλοκο έργο πριν καν γράψει τη πρώτη γραμμή κώδικα. Πολύ συχνά, το πρώτο που κάνει κάποιος που μαθαίνει μια νέα τεχνολογία είναι να ξοδέψει χρόνο προσπαθώντας να ανακαλύψει όλες αυτές τις περιττές λειτουργίες, ή την αντικατάσταση αυτών των λειτουργιών, όταν δεν πληρούν τις προϋποθέσεις. Το Express υιοθετεί μια διαφορετική προσέγγιση, αφήνοντας το χρήστη να προσθέσει μόνο αυτά που χρειάζεται.

Γ.2.1.2.3 Πλαίσιο Διαδικτυακών Εφαρμογών

Εδώ είναι που σημασιολογικά αρχίζει να γίνεται περίπλοκη. Τι είναι μια διαδικτυακή εφαρμογή; Μήπως αυτό σημαίνει ένας ιστότοπος ή μια ιστοσελίδα με το Express; Όχι, δεν ισχύει κάτι τέτοιο, με τον όρο ιστότοπο εννοούμε μια διαδικτυακή εφαρμογή. Όμως η διαδικτυακή εφαρμογή (web application) είναι κάτι μεγαλύτερο, μπορεί να παρέχει κάποια λειτουργικότητα σε άλλες διαδικτυακές εφαρμογές (web applications) καθώς και πολλά άλλα. Γενικότερα, ο όρος “εφαρμογή” χρησιμοποιείται για να σηματοδοτήσει κάποια λειτουργικότητα και δεν είναι απλά μια

στατική συλλογή περιεχομένου (αν και αυτό είναι ένα πολύ απλό παράδειγμα μιας διαδικτυακής εφαρμογής). Ενώ υπάρχει γενικότερα μια διάκριση μεταξύ μιας εφαρμογής (κάτι το οποίο τρέχει σε μια συσκευή) και μιας ιστοσελίδας (κάτι το οποίο προβάλλεται στη συσκευή σου μέσα από ένα δίκτυο), αυτή η διάκριση γίνεται μικρότερη με το καιρό χάρης σε έργα όπως το PhoneGap και την κίνηση της Microsoft να επιτρέπει HTML 5 εφαρμογές στην επιφάνεια εργασίας σαν να ήταν κανονικές native εφαρμογές. Βλέποντας αυτή την εξέλιξη, είναι εύκολο να φανταστούμε ότι μέσα σε λίγα χρόνια, δεν θα υπάρχει διάκριση μεταξύ εφαρμογής και ιστοσελίδας.

Γ.2.1.2.4 Single-page Διαδικτυακές εφαρμογές

Οι διαδικτυακές εφαρμογές μιας σελίδας (single page web applications) είναι μια σχετικά νέα ιδέα. Αντί μιας ιστοσελίδας που απαιτεί ένα αίτημα δικτύου (network request) κάθε φορά που περιηγείται ο χρήστης σε διαφορετική σελίδα της εφαρμογής, η διαδικτυακή εφαρμογή μιας σελίδας κατεβάζει ολόκληρο το site (ή ένα μεγάλο κομμάτι αυτού) στο πρόγραμμα περιήγησης. Μετά την αρχική λήψη, η περιήγηση είναι πολύ πιο γρήγορη επειδή δεν χρειάζεται να φορτώσει εκ νέου η σελίδα. Η ανάπτυξη διαδικτυακής εφαρμογής μιας σελίδας (single page web application development) διευκολύνεται με τη χρήση δημοφιλών πλαισίων όπως η Angular ή το Ember.

Γ.2.1.2.5 Multipage και Hybrid Διαδικτυακές Εφαρμογές

Οι διαδικτυακές εφαρμογές πολλών σελίδων (Multipage web applications) είναι μια παραδοσιακή προσέγγιση των ιστοσελίδων. Κάθε σελίδα στην ιστοσελίδα παρέχεται από ένα διαφορετικό αίτημα διακομιστή (server request). Ακριβώς επειδή αυτή η προσέγγιση είναι πιο παραδοσιακή δεν σημαίνει ότι δεν έχει κάποια αξία πλέον ή ότι η διαδικτυακή εφαρμογή μιας σελίδας (Single-page web application) είναι κατά κάποιο τρόπο καλύτερη. Απλά, πλέον υπάρχουν περισσότερες επιλογές, και μπορούμε να διαλέξουμε ποια μέρη του περιεχομένου πρέπει να παρέχονται σαν μια σελίδα διαδικτυακής εφαρμογής, και ποια πρέπει να παρέχονται με ξεχωριστά αιτήματα. Το “Υβριδικό” μοντέλο περιγράφει τις ιστοσελίδες που μπορούν να αξιοποιήσουν και τις δυο προσεγγίσεις.

Γ.2.1.2.6 Μια σύντομη ιστορική αναδρομή για το Express

Ο δημιουργός του Express, TJ Holowaychuk, περιγράφει το Express ως ένα πλαίσιο (framework) που εμπνεύστηκε από το Sinatra, το οποίο είναι ένα πλαίσιο (framework) στη Ruby. Δεν μας κάνει έκπληξη ότι το Express δανείζεται πολλά τη Ruby, η οποία δημιούργησε μια πληθώρα από ενδιαφέρουσες προσεγγίσεις στην ανάπτυξη δικτύου (web development), και στόχευε στην γρήγορη ανάπτυξη διαδικτυακών εφαρμογών (web development). Όσο και αν το Express εμπνεύστηκε από το Sinatra, είναι επίσης πολύ κοντά με το Connect, μια “plugin” βιβλιοθήκη για το Node. Το Connect θέσπισε τον όρο “middleware” για να περιγράφει πρόσθετα Node Modules τα οποία μπορούν να διαχειριστούν αιτήματα δικτύου (web requests) σε διάφορα επίπεδα. Μέχρι και πριν την έκδοση 4.0 το Express περιλάμβανε το Connect, στην έκδοση 4.0 το Connect (και άλλα middlewares εκτός του static) αφαιρέθηκαν για να επιτρέψουν σε αυτό το middleware να ενημερώνεται ανεξάρτητα.

Γ.2.1.2.7 Η αναβάθμιση στο Express 4.0

Η αναβάθμιση στην έκδοση 4.0 που έγινε στο Express είναι αρκετά μεγάλη καθώς έγιναν σημαντικές αλλαγές. Όπως αναφέρθηκε προηγουμένως, αφαιρέθηκε το Connect, με την εξαίρεση φυσικά του static middleware. Την ίδια στιγμή, το Connect μετακίνησε κάποια από τα middlewares του στα δικά του πακέτα (packages). Επίσης, ο αναλυτής σώματος (body parser) βρίσκεται πλέον σε δικό του πακέτο, το οποίο δεν περιλαμβάνει το πολλαπλό middleware, κλείνοντας ένα μεγάλο πρόβλημα ασφαλείας. Μάλιστα, δεν χρειάζεται πλέον να συνδεθεί ο Express δρομολογητής (router) στην εφαρμογή, με αποτέλεσμα να πρέπει να καταργηθεί το `app.use (app.router)` από τις υπάρχουσες εφαρμογές με την έκδοση Express 3.0. Τέλος καταργήθηκε το `app.configure`.

Γ.2.1.2.8 Η αναβάθμιση στο Express 5.0

Η έκδοση Express 5.0 βρίσκεται ακόμα στο alpha στάδιο έκδοσης, αλλά μπορεί κανείς να δει συνοπτικά τις αλλαγές που θα γίνουν από το 4.0 στο 5.0. Το Express 5 δεν διαφέρει πολύ από το

Express 4, όπως έγινε από το 3.0 στο 4.0. Παρόλα αυτά εφαρμογές που φτιάχτηκαν με το Express 4.0 μπορεί να μην δουλεύουν μετά την αναβάθμιση. Στη συνέχεια θα αναλυθούν μερικές από τις σημαντικές αλλαγές που έγιναν. Καταργήθηκε το `app.del()`. Αρχικά το `del` χρησιμοποιήθηκε αντί του `delete`, γιατί η λέξη `delete` δεσμεύτηκε από τη Javascript. Παρόλα αυτά μετά την έκδοση ECMAScript 6, το `delete` καθώς και άλλες δεσμευμένες λέξεις μπορούν πλέον να χρησιμοποιηθούν σαν ιδιότητα ονομάτων. Αφαιρέθηκε επίσης το `app.param(fn)` το οποίο είχε ήδη αποδοκιμαστεί από την έκδοση v4.11.0 αλλά πλέον στο Express 5.0 δεν υποστηρίζεται καθόλου. Το `req.param (name)` πλέον δεν χρησιμοποιείται, αντί αυτού πλέον χρησιμοποιείται το `req.params`, `req.body` ή `req.query`. Επίσης το `res.json(obj,status)` πλέον δεν υποστηρίζεται και χρησιμοποιείται το `res.json()` και το `res.status(status).json(obj)`. Επίσης, το `res.send(obj, status)` δεν υποστηρίζεται και άλλαξε σε `res.send()` και `res.status(status).send(obj)`. Άλλες αλλαγές που έγιναν είναι ότι το `app.router` το οποίο είχε αφαιρεθεί από το Express 4 πλέον μπορεί κάποιος που θέλει, να το ξαναχρησιμοποιήσει. Τέλος κάποιες από τις βελτιώσεις που έγιναν είναι το `res.render()`, η οποία επιτρέπει την ασύγχρονη συμπεριφορά σε όλες τις view μηχανές, αποφεύγοντας προβλήματα που δημιουργήθηκαν από αυτές, οι οποίες είχαν σύγχρονη εκτέλεση και παραβίαζαν τη προτεινόμενη διεπαφή.

Γ.2.2 MongoDB & Mongoose

Γ.2.2.1 Εισαγωγή στο MongoDB

Η MongoDB είναι μια πολύ ισχυρή, ευέλικτη και κλιμακούμενη βάση δεδομένων. Συνδυάζει την ικανότητα να κλιμακώνεται με τα πιο χρήσιμα χαρακτηριστικά των σχεσιακών βάσεων δεδομένων, όπως δευτερεύοντα ευρετήρια, ερωτήματα εύρους και ταξινόμηση. Επίσης η MongoDB έχει πολλές χρήσιμες λειτουργίες, όπως η ενσωματωμένη υποστήριξη για Map-reduce/style, συσσωμάτωση και γεωχωρικά ευρετήρια.

Δεν υπάρχει νόημα να δημιουργήσεις μια πολύ χρήσιμη τεχνολογία, αν είναι δύσκολο από αυτόν που τη χρησιμοποιεί να τη κατανοήσει και κατ' επέκταση να τη εφαρμόσει. Ένα τέτοιο πρόβλημα δεν θα συναντήσει κάποιος με τη MongoDB διότι οι σχεδιαστές της έχουν δώσει πολύ μεγάλη

έμφαση στο να την κάνουν εύκολη για να ξεκινήσει κάποιος να δουλεύει με αυτή. Η MongoDB έχει ένα φιλικό μοντέλο δεδομένων για τον προγραμματιστή, επιλογές διαμόρφωσης που είναι φιλικές προς τον διαχειριστή και μια πολύ φιλική γλώσσα. Η MongoDB προσπαθεί να φύγει από τη μέση, και απλά να αφήσει τον προγραμματιστή να δημιουργήσει αντί να ανησυχεί για την αποθήκευση των δεδομένων.

Η MongoDB είναι μια προσανατολισμένη σε έγγραφα (document-oriented) βάση δεδομένων, και όχι μια σχεσιακή βάση δεδομένων. Ο μεγαλύτερος λόγος για να μεταπηδήσει κανείς από το σχεσιακό μοντέλο είναι ότι η κλιμάκωση (scaling) γίνεται ευκολότερα, αλλά υπάρχουν και άλλα πλεονεκτήματα. Η βασική ιδέα είναι να αντικαταστήσουμε την έννοια της σειράς (row) με ένα πιο ευέλικτο μοντέλο, το έγγραφο (document). Επιτρέποντας ενσωματωμένα έγγραφα και πίνακες, η document-oriented προσέγγιση καθιστά δυνατή την αναπαράσταση σύνθετων ιεραρχικών σχέσεων με μια μόνο εγγραφή. Αυτό ταιριάζει πολύ φυσικά στο τρόπο με τον οποίο οι προγραμματιστές σκέφτονται χρησιμοποιώντας της σύγχρονες αντικειμενοστραφείς γλώσσες.

Επίσης η MongoDB δεν έχει σχήματα (schema-free) και τα κλειδιά ενός εγγράφου δεν είναι προκαθορισμένα ούτε σταθερά. Χωρίς το σχήμα να αλλάζει, μαζικές μετεγκαταστάσεις δεδομένων συνήθως είναι περιττές. Νέα ή χαμένα κλειδιά μπορούν να αντιμετωπιστούν σε επίπεδο εφαρμογής αντί να αναγκάζουν όλα τα δεδομένα να έχουν το ίδιο σχήμα. Αυτό δίνει μεγάλη ευελιξία στον χειρισμό εξελισσόμενων μοντέλων δεδομένων.

Αν και η MongoDB είναι πολύ ισχυρό εργαλείο και προσπαθεί να κρατήσει πολλές δυνατότητες από τα σχεσιακά συστήματα, στη πραγματικότητα δεν προορίζεται να κάνει τα πάντα τα οποία κάνει μια σχεσιακή βάση δεδομένων. Στη MongoDB βάση, ο διακομιστής της βάσης δεδομένων έχει την ικανότητα να εκφορτώνει την επεξεργασία και την λογική στη πλευρά του πελάτη (client side). Η διατήρηση αυτού του βελτιωμένου σχεδιασμού είναι ένας από τους λόγους για τους οποίους η MongoDB μπορεί να επιτύχει τόσο υψηλή απόδοση.

Η MongoDB προσπαθεί να απλοποιήσει τη διαχείριση των βάσεων δεδομένων. Εκτός από την εκκίνηση του διακομιστή βάσης δεδομένων, απαιτείται και πολύ μικρή διαχείριση. Εάν ένας κεντρικός διακομιστής τερματιστεί η MongoDB μπορεί αυτόματα να μεταφερθεί σε ένα εφεδρικό slave και να προάγει το slave σε ένα κύριο (master). Σε ένα καταναμημένο περιβάλλον, το σύμπλεγμα (cluster) μπορεί να δημιουργηθεί μόνο όταν υπάρχει ένας νέος κόμβος για να κάνει αυτόματη ενσωμάτωση και διαμόρφωση του. Η φιλοσοφία της διαχείρισης του MongoDB είναι ότι ο διακομιστής μπορεί να χειρίζεται όσο το δυνατόν περισσότερη διαμόρφωση αυτόματα, επιτρέποντας στους χρήστες να τροποποιήσουν τις ρυθμίσεις τους εάν χρειαστεί.

Γ.2.2.2 Εισαγωγή στο Mongoose

Το Mongoose χρησιμοποιείται συχνά σε συνδυασμό με το NodeJS το Express, το MongoDB και το npm. Το Mongoose είναι ένα εργαλείο μοντελοποίησης αντικειμένων για το MongoDB και το NodeJS. Πρακτικά αυτό σημαίνει ότι μπορεί κανείς να το ορίσει σαν ένα μοντέλο το δεδομένων του προγραμματιστή ενσωματωμένο μέσα στον κώδικα του. Δεν χρειάζεται να σχεδιάσει ένα σχήμα (Schema) στη βάση δεδομένων του, να το συνδεθεί σε ένα Object Relational Mapper (ORM) ή να την αντιστοιχίσει μέσα στα αντικείμενα και κλάσεις στο πρότζεκτ του. Είναι ένα πολύ γρήγορο και εύελκτο εργαλείο που βοηθάει πολύ στο να συνδεθεί κανείς εύκολα και να δημιουργήσει μέσα στη βάση δεδομένων του.

Για να ορίσει κανείς ένα σχήμα (Schema) στο κώδικα του απλά πρέπει να γράψει το παρακάτω κώδικα:

```
var userSchema = new mongoose.Schema({  
  firstname: String,  
  lastname: String,  
  createdAt: Date  
});
```

Όπως αναφέρθηκε και πιο πάνω το Mongoose είναι κυρίως χρήσιμο όταν θέλει κανείς να αλληλεπιδράσει με την βάση δεδομένων του που είναι στο MongoDB, αλλά για πιο λόγο είναι καλό σε αυτό;

Δίνει την δυνατότητα να οριστούν σχήματα (Schemas) για τα δικά του δεδομένα, και αυτό του δίνει τη ικανότητα να αλληλεπιδρά με τα δεδομένα, με ένα πιο δομημένο και επαναλαμβανόμενο τρόπο.

Το Mongoose βοηθάει με πολλές κοινές εργασίες του, το MongoDB και αφαιρεί κάποια επίπεδα πολυπλοκότητας, από τις επιστροφές (callback).

Το Mongoose επιστέφει τα δεδομένα σε μορφή αντικειμένου JSON που μπορεί να χρησιμοποιηθεί απευθείας και αυτό είναι από τα σημαντικότερα, αν όχι το πιο σημαντικό, πλεονέκτημα για να δουλέψει κανείς με το Mongoose.

Γ.2.3 Εισαγωγή στο PassportJS

Το PassportJS με απλά λόγια είναι ένα ενδιάμεσο (middleware) για το πλαίσιο (framework) Express. Επιτρέπει στους προγραμματιστές να ενσωματώνουν διάφορους τύπους ελέγχου ταυτότητας, εύκολα με τη χρήση ελάχιστου κώδικα. Για παράδειγμα, οι προγραμματιστές μπορούν να προσθέσουν διάφορους τρόπους πιστοποίησης του χρήστη, όπως με τον έλεγχο των στοιχείων του χρήστη που βρίσκονται σε μια βάση δεδομένων, αλλά και με τη χρήση του Facebook, Google, GitHub, Twitter κτλπ.

Επίσης μπορούν να συνδυαστούν όλοι οι παραπάνω τρόποι πιστοποίησης και ο χρήστης με αυτόν τον τρόπο θα έχει πολλές επιλογές. Είναι ένας πολύ πιο εύκολος τρόπος από το να δημιουργήσει κανείς μόνος του από την αρχή μια προσαρμοσμένη διαδικασία πιστοποίησης για την διαδικτυακή του εφαρμογή.

Γ.2.4 Εισαγωγή στο Socket.io

Το εργαλείο Socket.io είναι ένα πολύ ισχυρό εργαλείο γιατί δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει εφαρμογές σε πραγματικό χρόνο με αμφίδρομη επικοινωνία μεταξύ του διακομιστή (server) και πελάτη (client). Συνδυάζει τη μεγάλη δύναμη που έχουν τα WebSockets μαζί με πολλά άλλα χαρακτηριστικά, όπως του JSON long polling και του JSONP long polling μέσα από μια ενιαία API. Με τον όρο long polling εννοούμε την εξομοίωση δεδομένων ώθησης που εφαρμόζονται με επαναλαμβανόμενες αιτήσεις (polling) με καθυστερημένη απόκριση.

Συνήθως χρησιμοποιείται για τη δημιουργία αμφίδρομων αλληλεπιδράσεων, όπως σε πίνακες ελέγχου πραγματικού χρόνου, για εφαρμογές συνομιλίας (στη σελίδα socket.io το παράδειγμα που χρησιμοποιούν για να δείξουν στη πράξη πως δουλεύει είναι ένα web chat) και για παιχνίδια για πολλούς παίκτες. Πριν τη βιβλιοθήκη socket.io υπήρχαν διάφοροι τρόποι για να δημιουργήσουμε την επικοινωνία σε πραγματικό χρόνο του server με το client, αλλά ήταν δύσκολοι και περίεργοι. Μια μέθοδος ήταν να “χτυπάμε” τον διακομιστή (server) με μια κλήση Ajax κάθε λίγα δευτερόλεπτα. Επειδή ο server δεν ήξερε αν έγιναν κάποιες αλλαγές απαντούσε στο αίτημα με ένα αρχείο JSON και ο client έλεγε αν σε αυτό το JSON υπήρχαν ενημερώσεις. Αν υπήρχαν, ο client ενημέρωνε το χρήστη μέσα από την οθόνη. Αυτό είναι δύσκολο στην ανάπτυξη, αλλά ακόμα πιο δύσκολος είναι ο εντοπισμός των σφαλμάτων. Με το Socket.io μπορούμε να στέλνουμε μόνο τις αλλαγές που γίνονται και όταν γίνονται. Αντί να ορίσουμε ένα χρονικό διάστημα για να κάνουμε κλήσεις Ajax, μπορούμε να στείλουμε δεδομένα μόλις αυτά εισέλθουν. Με λίγα λόγια τα Websockets και το Socket.io μας κάνουν τη ζωή πιο εύκολη.

Το Socket.io είναι μια βιβλιοθήκη ανοιχτού κώδικα που δημιουργήθηκε από τον Guillermo Rauch. Είναι κατασκευασμένο με το Engine.IO, που είναι μια χαμηλού επιπέδου τεχνολογία πάνω από τη τεχνολογία του WebSocket. Το Socket.io χρησιμοποιείται για να επικοινωνούν αμφίδρομα ο server και ο client σε μια σύνταξη που μοιάζει σαν να πυροδοτεί και να ακούει γεγονότα. Το WebSocket API πρωτόκολλο τυποποιήθηκε το 2011. Πρόκειται για ένα πρωτόκολλο ελέγχου μετάδοσης (TCP) που βασίζεται στο HTTP μόνο για την αρχική χειραψία. Αφού ολοκληρωθεί η χειραψία, η σύνδεση αφήνεται ανοιχτή έτσι ώστε ο διακομιστής (server) και ο πελάτης (client) να μπορούν να περάσουν μηνύματα μπρος και πίσω όταν χρειάζεται.

Το Socket.io προχωρά ένα βήμα παραπέρα παρέχοντας μια εύκολη στη χρήση αλλά και ισχυρή API πάνω από τα WebSockets. Επίσης το συγκεκριμένο παρέχει την δυνατότητα της χρήσης άλλων πραγματικού χρόνου πρωτοκόλλων αν τα WebSockets δεν είναι διαθέσιμα. Για παράδειγμα, θα επιστρέψει στο JSON long polling στη απουσία υποστήριξης του WebSocket. Το Long polling είναι ουσιαστικά μια τεχνική για να μιμηθούμε τη συμπεριφορά των WebSockets. Μετά από το long polling αίτημα, αυτό διατηρείται στο διακομιστή (server), αντί να ανταποκρίνεται αμέσως όπως θα έκανε ένα παραδοσιακό αίτημα HTTP. Όταν τα δεδομένα γίνουν διαθέσιμα, το long polling αίτημα τερματίζεται κλείνοντας το βρόχο του μεγάλου κύκλου αιτήσεων. Σε αυτό το σημείο ένα νέο long polling αίτημα θα δημιουργηθεί. Αυτό δίνει τη ψευδαίσθηση της συνεχούς σύνδεσης που τα WebSockets παρέχουν. Παρόλο που το long polling είναι κάτι λιγότερο από ιδανικό στο τοπίο της σύγχρονης τεχνολογίας, είναι μια πολύ καλή εναλλακτική για κάποιον που θέλει συνεχή επικοινωνία διακομιστή (server) και πελάτη (client). Όταν στέλνουμε ένα μήνυμα με το Socket.io, η API για τα WebSockets και για το long polling είναι πανομοιότυπη, με αποτέλεσμα να μην χρειάζεται να ασχοληθούμε με τη διανοητική επιβάρυνση της ολοκλήρωσης δυο συντακτικά διαφορετικών τεχνολογιών. Παρόλο που υπάρχουν Socket.io εφαρμογές σε πολλές γλώσσες για την ανάπτυξη διακομιστών, θα χρησιμοποιήσουμε NodeJS σε αυτή τη πτυχιακή για την ανάπτυξη του διακομιστή (server).

Γ.3 Front end

Γ.3.1 Εισαγωγή στην HTML

Η HTML (HYPERTEXT MARKUP LANGUAGE) χρησιμοποιεί ένα σύστημα σήμανσης που αποτελείται από στοιχεία που αντιπροσωπεύουν ένα συγκεκριμένο περιεχόμενο. Σήμανση (markup) σημαίνει ότι με την HTML δηλώνουμε τι θα εμφανίζεται στο χρήστη στο πρόγραμμα περιήγησης του, όχι όμως πώς θα εμφανίζεται. Οι οπτικές αναπαραστάσεις δηλώνονται με τη χρήση της CSS (CASCADING STYLE SHEETS) και πραγματοποιούνται από το πρόγραμμα περιήγησης.

Η HTML καμία φορά αποκαλείται προγραμματιστική γλώσσα, όμως αυτό δεν ισχύει γιατί δεν έχει λογική, οπότε τη καλούμε γλώσσα σήμανσης (markup language). Οι ετικέτες (tags) της HTML παρέχουν σημασιολογική σημασία και μηχανική αναγνωσιμότητα στο περιεχόμενο της σελίδας, Ένα στοιχείο αποτελείται συνήθως από μια ετικέτα ανοίγματος (< όνομα στοιχείου >), και μια ετικέτα κλεισίματος (</ όνομα στοιχείου >), το οποίο περιέχει το όνομα του στοιχείου το οποίο περιβάλλεται από αγκύλες, και το περιεχόμενο ανάμεσα:

< όνομα στοιχείου >..... περιεχόμενο..... </ όνομα στοιχείου >

Υπάρχουν βέβαια και κάποια HTML στοιχεία τα οποία δεν έχουν ετικέτες κλεισίματος ή κανένα περιεχόμενο. Αυτά καλούνται κενά (void) στοιχεία. Στα κενά στοιχεία περιλαμβάνονται τα < img >, < meta >, < link > και < input >.

Τα ονόματα των στοιχείων μπορούν να θεωρηθούν ως περιγραφικές λέξεις - κλειδιά για το περιεχόμενο που έχουν, όπως βίντεο, ήχο, πίνακα, υποσέλιδα. Μια HTML σελίδα μπορεί να αποτελείται δυνητικά από εκατοντάδες στοιχεία τα οποία διαβάζονται από ένα πρόγραμμα περιήγησης ιστού, και στη συνέχεια ερμηνεύονται και αποδίδονται σε ανθρώπινα αναγνώσιμα ή ακουστικά περιεχόμενα στην οθόνη.

Για τη παρούσα πτυχιακή είναι σημαντικό να σημειωθεί η διαφορά μεταξύ στοιχείων και ετικετών:

Στοιχεία: βίντεο, ήχος, πίνακας, υποσέλιδα

Ετικέτες: < video >, < audio >, < table >, < footer >, < /html >, < / body >

Παρακάτω θα αναλυθεί μια ετικέτα για να μπορέσει ο αναγνώστης να κατανοήσει τη λογική με την οποία λειτουργεί η HTMLH ετικέτα < p > αντιπροσωπεύει μια κοινή παράγραφο. Τα στοιχεία έχουν συνήθως μια ετικέτα ανοίγματος και κλεισίματος.. Η ετικέτα ανοίγματος περιέχει το όνομα του στοιχείου μέσα σε αγκύλες. Η ετικέτα κλεισίματος είναι ίδια με τη με την ετικέτα ανοίγματος με τη διαφορά ότι υπάρχει το σύμβολο / πριν το όνομα του στοιχείου. Το περιεχόμενο βρίσκεται ανάμεσα σε αυτά, για παράδειγμα:

