ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΑΓΙΩΝ ΠΕΡΟΥΣΙΑΚΩΝ ΣΤΟΙΧΕΙΩΝ

Διπλωματική Εργασία

Ελένη Μαρία Λευκοπούλου

Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Σεπτέμβριος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΑΓΙΩΝ ΠΕΡΟΥΣΙΑΚΩΝ ΣΤΟΙΧΕΙΩΝ

Διπλωματική Εργασία

Ελένη Μαρία Λευκοπούλου

Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Σεπτέμβριος 2022

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DEVELOPMENT OF FIXED-ASSET MANAGEMENT SYSTEM

Diploma Thesis

Eleni Maria Lefkopoulou

Supervisor: Michael Vassilakopoulos

September 2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων **Μιχαήλ Βασιλακόπουλος**
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Γεώργιος Θάνος**
Μέλος Ε.ΔΙ.Π., Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Ελένη Τουσίδου**
Μέλος Ε.ΔΙ.Π., Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή αυτής της διπλωματικής εργασίας κύριο Βασιλακόπουλο Μιχαήλ για την πολύτιμη βοήθειά του κατά τη διάρκεια των μαθημάτων του, αλλά και κατά την επίβλεψη της τρέχουσας διπλωματικής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια μου και ειδικά τους γονείς μου που με στήριξαν και ήταν δίπλα μου κατά τη διάρκεια των σπουδών μου, αλλά και τους φίλους μου που αποτέλεσαν σημαντικό στήριγμα.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Η Δηλούσα

Ελένη Μαρία Λευκοπούλου

Διπλωματική Εργασία

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΑΓΙΩΝ ΠΕΡΟΥΣΙΑΚΩΝ ΣΤΟΙΧΕΙΩΝ

Ελένη Μαρία Λευκοπούλου

Περίληψη

Στις μέρες μας η ανάπτυξη λογισμικού είναι συνδεδεμένη με την επίλυση αναγκών και προβλημάτων της κοινωνίας. Ολοένα και περισσότερες εργασίες που εκτελούνταν με το χέρι ανατίθενται σε εφαρμογές και υπολογιστικά συστήματα. Μία από τις βασικότερες εργασίες που αναζητά αντικατάσταση είναι η καταγραφή των πάγιων περιουσιακών στοιχείων, η διαχείριση τους, καθώς και η αντιστοίχιση τους με κάποιον μοναδικό κωδικό. Επίσης οι γρήγοροι ρυθμοί και τα αυξημένα έξοδα επιζητούν ένα λογισμικό καταγραφής των καθημερινών συναλλαγών. Ωστόσο ο σχεδιασμός του λογισμικού δεν είναι απλή, ούτε εύκολη υπόθεση. Ο εκάστοτε μηχανικός σχεδιασμού λογισμικού πρέπει να είναι ενήμερος για τις νέες μεθοδολογίες και τα εργαλεία που προκύπτουν και να είναι σε θέση να αξιολογεί ποια είναι τα πιο κατάλληλα για την υλοποίηση της εφαρμογής του. Στα πλαίσια αυτής της διπλωματικής εργασίας τίθεται το θεωρητικό πλαίσιο για την ανάπτυξη μιας διαδικτυακής εφαρμογής και αναλύεται η διαδικασία σχεδιασμού ενός συστήματος κατάλληλου να διαχειρίζεται πάγια περιουσιακά στοιχεία και συναλλαγές. Πιο συγκεκριμένα τεκμηριώνονται και επεξηγούνται αναλυτικά η μεθοδολογία Agile ως τρόπος ανάπτυξης του λογισμικού, η χρήση της React.js σε συνδυασμό με την HTML και CSS για το σχεδιασμό του front-end καθώς και η χρήση της Node.js για το σχεδιασμό του back-end. Επίσης αναλύεται ο σχεδιασμός των βάσεων δεδομένων, η χρήση της mongoDB σε συνδυασμό με τη graphQL και το Apollo καθώς και η αυθεντικοποίηση του χρήστη μέσω του keycloak. Πρόκειται λοιπόν για έναν ολοκληρωμένο οδηγό σχεδίασης μιας διαδικτυακής εφαρμογής διαχείρισης πάγιων περιουσιακών στοιχείων και συναλλαγών.