<p> Αυτή είναι μια απλή παράγραφος </p>

Παρακάτω παρουσιάζεται ένα παράδειγμα ενός αρχείου HTML.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Hello!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is a simple paragraph.</p>
  </body>
</html>
```

Εικόνα 2: Παράδειγμα αρχείου HTML

Αυτές είναι οι ετικέτες που χρησιμοποιούνται στο παραπάνω παράδειγμα :

< !DOCTYPE > Ορίζει την έκδοση της HTML που χρησιμοποιείται. Σε αυτή τη περίπτωση είναι HTML5.

< **html** > Ανοίγει τη σελίδα. Καμία επισήμανση δεν πρέπει να υπάρχει μετά την ετικέτα κλεισίματος < /html>. Το χαρακτηριστικό lang δηλώνει τη γλώσσα της σελίδας χρησιμοποιώντας τους ISO κωδικούς γλώσσας.

< **head** > Ανοίγει την ενότητα κεφαλής, η οποία δεν εμφανίζεται στο κύριο παράθυρο του προγράμματος περιήγησης, αλλά περιέχει κυρίως πληροφορίες σχετικά με το έγγραφο html τα οποία ονομάζονται μετά-δεδομένα (metadata). Μάλιστα μπορεί να περιέχει εισαγωγές συνδέσμων για να χρησιμοποιούνται διάφορες βιβλιοθήκες και άλλα στοιχεία που δεν υπάρχουν στον υπολογιστή μας.

< **meta** > Δίνει στο πρόγραμμα περιήγησης ορισμένα μετά-δεδομένα σχετικά με το έγγραφο. Το χαρακτηριστικό charset δηλώνει το χαρακτήρα κωδικοποίησης. Τα σύγχρονα έγγραφα HTML πρέπει πάντα να χρησιμοποιούν UTF-8, παρόλο που δεν είναι απαραίτητο. Στην HTML, η ετικέτα < meta > δεν έχει ετικέτα κλεισίματος.

< **title** > Ο τίτλος της σελίδας. Το κείμενο γράφεται ανάμεσα στην ετικέτα ανοίγματος και κλεισίματος και εμφανίζεται στη καρτέλα της σελίδας ή στη γραμμή τίτλου του προγράμματος περιήγησης.

< **body** > Ανοίγει το τμήμα του εγγράφου που εμφανίζεται στους χρήστες δηλαδή όλο το ορατό ή ηχητικό περιεχόμενο της σελίδας. Κανένα περιεχόμενο δεν πρέπει να προστίθεται μετά το κλείσιμο ετικέτας.

Γ.3.2 Εισαγωγή στην CSS

Η HTML προήλθε από ένα πρωτότυπο μιας γλώσσας που δημιουργούσε ο Tim Berners-Lee το 1992. Ο Berners-Lee θεώρησε ότι υπήρχε η δυνατότητα σύνδεσης εγγράφων με τη χρήση υπερκειμένου (hypertext), και η έννοια της HTML εξελίχθηκε από αυτό. Μια σημαντική προσθήκη

σε αυτόν τον τομέα ήταν η εισαγωγή της CSS στην HTML 4.0. Πριν από την εισαγωγή της CSS οι σχεδιαστές ιστοσελίδων και προγραμματιστές, χρησιμοποιούσαν την HTML για σκοπούς μορφοποίησης. Ο σκοπός της HTML όμως είναι να καθορίζει μόνο τη δομή της σελίδας. Ο σκοπός τώρα της CSS ήταν να πάρει το στυλ από τη διαρθρωτική σήμανση. Με την εισαγωγή της CSS, μπορούμε πλέον να διαχωρίσουμε τη παρουσίαση από το περιεχόμενο. Σαν αποτέλεσμα, η παρουσίαση θα μπορούσε να αφαιρεθεί από το αρχείο HTML και να αποθηκευτεί σε ξεχωριστό αρχείο που θα έχει κατάληξη .css, το οποίο μπορούμε να συμπεριλάβουμε στο έγγραφο μας χρησιμοποιώντας μια ετικέτα συνδέσμου < link >. Με αυτόν τον τρόπο έχουμε μια καλύτερη κατανομή του κώδικας μας που μας προσφέρει ευελιξία και ευκολία στη χρήση.

Γ.3.3 Εισαγωγή στο Bootstrap

Υπάρχουν πολλοί λόγοι για να χρησιμοποιήσει κάποιος το Bootstrap, αλλά ας αναφέρουμε κάποιους από τους βασικούς. Ένα από τα πιο απογοητευτικά μέρη του να ξεκινάς ένα πρότζεκτ από την αρχή είναι να δημιουργήσεις τη βάση για την HTML, τη CSS και τη JavaScript. Είναι λογικό να επαναχρησιμοποιείται έτοιμος κώδικας και στη συνέχεια να χτίζει ο προγραμματιστής πάνω σε αυτόν. Κάποιοι προγραμματιστές προτιμούν να γράφουν το δικό τους πλαίσιο (framework), και σε άλλες περιπτώσεις είναι πιο εύκολο να χρησιμοποιείται ένα υπάρχον πλαίσιο. Το Bootstrap είναι ένα πολύ ισχυρό εργαλείο στο κόσμο των start-up. Το Bootstrap επιτρέπει να δημιουργήσεις κανείς γρήγορα ένα πρωτότυπο για να αποδείξει την ιδέα κάποιου χωρίς να αφιερώσει αυτός πάρα πολύ χρόνο στο frontend, για κάτι που τελικά μπορεί να μην χρησιμοποιηθεί. Επίσης, επειδή είναι πολύ γνωστό πλαίσιο, αν δουλέψει κανείς σε κάποια ομάδα είναι πολύ πιθανό να το γνωρίζουν όλοι, με αποτέλεσμα να υπάρχει καλύτερη κατανόηση αλλά και επικοινωνία μεταξύ των συνεργαζομένων. Αυτό θα επιτρέψει τη συνοχή του κώδικα από τη πρώτη μέρα. Δεν θα υπάρχει καμία διαφωνία πώς να ονομάσουμε τους επιλογείς (selectors) ή να βρούμε τον καλύτερο τρόπο για τη διαμόρφωση των CSS αρχείων. Το μεγαλύτερο μέρος, με τη χρήση της Bootstrap έχει ήδη ρυθμιστεί και μπορεί να συνεχιστεί η δημιουργία του έργου πολύ πιο γρήγορα, δίνοντας μεγαλύτερη έμφαση σε άλλα προβλήματα. Ο κυριότερος όμως λόγος που αξίζει να χρησιμοποιεί κανείς το Bootstrap είναι ότι βοηθάει στην κατασκευή ιστοσελίδων οι οποίες αποκρίνονται σε όλες τις οθόνες. Με τη χρήση μόνο της CSS το να κάνουμε μια σελίδα να αποκρίνεται σωστά σε όλες τις οθόνες είναι μια δύσκολη και χαοτική διαδικασία γιατί πρέπει να χρησιμοποιηθούν media queries για το κάθε πλάτος της κάθε οθόνης για το κάθε στοιχείο της html. Κάτι τέτοιο δεν συμβαίνει με το Bootstrap

που προσφέρει αυτές τις επιλογές μέσα σε μια γραμμή κώδικα. Βέβαια πρέπει να αναφερθεί ότι οποιοδήποτε έτοιμο πλαίσιο, ο οποιοσδήποτε έτοιμος κώδικας, μετά από ένα σημείο μας περιορίζει γιατί δεν έχουμε 100% την ελευθερία να κάνουμε όλες τις παραμετροποιήσεις που θέλουμε. Το ίδιο συμβαίνει και με το Bootstrap. Οπότε ένας νέος προγραμματιστής που δεν έχει καταφέρει ακόμη να διευρύνει τις γνώσεις του πάνω στη CSS αλλά θέλει να φτιάξει κάτι, τότε είναι μια καλή επιλογή σε αντίθετη περίπτωση δεν προτείνεται.

Κεφάλαιο Δ'

Δ.1 Σχεδίαση και ανάπτυξη εφαρμογής

Δ.1.1 Εισαγωγή στην εφαρμογή

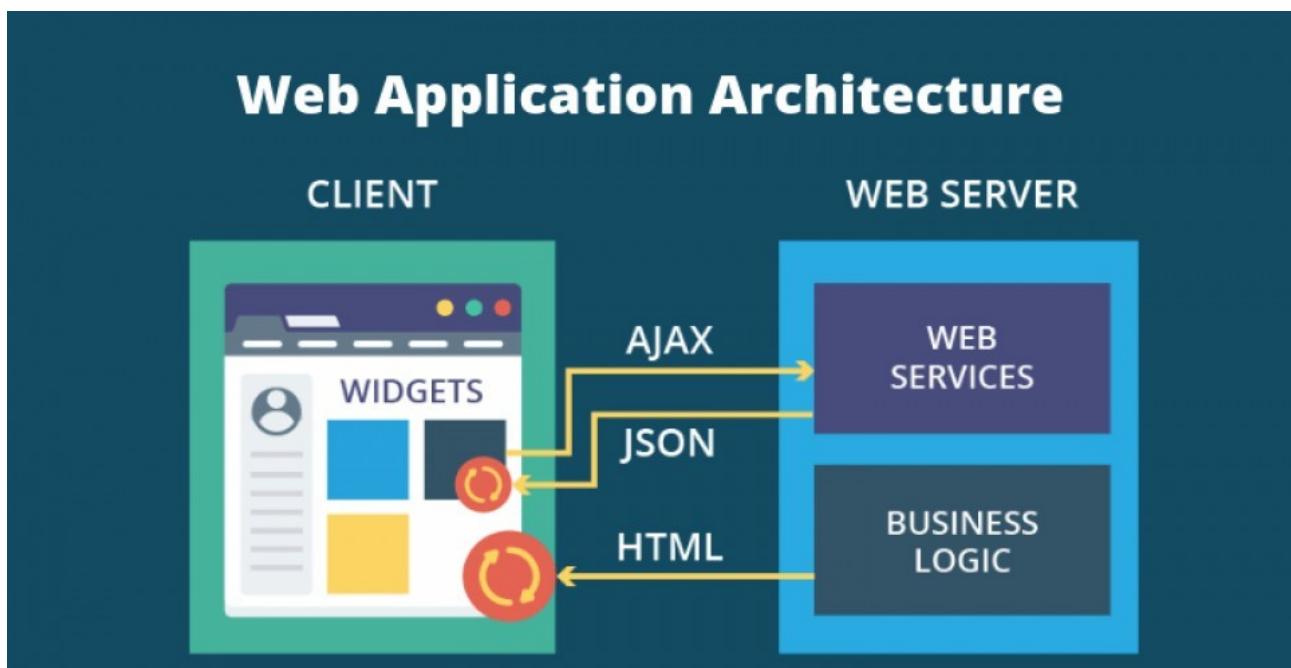
Η διαδικτυακή εφαρμογή Nstocks είναι μια εφαρμογή η οποία κατασκευάστηκε για την παρακολούθηση μετοχών. Περιλαμβάνει την οπτικοποίηση δεδομένων που στη συγκεκριμένη περίπτωση είναι οι μετοχές αλλά και τη συνεχή αλληλεπίδραση με το χρήστη καθώς ο χρήστης έχει τη δυνατότητα να παρακολουθεί πιο στενά τις αγαπημένες του μετοχές και να ενημερώνεται αυτόματα για τις πιθανές αυξομειώσεις των τιμών των μετοχών που τον ενδιαφέρουν. Τα προνόμια της ειδοποίησης για επιθυμητές αλλαγές στη τιμή των μετοχών την έχουν μόνο οι χρήστες που εγγράφονται στη σελίδα, για ευνόητους λόγους. Οι μετοχές παρουσιάζονται σε γραφικές παραστάσεις οι οποίες απεικονίζουν την τιμή της μετοχής της σημερινής ημέρας, δυο ημερών, ενός μήνα, πέντε μηνών, ενός χρόνου, πέντε χρόνων αλλά και την τιμή της μετοχής από τη στιγμή που εταιρία ξεκίνησε να είναι στο χρηματιστήριο. Επίσης για κάθε τιμή παρουσιάζεται η open τιμή, η high τιμή, η low τιμή και η previous close τιμή. Πληροφορίες που είναι πολύ χρήσιμες για έναν αναλυτή. Τέλος δείχνει το ποσοστό αύξησης ή πτώσης της τιμής σε σχέση με τη προηγούμενη. Ο χρήστης έχει τη δυνατότητα να αναζητήσει όποια μετοχή τον ενδιαφέρει πληκτρολογώντας το

σύμβολο της μετοχής ή ακόμα και ολόκληρο το όνομα της στην μπάρα αναζήτησης που βρίσκεται στη σελίδα stocks της διαδικτυακής εφαρμογής (web application) Nstocks.

Πέρα από αυτές τις λειτουργίες στη διαδικτυακή εφαρμογή ο χρήστης μπορεί να εγγραφεί χρησιμοποιώντας το email και αυτό του δίνει πρόσβαση στη σελίδα userpage στην οποία μπορεί να ακολουθήσει και να αποθηκεύσει τις αγαπημένες του μετοχές για να τις παρακολουθεί πιο στενά. Επίσης του δίνεται η δυνατότητα να ορίσει μία τιμή όπου αν η πραγματική τιμή της μετοχής πέσει κάτω από αυτή να τον ειδοποιήσει καθώς και να ορίσει μια τιμή όπου αν η τιμή της μετοχής την περάσει επίσης να τον ειδοποιήσει αντίστοιχα με το κατάλληλο μήνυμα. Οι ειδοποιήσεις γίνονται αυτόματα από το server μόλις εντοπίσει ότι η τιμή της μετοχής έπεσε ή ανέβηκε πάνω ή κάτω από τα αντίστοιχα όρια που θέτει ο χρήστης. Τέλος οι ειδοποιήσεις εμφανίζονται σε πραγματικό χρόνο στην εφαρμογή αν είναι συνδεδεμένος ο χρήστης αλλιώς αποθηκεύονται στη βάση δεδομένων για να μπορεί ο χρήστης να τις δει στην εφαρμογή την επόμενη φορά που θα συνδεθεί, παράλληλα όμως για να υπάρχει αμεσότητα και καλύτερη επικοινωνία εφαρμογής και χρήστη ο διακομιστής (server) στέλνει και ένα email στο χρήστη με ένα ανάλογο μήνυμα.

Παρακάτω θα αναλυθεί ο τρόπος που κατάφερα να δημιουργήσω αυτές τις λειτουργίες, τη διαδικασία που ακολουθήθηκε από το σχεδιασμό, την ανάπτυξη μέχρι και το διαμοιρασμό της διαδικτυακής εφαρμογής στο παγκόσμιο ιστό, χρησιμοποιώντας τεχνικές και εργαλεία που ένας αρχάριος προγραμματιστής θα έχει τη δυνατότητα να αναπτύξει και να εφαρμόσει. Τέλος θα παρουσιαστεί λεπτομερώς το πως μπορεί κανείς να εγκαταστήσει τα εργαλεία στον υπολογιστή και θα δημιουργηθεί το κατάλληλο προγραμματιστικό περιβάλλον για την ανάπτυξη μιας διαδικτυακής εφαρμογής με τα πιο σύγχρονα και δημοφιλή εργαλεία που υπάρχουν αυτή τη στιγμή στην βιομηχανία ανάπτυξης διαδικτυακών εφαρμογών και ιστοσελίδων.

Δ.1.2 Αρχιτεκτονική Εφαρμογής



Εικόνα 3: Αρχιτεκτονική Εφαρμογής

Δ.1.2.1 Ορισμός Αρχιτεκτονικής Διαδικτυακών Εφαρμογών

Η αρχιτεκτονική διαδικτυακών εφαρμογών περιγράφει τις αλληλεπιδράσεις μεταξύ των εφαρμογών, των βάσεων δεδομένων και συνδετικών συστημάτων (middleware) στο διαδίκτυο. Περιεκτικά η αρχιτεκτονική είναι αυτή η οποία εξασφαλίζει ότι πολλαπλές εφαρμογές λειτουργούν ταυτόχρονα. Για να καταλάβει κανείς αυτός που αναλύθηκε πιο πάνω θα χρησιμοποιηθεί ένα παράδειγμα ανοίγοντας μια ιστοσελίδα.

Όταν κάποιος που περιηγείται στο διαδίκτυο πατήσει το κουμπί για να αναζητήσει την ιστοσελίδα που πληκτρολόγησε στη μπάρα διεύθυνσης του περιηγητή, αυτό που κάνει είναι να στέλνει αίτηση (request) για αυτή τη μοναδική διαδικτυακή διεύθυνση. Ο server με τη σειρά του στέλνει τα αρχεία στο πρόγραμμα περιήγησης σαν απάντηση στην αίτηση που έκανε ο χρήστης. Ο περιηγητής στη συνέχεια εκτελεί αυτά τα αρχεία και εμφανίζει την σελίδα που έκανε αναζήτηση ο χρήστης.

Σε αυτή τη σελίδα τώρα ο χρήστης με την σειρά του αλληλεπιδρά με την διαδικτυακή εφαρμογή. Ο κώδικας που εκτελείται στον περιηγητή μπορεί, μπορεί και όχι, να έχει συγκεκριμένες οδηγίες με

τις οποίες καθοδηγεί τον περιηγητή για το πώς να ανταποκρίνεται στα διάφορα είδη των εισαγωγών που κάνουν οι χρήστες.

Ως εκ τούτου, η αρχιτεκτονική μιας διαδικτυακής εφαρμογής περιλαμβάνει όλα τα υποσυστήματα όπως και τις εξωτερικές αλληλεπιδράσεις για ολόκληρο το λογισμικό της εφαρμογής. Η αρχιτεκτονική είναι κάτι απαραίτητο για το σύγχρονο κόσμο καθώς μια μεγάλη μερίδα του παγκόσμιου δικτύου, όπως και οι περισσότερες εφαρμογές και συσκευές, κάνουν χρήση της web-based επικοινωνίας.

Η αρχιτεκτονική διαδικτυακής εφαρμογής δεν έχει μόνο να διαχειριστεί την αποδοτικότητα, αλλά και την αξιοπιστία, επεκτασιμότητα, ασφάλεια και ευρωστία.

Δ.1.2.1 Πως λειτουργεί

Σε όλες τις διαδικτυακές εφαρμογές, υπάρχουν δύο διαφορετικοί κώδικες οι οποίοι τρέχουν παράλληλα.

- Client-side – Ο κώδικας που είναι μέσα στο πρόγραμμα περιήγησης και ανταποκρίνεται στις εντολές του χρήστη.
- Server-side – Ο κώδικας που είναι μέσα στο διακομιστή (server) και ανταποκρίνεται στις HTTP αιτήσεις.

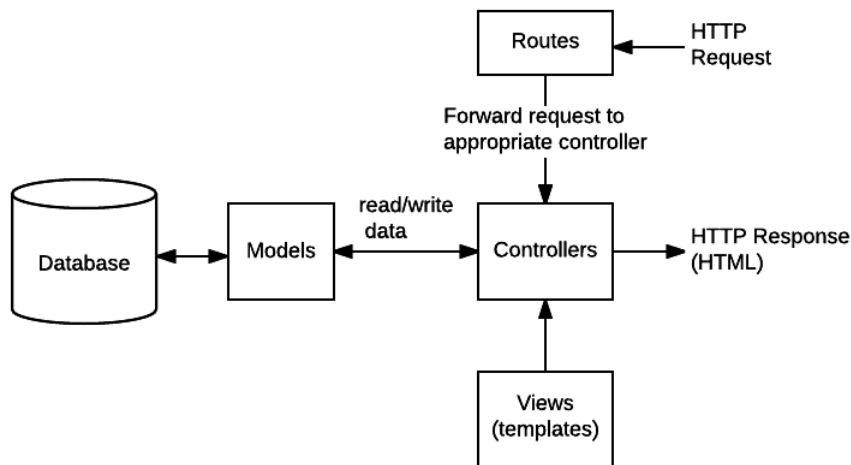
Δ.1.2.2 Συστατικά Διαδικτυακής Εφαρμογής

- **UI/UX Web Application Components** - Περιλαμβάνει αρχεία καταγραφής δραστηριότητας (activity logs), πίνακες ελέγχου (dashboards), ειδοποιήσεις (notifications), ρυθμίσεις (settings), στατιστικά (statistics) και άλλα πολλά. Αυτά τα συστατικά δεν έχουν να κάνουν με τη διαδικασία της αρχιτεκτονικής της διαδικτυακής εφαρμογής. Αντιθέτως είναι μέρος του σχεδίου (layout).

- **Structural Components** - Τα δύο βασικά κατασκευαστικά συστατικά μιας διαδικτυακής εφαρμογής είναι ο client και ο server.
- **Client Component** - Τα συστατικά του client αναπτύσσονται με τη χρήση των εργαλείων CSS, HTML και Javascript. Επίσης από τη στιγμή που αυτά υπάρχουν ήδη μέσα στο πρόγραμμα περιήγησης που χρησιμοποιεί ο χρήστης δεν χρειάζεται να γίνουν διορθώσεις και αλλαγές στο λειτουργικό σύστημα. Ο client ουσιαστικά παρουσιάζει το περιβάλλον μέσα από το οποίο ο χρήστης αλληλεπιδρά με την διαδικτυακή εφαρμογή.
- **Server Component** – Ο server τώρα με τη σειρά του μπορεί να χτιστεί με τον συνδυασμό πολλών προγραμματιστικών γλωσσών και πλαισίων (frameworks), τα οποία περιλαμβάνουν τη Java, το .Net, το NodeJS, τη PHP, τη Python, και τη Ruby on Rails. Ο server έχει τουλάχιστον δύο μέρη, την λογική της εφαρμογής (application logic) και τη βάση δεδομένων. Τι είναι η Application logic; Είναι μια λογική η οποία καθορίζει τα πραγματικά χαρακτηριστικά της εφαρμογής μας.

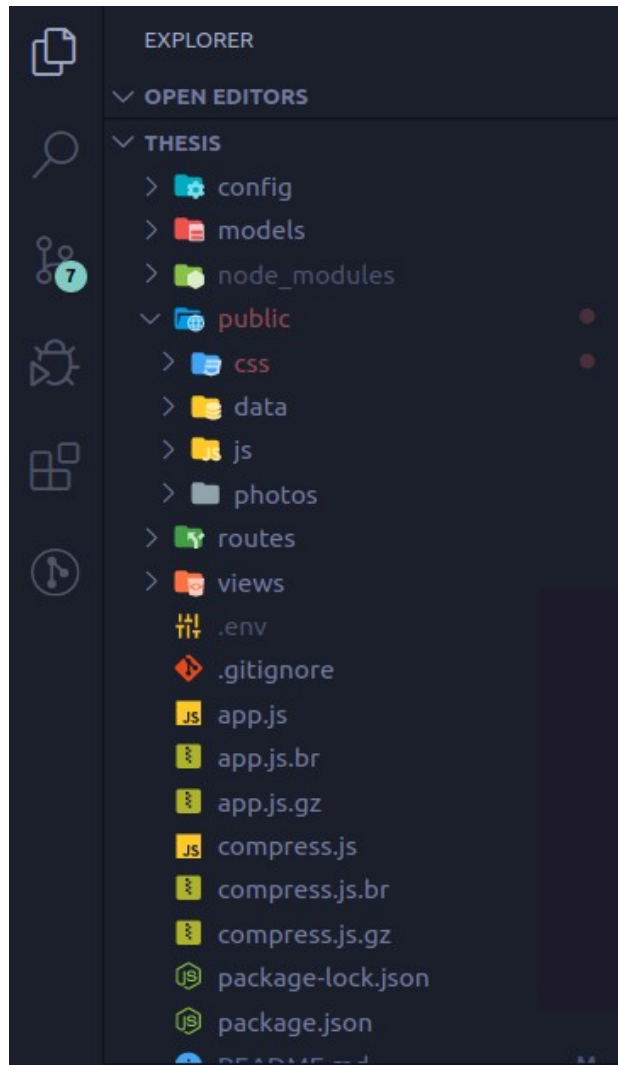
Δ.1.2.3 Αρχιτεκτονική της Διαδικτυακής Εφαρμογής Nstocks

Ας μιλήσουμε τώρα πιο συγκεκριμένα για την αρχιτεκτονική μιας διαδικτυακής εφαρμογής με τη χρήση του NodeJS και Express. Με τη βοήθεια αυτών των εργαλείων η αρχιτεκτονική της διαδικτυακής εφαρμογής είναι:



Εικόνα 4: Αρχιτεκτονική διαδικτυακής Εφαρμογής με Express

Στο διάγραμμα παραπάνω φαίνεται η πορεία των δεδομένων και των πραγμάτων που πρέπει να εκτελεστούν όταν χειρίζεται ένα HTTP αίτημα / απάντηση η εφαρμογή. Οι δρομολογητές (routes) προωθούν τα αιτήματα (requests) στις κατάλληλες συναρτήσεις ελέγχου (controller functions). Οι controller functions παίρνουν τα αιτούμενα δεδομένα και δημιουργώντας μια HTML σελίδα, εμφανίζουν τα δεδομένα, και τα επιστρέφουν στο χρήστη για να μπορέσει να τα δει στον περιηγητή του. Τα views (templates) χρησιμοποιούνται από τις controller συναρτήσεις για να προσφέρουν τα δεδομένα στο χρήστη.



Εικόνα 5: Δομή διαδικτυακής εφαρμογής Nstocks

- **app.js** – Το αρχείο αυτό δημιουργεί ένα Express αντικείμενο (που ονομάζεται app), όπου θα εγκαταστήσει την εφαρμογή με διάφορες ρυθμίσεις και ενδιάμεσα λογισμικά, και μετά θα εξάγει την εφαρμογή από την ενότητα (module).
- **models** - Περιλαμβάνει τα μοντέλα της Βάσης Δεδομένων.
- **config** – Περιλαμβάνει τα ευαίσθητα αρχεία με τα οποία μπορεί να συνδεθεί ο server με τη βάση δεδομένων αλλά και τα αρχεία που συνδέουν το χρήστη με την εφαρμογή και του παρέχουν και τα κατάλληλα δικαιώματα.

- **node-modules** - περιλαμβάνει τις απαραίτητες ενότητες που πρέπει να εγκατασταθούν για να λειτουργήσει μια διαδικτυακή εφαρμογή.
- **public** - Ο φάκελος που περιλαμβάνει όλα τα στατικά αρχεία όπως CSS, Javascript, CSV και διάφορες φωτογραφίες που χρησιμοποιούνται.
- **routes** - Σε αυτό το φάκελο περιλαμβάνονται όλες οι σελίδες που έχει η εφαρμογή. Δηλαδή αν πάει κανείς στη σελίδα Stocks τότε θα καλέσει το αρχείο index.js που βρίσκεται στο φάκελο routes και θα ανοίξει τη σελίδα /stocks. Με αυτό το τρόπο σπάει σε σελίδες η εφαρμογή και γίνεται ευκολότερα διαχειρίσιμη. Αρχικά, φορτώνει το Express module και το χρησιμοποιεί για να πάρει το express.Router αντικείμενο. Μετά ορίζει ένα route σε αυτό το αντικείμενο και τέλος εξάγει το route από το module (αυτή είναι η διαδικασία που επιτρέπει το αρχείο να ενσωματωθεί στο αρχείο App.js). Ο δρομολογητής δηλώνει μια callback συνάρτηση η οποία θα καλείται όταν γίνεται ένα HTTP GET αίτημα. Το πρότυπο που θα ταιριάζει είναι ο ορισμένος δρομολογητής, όταν η ενότητα εισαχθεί στο ('/stocks') συν ότι έχει οριστεί μέσα σε αυτό το αρχείο ('/'). Με άλλα λόγια, αυτός ο δρομολογητής θα χρησιμοποιηθεί όταν η διεύθυνση URL του / stocks / θα ληφθεί.
- **views** – Τα views (πρότυπα) βρίσκονται στο φάκελο /views (ορίζονται στο app.js) και έχουν την κατάληξη αρχείου .ejs. Η μέθοδος Response.render () χρησιμοποιείται για να προσφέρει ένα ορισμένο πρότυπο μαζί με τις τιμές των μεταβλητών που περνούν σε ένα αντικείμενο και στη συνέχεια στέλνει το αποτέλεσμα σαν απάντηση.

Δ.1.3 Εγκατάσταση προγραμματιστικού περιβάλλοντος

Η παρούσα πτυχιακή εκπονήθηκε σε λειτουργικό σύστημα Linux και συγκεκριμένα στα Ubuntu οπότε τα βήματα που θα δείξουμε παρακάτω θα αφορούν το συγκεκριμένο σύστημα και ίσως τα

βήματα εγκατάστασης του περιβάλλοντος ανάπτυξης να διαφέρει σε άλλα λειτουργικά συστήματα. Για αρχή τα απαραίτητα εργαλεία που θα χρειαστούν για την ανάπτυξη αυτής της διαδικτυακής εφαρμογή είναι το Git και το NodeJS. Παρακάτω θα αναλυθούν τι είναι το καθένα και πώς μπορεί κάποιος να τα εγκαταστήσει.

Α.1.3.1 Git

Το Git είναι ένα σύστημα ελέγχου εκδόσεων (λέγεται και σύστημα ελέγχου αναθεωρήσεων ή σύστημα ελέγχου πηγαίου κώδικα) με έμφαση στην ταχύτητα, στην ακεραιότητα και στην υποστήριξη για κατανεμημένες μη γραμμικές ροές εργασίας [8]. Έτσι, το Git είναι ένα σύστημα ελέγχου έκδοσης, αλλά τι σημαίνει αυτό; Όταν ένας προγραμματιστής αναπτύσσει κάτι (μια εφαρμογή), κάνει συνέχεια αλλαγές στον κώδικα. Ουσιαστικά το Git το χρειάζεται στον υπολογιστή του για να μπορεί να επικοινωνεί με το repository (αποθετήριο) του στο Github και να ανεβάζει τον κώδικα για να μπορεί να τον έχει αποθηκευμένο κάπου που θα είναι ασφαλής, αλλά και με αυτό το τρόπο θα μπορεί να ανατρέξει σε προηγούμενες εκδόσεις του κώδικα του. Τα συστήματα ελέγχου έκδοσης διατηρούν τις αναθεωρήσεις αυτές, αποθηκεύοντας τις τροποποιήσεις σε ένα κεντρικό αποθετήριο. Αυτό επιτρέπει στους προγραμματιστές να συνεργάζονται εύκολα, καθώς μπορούν να κατεβάσουν μια νέα έκδοση λογισμικού, να κάνουν αλλαγές και να ανεβάζουν την πιο πρόσφατη αναθεώρηση. Όσο μεγαλώνει η εφαρμογή που αναπτύσσεται τόσο πιο εύκολα θα γίνει κατανοητό ότι υπάρχουν μεγαλύτερες πιθανότητες για λάθη στο κώδικα τα οποία είναι αναπόφευκτα. Με αυτόν τον τρόπο θα μπορέσει κανείς να ανατρέξει σε προηγούμενες εκδόσεις του κώδικα του πριν κάνει αυτά τα λάθη και η εφαρμογή του είναι ακόμα λειτουργική. Αυτό θα γλυτώσει χρόνο από το κάθε προγραμματιστή και αυτομάτως θα του κάνει τη ζωή πιο εύκολη. Ομοίως, οι άνθρωποι που δεν έχουν καμία σχέση με την ανάπτυξη έργου μπορούν να κατεβάσουν τα αρχεία και να τα χρησιμοποιήσουν. Οι περισσότεροι χρήστες του λειτουργικού συστήματος Linux, πρέπει να είναι εξοικειωμένοι με αυτές τις διαδικασίες, όπως η χρήση του Git, του Subversion ή κάποιας άλλης παρόμοιας μεθόδου οι οποία είναι γνωστή για τη λήψη των απαραίτητων αρχείων. Το Git είναι το προτιμώμενο σύστημα ελέγχου εκδόσεων των περισσότερων προγραμματιστών, καθώς έχει πολλά πλεονεκτήματα σε σχέση με τα άλλα διαθέσιμα συστήματα. Αποθηκεύει αποτελεσματικότερα τις αλλαγές αρχείων και βελτιώνει την ακεραιότητα των αρχείων. Επίσης μέσα από το Github θα γίνει πιο εύκολη η διαδικασία διαμοιρασμού του κώδικα του στο διαδίκτυο αλλά και σε συνδυασμό με το Heroku μπορεί να ανεβάσει στο διαδίκτυο την εφαρμογή

του, η οποία θα είναι έτοιμη για χρήση από τον οποιοδήποτε. Το τελευταίο κομμάτι θα αναλυθεί καλύτερα παρακάτω.

Δ.1.3.1.1 Εγκατάσταση Git

Ο πιο εύκολος τρόπος και αυτός που συνιστάται είναι να εγκατασταθεί το Git με τη χρήση του apt πακέτου εργαλείου διαχείρισης του Ubuntu. Τα βήματα που πρέπει να ακολουθήσει κανείς αναλύονται παρακάτω:

1. Αρχικά θα γίνει ενημέρωση το packet index :

```
$ sudo apt update
```

Εικόνα 6: Ενημέρωση apt πακέτου

2. Στη συνέχεια θα τρέξει η ακόλουθη εντολή για την εγκατάσταση του Git:

```
$ sudo apt install git
```

Εικόνα 7: Εγκατάσταση git

3. Αφού επιβεβαιωθεί ότι έγινε η εγκατάσταση του Git τρέχοντας την ακόλουθη εντολή μπορεί κανείς να δει την έκδοση του git που εγκαταστάθηκε στην εφαρμογή του:

```
$ git --version
```

Εικόνα 8: Έλεγχος έκδοσης git

Στη παρούσα φάση, που εκπονείται αυτή η πτυχιακή η έκδοση Git είναι η ακόλουθη:

Output

```
git version 2.17.1
```

Εικόνα 9: Έκδοση git που χρησιμοποιήθηκε στη παρούσα πτυχιακή

Δ.1.3.2 NodeJS & NPM

Απαραίτητο εργαλείο επίσης είναι το NodeJS και κατ' επέκταση το NPM (Node Package Manager). Το NodeJS αναλύθηκε πιο πάνω οπότε τώρα θα αναλυθεί το npm. Το npm είναι το μεγαλύτερο παγκοσμίως μητρώο λογισμικών (Software Registry). Προγραμματιστές χρησιμοποιούν το npm για να μοιράζονται το λογισμικό, αλλά και οργανισμοί χρησιμοποιούν αυτό το εργαλείο για να διαχειριστούν την ιδιωτική ανάπτυξη κώδικα. Το npm είναι δωρεάν και μπορεί να το χρησιμοποιήσει όποιος θέλει, επίσης το npm περιλαμβάνει ένα CLI (Command Line Client) το οποίο μπορεί να χρησιμοποιηθεί για την εγκατάσταση και διαχείριση λογισμικού. Τα πακέτα που εγκαθιστώνται με τη χρήση του npm αυτομάτως προσδιορίζονται στο αρχείο package.json. Επίσης με το npm μπορούμε να διαχειριστούμε τις εξαρτήσεις (dependencies), όπου το npm μπορεί να τις εγκαταστήσει με μια μόνο εντολή. Τέλος ο κάθε προγραμματιστής έχει την δυνατότητα να διαμοιράσει το λογισμικό του με το εργαλείο npm, απλά πρέπει να εγγραφεί στη πλατφόρμα του και στη συνέχεια με τη χρήση λίγων εντολών, το λογισμικό του να βρίσκεται στο διαδίκτυο, έτοιμο να το κατεβάσει ο οποιοσδήποτε μέσω του npm και να το χρησιμοποιήσει. Αφού παρουσιάστηκε πόσο χρήσιμο εργαλείο είναι το npm, στη συνέχεια θα αναλυθεί η διαδικασία εγκατάστασης για να μπορέσει ο εκάστοτε προγραμματιστής να το χρησιμοποιήσει στη διαδικτυακή εφαρμογή που θα αναπτύξει.

Δ.1.3.2.1 Εγκατάσταση NodeJS & NPM

1. Τα NodeJS και npm πακέτα είναι διαθέσιμα από τα Ubuntu 18.04 αποθετήρια. Για να εγκατασταθούν αυτά τα πακέτα θα χρειαστεί να τρέξει κανείς τις ακόλουθες εντολές :

```
$ sudo apt update  
$ sudo apt install nodejs
```

Εικόνα 10: Εγκατάσταση npm και NodeJS

2. Αφού γίνει η ενημέρωση του packet index και η εγκατάσταση του NodeJS ελέγχεται αν όλα πήγαν σωστά και έχει εγκατασταθεί το πακέτο :

```
$ nodejs --version
```

Εικόνα 11: Έλεγχος της έκδοσης NodeJS που εγκαταστάθηκε

```
Output  
v8.10.0
```

Εικόνα 12: Έκδοση NodeJS που είναι εγκατεστημένη στο λειτουργικό σύστημα που εκπονείται η παρούσα πτυχιακή.

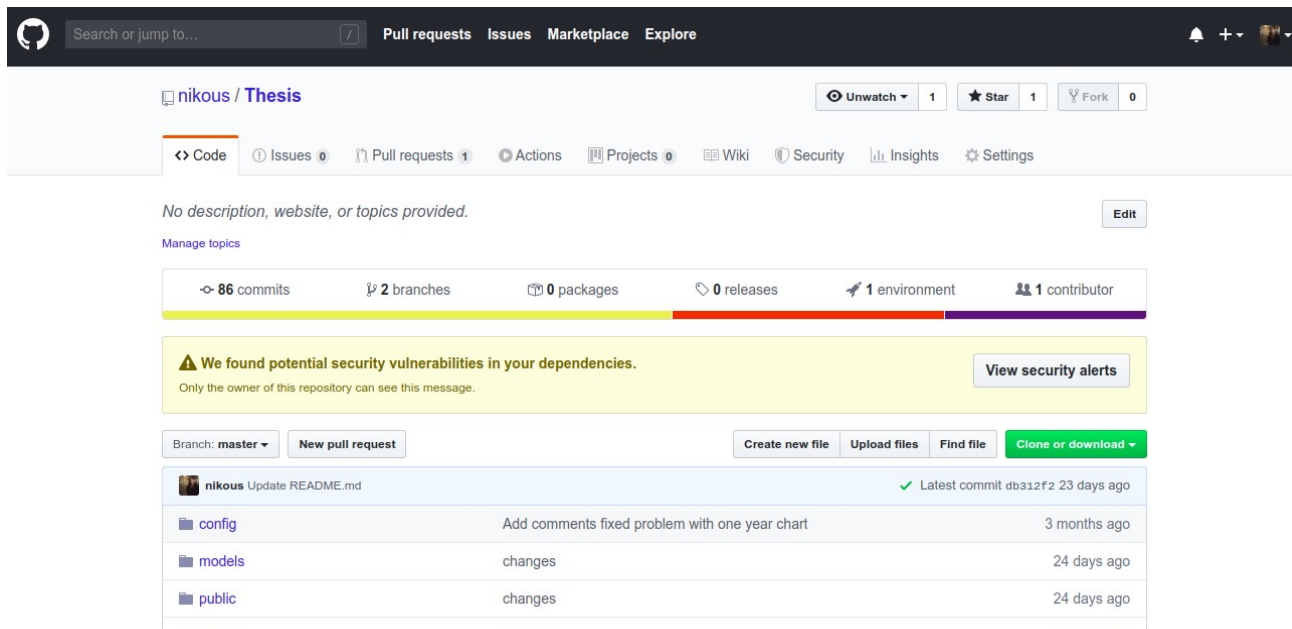
Η παραπάνω εγκατάσταση είναι μία από τις τρεις που μπορεί να πραγματοποιήσει ο προγραμματιστής για να κάνει εγκατάσταση το πακέτο στον υπολογιστή του. Οι άλλοι δυο είναι με τη χρήση του NVM (Node Version Manager) και με τη χρήση του NodeSource. Ο τρόπος που θα διαλέξει κανείς εξαρτάται από τις απαιτήσεις και προτιμήσεις που έχει. Στη παρούσα πτυχιακή χρησιμοποιήθηκε η εγκατάσταση του NodeJS από το αποθετήριο των Ubuntu και γιαυτό και περιγράφηκε.

Δ.1.3.3 GitHub

Όπως αναφέρθηκε και πιο πάνω, το Git είναι ένα σύστημα ελέγχου έκδοσης, από τα καλύτερα που κυκλοφορούν αυτή τη στιγμή. Αλλά τι το κάνει τόσο ξεχωριστό; Το Git είναι ένα εργαλείο γραμμών εντολών, αλλά το κέντρο γύρω από το οποίο όλα τα πράγματα που αφορούν το Git περιστρέφονται είναι το Hub όπου οι προγραμματιστές αποθηκεύουν τα σχέδιά τους και επικοινωνούν με άλλους προγραμματιστές. Γι αυτό θα αναλυθούν συγκεκριμένα τα κομμάτια που απαρτίζουν το GitHub και τα οποία χρησιμοποιούνται για την αποθήκευση και τον διαμοιρασμό της παρούσας πτυχιακής.

Δ.1.3.3.1 Αποθετήριο

Το αποθετήριο (repository) είναι ένα μέρος όπου βρίσκονται όλα τα αρχεία για ένα συγκεκριμένο πρότζεκτ. Κάθε πρότζεκτ όπως και η διαδικτυακή εφαρμογή Nstocks έχει το δικό της αποθετήριο (repository) και μπορεί να έχει πρόσβαση σε αυτό, ο καθένας μέσα από ένα ξεχωριστό σύνδεσμο (URL). Ο σύνδεσμος ο οποίος οδηγεί στο αποθετήριο αυτής της εφαρμογής είναι <https://github.com/nikous/Thesis>.



Εικόνα 13: Αποθετήριο διαδικτυακής εφαρμογής Nstocks στο GitHub

Για να δημιουργηθεί τώρα ένα καινούργιο αποθετήριο (repository), είναι σχετικά απλή διαδικασία. Ο προγραμματιστής πηγαίνει στο προφίλ του στο GitHub και πατάει new repository. Θα εμφανιστεί η παρακάτω σελίδα.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

Repository name *

 nikous ▾

 /

Nstocks 

Great repository names are short and memorable. Need inspiration? How about **congenial-potato**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

 |

Add a license: **None** ▾ 

Create repository

Εικόνα 14: Δημιουργία αποθετηρίου στο GitHub

Διαλέγει το όνομα του αποθετηρίου μας και αν θέλει να είναι δημόσιο ή ιδιωτικό (μπορεί να το αλλάξει αργότερα αν θέλει) και δημιουργεί το αποθετήριο. Αφού δημιουργηθεί το αποθετήριο, στη συνέχεια θα ακολουθηθούν τα βήματα για να συνδεθεί ο υπολογιστής με αυτό.

1. Ο προγραμματιστής πρέπει να δημιουργήσει ένα φάκελο στον υπολογιστή του στον οποίο θα βρίσκεται η εφαρμογή του. Για να γίνει αυτό πρέπει να ανοίξει το τερματικό (terminal) και να πάει σε αυτό το φάκελο.

```
cd /path/to/your/existing/code  
git init
```

Εικόνα 15: Εντολή git init

2. Στη συνέχεια πρέπει να αντιγράψει το αποθετήριο που έχει δημιουργήσει. Για να γίνει αυτό πρέπει να εκτελεστεί η ακόλουθεί εντολή:

```
git clone <repo url>
```

Εικόνα 16: Αντιγραφή αποθετηρίου τοπικά στον υπολογιστή

Καθώς αναπτύσσεται η εφαρμογή ο σωστός προγραμματιστής θα θέλει να αποθηκεύει και να ενημερώνει το αποθετήριο του συνεχώς. Για να γίνει αυτό θα πρέπει να πληκτρολογήσει τις παρακάτω εντολές:

```
git add README.md  
git commit -m "first commit"
```

Εικόνα 17: Εντολές προσθήκης αλλαγών στο κώδικα του αποθετηρίου μας

Το README.md θα αντικατασταθεί με οποιοδήποτε αρχείο δημιουργήθηκε ή μπορεί απλά να μπει στη θέση του μια τελεία (.) και το Git θα προσθέσει από μόνο του τα νέα αρχεία που δημιουργήθηκαν. Με τη δεύτερη εντολή ουσιαστικά παραπέμπονται οι αλλαγές στο αποθετήριο μαζί με ένα σχόλιο για να μπορεί αργότερα ο προγραμματιστής να ξεχωρίσει τις παραπομπές (commits) που έγιναν.

```
git push -u origin master
```

Εικόνα 18: Εντολή προσθήκης αλλαγών στο αποθετήριο μας

Τέλος με τη τρίτη εντολή γίνονται οι αλλαγές στο αποθετήριο και αν πλοηγηθεί σε αυτό μπορεί να τις δει.

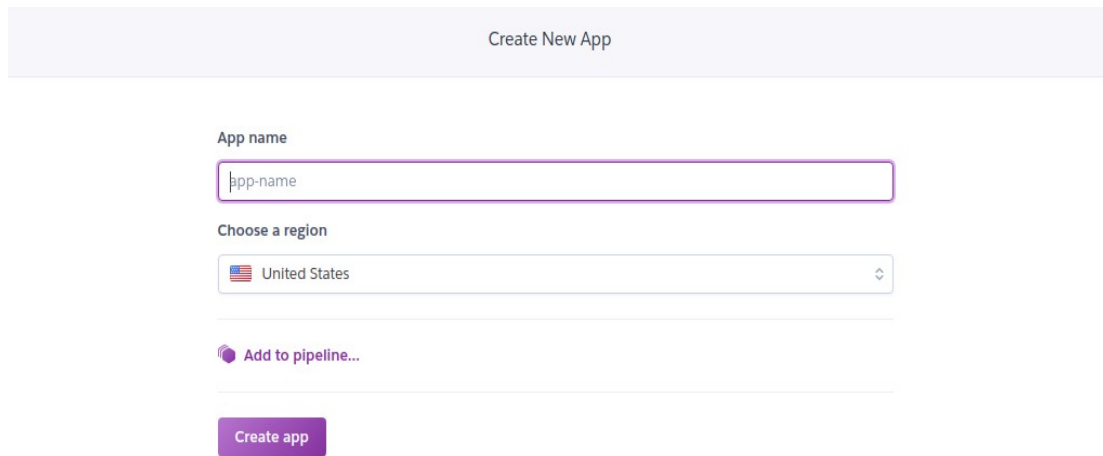
Δ.1.3.4 Heroku

Για να μπορέσει ο αναγνώστης αυτής της πτυχιακής να κατανοήσει τι είναι το Heroku και γιατί χρησιμοποιείται πρέπει πρώτα να γίνει μια σύντομη αναφορά στο Cloud computing. Είναι η διαθεσιμότητα πόρων του συστήματος υπολογιστών, όπως η αποθήκευση δεδομένων και η υπολογιστική ισχύς, χωρίς να υπάρχει η άμεση ενεργή διαχείριση από τον χρήστη. Ο όρος Cloud computing χρησιμοποιείται γενικά για να περιγραφούν κέντρα δεδομένων που είναι διαθέσιμα σε πολλούς χρήστες, μέσω του διαδικτύου. Υπάρχουν πολλές υπηρεσίες Cloud computing, για παράδειγμα υπάρχουν οι Infrastructure as a Service (IaaS) όπως το AWS, που επιτρέπουν στις επιχειρήσεις να μεταφέρουν τις λειτουργίες τους μακριά από τα παραδοσιακά κέντρα δεδομένων. Από την άλλη πλευρά, υπάρχουν οι Platform as Service (PaaS) πάροχοι όπως το Heroku, οι οποίοι προσφέρουν αφηρημένα περιβάλλοντα μέσα στα οποία οι προγραμματιστές μπορούν απλά να “πετάξουν” τον κώδικα τους και να αφήσουν την πλατφόρμα να φροντίσει για τις λεπτομέρειες. Μέσα από αυτή τη διαδικασία ο προγραμματιστής θα χάσει λίγο από τον έλεγχο της εφαρμογής του, αλλά θα κερδίσει την απλότητα και την ταχύτητα σε αντάλλαγμα. Το Heroku χρησιμοποιεί το Git για τη διαχείριση των εφαρμογών, οπότε το μόνο που χρειάζεται για να ανεβάσει κανείς την εφαρμογή του στο Heroku είναι να “σπρώξει” το αποθετήριο Git στους διακομιστές του. Παρακάτω θα αναλυθεί λεπτομερώς αυτή τη διαδικασία.

Δ.1.3.4.1 Σύνδεση με το Heroku

Αφού έχει εγκατασταθεί το git, έχει δημιουργηθεί το αποθετήριο της εφαρμογής στο GitHub και το έχει συνδεθεί με τον φάκελο της εφαρμογής που βρίσκεται τοπικά στον υπολογιστή, σειρά έχει τώρα να συνδεθεί με το Heroku ακολουθώντας τα παρακάτω βήματα:

1. Αρχικά πρέπει να δημιουργηθεί μια νέα εφαρμογή στο Heroku



Εικόνα 19: Δημιουργία εφαρμογής στο Heroku

2. Αφού δημιουργηθεί ο διακομιστής της εφαρμογής στο Heroku, τώρα πρέπει να εγκατασταθεί στον υπολογιστή εκτελώντας την ακόλουθη εντολή :

```
$ curl https://cli-assets.heroku.com/install-ubuntu.sh | sh
```

Εικόνα 20: Εγκατάσταση Heroku τοπικά στον υπολογιστή

3. Για να ελεγχθεί αν έγινε η εγκατάσταση σωστά εκτελείται η ακόλουθη εντολή η οποία δείχνει την έκδοση του Heroku που εγκαταστάθηκε :

```
$ heroku --version  
heroku/7.0.0 (darwin-x64) node-v8.0.0
```

Εικόνα 21: Εντολή ελέγχου έκδοσης Heroku που είναι εγκατεστημένη τοπικά στον υπολογιστή μας

4. Στη συνέχεια για να συνδεθεί ο υπολογιστής με το Heroku εκτελείται η ακόλουθη εντολή :

```
$ heroku login
heroku: Press any key to open up the browser to login or q to exit
> Warning: If browser does not open, visit
> https://cli-auth.heroku.com/auth/browser/**
heroku: Waiting for login...
Logging in... done
Logged in as me@example.com
```

Εικόνα 22: Εντολή σύνδεσης του υπολογιστή με τον λογαριασμό στο Heroku

5. Στη συνέχεια εκτελείται η εντολή για να καταλάβει ο υπολογιστής σε ποιον διακομιστή θα ανεβάσει τα αρχεία ή θα κάνει τις αλλαγές :

```
$ heroku git:remote -a nstockapplication
```

Εικόνα 23: Εντολή για να καταλάβει σε πιο διακομιστή του Heroku απευθύνεται ο προγραμματιστής

6. Τέλος εκτελούνται οι ακόλουθες εντολές για να ανεβεί τον κώδικα στο Heroku :

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

Εικόνα 24: Εντολές για να ανεβούν οι αλλαγές στο Heroku

Α.1.3.5 MongoDB

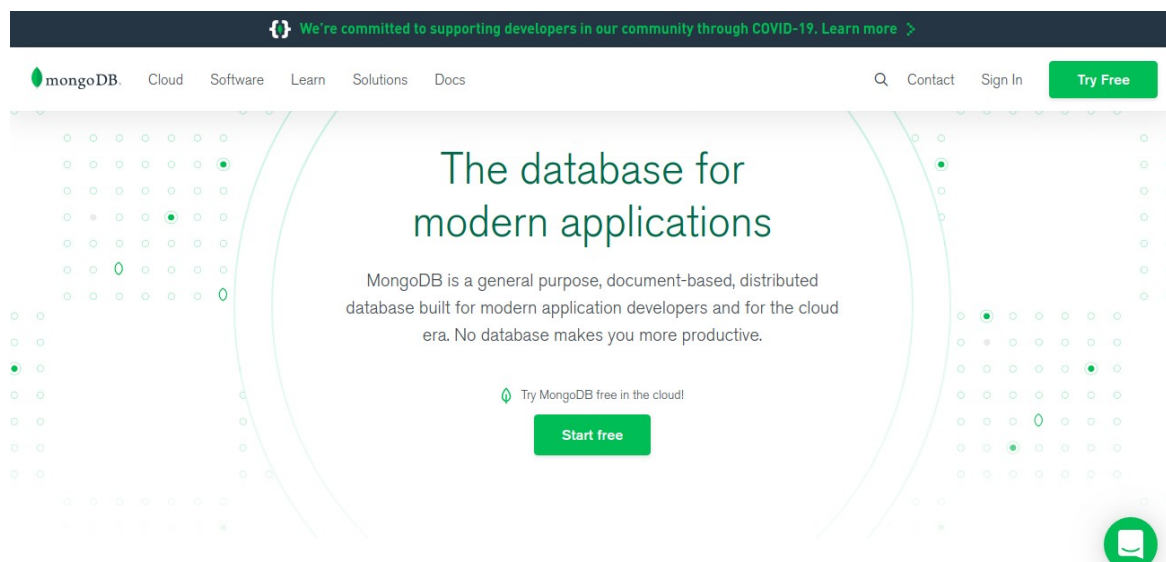
Για την ανάπτυξη αυτής της διαδικτυακής εφαρμογής θα χρειαστεί και μια βάση δεδομένων η οποία θα συνδέεται με την εφαρμογή και θα αποθηκεύονται σε αυτή τα στοιχεία των χρηστών που συνδέονται σε αυτή καθώς και κάποιες πληροφορίες για τις μετοχές που ο χρήστης αποφασίζει να παρακολουθεί πιο συχνά. Για να συμβεί αυτό θα χρησιμοποιηθεί η βάση δεδομένων MongoDB η

οποία είναι μια NoSQL βάση δεδομένων και συνδυάζεται πολύ καλά με το NodeJS και Express για την ανάπτυξη μιας διαδικτυακής εφαρμογής. Έχοντας αναλυθεί πιο πάνω η βάση δεδομένων MongoDB, παρακάτω θα αναλυθούν πιο τεχνικά κομμάτια, όπως ο τρόπος που δημιουργείται η βάση δεδομένων, καθώς και πως συνδέεται με τη διαδικτυακή εφαρμογή.

Δ.1.3.5.1 Σύνδεση με το MongoDB

Παρακάτω θα αναλυθούν τα βήματα που πρέπει να ακολουθήσει ο προγραμματιστής για να συνδεθεί στο MongoDB

1. Αρχικά πρέπει να δημιουργήσει λογαριασμό στη σελίδα του [MongoDB](#)



Εικόνα 25: Αρχική σελίδα της βάσης δεδομένων MongoDB

2. Στη συνέχεια θα δημιουργήσει ένα νέο πρότζεκτ. Για να γίνει αυτό πρέπει να πατήσει στο Next και στην επόμενη σελίδα να πατήσει Create project.

Create a Project

Name Your Project

Add Members

Next

Name Your Project
Project names have to be unique within the organization (and other restrictions).

Project Name

CancelNext

Εικόνα 26: Δημιουργία νέου πρότζεκτ στη βάση δεδομένων MongoDB

- Μετά έχοντας δημιουργήσει το πρότζεκτ, πρέπει να φτιάξει ένα νέο Cluster μέσα σε αυτό

Cloud Provider & Region

AWS, N. Virginia (us-east-1) ▾

aws

Google Cloud Platform

Azure

★ Recommended region ⓘ

NORTH AMERICA

🇺🇸 N. Virginia (us-east-1) ★

🇺🇸 Oregon (us-west-2) ★

EUROPE

🇮🇪 Ireland (eu-west-1) ★

🇩🇪 Frankfurt (eu-central-1) ★

ASIA

🇸🇬 Singapore (ap-southeast-1) ★

🇮🇳 Mumbai (ap-south-1)

AUSTRALIA

🇦🇺 Sydney (ap-southeast-2) ★

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage)
Encrypted >

Additional Settings

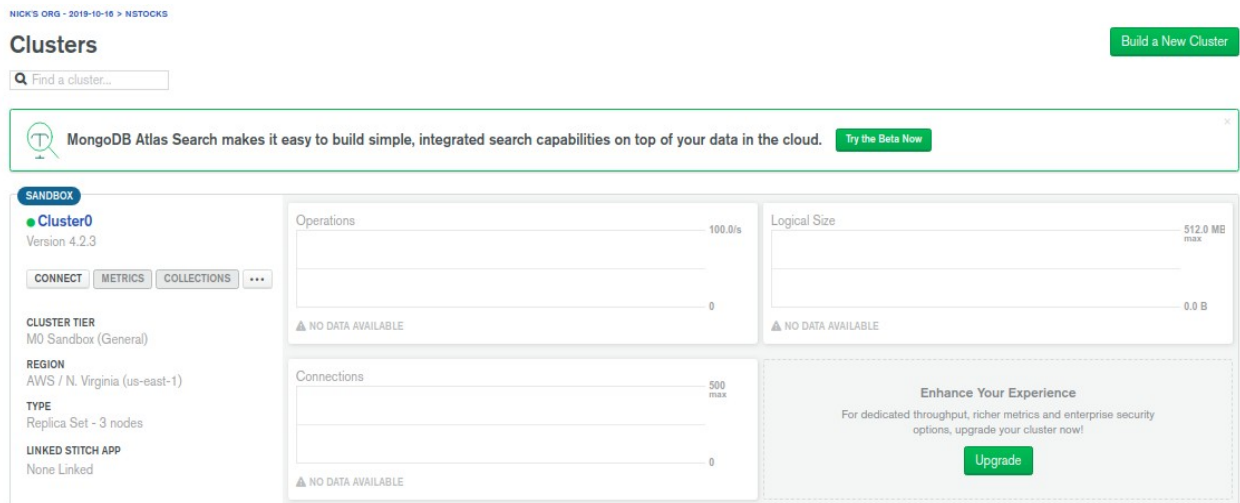
MongoDB 4.2, No Backup >

Cluster Name

Cluster0 >

Εικόνα 27: Δημιουργία cluster στη Βάση Δεδομένων MongoDB

4. Πρέπει να πατήσει το connect



Εικόνα 28: Σύνδεση στο cluster που δημιουργήθηκε στο MongoDB

5. Στη συνέχεια προσθέτει για IP address 0.0.0.0 και με αυτό το τρόπο θα μπορεί να έχει πρόσβαση στη βάση δεδομένων από παντού. Στη συνέχεια πρέπει να δημιουργηθεί ένας MongoDB χρήστης.

Connect to Cluster0

Setup connection security

Choose a connection method

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

You can't connect yet. Set up your firewall access and user security permission below.

1 Whitelist a connection IP address

IP Address	Description (Optional)
0.0.0.0	An optional comment describing this entry
Cancel Add IP Address	

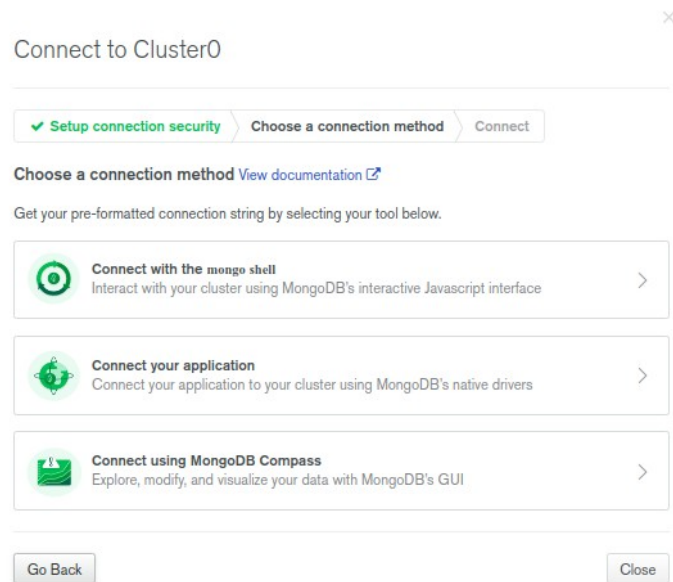
2 Create a MongoDB User

This first user will have [atlasAdmin](#) permissions for this project.
Keep your credentials handy, you'll need them for the next step.

Username	Password
ex. dbUser	ex. dbUserPassword Autogenerate Secure Password SHOW
Create MongoDB User	

Εικόνα 29: Ρυθμίσεις σύνδεσης στο cluster

6. Στη συνέχεια πρέπει να πατήσει Connect your application



Εικόνα 30: Σύνδεση στο cluster

7. Τέλος πρέπει να πάρει το link, που είναι αυτό που θα εισαχθεί στην εφαρμογή, με το οποίο θα επικοινωνούν η βάση δεδομένων και η εφαρμογή. Προς το παρόν δεν θα αναλυθεί άλλο η βάση δεδομένων, αλλά θα αναλυθεί λεπτομερώς παρακάτω. Η βάση δεδομένων και η επικοινωνία με την εφαρμογή είναι ένα πολύ σημαντικό κομμάτι και πρέπει να δοθεί μεγάλη έμφαση σε αυτό για να επιτύχουμε τη βέλτιστη λειτουργία.

Connect to Cluster0

✓ Setup connection security ✓ Choose a connection method Connect

1 Select your driver and version

DRIVER VERSION

Node.js 3.0 or later

2 Add your connection string into your application code

Connection String Only Full Driver Example

mongodb+srv://nickous:<password>@cluster0-4ky0e.mongodb.net/t Copy

Replace <password> with the password for the user, nickous, and ensure all special characters are URL encoded.

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back Close

Εικόνα 31: Σύνδεσμος σύνδεσης εφαρμογής με τη βάση που δημιουργήσαμε

Δ.1.3.6 Packages

Προτού περιγραφεί η διαδικασία της εγκατάστασης των πακέτων θα γίνει μια σύντομη αναφορά για αυτά. Το `npm` αρχείο περιλαμβάνει πακέτα (`packages`), πολλά από τα οποία είναι και `Node modules` ή περιλαμβάνουν `Node modules`. Οπότε τι είναι το πακέτο; Είναι ένα αρχείο ή ένας κατάλογος που χαρακτηρίζεται από ένα αρχείο `package.json`. Ένα πακέτο πρέπει να περιλαμβάνει ένα `package.json` αρχείο έτσι ώστε να μπορεί να δημοσιευτεί στο `npm` αρχείο.

Ένα πακέτο όπως αναφέρθηκε μπορεί και να είναι `node module`, τι είναι όμως το `module`; Είναι ένα αρχείο ή κατάλογος αρχείων μέσα στο φάκελο `node_modules` όπου μπορεί να φορτωθεί από το `NodeJS` με τη συνάρτηση `require()`. Για να συμβεί αυτό, το `module` πρέπει να είναι ένα από τα παρακάτω :

- Ένας φάκελος με ένα `package.json` αρχείο που περιέχει ένα `main` πεδίο
- Ένας φάκελος με ένα `index.js` αρχείο μέσα σε αυτό
- Ένα `Javascript` αρχείο

Ένα παράδειγμα με το φορτώνεται ένα `node_module` στο `NodeJS` είναι αυτό:

```
var req = require('request')
```

Εικόνα 32: Παράδειγμα προσθήκης `node_module` στο `NodeJS`

όπου η μεταβλητή `req` αναφέρεται στο καλούμενο `module`.

Για να ξεκινήσει η ανάπτυξη της διαδικτυακής εφαρμογής το πρώτο πράγμα που πρέπει να γίνει είναι να δημιουργηθεί ένα `package.json` αρχείο μέσα στο φάκελο της εφαρμογής. Αφού εγκατασταθεί το `NodeJS` και `npm`, αυτό που πρέπει να γίνει στη συνέχεια είναι να εκτελεστεί η ακόλουθη εντολή:

```
npm init
```

Εικόνα 33: Εντολή δημιουργίας `package.json`

και στη συνέχεια να απαντήσει ο προγραμματιστής στις ερωτήσεις που του γίνονται.

Δ.1.3.6.1 Εγκατάσταση πακέτων

Το θετικό με τα πακέτα (packages) είναι ότι είναι πολύ εύκολο να ενσωματωθούν στο κώδικα , με αυτό το τρόπο διευκολύνουν πάρα πολύ το προγραμματιστή γιατί δεν χρειάζεται να φτιάξει κάτι το οποίο υπάρχει ήδη και διανέμεται δωρεάν στο ίντερνετ. Όπως θα δειχθεί και πιο κάτω, σε αυτή τη πτυχιακή χρησιμοποιήθηκαν πολλά πακέτα / node modules τα οποία φορτώθηκαν στο NodeJS με τη βοήθεια του npm (Node Package Manager). Όπως θα φαίνεται και στη φωτογραφία όλα τα πακέτα που φορτώθηκαν στην εφαρμογή υπάρχουν στο αρχείο package.json

```
2  "name": "pt",
3  "version": "1.0.0",
4  "description": "",
5  "main": "app.js",
6  "engines": {
7    "node": "12.11.1",
8    "npm": "6.12.0"
9  },
10 "scripts": {
11   "start": "node app.js",
12   "dev": "nodemon app.js",
13   "compress": "node compress.js"
14 },
15 "keywords": [],
16 "author": "",
17 "license": "ISC",
```

Εικόνα 34: Αρχείο package.json της διαδικτυακής εφαρμογής

```
18  "dependencies": {
19    "ajax": "0.0.4",
20    "bcryptjs": "^2.4.3",
21    "connect-flash": "^0.1.1",
22    "css-element-queries": "^1.2.3",
23    "dotenv": "^8.2.0",
24    "easytimer.js": "^4.0.2",
25    "ejs": "^2.7.1",
26    "express": "^4.17.1",
27    "express-ejs-layouts": "^2.5.0",
28    "express-session": "^1.17.0",
29    "http": "0.0.0",
30    "iconv": "^2.3.5",
31    "jQuery": "^1.7.4",
32    "jquery": "^3.4.1",
33    "jsdom": "^15.2.0",
34    "local": "^0.3.3",
35    "mongodb": "^3.4.1",
36    "mongoose": "^5.7.5",
37    "node-fetch": "^2.6.0",
38    "nodemailer": "^6.4.2",
39    "passport": "^0.4.0",
40    "passport-local": "^1.0.0",
41    "socket.io": "^2.3.0"
42  },
```

Εικόνα 35: Dependencies από το αρχείο package.json της διαδικτυακής εφαρμογής

```
43   "devDependencies": {
44     "brotli": "^1.3.2",
45     "nodemon": "^1.19.4",
46     "zlib": "^1.0.5"
47   }
48 }
49
```

Εικόνα 36: *devDependencies* από το αρχείο *package.json* της διαδικτυακής εφαρμογής

Όπως φαίνεται στο *package.json* υπάρχουν τα :

- **name** όπου θέτει το όνομα της εφαρμογής / πακέτου
- **version** που είναι η έκδοση της εφαρμογής
- **description** που είναι η περιγραφή της εφαρμογής
- **engines** όπου δείχνει ποια έκδοση του NodeJs χρησιμοποιούμε
- **main** όπου θέτει το αρχικό σημείο της εφαρμογής μας
- **scripts** όπου ορίζει ένα σύνολο node scripts που μπορούν να τρέξουν. Αυτά τα scripts τρέχουν από το τερματικό. Μπορεί κανείς να τα τρέξει γράφοντας **npm run XXX**. Στην εφαρμογή μπορεί να εκτελεστεί το **npm run dev** και το **npm start** για να ξεκινήσει ο διακομιστής (server) της εφαρμογής να λειτουργεί.
- **Dependencies** όπου ορίζει μια λίστα από npm πακέτα τα οποία είναι αποθηκευμένα σαν εξαρτήσεις. Κάθε φορά γίνεται εγκατάσταση κάποιου πακέτου χρησιμοποιώντας το npm ή το yarn αυτό το πακέτο αυτομάτως προστίθεται στη λίστα

```
npm install <PACKAGENAME>
yarn add <PACKAGENAME>
```

Εικόνα 37: Εντολή εγκατάστασης πακέτου με τη χρήση του *npm* ή *yarn*

- **devDependencies** όπου ορίζει μια λίστα από *npm* πακέτα τα οποία είναι αποθηκευμένα σαν αναπτυξιακές εξαρτήσεις. Διαφέρουν από τις απλές εξαρτήσεις επειδή πρέπει να εγκατασταθούν μόνο στο περιβάλλον ανάπτυξης, και δεν χρειάζονται στον κώδικα παραγωγής. Για να εγκατασταθεί ένα αναπτυξιακό πακέτο με τη βοήθεια του *npm* ή *yarn*, πρέπει να εκτελεστούν οι ακόλουθες εντολές.

```
npm install --dev <PACKAGENAME>
yarn add --dev <PACKAGENAME>
```

Εικόνα 38: Εντολή εγκατάστασης αναπτυξιακών εξαρτήσεων με τη βοήθεια του *npm* ή *yarn*

Δ.1.3 Αναλυτική παρουσίαση κώδικα

Δ.1.3.1 Δημιουργία server

Το βασικό αρχείο (*main*) το οποίο εκτελείται πρώτο και θέτει σε λειτουργία όλη την εφαρμογή είναι το **app.js** εκεί φορτώνονται όλα τα απαραίτητα *modules* και ξεκινάει ο *server* να λειτουργεί.

```
// Web application framework for Node.js to build servers
const express = require('express');

// Require it to use ejs
const expressLayouts = require('express-ejs-layouts'); 2.4K (gzipped: 1005)

// Mongoose provides a straight-forward,
// schema-based solution to model your application data.
const mongoose = require('mongoose');
    You, 3 months ago • Add comments to code
// flash is a special area of the session used for storing messages
const flash = require('connect-flash'); 9.6K (gzipped: 3.6K)
```

Κώδικα 1: Εισαγωγή node_modules στο αρχείο app.js

Όπως φαίνεται και στη φωτογραφία με αυτόν τον τρόπο ενσωματώνονται τα modules στον server. Αυτό γίνεται με την εντολή require(), η οποία δεν είναι μια από τις βασικές εντολές της JavaScript, αλλά μια εντολή του NodeJs, οπότε αν για παράδειγμα θέλει κανείς να έχει πρόσβαση στο express module που υπάρχει στο τοπικό φάκελο node_modules απλά γράφει την εντολή όπως φαίνεται στη παραπάνω φωτογραφία. Τώρα για να μπορέσει όμως κάποιος να έχει πρόσβαση σε αυτό το module πρέπει και αυτό με τη σειρά του να “εκθέσει” τον εαυτό του. Για να γίνει αυτό πρέπει να ανατεθεί στα exports ή module.exports. Στην ακόλουθη φωτογραφία φαίνεται ο τρόπος που γίνεται αυτό.

```
const Stock = mongoose.model('Stock', StockValueSchema);

module.exports = Stock;
```

Κώδικα 2: Export μοντέλο Stock από το αρχείο Stock.js

Στη συνέχεια αυτό που πρέπει να γίνει είναι να αρχίσει ο προγραμματιστής να χρησιμοποιεί το express και να δείξει στο server σε ποια port να “ακούει”.

```
// Start express
const app = express();

// This app starts a server and listens on port for con
const PORT = process.env.PORT || 1200;

var http = require('http').createServer(app);
http.listen(PORT, function () {
  console.log(`Server started on port ${PORT}`);
});
```

Κώδικας 3: Δείχνει σε ποια port να "ακούει" ο server στο app.js

Όπως φαίνεται με τη χρήση του http module δημιουργείται ο server και με τη μέθοδο listen() δείχνει σε ποια port να ξεκινήσει ο server. Τώρα αυτό που πρέπει να γίνει είναι να ενσωματωθεί στο αρχείο app.js και τα υπόλοιπα αρχεία κώδικα της διαδικτυακής εφαρμογής. Για να γίνει αυτό θα χρησιμοποιηθεί πάλι το require() άλλα και το app.use το οποίο τα θέτει σε λειτουργία.

```
// Routes
app.use('/', require('./routes/index.js'));
app.use('/users', require('./routes/users.js'));
app.use('/', require('./routes/StockValue.js'));

// Middleware for my static files
app.use('/public', express.static('public'));
```

Κώδικας 4: Δημιουργία διαδρομών (routes) στο app.js

Με αυτόν τον τρόπο ο προγραμματιστής έχει πρόσβαση και στα αρχεία που βρίσκονται στο φάκελο routes αλλά και στο φάκελο public. Μόλις δηλαδή μπει στη διαδικαστική του εφαρμογή θα ξεκινήσει το αρχείο app.js και θα αρχίσουν να φορτώνουν και τα υπόλοιπα αρχεία που βρίσκονται σε αυτούς τους φακέλους.

Δ.1.3.2 Σύνδεση με Βάση Δεδομένων

Στη συνέχεια αυτό που θα γίνει είναι να συνδεθεί ο server με τη βάση δεδομένων που φτιάχτηκε πιο πάνω. Στη βάση δεδομένων υπάρχει ένα link που είναι για αυτό ακριβώς το σκοπό. Για να επιτευχθεί η επικοινωνία αυτών των δύο θα χρησιμοποιείται το module Mongoose, το οποίο παρέχει μια απλή λύση για τη μοντελοποίηση των δεδομένων της εφαρμογής. Προσφέρει πολλά εργαλεία που βοηθούν στο να διαχειριστεί πιο εύκολα και γρήγορα τα δεδομένα του. Στη παρακάτω φωτογραφία φαίνεται ο κώδικας για τη δημιουργία της.

```
// Connect to MongoDB
mongoose

  .connect(
    db,
    { useNewUrlParser: true, useUnifiedTopology: true }
  )

  .then(() => {
    console.log('MongoDB Connected')

    const notification = mongoose.model('User');

    // Create a change stream. The 'change' event gets emitted when there's a
    // change in the database
    notification.watch({ 'fullDocument': 'updateLookup' }).
      on('change', function (change) {
        io.emit('new-notification', change);
      });

  }).catch(err => console.log(err));
```

Κώδικας 5: Σύνδεση με βάση δεδομένων στο αρχείο app.js

Το κομμάτι που έχει το μεγαλύτερο ενδιαφέρον είναι το .connect και αυτό είναι που δημιουργεί το δίαυλο επικοινωνίας του server και της βάσης δεδομένων. Όπου db είναι το link που δόθηκε από το MongoDB και βρίσκετε σε ξεχωριστό αρχείο (Keys.js) για την καλύτερη οργάνωση του κώδικα.

```
// DB Config
const db = require('./config/keys').mongoURI;
```

Κώδικας 6: Ενσωμάτωση του keys.js στο app.js

Α.1.3.3 Δημιουργία διαδρομών (routes)

Αφού τέθηκε σε λειτουργία ο server σειρά έχει να δημιουργηθούν οι σελίδες για τη διαδικτυακή εφαρμογή. Στο **index.js** αρχείο στο φάκελο routes θα δηλωθούν όλες αυτές οι διαδρομές. Ο τρόπος που θα το γίνει αυτό είναι με την κλάση `express.Router()` όπου με τη βοήθεια αυτής θα δηλωθούν οι διαδρομές. Στη παρακάτω φωτογραφία φαίνεται ο κώδικας που χρησιμοποιήθηκε για να επιτευχθεί αυτό.

```
// Index page
router.get('/', (req, res) => {

  // If User connected send user data to client side
  if (passport.authenticate) {

    res.render('Homepage', {

      user: req.user,
      error: true

    })

  }

  else {

    res.render('Homepage', {

      error: false,

    })

  }

});
```

Κώδικα 7: Λαμβάνει το αίτημα για να πλοηγηθεί στην αρχική σελίδα

Με το `router.get` ουσιαστικά δηλώνεται η διαδρομή χρησιμοποιώντας το router Express αντικείμενο το οποίο ανταποκρίνεται στις HTTP μεθόδους. Για παράδειγμα το `router.get` που φαίνεται στην πιο πάνω φωτογραφία ανταποκρίνεται στο GET αίτημα που γίνεται από αυτόν που περιηγείται στην εφαρμογή για να πάει στην αρχική της σελίδα. Πιο συγκεκριμένα οι μέθοδοι διαδρομών καθορίζουν μια συνάρτηση επανάκλησης (callback function) η οποία καλείται όταν η εφαρμογή λαμβάνει ένα αίτημα στη συγκεκριμένη διαδρομή (endpoint) και στη μέθοδο HTTP. Με άλλα λόγια

η εφαρμογή “ακούει” στο αίτημα το οποίο ταιριάζει στις συγκεκριμένες διαδρομές και μεθόδους, και όταν εντοπίζει ότι ταιριάζουν, καλεί τη συγκεκριμένη συνάρτηση επανάκλησης (callback function). Το Passport που φαίνεται στη φωτογραφία δεν θα αναλυθεί ακόμα. Ο πιο πάνω κώδικας θα ξαναχρησιμοποιηθεί ώστε να δημιουργηθούν διαδρομές και για τις υπόλοιπες σελίδες της εφαρμογής, δηλαδή το Homepage, το About, το stocks , τη σελίδα για να κάνει Register ο χρήστης στην εφαρμογή, τη σελίδα για να κάνει Login ο χρήστης αλλά και τη σελίδα Userpage.

Α.1.3.4 Εγγραφή χρήστη (Register)

Το επόμενο στάδιο όσο αφορά την διαδικτυακή εφαρμογή είναι να οριστούν ποιες σελίδες είναι προσβάσιμες από όλους και ποιες μόνο για τους εγγεγραμμένους χρήστες. Για να γίνει αυτό πρέπει να δημιουργηθεί ένας τρόπος με τον οποίο ο χρήστης θα δημιουργεί λογαριασμό στην εφαρμογή και θα συνδέεται. Για να γίνει αυτό πρέπει να δημιουργηθεί μια φόρμα όπου ο χρήστης θα βάζει τα στοιχεία του. Προτού όμως κατασκευαστεί αυτή η φόρμα, αυτό πρέπει να γίνει είναι μια συλλογή στην οποία θα αποθηκεύονται οι χρήστες στη βάση δεδομένων, από την οποία μετά θα ελέγχεται με το Passport που αναφέρθηκε πιο πάνω αν ο χρήστης υπάρχει σε αυτή. Γι’ αυτόν τον λόγο θα αναλυθεί το αρχείο **User.js** που βρίσκεται στο φάκελο models.