Λέξεις-κλειδιά:

Ανάπτυξη διαδικτυακών εφαρμογών , σχεδιασμός λογισμικού, front-end, back-end, React, graphql ,Apollo, Keycloak MongoDB, πάγια περιουσιακά στοιχεία, έσοδα, έξοδα.

Diploma Thesis

DEVELOPMENT OF FIXED-ASSET MANAGEMENT SYSTEM

Eleni Maria Lefkopoulou

Abstract

In the modern world, software development is linked to problem-solving and societal needs. As time goes on, more and more tasks that were once taken care of by hand are now replaced by applications and computing systems. The registration of fixed assets, their maintenance, and their matching with a unique code are some of the most crucial jobs that need replacement. People and businesses also require an application that records their everyday transactions due to the rapid pace at which the world operates today and the rising cost of living. But creating this application is not a simple undertaking. The software designer must always stay up to date with contemporary approaches and tools. The theoretical foundation for developing a web application will be laid out in the following bachelor's thesis, and then we will build an application that addresses the issues we mentioned earlier. We will go into further detail on the Agile approach used to create the software, the usage of React.js in conjunction with HTML and CSS for the front-end design, and the use of Node.js for the back-end design. In addition, we'll examine the database architecture, the use of MongoDB in combination with graphQL and Apollo, and also the use of Keycloak for user authentication. So this is a comprehensive guide to designing a fixed asset and transaction management web application.

Keywords:

Web development, Full stack development, Front-end development, Back-end development , mongoDB, React.js , Node.js, graphql ,Apollo, Keycloak,fixed asset, incomes, outcomes.

Πίνακας περιεχομένων

Ευχαριστίες **ix**

Περίληψη **xii**

Abstract **xiii**

Πίνακας περιεχομένων **xv**

Κατάλογος σχημάτων **xix**

Κατάλογος πινάκων **xxiii**

1 Εισαγωγή **1**

1.1 Αντικείμενο της διπλωματικής 1

1.1.1 Συνεισφορά 2

1.2 Οργάνωση του τόμου 2

2 Ανάπτυξη διαδικτυακών εφαρμογών **5**

2.1 Πλεονεκτήματα και μειονεκτήματα διαδικτυακών εφαρμογών 6

2.1.1 Πλεονεκτήματα 6

2.1.2 Μειονεκτήματα 7

2.2 Front end 8

2.2.1 HTML, CSS και JavaScript 9

2.2.2 FRONT FRAMEWORKS 12

2.2.3 CSS FRAMEWORKS 14

2.3 Back end 14

2.3.1 Back-end Frameworks 15
2.4 Full-stack 17
2.5 Η λογική Πελάτη Εξυπηρετητή και
ο Παγκόσμιος Ιστός 18
2.5.1 Το Διαδίκτυο 18
2.5.2 Ο Παγκόσμιος Ιστός 18
2.5.3 Πελάτης-Εξυπηρετητής 18
2.6 Συμπέρασμα 19

3 Σχεδιασμός Λογισμικού 21
3.1 Τι είναι το λογισμικό 21
3.2 Μεθοδολογίες ανάπτυξης λογισμικού 22
3.2.1 Το μοντέλο του καταρράκτη 23
3.2.2 Το μοντέλο της πρωτοτυποποίησης 23
3.2.3 Το μοντέλο λειτουργικής επαύξησης 24
3.2.4 Το σπειροειδές μοντέλο 24
3.2.5 Η ευέλικτη μεθοδολογία ανάπτυξης- Agile 24
3.3 Διάταξη λογισμικού 25
3.4 Απαιτήσεις λογισμικού 26
3.5 Κωδικοποίηση 28

4 Η εφαρμογή Fixed Asset Management 31
4.1 Ορισμός του προβλήματος και τρέχουσα ανάγκη 31
4.2 Υπάρχουσες εφαρμογές 32
4.3 Απαιτήσεις εφαρμογής 36
4.3.1 Σύντομος ορισμός απαιτήσεων και ορολογία 36
4.3.2 Ομάδα χρηστών 37
4.3.3 Λειτουργικές απαιτήσεις 37
4.3.4 Μη Λειτουργικές απαιτήσεις 40
4.4 Επιλογή εργαλείων και μεθοδολογίας 40
4.4.1 Μεθοδολογία 41
4.4.2 back-end 41
4.4.3 front-end 41