```
// User model
const UserSchema = new mongoose.Schema({

  name: {

    type: String,
    required: true
  },
  email: {

    type: String,
    required: true
  },
  password: {
```

Κώδικας 8: Δημιουργία μοντέλου User

Όπως φαίνεται στη φωτογραφία, δημιουργείται ένα μοντέλο που απαρτίζεται από όλα τα απαραίτητα στοιχεία που πρέπει να έχει ένας χρήστης της διαδικτυακής εφαρμογής για να μπορέσει να γίνει μέλος. Με αυτό το τρόπο δημιουργείται το Schema, το οποίο κατασκευάζει τη συλλογή user στη βάση δεδομένων και καθορίζει το σχήμα των εγγράφων μέσα στη συλλογή.

Αφού κατασκευαστεί η συλλογή στη βάση δεδομένων της διαδικτυακής εφαρμογής, το επόμενο βήμα είναι να δημιουργηθεί μια φόρμα με την οποία θα περνάνε τα στοιχεία του χρήστη στη βάση δεδομένων. Στη παρακάτω φωτογραφία φαίνεται ο κώδικας της φόρμας ο οποίος βρίσκεται στο αρχείο **register.js** στο φάκελο views:

```

<form action="/users/register" method="POST">

  <!-- Field for name -->
  <div class="form-group">

    <label for="name">Name</label>

    <input type="name" id="name" name="name" class="form-control" placeholder="Enter Name"
      value="<%= typeof name != 'undefined' ? name : '' %>" />

  </div>

  <!-- Field for email -->
  <div class="form-group">

    <label for="email">Email</label>

    <input type="email" id="email" name="email" class="form-control" placeholder="Enter Email"
      value="<%= typeof email != 'undefined' ? email : '' %>" />

  </div>

  <!-- Field for password -->
  <div class="form-group">

    <label for="password">Password</label>

    <input type="password" id="password" name="password" class="form-control"
      placeholder="Create Password" value="<%= typeof password != 'undefined' ? password : '' %>" />

  </div>

  <!-- Field for password 2 -->
  <div class="form-group">

    <label for="password2">Confirm Password</label>

    <input type="password" id="password2" name="password2" class="form-control"
      placeholder="Confirm Password"
      value="<%= typeof password2 != 'undefined' ? password2 : '' %>" />

  </div>

  <!-- Submit btn -->
  <button type="submit" class="btn btn-primary btn-block" id="registerBtn">

    Register

  </button>

</form>

```

Κώδικα 9: Φόρμα για εγγραφή του χρήστη στη διαδικτυακή εφαρμογή

The image shows a 'Register' form with a user icon and the title 'Register'. It contains four input fields: 'Name' (placeholder 'Enter Name'), 'Email' (placeholder 'Enter Email'), 'Password' (placeholder 'Create Password'), and 'Confirm Password' (placeholder 'Confirm Password'). Below the fields is a dark red 'Register' button. At the bottom, there is a link 'Have An Account? Login'.

Εικόνα 39: Φόρμα εισαγωγής στοιχείων χρήστη

Η φόρμα αποτελείται από τρία πεδία όπου ο χρήστης εισάγει το email τον κωδικό πρόσβασης και την επαλήθευση κωδικού πρόσβασης, και ένα κουμπί όπου με το πάτημα του γίνεται ένα αίτημα POST στο server. Παρατηρείται ότι στον κώδικα υπάρχει η παράμετρος `values`, η οποία περιλαμβάνει τον έλεγχο για το αν συμπληρώθηκαν σωστά τα πεδία από το χρήστη. Συγκεκριμένα απορρίπτει αν αυτό που εισήγαγε ο χρήστης είναι `undefined` ή κενό. Πάμε τώρα στο αρχείο `users.js` στο φάκελο `routes` για να αναλυθεί πως διαχειρίζεται ο server τα στοιχεία του χρήστη που του στάλθηκαν με το POST αίτημα από τη φόρμα. Προτού όμως περιγραφεί αυτή η διαδικασία, καλό είναι να αναφερθεί ότι ο server με παρόμοιο τρόπο με αυτό που δείχθηκε πιο πάνω, διαχειρίζεται το GET αίτημα του χρήστη για να δρομολογηθεί στη σελίδα `register`.

```
// Register Page
router.get('/register', forwardAuthenticated, (req, res) => res.render('register'));
```

Κώδικα 10: Δρομολόγηση χρήστη στη σελίδα `register` της εφαρμογής

Αφού λοιπόν συμπληρωθούν τα στοιχεία στη φόρμα ο server δέχεται το POST αίτημα με το ακόλουθο τρόπο:

```
// Register
router.post('/register', (req, res) => {

  const { name, email, password, password2 } = req.body;
  const notification = 'Start';
  const dot = 0;
  let errors = [];
```

Κώδικα 11: Διαχείριση στοιχείων που εισάγει ο χρήστης στη φόρμα register

Σημαντικό είναι να αναφερθεί ο τρόπος που δέχεται ο server τις τιμές από τα πεδία της φόρμας. Όπως φαίνεται δέχεται τις τιμές από το req.body και τις περνάει σε νέες μεταβλητές. Το request body, είναι τα δεδομένα που στέλνονται από τον client (πρόγραμμα περιήγησης) στην API και από προεπιλογή είναι undefined στο express, για αυτό και πρέπει να χρησιμοποιηθεί το middleware express.urlencoded().

```
// Bodyparser
app.use(express.urlencoded({ extended: true }));
```

Κώδικα 12: Χρήση middleware express.urlencoded() στην εφαρμογή

Στη συνέχεια αφού δεχτεί ο server τα στοιχεία του χρήστη ελέγχει για την εγκυρότητα του καθώς πρέπει να πληρούν κάποια χαρακτηριστικά, όπως ο κωδικός να είναι μεγαλύτερος από 6 χαρακτήρες, να ταιριάζει με το κωδικό που εισάχθηκε στο πεδίο επαλήθευσης κωδικού αλλά και γενικότερα να έχει συμπληρώσει ο χρήστης και τα τρία πεδία.

```
// conditions to register client
if (!name || !email || !password || !password2) {
  errors.push({ msg: 'Please enter all fields' });
}

if (password !== password2) {
  errors.push({ msg: 'Passwords do not match' });
}

if (password.length < 6) {
  errors.push({ msg: 'Password must be at least 6 characters' });
}
```

Κώδικα 13: Έλεγχος κωδικού που εισήγαγε ο χρήστης στην φόρμα στη σελίδα register

Εάν όλα είναι σωστά, θα πάει στο επόμενο στάδιο ελέγχου εγκυρότητας των στοιχείων που εισήγαγε ο χρήστης όπου αν κάποια από τα στοιχεία είναι λάθος η εφαρμογή ενημερώνει τον χρήστη με το κατάλληλο μήνυμα να εισάγει ξανά τα στοιχεία του, ενώ αν όλα είναι σωστά ελέγχει αν το email του χρήστη υπάρχει στη βάση δεδομένων. Αν εντοπίσει τον χρήστη στη βάση δεδομένων τον ειδοποιεί με το κατάλληλο μήνυμα ενώ σε αντίθετη περίπτωση δημιουργεί τον χρήστη μέσα στη συλλογή user στη βάση δεδομένων.

```
// Find if Users email exist
User.findOne({ email: email }).then(user => {

  if (user) {

    errors.push({ msg: 'Email already exists' });
    res.render('register', {
      errors,
      name,
      email,
      password,
      password2
    });

  } else {

    // Make new User
    const newUser = new User({

      name,
      email,
      password,
      notification,
      dot
    });

  }

});
```

Κώδικα 14: Έλεγχος αν ο χρήστης που πάει να φτιάξει λογαριασμό υπάρχει ήδη στη βάση δεδομένων

Είναι πολύ σημαντικό η εφαρμογή να προστατεύει τον χρήστη και τα προσωπικά του δεδομένα, οπότε πριν καταχωρηθεί ο νέος χρήστης στη βάση δεδομένων υποχρέωση της διαδικτυακής εφαρμογής είναι να προστατεύσει το κωδικό πρόσβασης. Αυτό συμβαίνει γιατί ακόμα και αν κάποιος καταφέρει να έχει πρόσβαση σε αυτή δεν θα έχει τη δυνατότητα να έχει πρόσβαση σε κανένα λογαριασμό κανενός χρήστη της διαδικτυακής εφαρμογής. Για να το επιτευχθεί αυτό θα χρησιμοποιηθεί η συνάρτηση κατακερματισμού BCrypt, η οποία είναι βασισμένη στον αλγόριθμο κρυπτογράφησης συμμετρικού μπλοκ. Χρησιμοποιεί ένα κλειδί παράγοντα για την προσαρμογή του κόστους του κατακερματισμού, το οποίο κατά πάσα πιθανότητα είναι το πιο αξιοσημείωτο χαρακτηριστικό του BCrypt.

```

// encrypt password and save User to database
bcrypt.genSalt(10, (err, salt) => {
  bcrypt.hash(newUser.password, salt, (err, hash) => {
    if (err) throw err;
    newUser.password = hash;
    newUser
      .save()
      .then(user => {
        req.flash(
          'success_msg',
          'You are now registered and can log in'
        );
        res.redirect('/users/login');
      })
      .catch(err => console.log(err));
  });
});
}

```

Κώδικα 15: Κρυπτογράφηση κωδικού με τη χρήση του Bcrypt και αποθήκευση χρήστη στη βάση δεδομένων

Στο παραπάνω κώδικα φαίνεται ο τρόπος που χρησιμοποιήθηκε το BCrypt. Ουσιαστικά χρησιμοποιεί το Bcrypt.genSalt για να παραχθεί ένα Salt, για το κωδικό που χρειάζεται να κρυπτογραφηθεί. Στη κρυπτογραφία Salt σημαίνει ένα τυχαίο δεδομένο που χρησιμοποιείται σαν επιπλέον είσοδος σε μια one-way¹ συνάρτηση η οποία κατακερματίζει δεδομένα, έναν κωδικό ή φράση πρόσβασης. Δίπλα από το genSalt φαίνεται ο αριθμός 10, αυτή η παράμετρος είναι ένας λογαριθμικός αριθμός. Ο πραγματικός αριθμός που δηλώνει πόσες επαναλαμβανόμενες κατακερματισμένες επαναλήψεις θα γίνουν μέσα στο BCrypt είναι 2^{10} . Μετά, με τη χρήση του bcrypt.hash συνδυάζει τον κωδικό του χρήστη και το Salt που δημιουργήθηκε πιο πάνω και παράγει το κρυπτογραφημένο κωδικό. Μόλις ολοκληρωθεί και η διαδικασία της κρυπτογράφησης ο χρήστης καταχωρείται στη βάση δεδομένων και δρομολογείται στη σελίδα login που θα αναλυθεί παρακάτω.

Δ.1.3.5 Πιστοποίηση χρήστη (Login)

Έχοντας κατασκευάσει το τρόπο με τον οποίο ένας χρήστης δημιουργεί λογαριασμό στην εφαρμογή, το επόμενο στάδιο είναι να δημιουργηθεί ένας τρόπος με τον οποίο συνδέεται σε αυτή. Για να επιτευχθεί αυτό θα χρησιμοποιηθεί μιας μορφής πιστοποίησης. Με τον όρο αυτό εννοείται ο

¹ One-way είναι η συνάρτηση που είναι εύκολο να υπολογιστεί με κάθε είσοδο, αλλά δύσκολο να αντιστραφεί δίνοντας της, όλες τις πιθανές εξόδους που μπορεί να έχει για μια τυχαία είσοδο.

έλεγχος που γίνεται για να εξακριβωθεί εάν ο χρήστης ο οποίος εισάγει τα δεδομένα του στη φόρμα Login υπάρχει και αν ναι, ελέγχοντας το κωδικό του θα δει αν είναι αυτός που ισχυρίζεται πως είναι. Έχοντας αποθηκευμένους όλους τους χρήστες στη βάση δεδομένων, η όλη διαδικασία είναι πολύ απλή και θα χρησιμοποιηθεί ένα πακέτο που λέγεται Passport το οποίο αναφέρθηκε αναλυτικά και πιο πάνω. Ο λόγος όμως για τον οποίο θα χρησιμοποιηθεί αυτή η πιστοποίηση είναι γιατί η εφαρμογή δεν θα επιτρέπει σε όλους τους χρήστες να έχουν πρόσβαση σε κάποιες σελίδες της διαδικαστικής εφαρμογής παρά μόνο οι εγγεγραμμένοι. Για παράδειγμα στη σελίδα userpage στη διαδικτυακή εφαρμογή Nstocks δεν μπορεί να έχει ο οποιοσδήποτε χρήστης πρόσβαση για τον απλούστερο λόγο ότι σε αυτή τη σελίδα παρουσιάζονται κάποια από τα δεδομένα του εγγεγραμμένου χρήστη αλλά υπάρχουν και κάποιες επιπλέον λειτουργίες στις οποίες δεν πρέπει να έχουν πρόσβαση οι μη εγγεγραμμένοι χρήστες. Αν ένα απλός χρήστης είχε πρόσβαση στη σελίδα userpage, η σελίδα αυτή δεν θα παρουσίαζε σωστά το περιεχόμενο της κάτι το οποίο πρέπει να αποφευχθεί και γι' αυτόν τον λόγο χρειάζεται αυτός ο περιορισμός, με τη βοήθεια της πιστοποίησης. Στη συνέχεια θα αναλυθεί ο τρόπος με τον οποίο γίνεται αυτή η πιστοποίηση.

Αρχικά, όπως και στην εγγραφή χρηστών, χρειάζεται μία φόρμα στην οποία θα συμπληρώνει ο χρήστης τα στοιχεία του (email και κωδικό πρόσβασης) και στη συνέχεια αυτά τα στοιχεία θα στέλνονται πίσω στο server για να γίνει ο έλεγχος.

```

<div class="modal-body">
  <!-- Include error , success messages -->
  <% include ./partials/messages %>

  <!-- Send email password to server to login -->
  <form action="/users/login " method="POST">

    <div class="input-group Email mb-3" id="ema">

      <!-- Field for email-->
      <input type="email" class="form-control" id="email" name="email" aria-describedby="emailHelp"
        | placeholder="Enter Email" value="<%= typeof email != 'undefined' ? email : '' %>" />

    </div>

    <!-- Field for password-->
    <div class="input-group Password mb-3" id="pass">

      <input type="password" class="form-control" id="password" name="password"
        | aria-describedby="PasswordHelp" placeholder="Enter password"
        | value="<%= typeof password != 'undefined' ? password : '' %>" />

    </div>

    <!-- Submit btn -->
    <div class="text-center">

      <div class="Login">

        <button type="submit" class="btn btn-primary btn-lg" id="loginBtn">
          | Login
        </button>

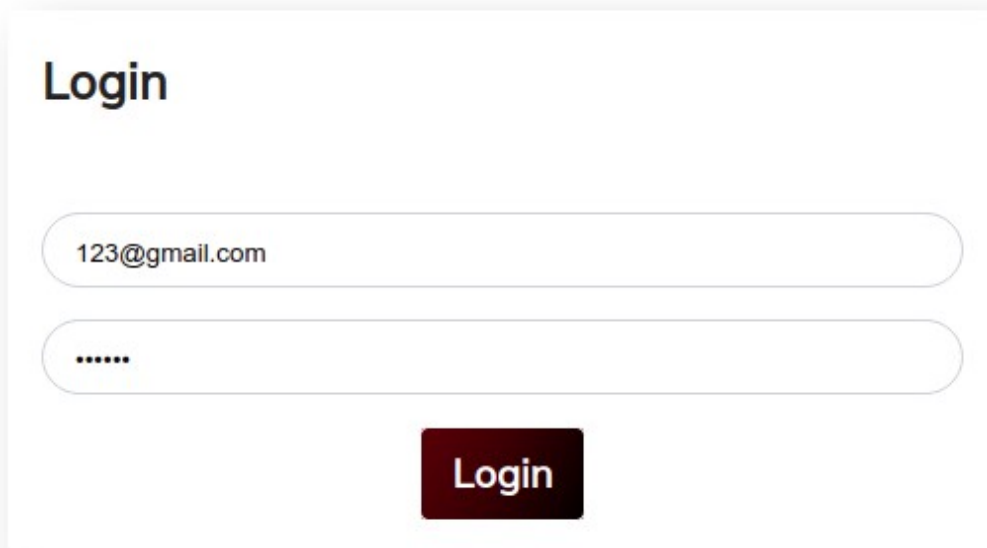
      </div>

    </div>

  </form>

```

Κώδικα 16: Φόρμα σύνδεσης χρήστη στη σελίδα login



The image shows a clean, modern login interface. At the top, the word 'Login' is displayed in a large, bold, black font. Below it, there are two rounded rectangular input fields. The first field contains the text '123@gmail.com'. The second field contains seven dots, indicating a password. Below these fields is a single, prominent dark red button with the word 'Login' in white, bold, sans-serif font.

Εικόνα 40: Φόρμα εισαγωγής στοιχείων χρήστη στη σελίδα login

Όπως φαίνεται στον παραπάνω κώδικα υπάρχουν δύο inputs όπου ο χρήστης εισάγει το email και το κωδικό πρόσβασης του και στη συνέχεια υπάρχει και ένα κουμπί login όπου όταν πατηθεί στέλνεται στο user/login ένα αίτημα POST το οποίο λαμβάνει ο server με το παρακάτω κώδικα.

```
// Login handle
router.post('/login', (req, res, next) => {
  // If successful login user redirect to Homepage else
  // redirect to login in page
  passport.authenticate('local', {
    successRedirect: '/dashboard',
    failureRedirect: '/users/login',
    failureFlash: true,
  })(req, res, next);
});
```

Κώδικα 17: Διαχείριση αιτήματος του χρήστη να περιηγηθεί στη σελίδα login

Στο αρχείο **users.js** ο server τώρα με τη σειρά του δέχεται το αίτημα POST από τη σελίδα login με την εντολή `router.post()` και στη συνέχεια με τη βοήθεια του Passport ελέγχει αν ο χρήστης υπάρχει, και αν ναι τον ανακατευθύνει στη σελίδα `userpage` αλλιώς παραμένει στη σελίδα `login`, έτσι ώστε ο χρήστης να ξαναβάλει τα στοιχεία του. Στη συνέχεια θα αναλυθεί το Passport το οποίο βρίσκεται στο αρχείο **passport.js**.

```

// Use passport
passport.use(

  // Search for email in database
  new LocalStrategy({ usernameField: 'email' }, (email, password, done) => {

    User.findOne({ // If found
      email: email
    }).then(User => {

      // If not return message
      if (!User) {
        return done(null, false, { message: 'That email is not registered' });
      }

      // Match password
      bcrypt.compare(password, User.password, (err, isMatch) => {

        if (err) throw err;

        if (isMatch) { // If match Log in user
          return done(null, User);
        }

        else { // Else throw error message
          return done(null, false, { message: 'Password incorrect' });
        }
      });
    });
  });
});

```

Κώδικα 18: Χρήση του Passport για να τον έλεγχο αν υπάρχει ο χρήστης

Στον κώδικα που παρουσιάζεται, δημιουργείται μια στρατηγική με την εντολή LocalStrategy όπου δέχεται το email και το κωδικό που στέλνει ο χρήστης και ψάχνει στη βάση δεδομένων αν αρχικά υπάρχει το email και αν ναι, αφού ελέγξει με τη bcrypt.compare() συνάρτηση τον κρυπτογραφημένο κωδικό, αν είναι σωστός επιστρέφει τη συνάρτηση done() η οποία περιέχει τα στοιχεία του χρήστη ενώ σε αντίθετη περίπτωση επιστρέφει τη συνάρτηση done() με την Boolean τιμή false.

Μετά στο αρχείο **auth.js** φτιάχνονται οι συναρτήσεις πιστοποίησης,

```

//Function which authenticate that user is signed up
ensureAuthenticated: function (req, res, next) {

  if (req.isAuthenticated()) {

    return next();

  }

  else {

    req.flash('error_msg', 'Please log in to view that resource');
    res.redirect('/users/login');

  }

},

//Function which forward user to sign up
forwardAuthenticated: function (req, res, next) {

  if (!req.isAuthenticated()) {

    return next();

  }

  res.redirect('/dashboard');

}

```

Κώδικα 19: Συναρτήσεις πιστοποίησης χρήστη

δηλαδή φτιάχνεται μια συνάρτηση (ensureAuthenticated), η οποία όταν ο χρήστης πηγαίνει στη σελίδα userpage τον ανακατευθύνει στη σελίδα login γιατί δεν μπορεί να έχει πρόσβαση σε αυτή και μια συνάρτηση (forwardAuthenticated) η οποία όταν συνδέεται ο χρήστης επιτυχώς στη σελίδα login τον ανακατευθύνει στη σελίδα userpage όπου πλέον μπορεί να έχει πρόσβαση.

Τέλος, αυτές οι συναρτήσεις χρησιμοποιούνται στα GET αιτήματα του χρήστη για να δρομολογηθεί στις σελίδες αυτές.

```

// Login Page
router.get('/login', forwardAuthenticated, (req, res) => res.render('login'));

```

Κώδικα 20: Δρομολόγηση χρήστη στη σελίδα Login

```

//UserPage
router.get('/dashboard', ensureAuthenticated, (req, res) => {

  res.render('dashboard', {

    user: req.user

  });

});

```

Κώδικα 21: Δρομολόγηση χρήστη στη σελίδα userpage

Αρα όπως φαίνεται και στο κώδικα παραπάνω, αν ο χρήστης συνδεθεί στη σελίδα login και έχει γίνει η πιστοποίηση, στη συνέχεια αν ξανά δρομολογηθεί στη σελίδα login θα τον ανακατευθύνει αυτόματα στη σελίδα userpage γιατί δεν χρειάζεται να ξανά βάλει τα στοιχεία του. Σε αντίθετη περίπτωση αν ο χρήστης πάει να δρομολογηθεί στη σελίδα userpage θα τρέξει η συνάρτηση `ensureAuthenticated()` και αν έχει γίνει η πιστοποίηση επιτυχώς θα του επιτρέψει να έχει πρόσβαση σε αυτή αλλιώς θα τον στείλει πίσω στο login.

Α.1.3.6 Διαχείριση δεδομένων API

Αφού αναλύθηκαν πλήρως οι βασικές λειτουργίες του server στη συνέχεια θα αναλυθεί για το πώς ο server καλεί την API του Alpha Vantage. Ουσιαστικά δέχεται τα δεδομένα και τα μεταφέρει στον client. Για αρχή θα περιγραφεί ο κώδικας στο αρχείο **api.js** όπου αυτό το κομμάτι του κώδικα δεν αναφέρεται στο server αλλά στο client. Στο συγκεκριμένο αρχείο θα σταλθούν αίτημα στον server με τη βοήθεια της Ajax για να καλέσει την API, να πάρει τα δεδομένα και να τα στείλει πίσω στον client.

```
// Send stocks name to server and Retrieves data from Api
await $.ajax({
  url: '/getApi/' + symbol + '',
  dataType: 'json',
  type: 'get',
  cache: false,
  success: function (data) {

    const time = "Weekly Time Series";
    const json_length = Object.keys(data[time]).length; // Length of json

    var length = json_length - 1; // Length of arrays
    var today = new Date(); // Current Date
    var counter = 0; // Counter
    var curMonth = today.getMonth() + 1; // Current Month
    var prvMonth = today.getMonth();
```

Κώδικα 22: Ajax call για να λάβει τα δεδομένα της μετοχής

Στον παραπάνω κώδικα φαίνεται αυτή τη διαδικασία. Ουσιαστικά με την Ajax στέλνετε ένα αίτημα GET στη διεύθυνση /getApi, καθώς στέλνετε και το σύμβολο της μετοχής για την οποία ζητάει ο χρήστης να δει τα δεδομένα της.

Με τη σειρά του ο server δέχεται αυτό το αίτημα καθώς και το σύμβολο της μετοχής που του στάλθηκε, μέσα από τη διεύθυνση και καλεί την API

```
// Weekly
app.get('/getApi/:symbol', async (request, response) => {

  const api_key = process.env.API_KEY;    // Takes api key from .env file
  const symbol = request.params.symbol;
  const api_url = 'https://www.alphavantage.co/query?function=TIME_SERIES_WEEKLY&symbol=' + symbol + '&apikey=';
  const fetch_response = await fetch(api_url);
  const json = await fetch_response.json();    // Wait to fetch json from api

  console.log(request.params.symbol);
  console.log(api_url);

  response.json(json);    // Send response to client
  exports.symbol = request.params.symbol;
});
```

Κώδικα 23: Ο server δέχεται την κλήση από το client για να καλέσει την api

Όπως φαίνεται στο κώδικα η app.get() συνάρτηση δέχεται το αίτημα GET από τον client μαζί με το σύμβολο που του έστειλε και καλεί την API αφού πρώτα περάσει μέσα σε αυτήν το σύμβολο της μετοχής. Προτού συνεχιστεί η ανάλυση του κώδικα, πρέπει να γίνει μια σύντομη αναφορά για το fetch. Η μέθοδος fetch (API) παρέχει μια διεπαφή για την ανάκτηση πόρων (συμπεριλαμβανομένου του δικτύου). Φαίνεται πολύ οικείο σε όσους έχουν χρησιμοποιήσει το XMLHttpRequest, αλλά αυτή η νέα μέθοδος για την ανάκτηση πόρων παρέχει ένα πιο ισχυρό και ευέλικτο σύνολο δυνατοτήτων. Αφού λοιπόν ολοκληρωθεί η διαδικασία του fetch περνάει στη μεταβλητή json το json αρχείο που ανακτήθηκε από την API. Στη συνέχεια με τη response.json(json) μέθοδο στέλνονται πίσω στο client όλα αυτά τα δεδομένα για τη συγκεκριμένη μετοχή.

```

  "Meta Data": {
    "1. Information": "Weekly Prices (open, high, low, close) and Vol",
    "2. Symbol": "MSFT",
    "3. Last Refreshed": "2020-04-21",
    "4. Time Zone": "US/Eastern"
  },
  "Weekly Time Series": {
    "2020-04-21": {
      "1. open": "176.6300",
      "2. high": "178.7500",
      "3. low": "166.1102",
      "4. close": "167.8200",
      "5. volume": "92496267"
    },
    "2020-04-17": {
      "1. open": "164.3500",
      "2. high": "180.0000",
      "3. low": "162.3000",
      "4. close": "178.6000",
      "5. volume": "238965670"
    },
    "2020-04-09": {
      "1. open": "160.3200",
      "2. high": "170.0000",
      "3. low": "157.5800",
      "4. close": "165.1400",
      "5. volume": "229630744"
    },
    "2020-04-03": {
      "1. open": "152.4400",
      "2. high": "164.7800",
      "3. low": "150.0100",
      "4. close": "153.8300",
      "5. volume": "290191457"
    }
  }
}

```

Εικόνα 41: Αρχείο json που στέλνεται στο server από το Alpha Vantage

Στην παραπάνω φωτογραφία φαίνεται η μορφή με την οποία φτάνουν τα δεδομένα στον client. Στη συνέχεια αυτό που πρέπει να γίνει είναι να περαστούν τα δεδομένα σε πίνακες για να μπορέσουν αργότερα να εμφανιστούν στις γραφικές παραστάσεις που θα κατασκευαστούν.

```

//Fill arrays with data from servers json response
for (var date in data["Weekly Time Series"]) {

  date_array.push(date);
  open_array.push(data[time][date]["1. open"]);
  close_array.push(data[time][date]["4. close"]);
  low_array.push(data[time][date]["3. low"]);
  high_array.push(data[time][date]["2. high"]);
  volume_array.push(data[time][date]["5. volume"]);

};

```

Κώδικα 24: Εισαγωγή δεδομένων που λήφθηκαν από την api σε πίνακες

Στον κώδικα, αυτό που γίνεται είναι τα δεδομένα που λήφθηκαν με επιτυχία στη συνέχεια να περνούν στους αντίστοιχους πίνακες Open, Close, Low, High, Volume και Date. Επίσης, επειδή χρειάζεται μόνο ένα μέρος από τα δεδομένα που λήφθηκαν, συγκεκριμένα οι τιμές των μετοχών για αυτόν τον μήνα, πρέπει να επεξεργαστεί ο πίνακας date για να μπορέσουν να διαχωριστούν. Στις παρακάτω γραμμές κώδικα φαίνεται ο τρόπος που υπολογίζεται η αρχή και το τέλος του χρόνου που θα χρησιμοποιηθεί για να περαστούν τα δεδομένα αυτού του μήνα σε έναν καινούργιο πίνακα. Επειδή τα δεδομένα που στέλνει η συγκεκριμένη API είναι για ακριβώς πέντε χρόνια δεν χρειάζεται να επεξεργαστούν τα δεδομένα, απλά περνάνε σε πίνακες και στέλνονται για οπτικοποίηση με την αντίστοιχη γραφική παράσταση.

```
// Find index of current month in previous year --> START  
var startMonth = date_labels.findIndex(element => element.includes(today.getFullYear() - 1) && element.endsWith(curMonth, 7));
```

Κώδικα 25: Εντοπισμός αρχής του μήνα στο πίνακα date

```
// Find index of the current month in current year --> END  
var endMonth = date_labels.findIndex(element => element.includes(today.getFullYear()) && element.endsWith(curMonth, 7));
```

Κώδικα 26: Εντοπισμός τέλος του μήνα στο πίνακα date

Έχοντας δημιουργηθεί οι πίνακες που θα χρησιμοποιηθούν στη γραφική παράσταση για τις τιμές των μετοχών για ένα χρόνο, στη συνέχεια περνάνε τα οι τιμές της μετοχής στους αντίστοιχους πίνακες στον κώδικα που βρίσκεται στο αρχείο **chart.js**. Προτού συνεχιστεί η ανάλυση του υπόλοιπου κώδικα όμως στο αρχείο **chart.js** και γενικότερα στην οπτικοποίηση των δεδομένων, σημαντικό είναι να αναλυθεί πως επεξεργάζονται τα δεδομένα και για τις υπόλοιπες γραφικές παραστάσεις που βρίσκονται στη διαδικτυακή εφαρμογή. Αφού περιγράφηκε η διαδικασία λήψης και διαχείρισης των δεδομένων του ενός έτους και των πέντε ετών, στη συνέχεια θα αναλυθούν και για τις υπόλοιπες γραφικές παραστάσεις που παρουσιάζονται στη διαδικτυακή εφαρμογή όπου είναι τους ενός μήνα και των πέντε μηνών.

```

//GetDataDaily wait Ajax function to retrieve data from daily Api
async function getDataDaily(symbol) {

  if (symbol == null) {

    symbol = 'MSFT';
  }

  // Send stocks name to server and Retrieves data from Api
  await $.ajax({

    url: '/getAp/' + symbol + '',
    dataType: 'json',
    type: 'get',
    cache: false,
    success: function (data) {

      const time = "Time Series (Daily)";
      const json_length = Object.keys(data[time]).length; // Length of json
      var length = json_length - 1; // Length of arrays

      var month = new Date(); // Current date
      var thisMonth = month.getMonth() + 1; // Current month
      var thisDate = month.getDate(); // Current day
      var prevDate = thisDate - 1; // Previous day
      var counter = 0;
    }
  });
}

```

Κώδικα 27: Ajax call για να καλέσει ο server την api για τη μετοχή που του στέλνεται

Όπως φαίνεται στον κώδικα παραπάνω, στο αρχείο **api.js** υπάρχει η συνάρτηση `getDaily()` η οποία δέχεται σαν όρισμα το σύμβολο της μετοχής που θέλει να βρει ο χρήστης και στη συνέχεια στέλνει ένα GET αίτημα στη διεύθυνση `/getApp()` στέλνοντας μαζί και την ονομασία του συμβόλου.

Το GET αίτημα, στη συνέχεια το δέχεται ο server με τη μέθοδο `router.get()` όπως φαίνεται και παρακάτω.

```

app.get('/getAp/:symbol', async (request, response) => {

  const api_key = process.env.API_KEY;
  const symbol = request.params.symbol;
  const api_url_daily = 'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=' + symbol + '&outputsize=compact&apikey=' + api_key;
  const fetch_response_daily = await fetch(api_url_daily);
  const dailyjson = await fetch_response_daily.json();

  console.log(request.params.symbol);
  console.log(api_url_daily);

  response.json(dailyjson);
});

```

Κώδικα 28: Ο server λαμβάνει το αίτημα για να καλέσει την api για τη μετοχή που του στάλθηκε

Αφού περάσει το σύμβολο της μετοχής στη μεταβλητή `symbol`, καλεί την API από το Alpha Vantage. Στη συνέχεια, αφού ολοκληρωθεί η διαδικασία του `fetch` με την οποία δέχεται τα δεδομένα αλλά και η διαδικασία κατά την οποία τα περνάει στο `json`, επιστρέφεται πίσω στο `client` σαν απάντηση το `json` αρχείο. Η `Ajax` τώρα με τη σειρά της, αφού ολοκληρωθεί με επιτυχία η απάντηση από το `server`, θέτει σε λειτουργία το `success` γεγονός όπου εκτελεί τη συνάρτηση που έχει σαν όρισμα το πίνακα με τα δεδομένα του `json`. Η συνάρτηση λαμβάνει τα δεδομένα και περνάνε στους αντίστοιχους πίνακες όπως έγινε και πιο πάνω για την API για τα έτη. Αφού ολοκληρωθεί η διαδικασία πρέπει τα δεδομένα να δεχθούν κάποια επεξεργασία έτσι ώστε να υπάρχουν μόνο οι τιμές των μετοχών για τον ένα και τους πέντε μήνες. Οπότε, με παρόμοιο τρόπο με αυτόν που έγινε πιο πάνω, περνούν μόνο τα απαραίτητα δεδομένα στους αντίστοιχους πίνακες που μετέπειτα θα εμφανιστούν στις γραφικές παραστάσεις. Σημαντικό είναι να αναφερθεί ότι η `Ajax` εκτός από το `success` γεγονός, με το οποίο διαχειρίζεται τις επιτυχείς απαντήσεις από το `server` έχει και ένα `error` γεγονός που διαχειρίζεται τις λανθασμένες απαντήσεις από το `server`.

Τέλος σειρά έχει να αναλυθεί η επεξεργασία των δεδομένων για τις γραφικές παραστάσεις της τωρινής και προηγούμενης μέρας. Όπως έγινε και πιο πάνω, με τη μέθοδο `Ajax` στέλνεται ένα αίτημα `GET` στο `server` και αφού αυτός καλέσει την API για τα δεδομένα πραγματικού χρόνου και στείλει την αντίστοιχη απάντηση στον `client`, αυτός με τη σειρά του, εφόσον έχει σταλθεί με επιτυχία η απάντηση, παίρνει τα δεδομένα και τα περνάει στους αντίστοιχους πίνακες. Όπως και πιο πάνω θα χρειαστεί ειδική επεξεργασία για να τα δεδομένα που έχουν ληφθεί.

Προτού παρουσιαστεί ο κώδικας, πρέπει να γίνει κατανοητό ποια είναι τα προβλήματα με τις γραφικές παραστάσεις της τωρινής και της προηγούμενης μέρας. Τα δυο βασικά προβλήματα είναι η μέρα και η ώρα. Το χρηματιστήριο ανοίγει 16:35 ώρα Ελλάδος και λειτουργεί μόνο τις καθημερινές. Οπότε πρέπει να ρυθμιστεί η γραφική παράσταση της πρώτης μέρας, όταν η μέρα είναι Σάββατο ή Κυριακή να δείχνει τα δεδομένα της Παρασκευής αλλά και όταν η ώρα είναι πριν της 16:35 να δείχνει πάλι τα δεδομένα της προηγούμενης μέρας ενώ όταν είναι καθημερινή και η ώρα είναι 16:35 να αρχίσει να δείχνει τα δεδομένα της σημερινής μέρας.

Για την γραφική παράσταση της προηγούμενης μέρας αυτό που πρέπει μόνο να προσέξει κάποιος είναι η μέρα. Όταν είναι Σάββατο πρέπει να εμφανίζει τα δεδομένα της Πέμπτης και όταν είναι

Κυριακή πάλι να δείχνει τα δεδομένα της Πέμπτης. Για να το γίνει αυτό αρχικά πρέπει να βρεθεί ποια είναι η τωρινή μέρα και ώρα, αυτό μπορεί να το πετύχει κάποιος με τη δημιουργία ενός νέου αντικειμένου με το `new Date()` της JavaScript.

```
var day = new Date();  
var curDay = day.getDate() - 1;  
var numDay = day.getDay();  
var numHour = day.getHours();  
var curMin = day.getMinutes();
```

Κώδικα 29: Δήλωση μέρας και ώρας με τη μέθοδο `.newDate()`

Όπως φαίνεται και στον παραπάνω κώδικα πλέον ξέροντας την ώρα, τα λεπτά, την ημερομηνία, αλλά και ποια μέρα της εβδομάδας είναι. Με αυτές τις μεταβλητές που δημιουργήθηκαν θα πραγματοποιηθούν κάποιοι έλεγχοι για να μπορέσουν τελικά να σταλθούν τα σωστά δεδομένα για σωστή μέρα που θα εμφανιστεί στη γραφική παράσταση.

```

// If its a day of the week except saturday, sunday and the hour is bigger than 16:35
if ((numHour == 16 && curMin >= 35) && (numDay != 6 || numDay != 0)) {
    curDay = curDay + 1;
}

// If its a day of the week except saturday, sunday and the hour is bigger than 16
if ((numHour > 16) && (numDay != 6 || numDay != 0)) {
    curDay = curDay + 1;
}

// If its Monday and hour<16:35 go to friday
if (numDay == 1 && numHour < 16) {
    curDay = curDay - 2;
}

// If Sunday go to friday
if (numDay == 0) {
    curDay = curDay - 2;
}

if (numDay == 6) {
    curDay = curDay - 1;
}

```

Κώδικα 30: Έλεγχος μέρας και ώρας

Αποτέλεσμα αυτών των ελέγχων είναι να περνάει η σωστή ημερομηνία στη μεταβλητή curDay. Ξέροντας πλέον τη σωστή ημερομηνία των δεδομένων που θα παρουσιαστεί, επόμενο βήμα είναι να τα περαστούν στους κατάλληλους πίνακες. Για να γίνει αυτό θα πρέπει να εκτελεστεί μια αναζήτηση στον προηγούμενο πίνακα που περιέχει όλα τα δεδομένα και θα ληφθούν μόνο αυτά που χρειάζονται.

```

for (var i = 0; i <= length; i++) {
    // If date ends with current date or previous 2 days add to array
    if (date_labels_Real[i].endsWith((aDays), 10)) {
        date_labels_2Days_chart[counter] = date_labels_Real[i];
        close_labels_2Days_chart[counter] = close_labels_Real[i];
        counter++;
    }
}

```

Κώδικα 31: Εντοπισμός δεδομένων μόνο για τη μέρα που χρειάζεται

Με την επανάληψη που φαίνεται στον παραπάνω κώδικα, γίνεται προσπάθεια να βρεθούν όλα αυτά τα δεδομένα όπου η ημερομηνία τους είναι ίδια με το `curDay`. Όπως φαίνεται χρησιμοποιείται η μέθοδος `.endsWith()` η οποία ψάχνει μέσα στο κείμενο στη 10η θέση αν βρίσκεται η σωστή ημερομηνία. Αφού ολοκληρωθεί και αυτή η διαδικασία ο πίνακας στέλνεται στο αρχείο **chart.js** που θα αναλυθεί παρακάτω.

Δ.1.3.7 Οπτικοποίηση δεδομένων (Chart.js)

Ήρθε η στιγμή να περιγραφεί ένα από τα πιο σημαντικά κομμάτια της πτυχιακής και αυτό είναι η οπτικοποίηση δεδομένων. Για να καταφέρει κάποιος να παρουσιάσει τα δεδομένα, που λήφθηκαν από την API στο χρήστη θα χρησιμοποιηθεί το Chart.js. Για να γίνει όμως αυτό πρέπει να δημιουργηθεί ένα canvas στο `ejs` αρχείο. Ο παρακάτω κώδικας είναι από το αρχείο **Stocks.ejs**.

```
<div class="tab-pane fade show active" id="nav-1Day" role="tabpanel"
aria-labelledby="nav-profile-tab">

<canvas id="myChart1Day" class="canvades">

  <script>

    // Wait for fuction chartIt1Day done first because it makes the Api call
    // then call the function chartIt3Days
    $.when(chartIt1Day()).done(function () { chartIt3Days(); });

  </script>

</canvas>

</div>
```

Κώδικα 32: Δημιουργία canva για την οπτικοποίηση δεδομένων

Όπως φαίνεται και στο κώδικα δημιουργείται ένα ξεχωριστό κιβώτιο (container) και μέσα σε αυτό τοποθετείται το canvas των γραφικών παραστάσεων. Ο συγκεκριμένος canvas που απεικονίζεται στη φωτογραφία είναι για τη γραφική παράσταση της μίας μέρας. Επίσης η γραμμή που βρίσκεται μέσα στο `< script >` στο παραπάνω κώδικα είναι για να θέτει σε λειτουργία τις συναρτήσεις που

δημιουργούν τις γραφικές παραστάσεις, κάθε φορά που ανοίγει κανείς τη σελίδα stocks της διαδικτυακής εφαρμογής. Στο αρχείο **chart.js** και συγκεκριμένα στη συνάρτηση `chartIt1Day()` η οποία όπως φαίνεται και παρακάτω είναι ασύγχρονη (async), αυτό συμβαίνει γιατί προτού τρέξει αυτή η συνάρτηση πρέπει να εκτελεστεί αλλά και να ολοκληρωθεί η συνάρτηση `getDataReal()`, η οποία είναι αυτή που δέχεται τα δεδομένα από το server. Οπότε όπως φαίνεται και στον κώδικα δίπλα από τη δεύτερη συνάρτηση υπάρχει η έκφραση `await` ενώ δίπλα από την αρχική συνάρτηση υπάρχει η έκφραση `async`. Η `async` συνάρτηση είναι αυτή που περιμένει την `await` συνάρτηση να ολοκληρωθεί και μετά να συνεχίσει να εκτελεί το κώδικα που περιλαμβάνεται σε αυτήν.

```
async function chartIt1Day(symbol, destroy) {  
  //ChartIt waits getData to complete and then starts to visualize data  
  await getDataReal(symbol);    You, 5 months ago • Add a list with followed stocks at user pages  
  
  // Take id from ejs file to find the container it will fit  
  const ctx = document.getElementById('myChart1Day').getContext("2d");
```

Κώδικα 33: Ασύγχρονη συνάρτηση οπτικοποίησης δεδομένων μιας μέρας

Αφού ολοκληρωθεί η συνάρτηση `getDataReal()` στη συνέχεια πρέπει να αποφασιστεί σε ποιο καμβά θα δημιουργηθεί η γραφική παράσταση. Για να γίνει αυτό θα χρησιμοποιηθεί η μέθοδος `document.getElementById()` η οποία επιστρέφει το αντικείμενο με το id που δηλώθηκε μέσα στη μέθοδο. Συγκεκριμένα στον κώδικα παραπάνω δηλώθηκε το id `myChart1Day` το οποίο είναι ο canvas που βρίσκεται στο αρχείο **Stocks.ejs**.

```
// Chart for 1 Day  
const myChart1Day = new Chart(ctx, {  
  type: 'line',    // Line chart  
  
  data: {  
    datasets: [generate(0, "close", colorStock)] //Put data and choose color for the chart  
  },
```

Κώδικα 34: Δημιουργία γραφικής παράστασης με το Chart.js

Σειρά έχει να δημιουργηθεί η γραφική παράσταση. Σύμφωνα με το `chart.js` πρέπει να δημιουργηθεί ένα νέο chart περνώντας σε αυτό το αντικείμενο `ctx` που δείχνει το canvas στο `ejs` αρχείο αλλά και

τα χαρακτηριστικά αυτής της γραφικής παράστασης. Με τον όρο `type` δηλώνεται ο τύπο της γραφικής παράστασης που στη προκειμένη περίπτωση είναι γράφημα γραμμής. Στη συνέχεια στον πίνακα `datasets` περνάει η συνάρτηση `generate` η οποία θα γεμίσει τον πίνακα με τις τιμές του `x` και του `y` άξονα για το κάθε σημείο της γραφικής παράστασης. Σαν ορίσματα αυτής της συνάρτησης περνάει το βήμα, που δεν πρέπει να έχει τον όρο `close` για τη ονομασία των ταμπελών, και το χρώμα της γραφικής παράστασης το οποίο θα αλλάζει ανάλογα με το αν πέφτει ή ανεβαίνει η τιμή του `close` σε σχέση με την τελευταία τιμή `close` που είχε η μετοχή τη προηγούμενη μέρα.

```
// Generate data and put them to axis x and y
function generate(shift, label, color) {

    var datas = [];

    for (var i = 0; i <= date_labels_1Day_chart.length; i++) {

        datas.push({ x: date_labels_1Day_chart[i], y: close_labels_1Day_chart[i] });

    }

}
```

Κώδικας 35: Δημιουργία πίνακα με τα x,y σημεία της γραφικής παράστασης

Η `generate` περνάει σε έναν νέο πίνακα τα δεδομένα που στάλθηκαν από το αρχείο **api.js** όπου βάζει τις ημερομηνίες στο `x` και τις `close` τιμές στο `y`.

Αφού έγιναν τα παραπάνω βήματα πλέον μπορεί κανείς να δει τη γραφική παράσταση στη σελίδα `stocks` της εφαρμογής. Πρέπει όμως να δεχθεί και κάποια επεξεργασία για να μπορεί να φαίνεται όσο καλύτερα στο χρήστη, αλλά και για να μπορεί ο χρήστης να αλληλεπιδρά μαζί της με τον καλύτερο δυνατό τρόπο. Ένα πρόβλημα που δημιουργήθηκε είναι ότι στις γραφικές παραστάσεις που έχουν πολλά δεδομένα, άρα και πολλά σημεία πάνω στη γραφική παράσταση, δεν μπορεί να δει ο χρήστης τα δεδομένα πάνω σε αυτό το σημείο γιατί το ποντίκι αλληλεπιδρά και με άλλα σημεία που υπήρχαν πολύ κοντά του. Για να λυθεί αυτό το πρόβλημα χρησιμοποιήθηκε μια προέκταση (plugin) που λέγεται `crosshair`, η οποία δημιουργεί μια πολύ λεπτή γραμμή παράλληλα με τον άξονα `y` η οποία ακολουθά τη κίνηση από το ποντίκι. Οπότε πλέον ο χρήστης μπορεί να παρατηρεί πολύ πιο εύκολα τα δεδομένα για το κάθε σημείο ξεχωριστά χωρίς να έχει κανένα πρόβλημα. Ο τρόπος που ενσωματώθηκε αυτό το plugin στη γραφική παράσταση φαίνεται στον παρακάτω κώδικα, στον οποίο φαίνεται το πώς ορίζεται το χρώμα και το πλάτος της γραμμής αλλά

επίσης και το πως απενεργοποιούνται κάποιες έξτρα λειτουργίες που είχε το plugin οι οποίες δεν χρειάζονται.

```
plugins: {  
  crosshair: { // add line( crosshair ) to chart  
    line: {  
      color: '#808080', // crosshair line color  
      width: 1 // crosshair line width  
    },  
    sync: {  
      enabled: false, // disable trace line syncing with other charts  
    },  
    zoom: {  
      enabled: false, // disable zooming  
    },  
  },  
}
```

Κώδικας 36: Χρήση plugin crosshair στη γραφική παράσταση

Έχουν γίνει αρκετές μικρές αλλαγές στο τρόπο που εμφανίζεται η γραφική παράσταση οι οποίες δεν θα παρουσιαστούν, αλλά κάτι που είναι αρκετά σημαντικό να αναφερθεί και βοηθάει τον χρήστη να μπορέσει να διαβάσει και να καταλάβει καλύτερα τη γραφική παράσταση είναι ο τρόπος που εμφανίζονται οι ημερομηνίες στον άξονα x.

```
scales: {
  xAxes: [{
    display: true,
    type: 'time', // Type of data at x axis
    distribution: 'series',
    position: 'bottom',
    id: 'x-axis-0',
    gridLines: {
      display: false,
      drawBorder: false
    },
  }],
},
```

Κώδικα 37: Ρυθμίσεις για τον άξονα x της γραφικής παράστασης

Στον κώδικα φαίνεται ότι συγκεκριμένα για τον άξονα x έχουν δηλωθεί ότι τα δεδομένα αφορούν τον χρόνο, ο τρόπος που εμφανίζονται πάνω στον άξονα (am, pm), και η θέση τους σε σχέση με τον άξονα πρέπει να είναι κάτω από αυτόν. Μετά από όλες αυτές τις αλλαγές οι γραφικές παραστάσεις θα έχουν αυτή τη μορφή:



Εικόνα 42: Γραφική παράσταση μιας μέρας

Ο τρόπος με τον οποίο κατασκευάστηκαν και οι υπόλοιπες γραφικές παραστάσεις είναι παρόμοιος, οπότε δεν θα αναλυθεί ο κώδικας τους. Στη συνέχεια θα παρουσιαστεί η κατασκευή της σελίδας **Stocks.ejs** και με ποιο τρόπο θα οργανωθούν οι γραφικές παραστάσεις έτσι ώστε να είναι εύκολο στον χρήστη να περιηγηθεί σε αυτές.

Α.1.3.8 Κατασκευή σελίδας **Stocks.ejs**

Αφού επιτυχώς κατασκευάστηκαν οι γραφικές παραστάσεις πρέπει να οργανωθούν με κάποιον τρόπο καθώς είναι πολλές για να υπάρχουν ταυτόχρονα στην ίδια οθόνη. Για το λόγο αυτόν θα κατασκευαστεί μια λίστα μέσα στην οποία θα τοποθετηθούν οι γραφικές παραστάσεις και κάθε μια θα έχει τη δική της ταμπέλα, με τις οποίες ο χρήστης θα μπορεί να τις εναλλάσσει, έχοντας πάντα μόνο μια ανοιχτή στο προσκήνιο.

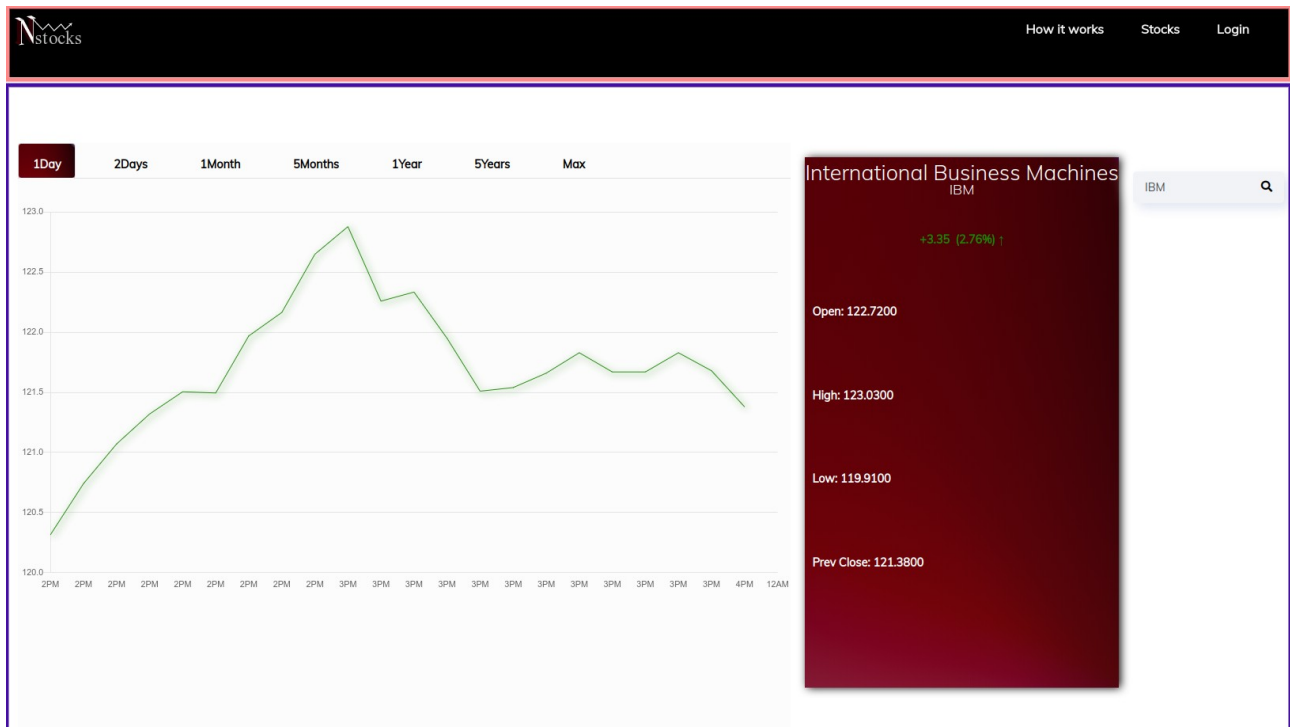
Πριν παρουσιαστεί όμως ο κώδικας για το κυρίως μέρος της σελίδας **stocks**, πρέπει να σημειωθεί ότι σε κάθε σελίδα της εφαρμογής μας πρέπει να υπάρχει η μπάρα πλοήγησης (**navigation bar**). Οπότε σωστό είναι να φτιαχτούν δυο ξεχωριστά **containers** που το ένα θα περιλαμβάνει τη μπάρα πλοήγησης και το άλλο την υπόλοιπη σελίδα. Στον παρακάτω κώδικα φαίνεται ο τρόπος που ενσωματώνεται το αρχείο **navbar.ejs** στη σελίδα **Stocks.ejs** αλλά και σε όλες τις άλλες σελίδες της διαδικτυακής εφαρμογής εκτός του **register.ejs** και **login.ejs**.

```
<div class="wrapper">
  <!-- Container for navbar -->
  <div class="container-navbar">

    <div class="row-nav">
      <% include ./navbar %>
    </div>
  </div>
```

Κώδικας 38: Εισαγωγή μπάρα πλοήγησης στα αρχεία *ejs*

Το container με την κλάση wrapper είναι το parent container το οποίο περιλαμβάνει το container-navbar και το container-stock, τα οποία αποτελούν τα child containers του wrapper. Αφού ξεκαθαρίστηκε αυτό, θα αναλυθούν τώρα τα υπόλοιπα κομμάτια που απαρτίζουν την σελίδα stocks.



Εικόνα 43: Containers στη σελίδα Stocks

Για να μπορέσει κανείς να διακοσμήσει τη σελίδα αλλά και να ορίσει ποιο μέρος της σελίδας θα περιλαμβάνει το κάθε container πρέπει να χρησιμοποιηθεί CSS. Υπάρχουν πολλά στοιχεία που άλλαξαν με τη χρήση της CSS στη παραπάνω φωτογραφία αλλά για την ώρα θα περιγραφεί η ιδιότητα flexbox, το οποίο είναι ένα εργαλείο που βοηθά να φτιαχτεί μια ευέλικτη και ανταποκρινόμενη δομή διάταξης. Όπως φαίνεται στη φωτογραφία παραπάνω η σελίδα χωρίζεται σε δυο γραμμές, για να επιτευχθεί αυτό πρέπει να εκτελεστεί ο παρακάτω κώδικας ο οποίος βρίσκεται στο αρχείο **stock.css**

```
.wrapper{
  display: flex;
  height:100vh;
  flex-direction: column;
}
```

Κώδικα 39: Χρήση ιδιότητας flexbox

Στο κώδικα αρχικά παρουσιάζεται το `display:flex` όπου η `display` ιδιότητα ορίζει εάν ένα στοιχείο που αντιμετωπίζεται ως μπλοκ (block) ή ενσωματωμένο (inline) στοιχείο και η διάταξη που χρησιμοποιείται για τα παιδιά του, όπου σε αυτή τη περίπτωση είναι `flex`. Μετά με την ιδιότητα `flex-direction` δηλώνετε πώς τα `flex` αντικείμενα τοποθετούνται μέσα στο `flex container` και τέλος δηλώνετε το ύψος του `parent container` όπου είναι `100vh` που σημαίνει ότι το ύψος θα είναι ίσο με το μέγεθος της οθόνης που κάθε χρήστης χρησιμοποιεί. Στη συνέχεια θα ρυθμιστεί το ύψος και πλάτος για τα δυο `child containers` που βρίσκονται στο `wrapper`.

```
.container-navbar{
  width: 100%;
  height:10%;
  background-color: black;
  position: normal!important;
}

.container-stock{
  width: 100%;
  height:90%;
  display: flex;
  flex-direction: column;
}
```

Κώδικα 40: Child containers μέσα στο parent container στη σελίδα Stocks

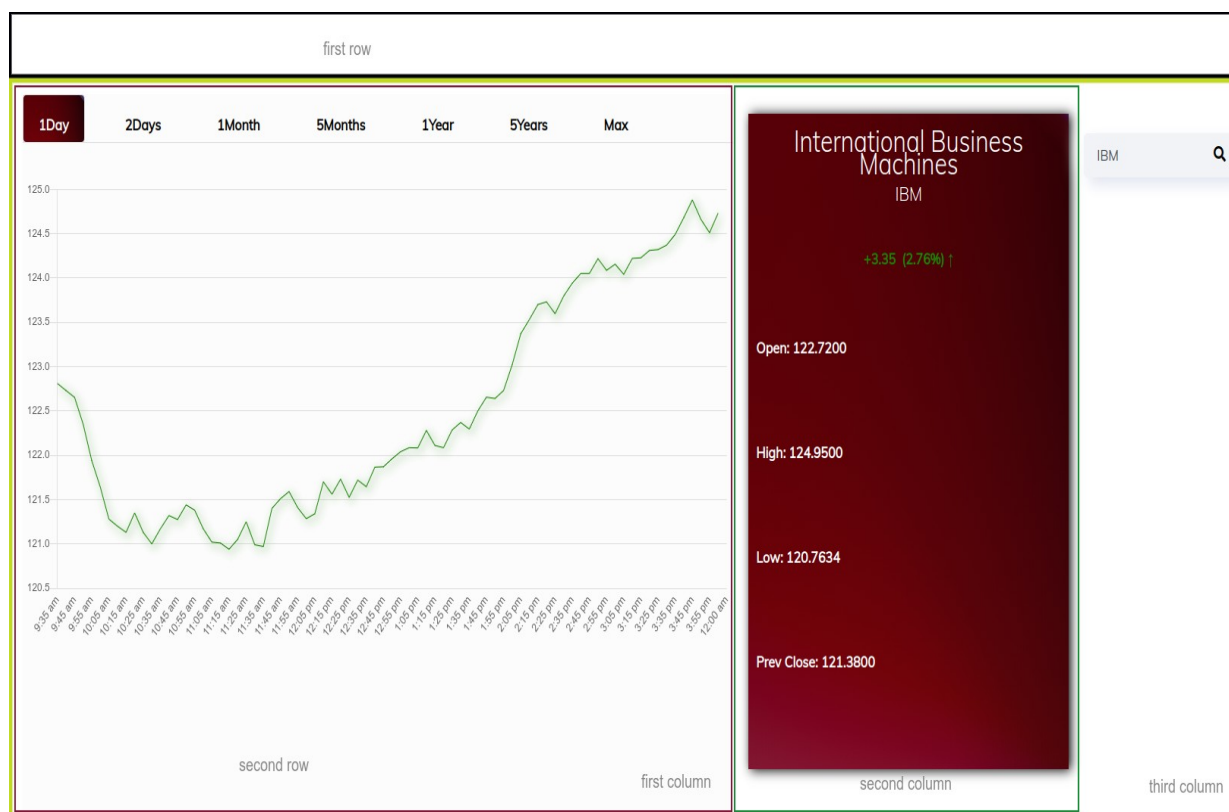
Το `container` για τη μπάρα πλοήγησης θα έχει σε ύψος μόνο το 10% της οθόνης ενώ η υπόλοιπη οθόνη θα καλύπτεται από το `container-stock`. Το πλάτος και για τα δυο `containers` θα καλύπτουν όλο το πλάτος της οθόνης. Φαίνεται ότι το `container-stock` έχει την ιδιότητα `flex`, αυτό συμβαίνει

γιατί και αυτό αποτελείται από child containers, συγκεκριμένα το row1 και row2, τα οποία χωρίζουν το parent container του σε δυο μέρη.

```
.row1{
  width: 100%;
  height:10%;
  background-color: #ffffff;
}

.row2{
  width: 100%;
  height:90%;
  background-color: #ffffff;
  display: flex;
}
```

Κώδικα 41: Child containers του container-stock



Εικόνα 44: Child containers στη σελίδα Stocks

Όπως φαίνεται στη φωτογραφία αλλά και στον κώδικα το row1 container αποτελεί το 10% του αρχικού container, και χρειάζεται γιατί σε αυτό θα εμφανίζεται το κουμπί follow όπου ο χρήστης θα ακολουθεί τη μετοχή που θέλει, αλλά και γιατί όσο μικραίνει η οθόνη η μπάρα αναζήτησης θα πρέπει να αλλάξει θέση και να ανέβει στο row1 container, ενώ το row2 container θα αποτελεί το 90% του αρχικού container. Τέλος το row2 container και αυτό με τη σειρά του αποτελείται από child container όπου το ένα είναι για τις γραφικές παραστάσεις, το δεύτερο είναι για τις πληροφορίες της μετοχής και το τελευταίο για τη μπάρα αναζήτησης. Σε αυτή την περίπτωση όμως φαίνεται ότι το row2 container δεν χωρίζεται σε γραμμές αλλά σε στήλες, αυτό συμβαίνει γιατί στο κώδικα του row2 δεν υπάρχει η ιδιότητα flex-direction και από μόνο του το display:flex δηλώνει ότι τα αντικείμενα θα τοποθετούνται μέσα στο flex container σε στήλες. Με λίγα λόγια η ιδιότητα του display:flex είναι να τοποθετεί τα child containers σε στήλες μέσα στο parent container και σε περίπτωση που κάποιος θέλει να το αλλάξει αυτό χρησιμοποιεί το flex-direction για να αποφασίσει αυτός πώς θα εμφανίζονται.

Μέσα στο col-chart container θα δημιουργηθούν δυο container, ένα για τη μπάρα πλοήγησης για τις γραφικές παραστάσεις και ένα μέσα στο οποίο θα εμφανίζονται αυτές. Αφού φτιαχτούν με τον ίδιο τρόπο που δείχθηκε και πιο πάνω σειρά έχει να κατασκευαστούν οι ταμπέλες

```
<section id="tabs">
  <nav>
    <div class="nav nav-tabs " id="nav-tab" role="tablist">
      <a class="nav-item nav-link active" id="nav-1Day-tab" data-toggle="tab" href="#nav-1Day" role="tab"
        aria-controls="nav-home" aria-selected="true">1Day</a>
      <a class="nav-item nav-link" id="nav-3Days-tab" data-toggle="tab" href="#nav-3Days" role="tab"
        aria-controls="nav-profile " aria-selected="false">2Days</a>
      <a class="nav-item nav-link" id="nav-1Month-tab" data-toggle="tab" href="#nav-1Month" role="tab"
        aria-controls="nav-contact" aria-selected="false">1Month</a>
    </div>
  </nav>
</section>
```

Κώδικα 42: Δημιουργία ταμπελών για τις γραφικές παραστάσεις

Ο κώδικας που φαίνεται παραπάνω δείχνει τον τρόπο που φτιάχνονται αυτές οι ταμπέλες. Με τη βοήθεια του Bootstrap θα φτιαχτούν εύκολα οι ταμπέλες για τις γραφικές παραστάσεις. Πρέπει απλά να προστεθεί το data-toggle="tab" σε κάθε στοιχείο. Χρησιμοποιείται το στοιχείο < a > το οποίο ορίζει έναν υπερ-σύνδεσμο, ο οποίος χρησιμεύει στο να μεταβαίνει ο χρήστης από τη μία

σελίδα στην άλλη. Στη συγκεκριμένη περίπτωση θα δημιουργηθεί ένα στοιχείο `a` για την κάθε γραφική παράσταση και θα συνδεθεί με αυτή, τοποθετώντας στο `href` του κάθε στοιχείου το `id` της αντίστοιχης γραφικής παράστασης, με αποτέλεσμα κάθε φορά που επιλέγει ο χρήστης μια ταμπέλα αυτομάτως θα εμφανίζεται και η γραφική παράσταση με την οποία είναι συνδεδεμένη.

Σημαντικό είναι όμως να οριστεί ποια γραφική παράσταση θα φαίνεται και ποιες θα κρύβονται στο παρασκήνιο μέχρι να τις διαλέξει ο χρήστης. Όπως φαίνεται από τον κώδικα χρησιμοποιείται το `aria-selected` για ακριβώς αυτόν τον σκοπό. Όταν είναι `true`, η ταμπέλα είναι ενεργοποιημένη και δείχνει την αντίστοιχη γραφική παράσταση με την οποία είναι συνδεδεμένη. Το τελευταίο σημαντικό κομμάτι που πρέπει να αναφερθεί για τις ταμπέλες είναι το πρόβλημα που θα δημιουργείται όταν ο χρήστης θα κάνει αναζήτηση μια μετοχής. Πρέπει με το που γίνεται αναζήτηση μετοχής να απενεργοποιούνται όλες οι ταμπέλες εκτός της αρχικής έτσι δηλαδή όπως το ορίσαμε. Για να το πετύχουμε αυτό θα χρησιμοποιήσουμε `jQuery`.

```
| // When click search go back to the first Tab      Yo  
$($('.nav-tabs a:first')).tab('show');
```

Κώδικας 43: Χρήση jQuery για την ενεργοποίηση της πρώτης ταμπέλας

Με τον παραπάνω κώδικα, ο οποίος βρίσκεται στο αρχείο **search.js**, αναγκάζεται το container με το `id nav-tabContent` να κάνει ανανέωση του περιεχομένου του χωρίς να γίνει ανανέωση της σελίδας. Ο ίδιος τρόπος θα χρησιμοποιηθεί παρακάτω για να γίνει ανανέωση του container των γραφικών παραστάσεων για να εμφανιστούν τα δεδομένα για τη νέα μετοχή που αναζητήθηκε.

Στη συνέχεια θα αναλυθεί το container `col-labels` στο οποίο παρουσιάζονται οι πληροφορίες των μετοχών.

```

</div>
<div class="col-labels">

  <div class="holder">
    <!-- Stocks name -->
    <p id="holder" style="margin-top:20px;">Microsoft Corp</p>

    <!-- Stocks symbol -->
    <p id="holderSymbol" >MSFT </p><a id="difference"></a>

  </div>
  <div class="description">
    <!-- Open -->
    <p class="card-text">Open:<a id="open"></a></p>

    <!-- High -->
    <p class="card-text">High:<a id="high"></a></p>

    <!-- Low -->
    <p class="card-text">Low:<a id="low"></a></p>

    <!-- Prev Close -->
    <p class="card-text">Prev Close:<a id="PrevClose"></a></p>

  </div>
</div>

```

Κώδικας 44: Δημιουργία html αντικειμένων για τις πληροφορίες της μετοχής

Στον κώδικα παραπάνω φαίνονται τα στοιχεία που φτιάχτηκαν και στα οποία θα περαστούν το όνομα της μετοχής, το σύμβολο της μετοχής, η τιμή previous close, η τιμή low, η τιμή high και η τιμή open. Για να περαστούν οι τιμές σε αυτά τα στοιχεία html που έχουμε φτιαχτεί πρέπει αναλυθεί ο κώδικας στο αρχείο **api.js** και συγκεκριμένα στη συνάρτηση `getDataReal()` η οποία παίρνει τα δεδομένα για την τωρινή ημέρα. Αρχικά πρέπει να βρεθούν τα δεδομένα για την ώρα που ανοίγει και κλείνει το χρηματιστήριο. Για να συμβεί αυτό θα χρησιμοποιηθεί ο παρακάτω κώδικας που ψάχνει να βρει στο πίνακα τα δεδομένα που αναζητά ο χρήστης ανάλογα με την ώρα.

```
var secIndex = date_array_Real.findIndex(element => element.endsWith("16:00:00")); // End hour
var openIndex = date_array_Real.findIndex(element => element.endsWith("09:45:00")); // Start hour
```

Κώδικα 45: Εντοπισμός θέσης στοιχείων της ώρας που ανοίγει και κλείνει το χρηματιστήριο στο πίνακα

Αφού βρεθούν οι θέσεις στις οποίες βρίσκονται οι ώρες που ανοίγει και κλείνει το χρηματιστήριο, τώρα μπορούν να περαστούν στα html στοιχεία που φτιάχτηκαν.

```
// Add open,high,low,previous close to cardBody at stocks and userpage
document.getElementById("open").innerHTML = " " + open_array_Real[openIndex];
document.getElementById("high").innerHTML = " " + high_labels_1Day[indexOfMax(close_labels_1Day_chart)];
document.getElementById("low").innerHTML = " " + low_labels_1Day[indexOfMin(close_labels_1Day_chart)];
document.getElementById("PrevClose").innerHTML = " " + close_array_Real[secIndex];
```

Κώδικα 46: Ανάθεση τιμών στα html στοιχεία με τις πληροφορίες της μετοχής

Θα χρησιμοποιηθούν πάλι το document.getElementById() με το οποίο θα δειχθεί σε ποιο στοιχείο θα σταλθούν οι τιμές, με βάση το id τους. Για το open στοιχείο θα εισαχθεί η τιμή για την ώρα που ανοίγει το χρηματιστήριο και για το PrevClose θα εισαχθεί η τιμή για την ώρα που κλείνει το χρηματιστήριο. Τώρα για το low και high πρέπει να γίνουν κάποιοι υπολογισμοί, πρέπει να βρεθούν η μεγαλύτερη και η μικρότερη close τιμή της ημέρας. Για τον λόγο αυτόν θα κατασκευαστούν δυο νέες συναρτήσεις, το indexOfMax() και το indexOfMin() στις οποίες θα εισαχθούν στον πίνακα με τις close τιμές και θα υπολογίζουν τη μεγαλύτερη και τη μικρότερη τιμή που έχει φτάσει η μετοχή.

```

// Function to find max of array
function indexOfMax(arr) {

    if (arr.length === 0) {

        return -1;

    }

    var max = arr[0];
    var maxIndex = 0;

    for (var i = 1; i < arr.length; i++) {

        if (arr[i] > max) {

            maxIndex = i;
            max = arr[i];

        }

    }

    return maxIndex;

}

```

Κώδικας 47: Εντοπισμός μεγαλύτερης τιμής στο πίνακα μας

Στη συνάρτηση `indexOfMax()` αρχικά δηλώνεται ότι το μεγαλύτερο στοιχείο είναι αυτό που βρίσκεται στη θέση 0 και το περνάει στη μεταβλητή `max` καθώς και τη θέση 0 στη μεταβλητή `maxIndex`, στη συνέχεια με τη χρήση μιας `for` επανάληψης ελέγχεται αν κάθε στοιχείο του πίνακα είναι μεγαλύτερο από το `max` και αν είναι, τότε αυτό το στοιχείο θα γίνεται το `max` και το `maxIndex` θα παίρνει τον αριθμό της θέσης αυτού του στοιχείου. Μόλις τελειώσει η επανάληψη επιστρέφεται το `maxIndex` το οποίο δείχνει σε ποια θέση του πίνακα βρίσκεται το μεγαλύτερο στοιχείο. Με παρόμοιο τρόπο λειτουργεί και η συνάρτηση `indexOfMin()` αλλά τώρα πρέπει να βρεθεί το μικρότερο στοιχείο του πίνακα, οπότε στη συγκεκριμένη περίπτωση η συνάρτηση θα ελέγχει αν το κάθε στοιχείο του πίνακα είναι μικρότερο από το άλλο. Στο τέλος θα επιστρέφει και αυτή, τη θέση με το μικρότερο στοιχείο του πίνακα `close`. Ένα ακόμα στοιχείο που παρουσιάζεται στον πίνακα με τις πληροφορίες κάθε μετοχής είναι η διαφορά ανάμεσα στη `close` τιμή όταν έκλεισε το χρηματιστήριο την προηγούμενη μέρα και τη τιμή του χρηματιστηρίου όταν άνοιξε. Στον παρακάτω κώδικα φαίνεται ο τρόπος που υπολογίζεται η διαφορά

```
// Difference for current close and previous close
var difference = close_array_Real[getIndex] - close_array_Real[secIndex];

// Percentage Change in Stock Price
var percentage = ((difference / close_array_Real[secIndex]) * 100);
```

Κώδικα 48: Υπολογισμός διαφοράς και ποσοστού μεταξύ των δυο τιμών της μετοχής

Τέλος, ανάλογα με το αν η διαφορά είναι αρνητική ή θετική ορίζεται και το χρώμα της γραφικής παράστασης, αν είναι θετικό που σημαίνει ότι ανέβηκε η μετοχή το χρώμα θα είναι πράσινο, σε αντίθετη περίπτωση θα είναι κόκκινο.