4.4.4 Βάσεις δεδομένων 42
4.4.5 Ασφάλεια της εφαρμογής 42
4.4.6 4-tier 42

5 Βάσεις Δεδομένων 43

5.1 Επιλογή Βάσης Δεδομένων 43
5.1.1 Σύγκριση σχεσιακών και μη σχεσιακών βάσεων 43
5.1.2 Η MongoDB 44
5.2 Βιβλιοθήκες, εργαλεία και προγράμματα 45
5.2.1 MongoDB Compass 45
5.2.2 Mongoose 47
5.2.3 GraphQL 47
5.2.4 Apollo 48
5.3 Σχήματα βάσης και συλλογές 50
5.3.1 Users 51
5.3.2 Teams 52
5.3.3 Accounts 53
5.3.4 Expenses 54
5.3.5 Categories 55
5.3.6 FixedAsset 56
5.3.7 Παράδειγμα κώδικα 57
5.4 Queries 58
5.4.1 Επεξήγηση όλων των queries 58
5.4.2 Ορισμός ενός querie στον κώδικα 60
5.4.3 Επεξήγηση find και aggregate 61
5.5 Mutations 62
5.5.1 Επεξήγηση όλων των mutation 62
5.5.2 Ορισμός ενός mutation στον κώδικα 65
5.5.3 Επεξήγηση delete, create και update 66
5.5.4 Resolvers 67

6 Αυθεντικοποίηση και εξουσιοδότηση 69

6.1 Keycloak 69

6.1.1 Authentication,Cookies,Session 72
6.2 Σύνδεση στο Keycloak 73
6.3 Σύνδεση χρήστη-LOGIN 74
6.4 Εγγραφή χρήστη - SIGNUP 75
6.5 Αποσύνδεση χρήστη - LOGOUT 76

7 Το γραφικό περιβάλλον και οι λειτουργικές απαιτήσεις 77
7.1 React και HTML 77
7.2 Κεντρική σελίδα 79
7.3 Η λειτουργία Teams 80
7.4 Η λειτουργία accounts 83
7.5 Η λειτουργία settings 84
7.6 Η λειτουργία transactions 88
7.7 Η λειτουργία fixed asset 91
7.7.1 QR code 95
7.8 Επεξεργασία λαθών 96

8 Συμπεράσματα 99
8.1 Σύνοψη και συμπεράσματα 99
8.2 Μελλοντικές επεκτάσεις 100

Βιβλιογραφία 101

ΠΑΡΑΡΤΗΜΑΤΑ 105

A Η χρήση της εφαρμογής 107

Κατάλογος σχημάτων

4.1 Στιγμιότυπα εκτέλεσης της εφαρμογής WMSOne 33
4.2 Στιγμιότυπα εκτέλεσης της εφαρμογής Infinity Asset 34
4.3 Στιγμιότυπα εκτέλεσης της εφαρμογής Asset Panda 34
4.4 Στιγμιότυπα εκτέλεσης της εφαρμογής Γρήγορος προϋπολογισμός 35
4.5 Στιγμιότυπα εκτέλεσης της εφαρμογής Διαχείριση εξόδων 35
4.6 Στιγμιότυπα εκτέλεσης της εφαρμογής Monefy 36
4.7 Βήματα του Agile μοντέλου 41

5.1 Αρχική σελίδα του compass 46
5.2 Στιγμιότυπο από την χρήση του compass 46
5.3 Σύνδεση της Mongoose στη βάση δεδομένων 47
5.4 Σύνδεση του διακομιστή στο Apollo server 49
5.5 Στιγμιότυπο από το playground της Graph 49
5.6 Σύνδεση του πελάτη στο Apollo client 50
5.7 Σύνδεση του πελάτη στο Apollo client 50
5.8 Στιγμιότυπο από τον ορισμό σχήματος στην mongoose 57
5.9 Στιγμιότυπο από τον ορισμό τύπων και union στην graph 58
5.10 Στιγμιότυπο από τον ορισμό union στους resolvers 58
5.11 Όλα τα queries που χρησιμοποιούνται 58
5.12 Κώδικας που εκτελεί το query και ορίζεται στους resolvers 61
5.13 Ορισμός του query στο front 61
5.14 Κλήση στο front μέσω της useQuery 61
5.15 Όλα τα mutations που χρησιμοποιούνται 62
5.16 Κώδικας που εκτελούν mutation update και create 65
5.17 Ορισμός των mutation στο front 66

5.18 Κλήση στο front μέσω της useMutation 66
5.19 Χρήση του mutation σε συνάρτηση 66