```
If Percentage Change is Increased add + symbol ,green color and up arrow
(close_array_Real[getIndex] >= close_array_Real[secIndex]) {

    document.getElementById("difference").innerHTML = " + " + difference.toFixed(2) + "&nbsp;(" + percentage.toFixed(2) + "%)&nbsp;" + "&uarr;";
    document.getElementById("difference").style.color = "green";
    colorStock = "green";

If Percentage Change is decreased red color and down arrow
e {

    document.getElementById("difference").innerHTML = difference.toFixed(2) + "&nbsp;(" + percentage.toFixed(2) + "%)&nbsp;" + "&darr;";
    document.getElementById("difference").style.color = "red";
    colorStock = "red";
```

Κώδικα 49: Εισαγωγή χρώματος στη γραφική παράσταση τωρινής μέρας

Έχοντας ολοκληρωθεί και το δεύτερο container, παρουσιάζεται το τελευταίο container το οποίο περιλαμβάνει την μπάρα αναζήτησης. Μέσα από αυτή τη μπάρα ο χρήστης θα μπορεί να αναζητεί το σύμβολο ή την ονομασία της μετοχής και όταν πατάει το κουμπί, αυτομάτως τα δεδομένα της νέας μετοχής θα περνάνε στη γραφικές παραστάσεις.

Για να δημιουργηθεί η μπάρα αναζήτησης θα χρησιμοποιηθεί το form. Το form αποτελείται από στοιχεία, όπου μπορεί να είναι πολλών ειδών αλλά σε αυτή τη περίπτωση είναι μία μπάρα εισόδου, όπου ο χρήστης εισάγει τη μετοχή που θέλει να ψάξει και ένα κουμπί που όταν το πατάει ο χρήστης το σύμβολο της μετοχής στέλνεται πίσω στον server.

```

<!-- form to send stocks name to server -->
<form id="form-search" action="javascript:void(0)" class="form-inline md-form mr-auto mb-2" autocomplete="off">
  <div class="input-group">
    <input id="myInput" type="text" name="Stocks" class="form-control mr-sm-2" type="text"
      placeholder="Search" aria-label="Search">
    <button id="suButton" type="submit" ><i class="fa fa-search"></i></button>
  </div>
  <!-- Btn for submitting stock -->
  <!-- <button class="btn aqua-gradient btn-rounded btn-sm my-0" type="submit" id="suButton">Search</button> -->
</form>

```

Κώδικα 50: Δημιουργία form στη σελίδα Stocks

Όπως φαίνεται στον κώδικα δίπλα από το form, έχει περαστεί στο actions το javascript:void(0) το οποίος είναι ένας τρόπος για να εμποδιστεί η form να εκτελέσει τη λειτουργία που είναι προγραμματισμένη να κάνει, αλλά να πραγματοποιήσει αυτό που θα ορίσει ο προγραμματιστής. Στη συνέχεια θα παρουσιαστεί ο κώδικας που βρίσκεται στο αρχείο **search.js** :

```

// Execute function when someone clicks in the document
document.getElementById('suButton').addEventListener("click", function (e) {

```

Κώδικα 51: Με τη χρήση της JavaScript ακούμε όταν ο χρήστης πατάει το κουμπί αναζήτησης

Δηλώνετε το id που έχει το κουμπί αναζήτησης και λέει ότι όταν γίνει η ενέργεια click θέλει να εκτελέσει τη συνάρτηση μέσα στην οποία θα γίνει ανανέωση όλων των στοιχείων που πρέπει να αλλάξουν, γραφικές παραστάσεις, πληροφορίες μετοχής, κουμπί (αναζήτησης μετοχής).

```

// Refresh function
async function reload() {

    // Refresh tabs and stock without refreshing the page using jquery
    $("#nav-tabContent").load(window.location.href + " #nav-tabContent");
    $("#stock").show();

    // Wait to refresh div and then add the new charts
    sleep(1000).then(() => {

        // When click search go back to the first Tab
        $('.nav-tabs a:first').tab('show');

        // Refresh charts without refreshing the page
        chartIt1Day(symbol, change);
        chartIt3Days(symbol, change);
        chartIt1Month(symbol, change);
        chartIt4Months(symbol, change);
        chartIt(symbol, change);
        chartIt1Year(symbol, change);
        chartIt5Years(symbol, change);

        // Refresh cardBody without refreshing the page
        document.getElementById("holder").innerHTML = define[number];
        document.getElementById("holderSymbol").innerHTML = symbol;

    });
};
}, { passive: true });

```

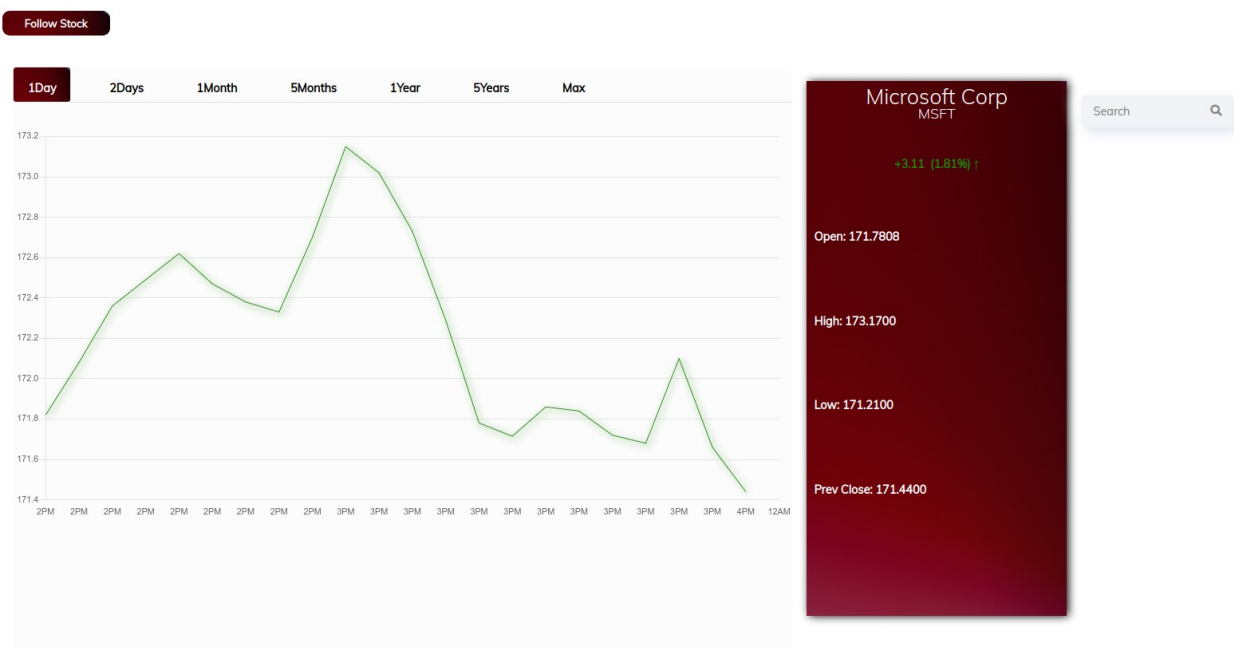
Κώδικας 52: Συνάρτηση ανανέωσης html στοιχείων στη σελίδα Stocks

Αφού γίνει ανανέωση των συγκεκριμένων containers μετά από μια χρόνο-καθυστέρηση ενός δευτερολέπτου, αρχίζουν να περνούν τα νέα δεδομένα στις γραφικές παραστάσεις καλώντας τις αντίστοιχες συναρτήσεις. Τέλος περνούν και τα δεδομένα για τη καινούργια μετοχή στο πίνακα με τις πληροφορίες. Οι συναρτήσεις που καλούνται στέλνουν το σύμβολο της μετοχής που ο χρήστης εισήγαγε στον server, ο οποίος και καλεί την αντίστοιχη api. Στη συνέχεια με τα νέα δεδομένα γεμίζουν ξανά οι γραφικές παραστάσεις με τις τιμές της νέας μετοχής που αναζήτησε ο χρήστης της διαδικτυακής εφαρμογής.

Δ.1.3.8 Αποθήκευση Μετοχών στη Βάση Δεδομένων

Όταν κάποιος δημιουργεί λογαριασμό και συνδέεται στην εφαρμογή, έχει τη δυνατότητα να ακολουθεί μετοχές στη σελίδα stocks και στη συνέχεια να εμφανίζονται στο userpage του. Για να έχει ο χρήστης όμως αυτή τη λειτουργία, στον συνδεδεμένο χρήστη πρέπει να παρουσιάζεται ένα

κουμπί το οποίο θα έχει σαν τιμή το σύμβολο της μετοχής και μόλις το πατήσει ο χρήστης αυτομάτως αυτό θα στέλνεται στον server και μετά στη Βάση δεδομένων.



Εικόνα 45: Κουμπί αποθήκευσης μετοχής στη σελίδα Stocks

Στη φωτογραφία φαίνεται το κουμπί με το οποίο ακολουθεί ο χρήστης μια μετοχή και εμφανίζεται μόνο στους συνδεδεμένους χρήστες.

```
<!-- Follow btn -->
<div class="FollowStock">

  <%if (user) { %>      You, 2 months ago • Add real time notifications fixed UI

  <form id="send" action="javascript:void(0)">

    <!-- Send stock name to server -->
    <button type="submit" name="stock" id="stock" class="buttonStock" style="vertical-align:middle"
      value="stock :">

      <span>Follow Stock </span>
    </button>

  </form>

  <%}%>

</div>
```

Κώδικα 53: Κουμπί αποθήκευσης μετοχής στη βάση δεδομένων

Στον κώδικα παραπάνω φαίνεται ότι με τη βοήθεια της `ejs` γίνεται ο έλεγχος για να καταλάβει η εφαρμογή αν είναι συνδεδεμένος ο χρήστης που την χρησιμοποιεί, στη συνέχεια χρησιμοποιείται το `form` μέσα στο οποίο υπάρχει το κουμπί. Όπως φαίνεται στον κώδικα, πάλι με τη χρήση του `javascript:void(0)` εμποδίζεται το `form` να εκτελέσει την προγραμματισμένη του λειτουργία και θα οριστεί μια νέα λειτουργία για το ποια λειτουργία πρέπει να εκτελέσει όταν ο χρήστης πατήσει το κουμπί που βρίσκεται μέσα στη αυτήν.

```
// Send stock to server and stops form to reload the page
$('#send').submit(function (e) {

    // Stop form from refreshing page
    e.preventDefault();

    // Send stock to server
    $.ajax({

        url: '/getStock/' + symbol + '',
        dataType: 'text',
        type: 'post',
        cache: false,

    });
});
```

Κώδικα 54: Στέλνουμε το σύμβολο της μετοχής στον server για να το αποθηκεύσει στη βάση δεδομένων

Με τη χρήση της `jQuery` και της `Ajax` στέλνεται ένα `POST` αίτημα στον `server` με το σύμβολο της μετοχής που ζητάει ο χρήστης να αποθηκεύσει. Στη συνέχεια ο `server` δέχεται το σύμβολο της μετοχής και το περνάει στη βάση δεδομένων.

```

// Get stock's name
router.post('/getStock/:symbol', (req, res) => {

  // Find user in database by id
  const user = req.user._id;
  const symbol = req.params.symbol;
  //Find if user has already followed stock
  User.findOne({ _id: user, stock: symbol }).then(result => {

    if (result) {

      console.log("already exist");
    }

    else {

      // Add stock to database
      User.findOneAndUpdate({ _id: user }, { '$push': { stock: req.params.symbol } }, (err, doc) => {

        if (err) {

          console.log("Something wrong when updating data!");
        }

        console.log(doc);
      })
    }
  });
});

```

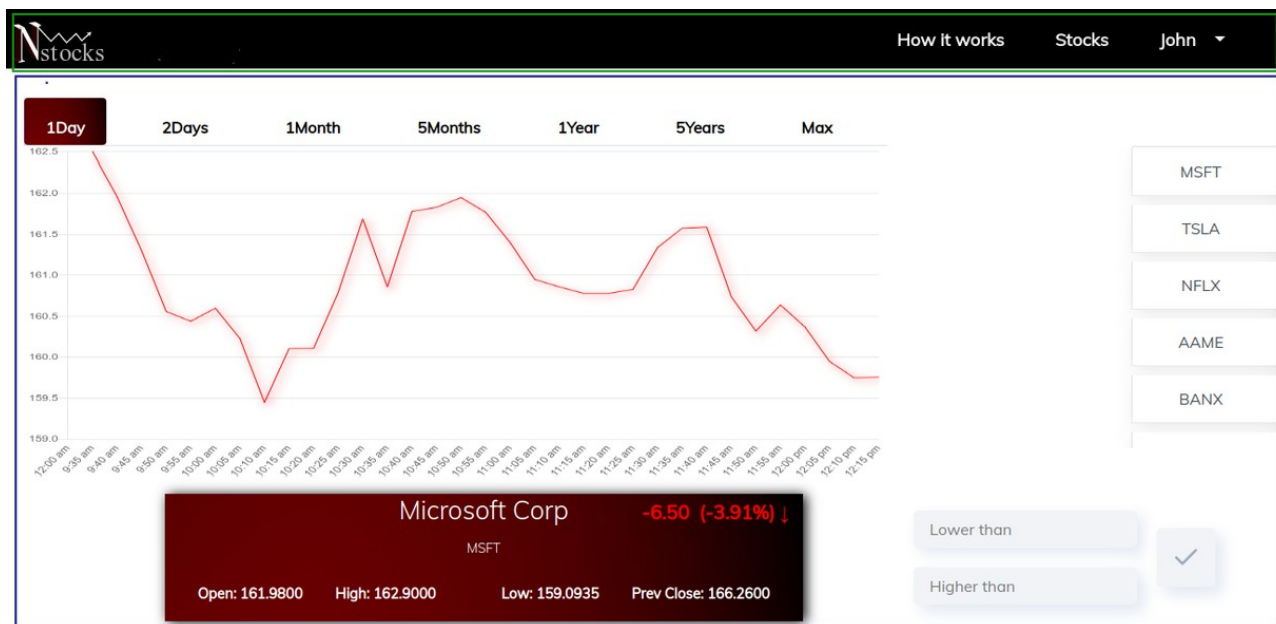
Κώδικας 55: Ο server δέχεται το σύμβολο και το περνάει στη βάση δεδομένων

Για να γίνει η αποθήκευση πρέπει πρώτα να δούμε αν υπάρχει ήδη αποθηκευμένο αυτό το σύμβολο στη βάση δεδομένων του χρήστη. Αυτό θα το επιτευχθεί με τη `findOne()` μέθοδο η οποία επιστρέφει `true` ή `false` ανάλογα με το αν βρέθηκε ή όχι το σύμβολο της μετοχής. Στη συνέχεια αν δεν βρεθεί, χρησιμοποιείται η μέθοδος `findOneAndUpdate()` στη οποία περνάει σαν όρισμα το `id` του χρήστη καθώς και η νέα μετοχή, όπου με τη χρήση του `$push` ζητάει να προστεθεί στο πίνακα των μετοχών ένα ακόμα στοιχείο. Το `$push` χρησιμοποιείται επειδή το `stock` στη βάση δεδομένων είναι πίνακας.

Α.1.3.9 Δημιουργία σελίδας `userpage`

Όπως αναφέρθηκε και πιο πάνω, για να μπορέσει να περιηγηθεί ο χρήστης στη βάση δεδομένων πρέπει πρώτα να συνδεθεί στη διαδικτυακή εφαρμογή. Αφού περιηγηθεί ο χρήστης στη σελίδα `userpage` εκεί έχει τη δυνατότητα να παρακολουθεί τις μετοχές που έχει ακολουθήσει καθώς και να ορίσει το ανώτερο και κατώτερο όριο μιας μετοχής έτσι ώστε ο server να παρακολουθεί τη τιμή της μετοχής αυτής και αν πέρασε αυτά τα όρια να τον ειδοποιεί. Οι γραφικές παραστάσεις στη σελίδα

Userpage κατασκευάστηκαν με τον ίδιο τρόπο με αυτόν στη σελίδα stocks οπότε δεν θα αναλυθεί ο κώδικας τους.



Εικόνα 46: Containers σελίδας userpage

Στη φωτογραφία παραπάνω φαίνονται τα δυο κύρια containers που χρησιμοποιήθηκαν στην εφαρμογή, τα οποία είναι child containers του parent container wrapper.

```
<div class="wrapper">
  <!-- Container for navbar-->
  <div class="container-navbar">
    You, 4 months ago • Add comments to code
    <div class="row-nav">
      <% include ./navbar %>
    </div>
  </div>

  <!-- Container for chart, search, follow btn and card body-->
  <div class="container-stock">
```

Κώδικας 56: container για την εισαγωγή μπάρας πλοήγησης στη σελίδα userpage

Στη συνέχεια θα παρουσιαστεί ο κώδικας στο αρχείο **dashboard.ejs**. Όπως φαίνεται δημιουργείται το container-navbar στο οποίο τοποθετείτε το αρχείο navbar με τη βοήθεια της `ejs` και το container-stock στο οποίο τοποθετούνται όλα τα στοιχεία τα οποία θα έχει η σελίδα μας. Βλέποντας το αρχείο **dashboard.css** χρησιμοποιείται πάλι η ιδιότητα `display:flex` για να οριστεί ο τρόπος με τον οποίο αυτά τα δυο containers θα κατανεμηθούν στο χώρο.

```
body{
  height: 100vh;
}
.wrapper{
  display: flex;
  height:100vh;
  flex-direction: column;
}
```

Κώδικα 57: Χρήση ιδιότητας *flexbox* στη σελίδα *userpage*

Στη συνέχεια χρησιμοποιείται το `flex:direction` για να τοποθετηθούν τα container κατά σειρά.

Συγκεκριμένα για το container-stock το οποίο αποτελείται από τρεις σειρές `row0`, `row1` και `row2`.

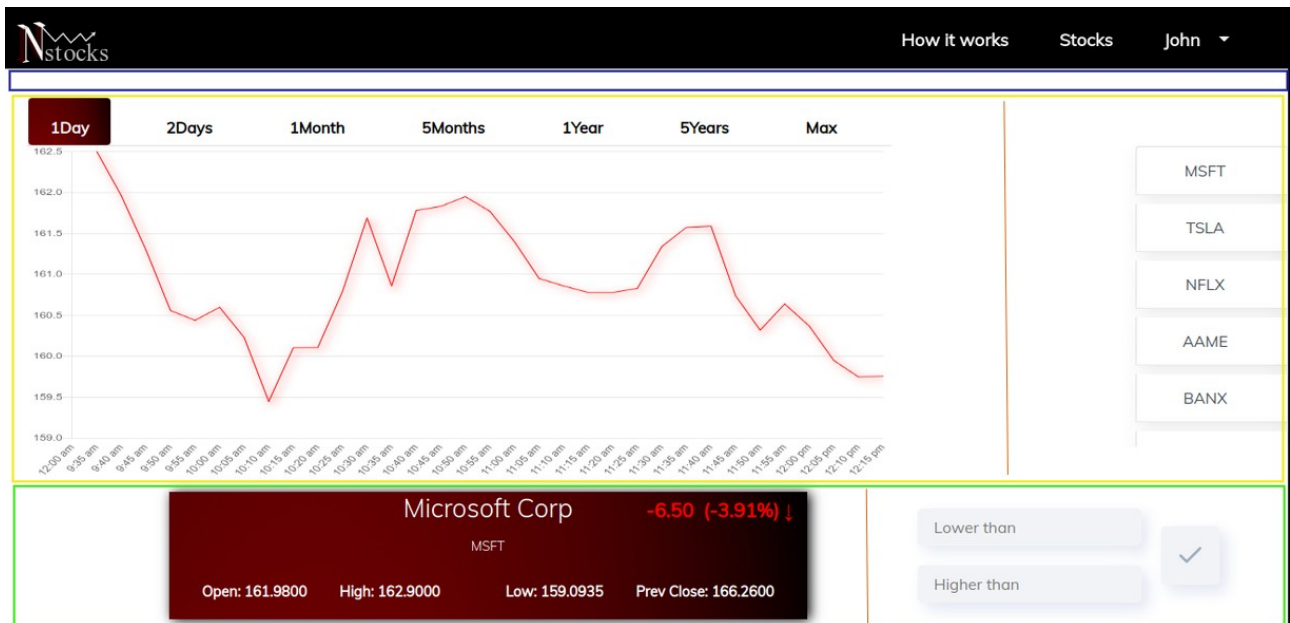
```
.row0{
  height:5%;
  width: 100%;
}
.row1{
  display: flex;
  flex-direction: row;
  width: 100%;
  height:70%;
  background-color: #ffffff;
}

.row2{
  width: 100%;
  height:25%;
  background-color: #ffffff;
  display: flex;

  justify-content: space-between;
  align-items: center;
}
```

Κώδικα 58: Child containers του container-stock στη σελίδα *userpage*

Το row0 αποτελεί το 5% του ύψους της σελίδας, το row1 αποτελεί το 70% της σελίδας και το row2 αποτελεί το 25% της σελίδας, όπου συνολικά αυτά τα τρία containers μαζί αποτελούν το 100% της κάθε οθόνης στην οποία ανοίγεται η σελίδα userpage. Στη φωτογραφία παρακάτω φαίνονται και τα τρία containers, πως έχουν στοιβαχθεί το ένα πάνω από το άλλο κατά σειρά. Αυτό έγινε με τη χρήση της ιδιότητας flex-direction:column.



Εικόνα 47: Τα τρία βασικά containers της σελίδας Userpage

Ας δούμε τώρα το row1 container το οποίο αποτελείται από τα child containers col-chart και col-list, μέσα στα οποία βρίσκονται οι γραφικές παραστάσεις και η λίστα με τις αποθηκευμένες μετοχές του χρήστη. Πάλι με τη χρήση του flexbox χωρίζεται το parent container row1 κατά στήλη, όπου το 70% του πλάτους του row1 αποτελείται από το col-chart και το 30% από το col-list.

```

.col-chart{
  width: 70%;
  height: 100%;
  background-color: #ffffff;
}

.col-list{
  position:static;
  height: 100%;
  width: 30%;
  background-color: #ffffff;
  display:flex;
  justify-content:flex-end;
}

```

Κώδικας 59: Child containers του row1 container τη σελίδα userpage

Τέλος το row2 και αυτό με τη σειρά του αποτελείται από δυο child containers, το col-labels και το col-searchValues. Μια από τις καινούργιες λειτουργίες που συναντάτε σε αυτή τη σελίδα είναι η λίστα με τις μετοχές. Για να φτιαχτεί χρησιμοποιείται αρχικά html για αυτό στη συνέχεια θα αναλυθεί ο κώδικας που υπάρχει στο αρχείο **dashboard.ejs**.

Δ.1.3.10 Δημιουργία λίστας μετοχών

```

<div class=" list-container">
  <ul class="list-group-stocks" id="listStocks"> </ul>
</div>

```

Κώδικας 60: Container για τη λίστα με τις μετοχές του χρήστη στη σελίδα userpage

Το < ul > είναι ένα block στοιχείο το οποίο χρησιμοποιείτε για να προσδιοριστεί μια μη ταξινομημένη λίστα. Μέσα σε αυτή τη λίστα πρέπει να υπάρχουν στοιχεία < li > τα οποία αποτελούν και τα αντικείμενα της λίστας. Όπως φαίνεται στον κώδικα παραπάνω η λίστα δεν

περιέχει αντικείμενα. Αυτό συμβαίνει γιατί δεν γίνεται να γνωρίζετε το περιεχόμενο της λίστας καθώς αυτό εξαρτάται από τον κάθε χρήστη. Αυτό που θα γίνει είναι να συνδεθεί η λίστα με τη βάση δεδομένων έτσι ώστε κάθε φορά που ένας χρήστης πηγαίνει στη σελίδα userpage αυτόματα να γεμίζει με τις μετοχές που αυτός έχει ακολουθήσει. Θα αφιερωθεί λίγος χρόνος παραπάνω σε αυτό το κομμάτι γιατί είναι αρκετά ενδιαφέρον. Το πρώτο πράγμα που ζητείται είναι να δημιουργηθεί μία επανάληψη έτσι ώστε μέσα σε αυτήν να δημιουργούνται τα αντικείμενα της λίστας, για να συμβεί αυτό πρέπει πρώτα να γνωρίζεται πόσες μετοχές έχει ακολουθήσει ο χρήστης. Για να επιτευχθεί αυτό θα πρέπει να σταλθεί αίτημα στο server για να στείλει αυτές τις πληροφορίες.

```
$.ajax({
    url: '/getUser',
    dataType: 'json',
    type: 'get',
    cache: false,
    success: function (data) {

        var place = 0; // Counter

        // Put User's stocks to the list on userpage
        for (var Userstocks in data["stock"]) {

            if (data["stock"][Userstocks] != "null") {

                followed_stocks.push(data["stock"][Userstocks]);

            }

        }

    }
});
```

Κώδικας 61: Αίτημα από το client για να στείλει ο server τις μετοχές που έχει ακολουθήσει ο χρήστης

Αφού σταλθεί το αίτημα, ο server πρέπει να λάβει τα δεδομένα από τη βάση δεδομένων και να τα επιστρέψει στον client. Έχει ξανά αναφερθεί αυτή η μέθοδος αρκετές φορές πιο πάνω, οπότε δεν θα αναλυθεί. Όταν η απάντηση από το server επιστρέψει με επιτυχία, σειρά έχει να περάσουμε τα δεδομένα μόνο για τις μετοχές που ακολουθεί ο χρήστης, σε έναν πίνακα. Έχοντας όλες τις μετοχές σε έναν πίνακα πλέον είναι γνωστό ποιες είναι αυτές αλλά και πόσες είναι, οπότε τώρα πρέπει να δημιουργηθεί η επανάληψη.

```
// Create elements at list. Add stock which user follows
for (var i = 0; i < followed_stocks.length; i++) {

    var x = document.createElement("LI");
    var t = document.createTextNode(followed_stocks[i]);

    x.appendChild(t);
    x.id = followed_stocks[i];
    document.getElementById("listStocks").appendChild(x);
    document.getElementById(x.id).className = "list-group-item list-group-item-action";
}
```

Κώδικας 62: Επανάληψη που εισάγει τις μετοχές του χρήστη από τον πίνακα στη λίστα

Στον κώδικα παραπάνω, για κάθε στοιχείο του πίνακα δημιουργείται ένα html στοιχείο `<li>`, το οποίο επιτυγχάνεται χρησιμοποιώντας τη μέθοδο `.createElement()`. Αφού δημιουργηθεί, με τη μέθοδο `.createTextNode()` ορίζεται το κείμενο που θα έχει αυτό το στοιχείο, στη συγκεκριμένη περίπτωση το όνομα της μετοχής. Για να συνδεθούν όμως αυτά τα δυο μεταξύ τους χρησιμοποιείται η μέθοδος `.appendChild()`. Για να μπορούν μετά να διαχειριστούν αυτά πρέπει να δοθεί στο καθένα ένα ξεχωριστό ένα μοναδικό id, όπου σε αυτή τη περίπτωση θα είναι το όνομα της μετοχής. Όταν ολοκληρωθεί η διαδικασία και φτιαχτεί το αντικείμενο, μετά πρέπει να περαστεί μέσα στη λίστα που φτιάχτηκε στο html αρχείο, το οποίο θα γίνει χρησιμοποιώντας το `.appendChild()`, όπως και πιο πάνω.



Εικόνα 48: Αντικείμενο λίστας μετοχών στη σελίδα Userpage

Μια ακόμη σημαντική λειτουργία για τη λίστα είναι να μπορεί να διαγραφούν κάποιες μετοχές τις οποίες δεν θα θέλει πια ο χρήστης να παρακολουθεί. Για να το λόγο αυτό όταν ο χρήστης πάει το κέρσορα πάνω στη μετοχή εμφανίζεται μια άσπρη γραμμή η οποία όταν πατηθεί διαγράφει αρχικά τη μετοχή από λίστα και έπειτα τη μετοχή από τη βάση δεδομένων.

```

var DeleteStock = document.createElement('i');
DeleteStock.className = "fa fa-minus";
DeleteStock.id = followed_stocks[i] + "s";
DeleteStock.onclick = function (e) {
    event.stopPropagation();
    var target = $(this).closest('li').attr('id');
    a few seconds ago • Uncommitted changes

```

Κώδικας 63: Διαγραφή μετοχής από τη λίστα του χρήστη στη σελίδα Userpage

Για επιτευχθεί αυτό μέσα στη επανάληψη που δείχθηκε πιο πάνω θα δημιουργηθεί ένα html στοιχείο `< i >` του οποίου θα δοθεί σαν όνομα κλάσης `fa fa-minus`. Αυτό θα δημιουργήσει τη γραμμή στην οποία θα δοθεί και ένα ξεχωριστό `id` για το αντικείμενο της λίστας. Ένα πρόβλημα που δημιουργείται τώρα είναι ότι επειδή η γραμμή για τη διαγραφή της μετοχής βρίσκεται μέσα στο `container` της μετοχής, όπου και να πατήσει ο χρήστης μέσα σε αυτό δεν θα έχει σημασία. Επειδή θέλουμε ο χρήστης να μπορεί να πατήσει πάνω στη μετοχή για να του δείξει τη γραφική της παράσταση και μόνο όταν πατάει πάνω στη γραμμή να διαγράφεται, θα χρησιμοποιηθεί μια μέθοδος που λέγεται `.stopPropagation()`. Η συγκεκριμένη μέθοδος εμποδίζει το `child` στοιχείο να μεταδώσει τη λειτουργία του στο `parent container`.

```

$(this).parent().fadeOut(1000, 0.01, function () {
    $(this).slideUp(150, function () {
        $(this).remove();
    });
});

$.ajax({
    url: '/deleteStock/' + target + '',
    dataType: 'text',
    type: 'post',
    cache: false,
    success: function (result) {
        console.log(result);
        if (result.status == 200) {
            console.log("Status:200");
        }
    },
    error: function (result) {
        console.log(result);
    }
});
});

};

x.appendChild(DeleteStock);

```

Κώδικας 64: Αποστολή αιτήματος διαγραφής μετοχής στον server για να τη διαγράψει από τη βάση δεδομένων

Μετά θα χρησιμοποιηθούν animations για να εξαφανίζεται το container της μετοχής από τη λίστα και με τη βοήθεια της Ajax θα στέλνεται το POST αίτημα στο server για να διαγράψει τη μετοχή από τη βάση δεδομένων. Τέλος, με τη χρήση πάλι του .appendChild() εισάγεται στη γραμμή σε κάθε στοιχείο της λίστας.

```

router.post('/deleteStock/:target', (req, res) => {
  const target = req.params.target;
  const user = req.user._id;

  User.updateOne({ _id: user }, { '$pull': { "stock": { $in: [target] } } }, (err, doc) => {
    if (err) {
      console.log("Something wrong when updating data!");
    }
    console.log(doc);
  });

  res.json({ success: "Updated Successfully", status: 200 });
});

```

Κώδικα 65: Διαγραφή μετοχής από τη βάση δεδομένων

Στον server αυτό που συμβαίνει είναι ότι με τη μέθοδο `.updateOne()` φανερώνεται το user με το id του χρήστη που είναι συνδεδεμένος και με τη χρήση του `$pull` δηλώνεται ότι ζητείται να αφαιρεθεί από το πίνακα `stocks` η συγκεκριμένη μετοχή. Το `$in` χρησιμοποιείται για να δηλωθεί το όνομα της μετοχής που βρίσκεται στη μεταβλητή `target`. Αφού ολοκληρωθεί η διαδικασία, ο server στέλνει σαν απάντηση ένα `.json` με το `status:200` το οποίο δέχεται η Ajax συνάρτηση και καταλαβαίνει ότι πραγματοποιήθηκε επιτυχώς το αίτημα της.

```

// When clicking a stock at list search for the stock and visualize it with the charts
document.getElementById("listStocks").addEventListener("click", function (e) {

  // e.target is our targetted element.
  if (e.target && e.target.nodeName == "LI") {
    symbolStock = e.target.id;
  }
});

```

Κώδικα 66: Κώδικας με τον οποίο αντιλαμβάνεται η εφαρμογή ότι ο χρήστης έκανε `click` ένα αντικείμενο της λίστας

Τώρα θα παρουσιαστεί ο τρόπος με τον οποίο εμφανίζεται η γραφική παράσταση της μετοχής που ψάχνει ο χρήστης από τη λίστα του. Όταν πατήσει το κουμπί της μετοχής, το id του κουμπιού της που όπως φαίνεται και πιο πάνω είναι το όνομα της μετοχής, περνάει στη μεταβλητή `symbolStock`. Στη συνέχεια με τη βοήθεια της Ajax στέλνεται το σύμβολο πίσω στον server για να ανακτήσει τα δεδομένα, τα οποία επιστρέφονται για να τα οπτικοποιηθούν στις γραφικές παραστάσεις. Η διαδικασία είναι ακριβώς ίδια με αυτή στη σελίδα `stocks`, οπότε δεν θα αναλυθεί περαιτέρω.

Τέλος, μια λειτουργία που αξίζει να αναφερθεί είναι ότι η γραμμή διαγραφής της μετοχής εμφανίζεται όταν ο χρήστης έχει το ποντίκι πάνω στο container της μετοχής, και εξαφανίζεται όταν παίρνει το ποντίκι. Υπό άλλες συνθήκες αυτό μπορεί να γίνει εύκολα με τη css και συγκεκριμένα τη pseudo-class :hover, αλλά στη συγκεκριμένη περίπτωση δεν γίνεται. Για να το πραγματοποιηθεί η συγκεκριμένη λειτουργία θα χρησιμοποιηθεί javascript με την οποία θα δηλωθεί ότι όταν το ποντίκι είναι πάνω σε οποιοδήποτε σημείο του container της μετοχής τότε θα προστίθεται στο < i > στοιχείο του συγκεκριμένου parent container η κλάση showFa. Αυτό γίνεται με τη μέθοδο .addClass(), όπου με τη χρήση της css έχει ορίσει την ιδιότητα opacity: 1. Επειδή αρχικά το opacity του στοιχείου έχει οριστεί 0, δεν φαίνεται, όταν όμως ο χρήστης πάει το ποντίκι πάνω από το parent container του, το opacity του γίνεται 1 και μπορεί πλέον ο χρήστης να το δει. Το ακριβώς αντίθετο συμβαίνει όταν πάρει κάποιος το ποντίκι πάνω από το container, όπου εφαρμόζεται η μέθοδος .removeClass() και η κλάση showFa διαγράφεται από το στοιχείο < i >, με αποτέλεσμα το opacity να γίνεται πάλι 0.

```
document.getElementById("listStocks").addEventListener("mouseover", function (e) {  
    // e.target is our targetted element.  
    if (e.target && e.target.nodeName == "LI") {  
        symbolStocks = e.target.id;  
        var s = "s";  
        $("#" + symbolStocks + "" + s + "").addClass('showFa');  
        document.getElementById(symbolStocks + s).addEventListener("mouseover", function (e) {  
            $("#" + symbolStocks + "" + s + "").addClass('showFa');  
        });  
    }  
});
```

Κώδικας 67: Εμφάνιση συμβόλου διαγραφής μετοχής όταν το ποντίκι βρίσκεται πάνω σε κάποιο στοιχείο της λίστας

Δ.1.3.10 Ορισμός ανώτερου, κατώτερου ορίου τιμής μετοχών

Μια πολύ σημαντική λειτουργία της διαδικτυακής εφαρμογής είναι η δυνατότητα που δίνεται στον χρήστη να ορίζει κάποια όρια και αν η μετοχή πέσει ή ανέβει πάνω ή κάτω από αυτά να τον ειδοποιεί με το κατάλληλο μήνυμα. Για να κατασκευαστεί αυτή η λειτουργία, αρχικά χρειάζεται μία

φόρμα μέσα από την οποία ο χρήστης θα εισάγει αυτές τις τιμές και με τη βοήθεια ενός κουμπιού θα στέλνει τις τιμές στον server, όπου από εκεί στέλνονται στη βάση δεδομένων.

```
<form id="formInput" action="javascript:void(0)" class="form-inline md-form mr-auto mb-2"
  autocomplete="off">
  <div class="PutValue">
    <!-- Input text for min stock value -->
    <input id="min" type="text" name="minStock" class="form-control mr-sm-2" type="text"
      placeholder="Lower than" aria-label="Input">

    <!-- Input text for max stock value -->
    <input id="max" type="text" name="minStock" class="form-control mr-sm-2" type="text"
      placeholder="Higher than " aria-label="Input">
  </div>
  <div class="PutBtn">
    <!--Btn for submitting stock -->
    <button class="btn aqua-gradient btn-rounded btn-sm my-0" type="submit" id="subBtn"> <i
      data-feather="check" class='icon'></i>
    </button>
  </div>
</form>
```

Κώδικα 68: Δημιουργία φόρμας για την εισαγωγή ανώτερης και κατώτερης τιμής

Όπως φαίνεται στον κώδικα παραπάνω δημιουργούνται δυο inputs με id min και max, καθώς και ένα κουμπί που θα στέλνει τις τιμές στον server. Προτού αναλυθεί η λειτουργικότητα αυτών, θα ήταν σημαντικό να γίνει μια σύντομη αναφορά για την εμφάνιση των inputs και του κουμπιού.

Για τη διακόσμηση χρησιμοποιείται μια νέα μόδα που ονομάζεται Neumorphism, είναι μια minimal διακόσμηση που δίνει την εντύπωση ότι τα html στοιχεία είναι συνδεδεμένα με το φόντο. Το πιο σημαντικό σε αυτή τη διακόσμηση είναι να επιλεγεί η σωστή παλέτα χρωμάτων. Για αυτό χρειάζονται τρεις διαφορετικές αποχρώσεις του ίδιου χρώματος:

- Ένα χρώμα πιο ανοιχτό για τη σκιά φωτός
- Ένα για το βασικό φόντο και το χρώμα του html στοιχείου
- Ένα για τη σκοτεινή σκιά

Για να καταλάβει κανείς τι συμβαίνει όταν πατήσει ο χρήστης το κουμπί πρέπει να αναλυθεί ο κώδικας στο αρχείο **dash-search.js**

```
document.getElementById('subBtn').addEventListener("click", function (e) {

    // Take values from inputs in Userpage
    min = document.getElementById('min').value;
    max = document.getElementById('max').value;
    console.log(symbolStock);
    if (symbolStock == undefined) {

        symbolStock = followed_stocks[0];
    }
    console.log(symbolStock);
    if (min == '') {
        min = null;
    }
    if (max == '') {
        max = null;
    }
    // Send symbol min max to database
    $.ajax({
        url: '/getValue/' + symbolStock + '/' + min + '/' + max + '',
        datatype: "json",
        type: 'post',
        cache: false,
        timeout: 1000000,
        success: function (result) {
            console.log(result);
            if (result.status == 200) {
                console.log("Status:200");
            }
        },
        error: function (result) {
            console.log(result);
        }
    });
});
```

Κώδικας 69: Αποστολή τιμών που εισήγαγε ο χρήστης στη σελίδα userpage πίσω στο server

Όπως φαίνεται στον κώδικα παραπάνω όταν πατηθεί το κουμπί, οι τιμές από τα inputs αποθηκεύονται στις αντίστοιχες μεταβλητές. Στη συνέχεια με τη χρήση της Ajax στέλνεται ένα αίτημα POST στον server.

```
// Take user's id min max and stocks symbol from userpage
router.post('/getValue/:symbol/:min/:max', (req, res) => {

  //Define them and create new Schema in database
  const user = req.user._id;
  const symbol = req.params.symbol;
  const mins = req.params.min;
  const maxs = req.params.max;
  const { id, stockName, min, max } = { id: user, stockName: symbol, min: mins, max: maxs };

  // Find if user has already add values for min max for the stock
  Stock.findOne({ id: user, stockName: symbol }).then(result => {
```

Κώδικα 70: Διαχείριση ανώτερου και κατώτερου ορίου μετοχής που εισήγαγε ο χρήστης, από τον server

Ο server δέχεται τις τιμές ή τη τιμή που εισήγαγε ο χρήστης καθώς και το σύμβολο της μετοχής για τον οποίο θέτει τα όρια, αλλά και το id του χρήστη. Μετά με τη μέθοδο `.findOne()` ψάχνει να βρει αν ο χρήστης έχει ήδη θέσει όρια για αυτή τη μετοχή.

```
if (result) {

  if (mins !== 'null') {

    // Update min value
    Stock.findOneAndUpdate({ id: user, stockName: symbol }, { '$set': { min: mins } }, (err, doc) => {

      if (err) {

        console.log("Something wrong when updating data!");
      }

      console.log(doc);
    });
  }
}
```

Κώδικα 71: Έλεγχος αν προϋπάρχουν όρια για τη μετοχή στη βάση δεδομένων

Αφού γίνει ο έλεγχος, αν βρει ότι υπάρχουν, ελέγχει ξεχωριστά ποιο από τα όρια υπάρχει ήδη στη βάση δεδομένων (μπορεί να υπάρχουν και τα δυο) και όταν τα βρει με τη μέθοδο `.findOneAndUpdate()` διαγράφει την παλιά τιμή και εισάγει την καινούργια. Όταν ολοκληρωθεί επιστρέφει κατάλληλο μήνυμα ότι η διαδικασία ολοκληρώθηκε επιτυχώς, το οποίο δέχεται η `success` συνάρτηση της Ajax. Στην περίπτωση που δεν υπάρχει κανένα όριο για τη μετοχή, τότε ο server δημιουργεί ένα ξεχωριστό έγγραφο στη βάση δεδομένων το οποίο περιλαμβάνει το id του χρήστη που έθεσε αυτά τα όρια, το όνομα της μετοχής και το ανώτερο, κατώτερο όριο της τιμής της μετοχής.

```
// If user hasn't set min max for the stock create new at database
else {

    const newStock = new Stock({

        id,
        stockName,
        min,
        max,

    });

    newStock.save();
}
```

Κώδικα 72: Δημιουργία ανώτερου και κατώτερου ορίου μετοχής στη βάση δεδομένων

Στη συνέχεια πρέπει να βρεθεί ένας τρόπος με τον οποίο να γίνεται συνεχής έλεγχος της τιμής της μετοχής και αν ξεπερνά ένα από αυτά τα όρια να ειδοποιεί τον χρήστη. Για να επιτευχθεί αυτό θα χρησιμοποιηθεί η μέθοδος `setInterval()`, όπου όποιος κώδικας υπάρχει μέσα σε αυτή εκτελείται κάθε φορά μετά από ένα χρονικό διάστημα.

```
// Loop running every 5 minutes and call Apis
setInterval(function () {
    You, 3 months ago • Add button to remind stock values
    var StartDate = new Date();
    var StartDay = StartDate.getDay();
    var StartHour = StartDate.getHours();
    var StartMin = StartDate.getMinutes();

    if (StartDay !== 6 || StartDay !== 0) {
```

Κώδικα 73: `SetInterval()` μέθοδος για την εκτέλεση του κώδικα κάθε 5 λεπτά

Μέσα στη μέθοδο ελέγχεται ποια μέρα της εβδομάδας είναι, γιατί αν είναι Σάββατο ή Κυριακή το χρηματιστήριο είναι κλειστό, με αποτέλεσμα η τιμή της μετοχής να μην αλλάζει άρα να μην χρειάζεται να γίνει κάποιος έλεγχος.

```

if (StartHour >= 16 && StartHour <= 23) {

  if (((StartHour == 16 && StartMin >= 35) || (StartHour >= 17 && StartHour <= 22) || (StartHour == 23 && StartMin == 0)))

    Stock.findOneAndDelete({ max: { '$exists': false }, min: { '$exists': false } }).then(result => { });

  // Find in DB the reminders for stock price and put it to an array
  Stock.find({}).then(result => {

    result = JSON.parse(JSON.stringify(result));
    const name = 'stockName';
    const time = "Time Series (5min)";
    var length;
  });
}

```

Κώδικας 74: Έλεγχος αν η τωρινή τιμή της μετοχής ξεπερνάει κάποιο όριο

Στη συνέχεια ελέγχεται η ώρα της ημέρας γιατί το χρηματιστήριο ανοίγει 16:35 ώρα Ελλάδος και κλείνει 23:00, οπότε ζητείται μόνο αυτές τις ώρες να γίνεται ο έλεγχος καθώς τις υπόλοιπες ώρες της ημέρας είναι κλειστό το χρηματιστήριο. Αρχικά ελέγχεται η βάση δεδομένων για να φανεί ποια min και max δεν έχουν τιμή (το γιατί δεν έχουν τιμή θα αναλυθεί παρακάτω) και διαγράφονται. Στη συνέχεια βρίσκονται όλες οι μετοχές που έχουν ακολουθήσει οι χρήστες και περνάν σε ένα πίνακα.

```

// Async for loop which wait the promises to continue iteration
(async () => {

  // For every reminder in db
  for (let i = 0; i < result.length; i++) {

    // Call Api
    await HTTPget(i);
  }
});

```

Κώδικας 75: Επανάληψη που καλεί τη συνάρτηση HTTP get για κάθε στοιχείο του πίνακα

Αφού γεμίσει ο πίνακας με τις μετοχές, εκτελείται η ασύγχρονη ανώνυμη συνάρτηση που φαίνεται παραπάνω στο κώδικα. Η συγκεκριμένη συνάρτηση θα εκτελεστεί μόνη της, και με τη χρήση μιας επανάληψης θα καλέσει για κάθε μια μετοχή του πίνακα τη συνάρτηση HTTPget(), την οποία θα περιμένει να ολοκληρωθεί για να συνεχιστεί η επανάληψη.

```

// Function to make HTTP request on server to call Api
function HTTPget(i) {

    // Return promise with the data from api call
    return new Promise(function (resolve) {

        var options = {
            // You, 2 months ago • Add realt time notifications fixed UI

            port: process.env.PORT || 1200, // the port
            path: '/getAps/' + result[i][name] + '',
            method: 'get',
        };

        http.request(options, function (res) {

            res.setEncoding('utf8'); // Set encode to data

            resolve(res.on('data', function (chunk) {

                number = 1;
                var temp = JSON.parse(chunk.toString()); // Convert chunk to string and then to json
                const json_length = Object.keys(temp[time]).length; // Length of json
                length = json_length - 1;

                for (var date in temp["Time Series (5min)"]) { // Fill array with data
                    close_array_Real.push(temp[time][date]["4. close"]);
                };
            }));
        }).end();
    });
}

```

Κώδικας 76: Συνάρτηση HTTP get()

Η συνάρτηση με τη σειρά της επιστρέφει μια υπόσχεση, την οποία περιμένει η επανάληψη που αναλύθηκε πιο πάνω για να μπορέσει να συνεχίσει. Μέσα σε αυτή την υπόσχεση καλείται μια συνάρτηση στέλνοντας ένα http αίτημα μέσα από το οποίο υπολογίζονται οι τιμές της μετοχής και περνούν σε ένα πίνακα μόνο οι τιμές close της συγκεκριμένης μετοχής. Αφού ολοκληρωθεί και σταλθεί η υπόσχεση επιστρέφεται στην ανώνυμη συνάρτηση και συγκεκριμένα μέσα στη επανάληψη for η οποία τώρα μπορεί να συνεχιστεί.

```

// If reminder is true send email and delete max from db
if (close_array_Real[length] >= result[i]['max']) {

    const text = result[i]['stockName'] + " value got higher than " + result[i]['max'];

    await findEmail(i, text);

    //Delete max
    Stock.findOneAndUpdate({ stockName: result[i]['stockName'], id: result[i]['id'] }, { '$unset': { max: "" } }, (err, doc) => {

        if (err) {

            console.log("Something wrong when updating data!");

        }

        console.log(doc);

    });

}

```

Κώδικας 77: Αν η τωρινή τιμή της μετοχής είναι μεγαλύτερη από το ανώτερο όριο καλείται η συνάρτηση `findEmail()`

Έχοντας την τιμή `close` της μετοχής για τη συγκεκριμένη μέρα και ώρα, ελέγχεται αν είναι μεγαλύτερη ή ίση από το ανώτερο όριο που έχει ορίσει ο χρήστης. Αν αυτό ισχύει, περνάει στη μεταβλητή `text` το κατάλληλο μήνυμα που θα σταλθεί στο χρήστη και καλείται η συνάρτηση `.findEmail()` η οποία έχει δηλωθεί σαν `await`, οπότε πρέπει να ολοκληρωθεί για να συνεχίσει στον παρακάτω κώδικα. Στη συνέχεια θα αναλυθεί η συνάρτηση `.findEmail()`

```

// Find where to send the email      You, 2 months ago • Add realt time notifications fixed UI
function findEmail(i, text) {

    return new Promise(function (resolve) {

        //Find the User
        User.find({ _id: result[i]['id'] }).then(user => {

            User.findOneAndUpdate({ _id: result[i]['id'] }, { '$push': { notification: text } }, (err, doc) => {

                if (err) {

                    console.log("Something wrong when updating data!");

                }

                console.log(doc);

            })

            User.findOneAndUpdate({ _id: result[i]['id'] }, { $set: { "dot": 1 } }, (err, doc) => {

                if (err) {

                    console.log("Something wrong when updating data!");

                }

                console.log(doc);

            })

        })

    })

}

```

Κώδικας 78: Συνάρτηση `findEmail()`

Η συνάρτηση δέχεται σαν ορίσματα τη μεταβλητή text και τον αριθμό που έχει η μετοχή μέσα στο πίνακα με όλες τις μετοχές. Αυτή με τη σειρά της επιστρέφει μια υπόσχεση. Μέσα σε αυτή εκτελείται μια συνάρτηση στην οποία με τη μέθοδο .find() ψάχνει να βρει τον χρήστη με βάση το id της μετοχής και στη συνέχεια πηγαίνει σε αυτόν τον χρήστη και εισάγει στον πίνακα του τις ειδοποιήσεις του, δηλαδή το μήνυμα που περάστηκε στη μεταβλητή text. Μετά αλλάζει η τιμή του dot σε 1. Το dot είναι η κουκίδα που θα εμφανίζεται, στο όνομα του χρήστη πάνω στη μπάρα πλοήγησης, όταν έχει κάποια ειδοποίηση. Όταν το dot είναι 1 ενεργοποιείται η κουκίδα ενώ όταν είναι 0 απενεργοποιείται.

```
user = JSON.parse(JSON.stringify(user));
userEmail = user[0]['email'];
console.log(userEmail);

var mailOptions = {
  from: 'nickzte@gmail.com',
  to: userEmail,
  subject: 'Sending Email using Node.js',
  text: text
};
button to remind stock values
// Send email
resolve(sendEmail(mailOptions));
```

Κώδικας 79: Ρυθμίσεις για την αποστολή email στον χρήστη

Τέλος περνάει το email του χρήστη στο αντικείμενο options όπου δηλώνεται το email του χρήστη που θα ειδοποιηθεί για τις αλλαγές που έγιναν στη τιμή της μετοχής που τον ενδιαφέρει. Το αντικείμενο object περνάει σαν όρισμα στη συνάρτηση sendEmail() η οποία καλείται στη συνέχεια.

```
// Send email to the User
function sendEmail(mailOptions) {

  return new Promise(function (resolve) {

    var transporter = nodemailer.createTransport({

      service: 'gmail',
      auth: {

        user: 'nstocksss@gmail.com',
        pass: Emailpass
      }
    });

    resolve(transporter.sendMail(mailOptions, function (error, info) {

      if (error) {

        console.log(error);
      } else {

        console.log('Email sent: ' + info.response);
      }
    }));
  });
};
```

Κώδικας 80: Αποστολή email στο χρήστη

Στη συνάρτηση sendEmail() χρησιμοποιείται το module nodemailer με το οποίο στέλνεται email στον χρήστη. Στη συνέχεια θα αναλυθεί η ανώνυμη συνάρτηση μέσα στον έλεγχο που γίνεται για το αν η τιμή της μετοχής είναι μεγαλύτερη από το max.

```
//Delete max
Stock.findOneAndUpdate({ stockName: result[i]['stockName'], id: result[i]['id'] }, { '$unset': { max: "" } }, (err, doc) => {

  if (err) {

    console.log("Something wrong when updating data!");
  }

  console.log(doc);
});
```

Κώδικας 81: Διαγραφή min ή max από τη βάση δεδομένων αφού σταλθεί ενημέρωση στο χρήστη

Μόλις σταλθεί το email στον χρήστη, διαγράφεται η τιμή max που εισήγαγε. Με τον ίδιο ακριβώς τρόπο κάνει και τον έλεγχο για το αν η τιμή της μετοχής είναι μικρότερη από το κατώτερο όριο που έχει ορίσει ο χρήστης. Αν ισχύει ειδοποιεί τον χρήστη και διαγράφει το min από τη βάση δεδομένων.

```
], the_interval);
```

You, 3 months ago • Add button to remind stock values

Κώδικα 82: Μεταβλητή που περνάει το χρόνο που θα εκτελείται η μέθοδος interval()

Στο τέλος της μεθόδου interval περνάει μια μεταβλητή η οποία ορίζει κάθε πότε θα εκτελείται η συνάρτηση μέσα σε αυτή. Στη συγκεκριμένη περίπτωση θα εκτελείται κάθε 5 λεπτά και θα ελέγχει αν κάποια μετοχή πέρασε τα όρια που της έθεσε ο χρήστης.

Α.1.3.11 Κατασκευή μπάρας πλοήγησης

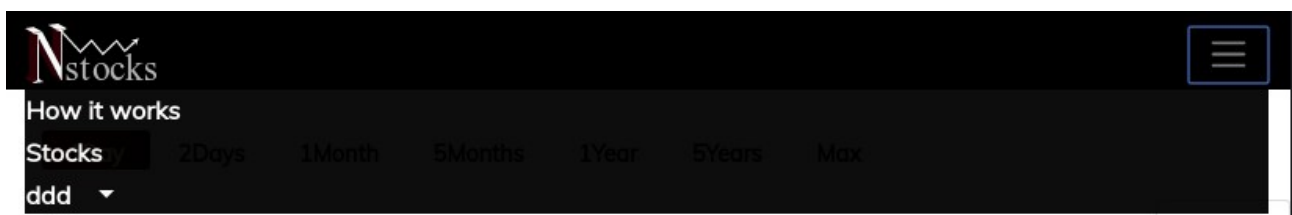
Η πλοήγηση σε μια ιστοσελίδα χαρακτηρίζεται από μια συλλογή από τους συνδέσμους που σχηματίζουν την μπάρα πλοήγησης. Όπως φαίνεται και στους κώδικες από τις διάφορες σελίδες της διαδικτυακής εφαρμογής σε όλες περιλαμβάνεται το αρχείο navigation. Δεν δημιουργήθηκε για κάθε σελίδα ξεχωριστά μια μπάρα πλοήγησης γιατί πολύ απλά πρέπει σε όλες τις σελίδες να είναι ίδια. Είναι το βασικό στοιχείο που υπάρχει παντού, για το λόγο αυτό τοποθετήθηκε το logo της διαδικτυακής εφαρμογής πάνω σε αυτό καθώς και χρησιμοποιήθηκε ένα ουδέτερο χρώμα το οποίο θα μπορεί να ταιριάξει σε οποιαδήποτε σελίδα της εφαρμογής. Επίσης προστέθηκαν οι σύνδεσμοι από τις υπόλοιπες σελίδες της εφαρμογής για να βοηθήσει τον χρήστη να μπορεί να περιηγείται πιο άνετα σε αυτή.



Εικόνα 49: Μπάρα πλοήγησης

Όπως φαίνεται στη φωτογραφία η μπάρα πλοήγησης αποτελείται από δυο containers: ένα για το logo που μπορεί ο χρήστης να το πατήσει και να γυρίσει στην αρχική σελίδα της διαδικτυακής εφαρμογής και ένα για τους υπόλοιπους συνδέσμους.

Εκτός από τα δυο containers υπάρχει και ένα κουμπί το οποίο ενεργοποιείται όταν γίνει κατάρρευση της μπάρας πλοήγησης, δηλαδή όταν το πλάτος του container της μπάρας πλοήγησης γίνει μικρότερο από ένα συγκεκριμένο αριθμό pixels. Το οποίο αντικαθιστά τους αρχικούς συνδέσμους και δημιουργεί ένα dropdown menu.



Εικόνα 50: Dropdown-menu

```
<!-- Add button to nav bar when collapse -->
<button class="navbar-dark navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavDropdown"
  aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">

  <span class="navbar-toggler-icon"></span>

</button>
```

Κώδικας 83: Dropdown-menu

Όπως φαίνεται και στον κώδικα, όταν κάποιος πατήσει το toggler-icon με τη χρήση του data-target ορίζεται ποιο html στοιχείο θα ενεργοποιηθεί.

Όταν ο χρήστης συνδέεται στη διαδικτυακή εφαρμογή τότε το κουμπί login αλλάζει στο όνομα του χρήστη που συνδέθηκε και εμφανίζεται ένα dropdown-menu κάτω από αυτό που περιέχει ένα κουμπί για να μπορέσει ο χρήστης να περιηγηθεί στο userpage και ένα για να αποσυνδεθεί.



Εικόνα 51: Dropdown-menu με τις επιπλέον επιλογές του συνδεδεμένου χρήστη

Αυτό το επιτεύχθηκε με τη χρήση της `ejs`. Όπως θα παρουσιαστεί και στο κώδικα παρακάτω, χρησιμοποιούνται έλεγχοι για το αν είναι εγγεγραμμένος ο χρήστης ή όχι και ανάλογα του εμφανίζονται οι κατάλληλοι σύνδεσμοι στη μπάρα πλοήγησης.

```
<%if (user) { %> You, 2 months ago • Add realt time notifications fixed
```

Κώδικα 84: Έλεγχος αν είναι συνδεδεμένος χρήστης με τη βοήθεια της `ejs`

Όπως φαίνεται στον κώδικα αν υπάρχει `user` τότε εμφανίζονται διαφορετικά `html` στοιχεία στη σελίδα. Ο `user` περνάει στις σελίδες της εφαρμογής από το `server`. Στη συνέχεια θα παρουσιαστεί ένα παράδειγμα για το πώς περνάει στη σελίδα `stocks`.

```
// Stock Page
router.get('/stocks', (req, res) => {

  // If User connected send user data to client side
  if (passport.authenticate) {

    res.render('Stocks', {

      // Send Stock name from server to client
      symbol: symbol.symbol,
      user: req.user,
      error: true
    })
  }

  else {

    res.render('Stocks', {

      error: false,
      symbol: symbol.symbol
    })
  }
});
```

Κώδικα 85: Αποστολή στοιχείων `user` στη σελίδα `Stocks`

Αν επιστρέφει true το passport.authenticate σημαίνει ότι ο χρήστης υπάρχει στη βάση δεδομένων άρα στέλνεται στη σελίδα stocks το user το οποίο περιλαμβάνει τα στοιχεία του εγγεγραμμένου χρήστη. Με αποτέλεσμα πλέον να υπάρχει πλήρης πρόσβαση σε αυτά από το αρχείο ejs.

```
<li class="nav-item dropdown">
  <!-- Button If logged in show user names -->
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" data-toggle="dropdown"
    aria-haspopup="true" aria-expanded="false" style="font-family: monospace;font-weight:600;">
    <%=user.name %>
  </a>
  <ul class="dropdown-menu">
    <li><a href="#">Profile</a></li>
    <li><a href="/logout">Log out</a></li>
    <li><div class="dropdown-menu-notification">
      <%= if(user.notification[1]) { %>
      <div class="notification">
        <div class="notification-header">
          <div class="notification-close"><button class="close"></button></div>
          <div class="notification-text"><p><%=user.notification[i] %></p></div>
        </div>
      </div>
      <%= } %>
      <div class="notification">
        <div class="notification-header">
          <div class="notification-close"><button class="close"></button></div>
          <div class="notification-text"><p><%=user.notification[i] %></p></div>
        </div>
      </div>
      <%= } else { %>
      <div class="notification">
        <div class="notification-header">
          <div class="notification-close"><button class="close"></button></div>
          <div class="notification-text"><p><%=user.notification[i] %></p></div>
        </div>
      </div>
      <%= } %>
    </li>
  </ul>
</li>
```

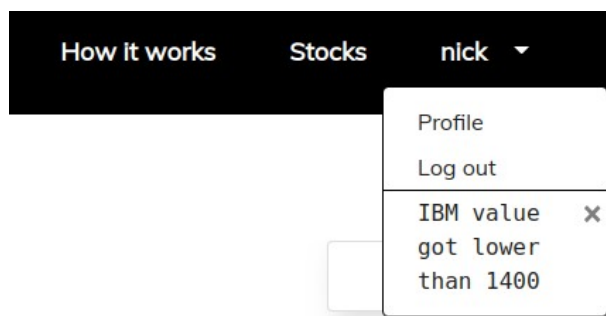
Κώδικας 86: Δημιουργία dropdown-menu χρήστη πάνω στη μπάρα πλοήγησης

Οπότε, εφόσον υπάρχει ο user, εμφανίζεται μια λίστα στην οποία το πρώτο αντικείμενο της είναι αυτό που θα φαίνεται και θα έχει το όνομα του χρήστη που πάλι περάσε μέσα στην html με τη χρήση του ejs. Με την ιδιότητα data-target δίνεται η δυνατότητα στον χρήστη, όταν πατάει πάνω σε αυτό να εμφανίζει ένα άλλο html στοιχείο.

```
<div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink" id="droDownMenu"
  style="max-width: 158px; border: solid 1px black; font-family: monospace;">
  <!-- Button to go to userpage -->
  <a class="dropdown-item" id="profileBtn" href="/users/login">Profile</a>
  <!-- Button to Log out -->
  <a class="dropdown-item" id="logoutBtn" href="/users/logout">Log out</a>
  <div class="dropdown-menu-notification">
    <%= if(user.notification[1]) { %>
    <div class="notification">
      <div class="notification-header">
        <div class="notification-close"><button class="close"></button></div>
        <div class="notification-text"><p><%=user.notification[i] %></p></div>
      </div>
    </div>
    <%= } %>
    <div class="notification">
      <div class="notification-header">
        <div class="notification-close"><button class="close"></button></div>
        <div class="notification-text"><p><%=user.notification[i] %></p></div>
      </div>
    </div>
    <%= } else { %>
    <div class="notification">
      <div class="notification-header">
        <div class="notification-close"><button class="close"></button></div>
        <div class="notification-text"><p><%=user.notification[i] %></p></div>
      </div>
    </div>
    <%= } %>
  </div>
</div>
```

Κώδικας 87: κατασκευή dropdown-menu χρήστη

Αφού πατηθεί το στοιχείο με το όνομα του χρήστη, ανοίγει το container με την κλάση dropdown-menu στην οποία βρίσκεται ο σύνδεσμος για το userpage, ο σύνδεσμος για αποσύνδεση και οι ειδοποιήσεις (αν υπάρχουν) για τυχόν σημαντικές μεταβολές των τιμών των μετοχών που έχουν οριστεί. Για να εμφανιστούν οι ειδοποιήσεις ελέγχεται πάλι με τη χρήση της ejjs αν υπάρχουν τέτοιες στα δεδομένα του χρήστη που στάλθηκαν στο server από τη βάση δεδομένων. Αφού γίνει ο έλεγχος και γίνει αντιληπτό ότι υπάρχουν, εκτελείται μια επανάληψη for και για κάθε ειδοποίηση που υπάρχει δημιουργείται ένα < a > στοιχείο το οποίο αποτελεί το parent container του στοιχείου < span > το οποίο είναι ένα κουμπί με το οποίο σβήνει ο χρήστης την ειδοποίηση και το < p > στοιχείο το οποίο θα περιέχει το κείμενο της ειδοποίησης.

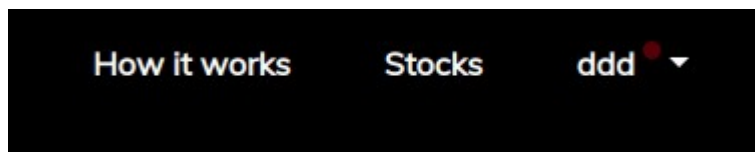


Εικόνα 52: Ειδοποίηση χρήστη για μεταβολές τιμών της μετοχής

```
<% if(user.dot==1) { %>
<span class="dot show" id="dot"></span></a>
<%}%>
<%if(user.dot==0) {%>
<span class="dot" id="dot"></span></a>
<%}%>
```

Κώδικα 88: Εισαγωγή κουκίδας ειδοποίησης

Επίσης ελέγχεται αν το dot είναι 0 ή 1 και αν είναι 1 εμφανίζεται με την κόκκινη κουκίδα δίπλα από το όνομα.



Εικόνα 53: Κουκίδα ειδοποίησης

Τώρα πρέπει να μπορεί να διαγραφεί αυτή η ειδοποίηση αλλά και η κουκίδα. Όταν ανοίγει κάποιος το dropdown-menu η κουκίδα δίπλα από το όνομα του χρήστη πρέπει να φεύγει και αυτό θα σημαίνει ότι διαβάστηκε.

```
document.getElementById("navbarDropdownMenuLink").addEventListener("click", function (e) {  
  
    console.log("yes");  
    $("#dot").removeClass("show");  
    const target = 0;  
    $.ajax({  
  
        url: '/deleteDot/' + target + '',  
        dataType: 'number',  
        type: 'post',  
        cache: false,  
        timeout: 1000000,  
        success: function (result) {  
            console.log(result);  
            if (result.status == 200) {  
                console.log("Status:200");  
            }  
        },  
        error: function (result) {  
            console.log(result);  
        }  
    });  
});
```

Κώδικα 89: Διαγραφή κουκίδας αφού διαβαστεί η ειδοποίηση από το χρήστη

Αφαιρείται το show από τη κλάση του dot και αυτομάτως αλλάζει το opacity του από 1 σε 0 και εξαφανίζεται, όμως πρέπει να διαγραφεί και από τη βάση δεδομένων. Για να γίνει αυτό θα πρέπει να γίνει ένα αίτημα POST στο server.

```

router.post('/deleteDot/:target', (req, res) => {
  const target = req.params.target;
  const user = req.user._id;

  User.findOneAndUpdate({ _id: user }, { $set: { "dot": 0 } }, (err, doc) => {
    if (err) {
      console.log("Something wrong when updating data!");
    }
    console.log(doc);
  })

  res.json({ success: "Updated Successfully", status: 200 });
});

```

Κώδικα 90: Διαγραφή κουκίδας από τη βάση δεδομένων

Ο server με τη σειρά του με τη μέθοδο `findOneAndUpdate()` θα βρει το dot του συγκεκριμένου χρήστη και θα του αλλάξει τη τιμή από 1 σε 0 και στη συνέχεια θα στείλει απάντηση στην Ajax ότι η διαδικασία ολοκληρώθηκε επιτυχώς.

Ένα πρόβλημα που δημιουργείται είναι ότι οι ειδοποιήσεις και η κόκκινη κουκίδα στο όνομα του χρήστη, με την οποία τον ενημερώνει ότι έχει κάποια ειδοποίηση, δεν εμφανίζονται σε πραγματικό χρόνο αλλά πρέπει να ανανεωθεί η σελίδα για να μπορέσουν να εμφανιστούν στη διαδικτυακή εφαρμογή. Για να λυθεί αυτό το πρόβλημα αυτό πρέπει να χρησιμοποιηθεί η τεχνολογία websocket όπου σε πραγματικό χρόνο θα περνάει τα νέα html στοιχεία στο UI. Ουσιαστικά το websocket είναι μια τεχνολογία που καθιστά δυνατό να ανοίξει μια αμφίδρομη διαδραστική συνεδρία επικοινωνίας, μεταξύ του προγράμματος περιήγησης του χρήστη και του διακομιστή. Για εφαρμοστεί αυτό θα χρησιμοποιηθεί το module `Socket.io`. Πάμε στο κώδικα τώρα στο αρχείο **App.js**.

```

const io = require('socket.io')(http);

```

Κώδικα 91: Ενσωμάτωση Socket.io

Αρχικά ενσωματώνεται το Socket.io στην εφαρμογή και το προσκολλάται στον HTTP server, έτσι ώστε να μπορεί να διακομίζει το δικό του client κώδικα μέσα από το socket.io/socket.io.js.

```
io.on('connection', function (socket) {  
  console.log('a user connected');  
  socket.on('disconnect', function () {  
    console.log('user disconnected');  
  });  
  socket.on('new-notification', function (msg) {  
    io.emit('new-notification', msg);  
  });  
});
```

Κώδικας 92: Socket.io .on() μέθοδος

Πριν αναλυθεί ο κώδικας, καλό είναι να αναφερθεί η διαφορά του io και του socket. Το io είναι ο Socket.io server και το socket είναι προσκολλημένο σε μια αναφορά του HTTP.server, ακούγοντας για εισερχόμενα γεγονότα. Από την άλλη το socket που περνά στη συνάρτηση που ακούει το γεγονός Socket του connection, είναι ένα αντικείμενο που αναπαριστά μια εισερχόμενη socket σύνδεση από τον client. Και τα δυο ακούνε τα γεγονότα με τη βοήθεια της μεθόδου .on(). Επιστρέφοντας στον κώδικα φαίνεται ότι με τη μέθοδο .on() ακούει το γεγονός connection, το οποίο είναι ένα γεγονός που περιλαμβάνει το Socket.io, από το πρόγραμμα περιήγησης του χρήστη και καταλαβαίνει κανείς ότι κάποιος συνδέθηκε. Αντίστοιχα με τη μέθοδο .on() ακούει τον περιηγητή του χρήστη και με το γεγονός disconnect, που υπάρχει ήδη μέσα στη βιβλιοθήκη του Socket.io, καταλαβαίνει ότι ο χρήστης αποσυνδέθηκε από την εφαρμογή μας. Πέρα από τα προ υπάρχοντα γεγονότα που έχει το Socket.io, μπορεί κανείς να εκπέμψει και να αποδεχτεί γεγονότα που ο προγραμματιστής έφτιαξε. Οπότε δημιουργείται το γεγονός new-notification που θα παρουσιαστεί παρακάτω και με τη μέθοδο .on() δείχνει στο socket αντικείμενο που έχει από το connection, να ακούει αυτό το γεγονός. Στη συνέχεια, με το io.emit() εκπέμπει το μήνυμα σε όλα τα sockets και εμφανίζεται στους χρήστες που το χρησιμοποιούν εκείνη τη στιγμή.

```

mongoose

    .connect(
      db,
      { useNewUrlParser: true, useUnifiedTopology: true }
    )

    .then(() => {
      console.log('MongoDB Connected')

      const notification = mongoose.model('User');

      // Create a change stream. The 'change' event gets emitted when there's a
      // change in the database
      // notification.watch({}).
      notification.watch({ 'fullDocument': 'updateLookup' }).
        on('change', function (change) {
          io.emit('new-notification', change);
          // console.log(change);
          console.log(new Date(), change);
        });

      // data => console.log(new Date(), data)
    })
    .catch(err => console.log(err));

```

Κώδικα 93: Σύνδεση socket.io με τη βάση δεδομένων

Για επιτευχθεί η αποστολή των ειδοποιήσεων σε πραγματικό χρόνο, πρέπει να ξέρει ο server πότε γίνονται αλλαγές στη βάση δεδομένων και συγκεκριμένα στις ειδοποιήσεις των χρηστών. Για τον λόγο αυτόν, στον κώδικά μας, μετά από τη σύνδεση με τη βάση δεδομένων χρησιμοποιείται η μέθοδος `watch()` με την οποία παρακολουθούνται οι αλλαγές στη βάση δεδομένων. Αυτή η μέθοδος επιστρέφει ένα έγγραφο με τις αλλαγές που έγιναν. Με τη μέθοδο `.on()` ακούει ο Socket.io server τις αλλαγές και τις στέλνει στον client.

Μέχρι στιγμής έχει κατασκευαστεί ο server ο οποίος παρακολουθεί τις αλλαγές στη βάση δεδομένων και στη συνέχεια μέσα από ένα websocket στέλνει στο client το έγγραφο με τις αλλαγές που έγιναν στη βάση δεδομένων. Τώρα θα παρουσιαστεί τι κάνει ο client για να δέχεται το έγγραφο από το server. Για το λόγο αυτό θα μελετηθεί ο κώδικας στο αρχείο **navbar.js**.

```
var socket = io();
socket.on('new-notification', async function (msg) {
```

Κώδικας 94: Αποδοχή ειδοποιήσεων από το client μέσω websocket

Όπως φαίνεται δημιουργείται μια μεταβλητή που συνδέεται με τον Socket.io server και συγκεκριμένα ακούει τις αλλαγές που γίνονται στο γεγονός new-notification. Στη συνέχεια εκτελεί μια ασύγχρονη συνάρτηση στην οποία περνάει το msg που είναι το έγγραφο που έλαβε από τον client.

```
▼ {_id: {...}, operationType: "update", clusterTime: "6822052497356488709", fullDocument: {...}, ns: {...}, ...}
  clusterTime: "6822052497356488709"
  ▼ documentKey:
    _id: "5e697aa44645437014850fba"
    ► __proto__: Object
  ▼ fullDocument:
    date: "2020-03-11T23:56:20.633Z"
    dot: 1
    email: "123@gmail.com"
    name: "nick"
  ▼ notification: Array(2)
    0: "Start"
    1: "IBM value got lower than 1600"
    length: 2
    ► __proto__: Array(0)
  password: "$2a$10$13wXaN3g0mkHIS50s0TSfeqN912vjtdEgoLBIAH35dcZZmWL0tHTG"
  ► stock: ["IBM"]
  v: 0
  _id: "5e697aa44645437014850fba"
  ► __proto__: Object
  ns: {db: "test", coll: "users"}
  operationType: "update"
  updateDescription: {updatedFields: {...}, removedFields: Array(0)}
  ► _id: {data: "825EACCC9A000000052B022C0100296E5A1004EC4D925F747D...053F55646645F696400645E697AA44645437014850FBA0004"}
  ► __proto__: Object
```

Εικόνα 54: Αρχείο json που στέλνει η βάση δεδομένων με τις αλλαγές που έγιναν

Στην παραπάνω φωτογραφία φαίνεται το αρχείο json που έστειλε ο server στον client, το οποίο περιλαμβάνει τη τιμή του dot που στη συγκεκριμένη περίπτωση είναι 1, καθώς και την ειδοποίηση που έλαβε ο χρήστης.

```

$("#dot").addClass("show");
var element = document.createElement("a");
var text = document.createElement("p");
var btn = document.createElement("button");
var span = document.createElement("span");
var Deletarget = 0;

span.className = "closeSpan";
btn.className = "close";
text.className = "notification-text";
span.innerHTML = "&times;";
text.innerHTML = msg["updateDescription"]["updatedFields"]['notification'];
element.className = "dropdown-item notification";

```

Κώδικας 95: Δημιουργία ειδοποίησης

Στη συνέχεια, με τη βοήθεια της JavaScript δημιουργούνται τα html στοιχεία τα οποία θα περιλαμβάνουν το κείμενο της ειδοποίησης καθώς και το κουμπί για να διαγράψει την ειδοποίηση ο χρήστης. Επίσης προστίθεται η κλάση show στο dot για να αλλάξει το opacity σε 1 και να φαίνεται δίπλα από το όνομα. Αφού δημιουργηθούν τα html στοιχεία με τη χρήση της .innerHTML, περνάει μέσα στο στοιχείο, το κείμενο της ειδοποίησης και το σύμβολο X που θα μπορεί να πατήσει ο χρήστης για να τη διαγράψει.

```

element.appendChild(btn);
btn.appendChild(span);
element.appendChild(text);
document.getElementById('droDownMenu').appendChild(element);

```

Κώδικας 96: Εισαγωγή html στοιχείου στο dropdown-menu του χρήστη

Στο τέλος περνάνε αυτά τα html στοιχεία που κατασκευάστηκαν μέσα στο dropdown-menu που βρίσκεται κάτω από το όνομα του χρήστη στη μπάρα πλοήγησης.

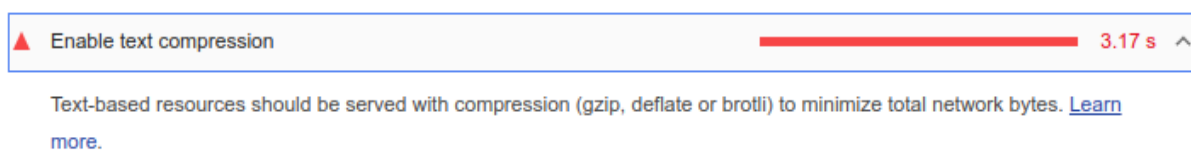
```
$.ajax({
  url: '/deleteNotif/' + Deletarget + '',
  dataType: 'number',
  type: 'post',
  cache: false,
  timeout: 1000000,
  success: function (result) {
    console.log(result);
    if (result.status == 200) {
      console.log("Status:200");
    }
  },
  error: function (result) {
    console.log(result);
  }
});
```

Κώδικας 97: Διαγραφή ειδοποίησης από το dropdown-menu

Μόλις πατηθεί το κουμπί διαγραφής της ειδοποίησης στέλνεται κατευθείαν ένα αίτημα POST με τη χρήση της Ajax στο server και αυτός με τη σειρά του διαγράφει την ειδοποίηση από τη συλλογή του συγκεκριμένου user.

Δ.1.3.8 Αλγόριθμοι συμπίεσης gzip και brotli

Χρησιμοποιώντας το Audits που βρίσκεται στο Chrome DevTools μπορεί κανείς πολύ εύκολα να τρέξει το Lighthouse το οποίο ελέγχει τη απόδοση μιας ιστοσελίδας/ διαδικτυακής εφαρμογής. Μια από τις σημαντικότερες βελτιώσεις που προτείνει να γίνει για να βελτιωθεί η απόδοση, είναι να χρησιμοποιηθούν οι αλγόριθμοι συμπίεσης όπως το brotli και το gzip.



Εικόνα 55: Audits χρήση αλγόριθμων συμπίεσης

Όπως φαίνεται και στη φωτογραφία πιο πάνω, το Lighthouse προτείνει ακόμα και τους αλγόριθμους συμπίεσης που θα μπορούσαν να τρέξουν στη διαδικτυακή εφαρμογή. Είναι ένα πολύ χρήσιμο εργαλείο που δείχνει το σκορ της εφαρμογής αλλά και λύσεις για να μπορέσει κανείς να τη βελτιώσει. Είναι ένα πολύ χρήσιμο εργαλείο που όλοι οι web developers πρέπει να το χρησιμοποιούν για καλύτερα αποτελέσματα.

Στη συνέχεια θα αναλυθεί ο κώδικας στο αρχείο **compress.js** και να παρουσιαστεί πώς γίνεται να χρησιμοποιηθούν αυτοί οι αλγόριθμοι για να βελτιωθούν οι επιδόσεις του NodeJS.

```
const brotliSettings = {  
  extension: 'br',  
  skipLarger: true,  
  mode: 0, // 0 = generic, 1 = text, 2 = font (WOFF2)  
  quality: 10, // 0 - 11,  
  lgwin: 12 // default  
};
```

Κώδικας 98: Ρυθμίσεις για brotli

Στον κώδικα φαίνεται πώς δημιουργείται το αντικείμενο με τις ρυθμίσεις για τον αλγόριθμο brotli. Δηλώνει την επέκταση που έχουν τα αρχεία μετά τη συμπίεση, με το mode:0. Ουσιαστικά δηλώνεται το περιεχόμενο των αρχείων που θα συμπιεστούν. Στη συνέχεια δηλώνονται οι φάκελοι στους οποίους υπάρχουν τα αρχεία που θα συμπιεστούν και για κάθε φάκελο τρέχει μια συνάρτηση

```
// Choose files from these folders to compress  
var dirs = ['./thesis', 'public/css', 'routes', 'views', 'models', 'config', 'public/js']  
  
// For each dir if file ends to .js, .css, .ejs, .html compress them  
dirs.forEach(dir => {
```

Κώδικας 99: Φάκελοι με τα αρχεία που θα συμπιεστούν

Στη συνέχεια χρησιμοποιώντας το file system module υπάρχει η δυνατότητα να επεξεργαστούν τα αρχεία από τον υπολογιστή. Με τη μέθοδο `.readdirSync()` διαβάζεται το περιεχόμενο του φακέλου.

Αυτή η μέθοδος επιστρέφει έναν πίνακα με όλα τα ονόματα των αρχείων ή αντικειμένων του φακέλου.

```
fs.readdirSync(dir).forEach(file => {  
  
  if (file.endsWith('.js') || file.endsWith('.css') || file.endsWith('.html') || file.endsWith('.ejs') |  
  
    // brotli  
    const result = brotli.compress(fs.readFileSync(dir + '/' + file), brotliSettings);  
    fs.writeFileSync(dir + '/' + file + '.br', result);  
  
    // gzip  
    const fileContents = fs.createReadStream(dir + '/' + file);  
    const writeStream = fs.createWriteStream(dir + '/' + file + '.gz');  
    const zip = zlib.createGzip();
```

Κώδικας 100: Δημιουργία brotli αρχείων

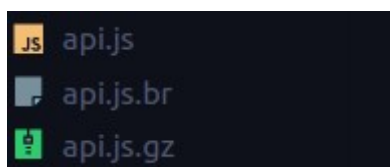
Μετά συμπιέζεται το αρχείο και με τη μέθοδο `writeFileSync()` περνάει το συμπιεσμένο αρχείο σε ένα καινούργιο αρχείο με τη κατάληξη `.br`. Τώρα θα παρουσιαστεί ο αλγόριθμος `gzip`. Χρησιμοποιείται η μέθοδος `.createReadStream()` με την οποία ανοίγει ένα `file/ Stream` για να διαβάσει τα δεδομένα μέσα σε αυτό, στη συνέχεια ανοίγει ένα `file/ Stream` για γράφει μέσα σε αρχεία με κατάληξη `.gzip` και η μέθοδος `.createGzip()` επιστρέφει ένα `gzip` αρχείο.

```
fileContents  
  
  .pipe(zip)  
  .on('error', err => console.error(err))  
  .pipe(writeStream)  
  .on('error', err => console.error(err));
```

Κώδικας 101: Ρυθμίσεις για το Gzip

Με τη μέθοδο `.pipe()` συνδέεται το stream για να διαβάσει αρχεία με το `gzip`, με αυτόν τον τρόπο συμπιέζεται το περιεχόμενο του αρχείου και στη συνέχεια πάλι με τη μέθοδο `.pipe()` περνάει τα συμπιεσμένα δεδομένα στο αρχείο που άνοιξε για να γράφει μέσα σε αυτό. Η όλη διαδικασία γίνεται για όλα τα αρχεία μέσα στους φακέλους της διαδικτυακής εφαρμογής, τα οποία έχουν

κατάληξη .ejs, .css, .js, .svg, .csv και για κάθε αρχείο δημιουργούνται δυο νέα αρχεία ένα με κατάληξη .gz και ένα με κατάληξη .br.



Εικόνα 56: Αρχεία brotli και gzip

Κεφάλαιο Ε΄

Ε.1 Στιγμιότυπα Διαδικτυακής Εφαρμογής

Ε.2 Μελλοντικές Βελτιώσεις

Η συγκεκριμένη πτυχιακή αν και είναι λειτουργική σίγουρα έχει περιθώρια για αρκετές βελτιώσεις. Ένα από τα κυριότερα προβλήματα που υπάρχουν είναι η εξωτερική API που χρησιμοποιήθηκε για να παίρνει τα δεδομένα των μετοχών. Η συγκεκριμένη API με περιόρισε πολύ ως προς τον αριθμό των αιτημάτων που μπορούσα να στείλω, αλλά και ως προς τις πληροφορίες που μπορούσε να μου προσφέρει για τις μετοχές. Με μια διαφορετική API θα μπορούσα να έχω περισσότερες πληροφορίες για τη μετοχή, με αποτέλεσμα να μπορούσα να προσθέσω το book to ratio, αλλά και

ειδήσεις (νέα) για την κάθε μετοχή, έτσι ώστε να μπορώ να δώσω μια πιο ολοκληρωμένη εικόνα της μετοχής στον επενδυτή που χρησιμοποιεί στη διαδικτυακή εφαρμογή μου.

Μια άλλη βελτίωση που θα μπορούσα να κάνω είναι να χρησιμοποιήσω WebSocket για να εμφανίζονται τα δεδομένα της μετοχής στη γραφική παράσταση της τωρινής μέρας, σε πραγματικό χρόνο χωρίς να χρειάζεται ο χρήστης να κάνει ανανέωση της σελίδας. Επίσης θα μπορούσα να χρησιμοποιήσω WebSockets για να ανανεώνονται τα html στοιχεία που υπάρχουν στις σελίδες και πρέπει να αλλάζουν κάθε φορά που αναζητά ο χρήστης μια νέα μετοχή. Ο τρόπος που χρησιμοποιώ τώρα, κάνει τη δουλειά αλλά δεν σωστός.

Μια μελλοντική βελτίωση που αφορά τη διακόσμηση της εφαρμογής είναι να αφαιρέσω τελείως το Bootstrap και να χρησιμοποιήσω μόνο CSS. Κάποια στοιχεία όπως κουμπιά, μενού, μπάρες πλοήγησης και άλλα, που έχουν φτιαχτεί με τη χρήση του Bootstrap ήταν δύσκολο να τα επεξεργαστώ και να αλλάξω την εμφάνιση τους.

Τέλος, μια μελλοντική βελτίωση που θα με βοηθούσε, θα ήταν να δημιουργήσω μόνος μου τις γραφικές παραστάσεις με τη βοήθεια του p5.js γιατί με αυτό το τρόπο θα μπορούσα να τις κατασκευάσω όπως ακριβώς ήθελα. Στη συγκεκριμένη περίπτωση χρησιμοποίησα τη βιβλιοθήκη chart.js για να κατασκευάσω τις γραφικές παραστάσεις, όπου στην αρχή μου ήταν πολύ εύκολο με τη βοήθεια αυτής της βιβλιοθήκης να οπτικοποιήσω τα δεδομένα, αλλά στη πορεία που ήθελα να κάνω αλλαγές και να προσθέσω περισσότερα χαρακτηριστικά ήταν αρκετά δύσκολο.

E.2 Συμπεράσματα

Στα πλαίσια αυτής της πτυχιακής αναπτύχθηκε μια διαδικτυακή εφαρμογή. Τα εργαλεία που χρησιμοποιήθηκαν κάποιες φορές με βοήθησαν και άλλες με περιόρισαν. Είναι πολύ σημαντικό να γνωρίζει κανείς καλά τις τεχνολογίες που χρησιμοποιεί για να μπορεί να κατανοήσει αν είναι κατάλληλες γι αυτό που θέλει να δημιουργήσει. Επίσης η σωστή οργάνωση και ο καλός και συνεχής σχολιασμός του κώδικα διευκολύνει κατά πολύ την ανάπτυξη μια εφαρμογής, η οποία

καθώς μεγαλώνει γίνεται όλο και πιο περίπλοκη. Παρόλο αυτά, με τη γνώση των βασικών αρχών και των τρόπων ανάπτυξης, μπορεί κανείς να δημιουργήσει οποιαδήποτε διαδικτυακή εφαρμογή θέλει.

Βιβλιογραφία – Δικτυογραφία

1. Stock market definition. Διαθέσιμο από: <https://www.investopedia.com/terms/s/stockmarket.asp>
2. MDN Web Docs. What is JavaScript? Διαθέσιμο από: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
3. NodeJS at PayPal. Διαθέσιμο από: <https://engineerbabu.com/blog/node-js-the-rising-technology-behind-paypal/>
4. Stack Overflow. Developer Survey Results 2019. Διαθέσιμο από: <https://insights.stackoverflow.com/survey/2019#technology>
5. ECMA International. ECMAScript 2018 Language Specification. Διαθέσιμο από: <https://www.ecma-international.org/ecma-262/9.0/index.html>
6. Express: Nodejs framework. Διαθέσιμο από: <https://expressjs.com/>
7. Web development with Node & Express. Ethan Brown. Ιούλιος 2014. Διαθέσιμο από: https://www.vanmееgern.de/fileadmin/user_upload/PDF/Web_Development_with_Node_Express.pdf
8. Express: production best practices: performance and reliability. Διαθέσιμο από: <http://expressjs.com/en/advanced/best-practice-performance.html>
9. Passport: Middleware for Node. Διαθέσιμο από: <http://www.passportjs.org/docs/downloads/html/>
10. Socket.io CookBook. Tyson Cadenhead. Οκτώβριος 2015. Διαθέσιμο από: <https://github.com/NedyUdombat/Free-Web-Development-eBooks/blob/master/book/Socket%20io/Socket.IO%20Cookbook.pdf>
11. HTML. Διαθέσιμο από: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>
12. CSS. Διαθέσιμο από: <https://developer.mozilla.org/en-US/docs/Web/CSS>
13. Single Page Applications vs Multiple Page Applications. Διαθέσιμο από: <https://medium.com/@goldybenedict/single-page-applications-vs-multiple-page-applications-do-you-really-need-an-spa-cf60825232a3>

14. Web Application Architecture: How the web Works. Διαθέσιμο από:
<https://www.altexsoft.com/blog/engineering/web-application-architecture-how-the-web-works/>
15. What is Web Application Architecture? Components, Models and Types. Διαθέσιμο από:
<https://hackr.io/blog/web-application-architecture-definition-models-types-and-more>
16. About packages and modules. Διαθέσιμο από: <https://docs.npmjs.com/about-packages-and-modules>
17. Github. Διαθέσιμο από: <https://github.com/about>
18. Mongoose for application development. Simon Holmes. 2013. Διαθέσιμο από:
<https://www.programmer-books.com/mongoose-for-application-development/>
19. MongoDB. Διαθέσιμο από: <https://www.mongodb.com/>
20. Git. Διαθέσιμο από: <https://git-scm.com/>
21. Heroku. Διαθέσιμο από: <https://www.heroku.com/>
22. Flexbox. Διαθέσιμο από: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox
23. Neumorphism. Διαθέσιμο από: <https://css-tricks.com/neumorphism-and-css/>

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

Εικόνα 1: Stack Overflow Developer Survey 2019.....	7
Εικόνα 2: Παράδειγμα αρχείου HTML.....	25
Εικόνα 3: Αρχιτεκτονική Εφαρμογής.....	30
Εικόνα 4: Αρχιτεκτονική διαδικτυακής Εφαρμογής με Express.....	33
Εικόνα 5: Δομή διαδικτυακής εφαρμογής Nstocks.....	34
Εικόνα 6: Ενημέρωση apt πακέτου.....	37
Εικόνα 7: Εγκατάσταση git.....	37
Εικόνα 8: Έλεγχος έκδοσης git.....	37
Εικόνα 9: Έκδοση git που χρησιμοποιήθηκε στη παρούσα πτυχιακή.....	38
Εικόνα 10: Εγκατάσταση npm και NodeJS.....	39
Εικόνα 11: Έλεγχος της έκδοσης NodeJS που εγκαταστήθηκε.....	39
Εικόνα 12: Έκδοση NodeJS που είναι εγκατεστημένη στο λειτουργικό σύστημα που εκπονείται η παρούσα πτυχιακή.....	39
Εικόνα 13: Αποθετήριο διαδικτυακής εφαρμογής Nstocks στο GitHub.....	41
Εικόνα 14: Δημιουργία αποθετηρίου στο GitHub.....	42

Εικόνα 15: Εντολή git init.....	42
Εικόνα 16: Αντιγραφή αποθετηρίου τοπικά στον υπολογιστή.....	43
Εικόνα 17: Εντολές προσθήκης αλλαγών στο κώδικα του αποθετηρίου μας.....	43
Εικόνα 18: Εντολή προσθήκης αλλαγών στο αποθετήριο μας.....	43
Εικόνα 19: Δημιουργία εφαρμογής στο Heroku.....	45
Εικόνα 20: Εγκατάσταση Heroku τοπικά στον υπολογιστή.....	45
Εικόνα 21: Εντολή ελέγχου έκδοσης Heroku που είναι εγκατεστημένη τοπικά στον υπολογιστή μας.....	45
Εικόνα 22: Εντολή σύνδεσης του υπολογιστή με τον λογαριασμό στο Heroku.....	46
Εικόνα 23: Εντολή για να καταλάβει σε πιο διακομιστή του Heroku απευθύνεται ο προγραμματιστής.....	46
Εικόνα 24: Εντολές για να ανεβούν οι αλλαγές στο Heroku.....	46
Εικόνα 25: Αρχική σελίδα της βάσης δεδομένων MongoDB.....	47
Εικόνα 26: Δημιουργία νέου πρότζεκτ στη βάση δεδομένων MongoDB.....	48
Εικόνα 27: Δημιουργία cluster στη Βάση Δεδομένων MongoDB.....	48
Εικόνα 28: Σύνδεση στο cluster που δημιουργήθηκε στο MongoDB.....	49
Εικόνα 29: Ρυθμίσεις σύνδεσης στο cluster.....	50
Εικόνα 30: Σύνδεση στο cluster.....	51
Εικόνα 31: Σύνδεσμος σύνδεσης εφαρμογής με τη βάση που δημιουργήσαμε.....	52
Εικόνα 32: Παράδειγμα προσθήκης node_module στο NodeJS.....	53
Εικόνα 33: Εντολή δημιουργίας package.json.....	53
Εικόνα 34: Αρχείο package.json της διαδικτυακής εφαρμογής.....	54
Εικόνα 35: Dependencies από το αρχείο package.json της διαδικτυακής εφαρμογής.....	55
Εικόνα 36: devDependencies από το αρχείο package.json της διαδικτυακής εφαρμογής.....	56
Εικόνα 37: Εντολή εγκατάστασης πακέτου με τη χρήση του npm ή yarn.....	57
Εικόνα 38: Εντολή εγκατάστασης αναπτυξιακών εξαρτήσεων με τη βοήθεια του npm ή yarn.....	57
Εικόνα 39: Φόρμα εισαγωγής στοιχείων χρήστη.....	65
Εικόνα 40: Φόρμα εισαγωγής στοιχείων χρήστη στη σελίδα login.....	70
Εικόνα 41: Αρχείο json που στέλνεται στο server από το Alpha Vantage.....	76
Εικόνα 42: Γραφική παράσταση μιας μέρας.....	86
Εικόνα 43: Containers στη σελίδα Stocks.....	88
Εικόνα 44: Child containers στη σελίδα Stocks.....	90
Εικόνα 45: Κουμπί αποθήκευσης μετοχής στη σελίδα Stocks.....	99
Εικόνα 46: Containers σελίδας userpage.....	102
Εικόνα 47: Τα τρία βασικά containers της σελίδας Userpage.....	104
Εικόνα 48: Αντικείμενο λίστας μετοχών στη σελίδα Userpage.....	107
Εικόνα 49: Μπάρα πλοήγησης.....	121
Εικόνα 50: Dropdown-menu.....	122
Εικόνα 51: Dropdown-menu με τις επιπλέον επιλογές του συνδεδεμένου χρήστη.....	123
Εικόνα 52: Ειδοποίηση χρήστη για μεταβολές τιμών της μετοχής.....	126
Εικόνα 53: Κουκίδα ειδοποίησης.....	126
Εικόνα 54: Αρχείο json που στέλνει η βάση δεδομένων με τις αλλαγές που έγιναν.....	131
Εικόνα 55: Audits χρήση αλγόριθμων συμπίεσης.....	133
Εικόνα 56: Αρχεία brotli και gzip.....	136

ΕΥΡΕΤΗΡΙΟ ΤΜΗΜΑΤΩΝ ΚΩΔΙΚΑ

Κώδικα 1: Εισαγωγή node_modules στο αρχείο app.js.....	58
Κώδικα 2: Export μοντέλο Stock από το αρχείο Stock.js.....	58

Κώδικα 3: Δείχνει σε ποια port να "ακούει" ο server στο app.js.....	59
Κώδικα 4: Δημιουργία διαδρομών (routes) στο app.js.....	59
Κώδικα 5: Σύνδεση με βάση δεδομένων στο αρχείο app.js.....	60
Κώδικα 6: Ενσωμάτωση του keys.js στο app.js.....	60
Κώδικα 7: Λαμβάνει το αίτημα για να πλοηγηθεί στην αρχική σελίδα.....	61
Κώδικα 8: Δημιουργία μοντέλου User.....	63
Κώδικα 9: Φόρμα για εγγραφή του χρήστη στη διαδικτυακή εφαρμογή.....	64
Κώδικα 10: Δρομολόγηση χρήστη στη σελίδα register της εφαρμογής.....	65
Κώδικα 11: Διαχείριση στοιχείων που εισάγει ο χρήστης στη φόρμα register.....	66
Κώδικα 12: Χρήση middleware express.urlencoded() στην εφαρμογή.....	66
Κώδικα 13: Έλεγχος κωδικού που εισήγαγε ο χρήστης στην φόρμα στη σελίδα register.....	66
Κώδικα 14: Έλεγχος αν ο χρήστης που πάει να φτιάξει λογαριασμό υπάρχει ήδη στη βάση δεδομένων.....	67
Κώδικα 15: Κρυπτογράφηση κωδικού με τη χρήση του Bcrypt και αποθήκευση χρήστη στη βάση δεδομένων.....	68
Κώδικα 16: Φόρμα σύνδεσης χρήστη στη σελίδα login.....	70
Κώδικα 17: Διαχείριση αιτήματος του χρήστη να περιηγηθεί στη σελίδα login.....	71
Κώδικα 18: Χρήση του Passport για να τον έλεγχο αν υπάρχει ο χρήστης.....	72
Κώδικα 19: Συναρτήσεις πιστοποίησης χρήστη.....	73
Κώδικα 20: Δρομολόγηση χρήστη στη σελίδα Login.....	73
Κώδικα 21: Δρομολόγηση χρήστη στη σελίδα userpage.....	73
Κώδικα 22: Ajax call για να λάβει τα δεδομένα της μετοχής.....	74
Κώδικα 23: Ο server δέχεται την κλήση από το client για να καλέσει την api.....	75
Κώδικα 24: Εισαγωγή δεδομένων που λήφθηκαν από την api σε πίνακες.....	76
Κώδικα 25: Εντοπισμός αρχής του μήνα στο πίνακα date.....	77
Κώδικα 26: Εντοπισμός τέλους του μήνα στο πίνακα date.....	77
Κώδικα 27: Ajax call για να καλέσει ο server την api για τη μετοχή που του στέλνεται.....	78
Κώδικα 28: Ο server λαμβάνει το αίτημα για να καλέσει την api για τη μετοχή που του στάλθηκε.....	78
Κώδικα 29: Δήλωση μέρας και ώρας με τη μέθοδο .newDate().....	80
Κώδικα 30: Έλεγχος μέρας και ώρας.....	81
Κώδικα 31: Εντοπισμός δεδομένων μόνο για τη μέρα που χρειάζεται.....	81
Κώδικα 32: Δημιουργία canva για την οπτικοποίηση δεδομένων.....	82
Κώδικα 33: Ασύγχρονη συνάρτηση οπτικοποίησης δεδομένων μιας μέρας.....	83
Κώδικα 34: Δημιουργία γραφικής παράστασης με το Chart.js.....	83
Κώδικα 35: Δημιουργία πίνακα με τα x,y σημεία της γραφικής παράστασης.....	84
Κώδικα 36: Χρήση plugin crosshair στη γραφική παράσταση.....	85
Κώδικα 37: Ρυθμίσεις για τον άξονα x της γραφικής παράστασης.....	86
Κώδικα 38: Εισαγωγή μπάρα πλοήγησης στα αρχεία ejs.....	87
Κώδικα 39: Χρήση ιδιότητας flexbox.....	89
Κώδικα 40: Child containers μέσα στο parent container στη σελίδα Stocks.....	89
Κώδικα 41: Child containers του container-stock.....	90
Κώδικα 42: Δημιουργία ταμπελών για τις γραφικές παραστάσεις.....	91
Κώδικα 43: Χρήση jQuery για την ενεργοποίηση της πρώτης ταμπέλας.....	92
Κώδικα 44: Δημιουργία html αντικειμένων για τις πληροφορίες της μετοχής.....	93
Κώδικα 45: Εντοπισμός θέση στοιχείων της ώρας που ανοίγει και κλείνει το χρηματιστήριο στο πίνακα.....	94
Κώδικα 46: Ανάθεση τιμών στα html στοιχεία με τις πληροφορίες της μετοχής.....	94
Κώδικα 47: Εντοπισμός μεγαλύτερης τιμής στο πίνακα μας.....	95
Κώδικα 48: Υπολογισμός διαφοράς και ποσοστού μεταξύ των δυο τιμών της μετοχής.....	96

Κώδικα 49: Εισαγωγή χρώματος στη γραφική παράσταση τωρινής μέρας.....	96
Κώδικα 50: Δημιουργία form στη σελίδα Stocks.....	97
Κώδικα 51: Με τη χρήση της javascript ακούμε όταν ο χρήστης πατάει το κουμπί αναζήτησης.....	97
Κώδικα 52: Συνάρτηση ανανέωσης html στοιχείων στη σελίδα Stocks.....	98
Κώδικα 53: Κουμπί αποθήκευσης μετοχής στη βάση δεδομένων.....	99
Κώδικα 54: Στέλνουμε το σύμβολο της μετοχής στον server για να το αποθηκεύσει στη βάση δεδομένων.....	100
Κώδικα 55: Ο server δέχεται το σύμβολο και το περνάει στη βάση δεδομένων.....	101
Κώδικα 56: container για την εισαγωγή μπάρας πλοήγησης στη σελίδα userpage.....	102
Κώδικα 57: Χρήση ιδιότητας flexbox στη σελίδα userpage.....	103
Κώδικα 58: Child containers του container-stock στη σελίδα userpage.....	103
Κώδικα 59: Child containers του row1 container τη σελίδα userpage.....	105
Κώδικα 60: Container για τη λίστα με τις μετοχές του χρήστη στη σελίδα userpage.....	105
Κώδικα 61: Αίτημα από το client για να στείλει ο server τις μετοχές που έχει ακολουθήσει ο χρήστης.....	106
Κώδικα 62: Επανάληψη που εισάγει τις μετοχές του χρήστη από τον πίνακα στη λίστα.....	107
Κώδικα 63: Διαγραφή μετοχής από τη λίστα του χρήστη στη σελίδα Userpage.....	108
Κώδικα 64: Αποστολή αιτήματος διαγραφής μετοχής στον server για να τη διαγράψει από τη βάση δεδομένων.....	109
Κώδικα 65: Διαγραφή μετοχής από τη βάση δεδομένων.....	110
Κώδικα 66: Κώδικας με τον οποίο <i>αντιλαμβάνεται η εφαρμογή</i> ότι ο χρήστης <i>έκανε click</i> ένα αντικείμενο της λίστας.....	110
Κώδικα 67: Εμφάνιση συμβόλου διαγραφής μετοχής όταν το ποντίκι βρίσκεται πάνω σε κάποιο στοιχείο της λίστας.....	111
Κώδικα 68: Δημιουργία φόρμας για την εισαγωγή ανώτερης και κατώτερης τιμής.....	112
Κώδικα 69: Αποστολή τιμών που εισήγαγε ο χρήστης στη σελίδα userpage πίσω στο server.....	113
Κώδικα 70: Διαχείριση ανώτερου και κατώτερου ορίου μετοχής που εισήγαγε ο χρήστης, από τον server.....	114
Κώδικα 71: Έλεγχος αν προϋπάρχουν όρια για τη μετοχή στη βάση δεδομένων.....	114
Κώδικα 72: Δημιουργία ανώτερου και κατώτερου ορίου μετοχής στη βάση δεδομένων.....	115
Κώδικα 73: setInterval() μέθοδος για την εκτέλεση του κώδικα κάθε 5 λεπτά.....	115
Κώδικα 74: Έλεγχος αν η τωρινή τιμή της μετοχής ξεπερνάει κάποιο όριο.....	116
Κώδικα 75: Επανάληψη που καλεί τη συνάρτηση HTTP get για κάθε στοιχείο του πίνακα.....	116
Κώδικα 76: Συνάρτηση HTTP get().....	117
Κώδικα 77: Αν η τωρινή τιμή της μετοχής είναι μεγαλύτερη από το ανώτερο όριο καλείται η συνάρτηση .findEmail().....	118
Κώδικα 78: Συνάρτηση .findEmail().....	118
Κώδικα 79: Ρυθμίσεις για την αποστολή email στον χρήστη.....	119
Κώδικα 80: Αποστολή email στο χρήστη.....	120
Κώδικα 81: Διαγραφή min ή max από τη βάση δεδομένων αφού σταλθεί ενημέρωση στο χρήστη.....	120
Κώδικα 82: Μεταβλητή που περνάει το χρόνο που θα εκτελείται η μέθοδος interval().....	121
Κώδικα 83: Dropdown-menu.....	122
Κώδικα 84: Έλεγχος αν είναι συνδεδεμένος χρήστης με τη βοήθεια της ejs.....	123
Κώδικα 85: Αποστολή στοιχείων user στη σελίδα Stocks.....	123
Κώδικα 86: Δημιουργία dropdown-menu χρήστη πάνω στη μπάρα πλοήγησης.....	124
Κώδικα 87: κατασκευή dropdown-menu χρήστη.....	125
Κώδικα 88: Εισαγωγή κουκίδας ειδοποίησης.....	126
Κώδικα 89: Διαγραφή κουκίδας αφού διαβαστεί η ειδοποίηση από το χρήστη.....	127
Κώδικα 90: Διαγραφή κουκίδας από τη βάση δεδομένων.....	127

Κώδικα 91: Ενσωμάτωση Socket.io.....	128
Κώδικα 92: Socket.io .on() μέθοδος.....	128
Κώδικα 93: Σύνδεση socket.io με τη βάση δεδομένων.....	129
Κώδικα 94: Αποδοχή ειδοποιήσεων από το client μέσω websocket.....	130
Κώδικα 95: Δημιουργία ειδοποίησης.....	131
Κώδικα 96: Εισαγωγή html στοιχείου στο dropdown-menu του χρήστη.....	132
Κώδικα 97: Διαγραφή ειδοποίησης από το dropdown-menu.....	132
Κώδικα 98: Ρυθμίσεις για brotli.....	134
Κώδικα 99: Φάκελοι με τα αρχεία που θα συμπιεστούν.....	134
Κώδικα 100: Δημιουργία brotli αρχείων.....	135
Κώδικα 101: Ρυθμίσεις για το Gzip.....	135