6.1 Γενικές ρυθμίσεις για το Realm 71
6.2 Ρυθμίσεις για το Login 71
6.3 Ρυθμίσεις για τα Tokens 71
6.4 Ρυθμίσεις για τους ρόλους 72
6.5 Ρυθμίσεις για τον client του back 72
6.6 Ρυθμίσεις για τον client του front 72
6.7 Keycloak sidebar 72
6.8 Σύνδεση στο Keycloak από τη μεριά του Server 74
6.9 Στιγμιότυπο κώδικα από επικοινωνία με το Keycloak 74
6.10 Σελίδα για είσοδο του χρήστη στην εφαρμογή 75
6.11 Περίπτωση εισόδου λανθασμένων στοιχείων 75
6.12 Σελίδα εγγραφής του χρήστη στην εφαρμογή 76

7.1 Main page πριν συνδεθεί ο χρήστης 80
7.2 Dropdown μενού για την περιήγηση στην εφαρμογή 80
7.3 Παράσταση όλων των ομάδων που είναι εγγεγραμένος ο χρήστης 81
7.4 Αναδυόμενο παράθυρο για την δημιουργία ομάδας 81
7.5 Αναδυόμενο παράθυρο για την εγγραφή σε κάποια ομάδα με βάση τον κωδικό της 82
7.6 Αναδυόμενο παράθυρο για επιβεβαίωση απεγγραφής 82
7.7 Αναδυόμενο παράθυρο για τον κωδικό της ομάδας 82
7.8 Κουμπί επιλογής σύνδεσης 82
7.9 Εμφάνιση όλων των λογαριασμών 83
7.10 Αναδυόμενο παράθυρο για τη δημιουργία νέου λογαριασμού 84
7.11 Αναδυόμενο παράθυρο για την επεξεργασία λογαριασμού 84
7.12 Αναδυόμενο παράθυρο για την διαγραφή λογαριασμού 84
7.13 Προβολή στοιχείων χρήστη με δυνατότητα επεξεργασίας 85
7.14 Προβολή στοιχείων χρήστη που είναι συνδεδεμένος ως ομάδα 85
7.15 Κατηγορίες εξόδων 86
7.16 Κατηγορίες εσόδων 86

7.17 Αναδυόμενο παράθυρο για προσθήκη κατηγορίας 87
7.18 Αναδυόμενο παράθυρο για επεξεργασία κατηγορίας 87
7.19 Αναδυόμενο παράθυρο για επιλογή χρώματος 87
7.20 Αναδυόμενο παράθυρο για επιλογή εικονιδίου 88
7.21 Η εμφάνιση της σελίδας όταν δεν έχουν προστεθεί συναλλαγες 89
7.22 Διάγραμμα εξόδων για ένα μήνα 89
7.23 Αναδυόμενο παράθυρο για προσθήκη, την ώρα που ο χρήστης επιλέγει λογαριασμό 90
7.24 Αναδυόμενο παράθυρο για προσθήκη όταν ο χρήστης επιλέγει κατηγορία . 91
7.25 Αναδυόμενο παράθυρο για δημιουργία κατηγορίας 91
7.26 Αναδυόμενο παράθυρο για επεξεργασία κατηγορίας 92
7.27 Όλες οι κατηγορίες 92
7.28 Όλα τα περιουσιακά στοιχεία μίας κατηγορίας 93
7.29 QR code 93
7.30 Φόρμα προσθήκης πάγιου περιουσιακού στοιχείου 94
7.31 Φόρμα προσθήκης πάγιου περιουσιακού στοιχείου συνέχεια 94
7.32 Επεξεργασία πάγιου περιουσιακού στοιχείου 95
7.33 Ειδοποίηση επιτυχούς ενημέρωσης 97
7.34 Παράδειγμα χρήσης validators 97

A.1 Στιγμιότυπο από την επιτυχή εκκίνηση του Keycloak 108
A.2 Στιγμιότυπο από την επιτυχή εκκίνηση του back-end 109

Κατάλογος πινάκων

4.1 Λειτουργικές απαιτήσεις 39
4.2 Μη Λειτουργικές απαιτήσεις 40

5.1 User Model 51
5.2 Scenario Model 52
5.3 Team Model 53
5.4 Account Model 54
5.5 Expense Model 55
5.6 Date Model 55
5.7 Category Model 56
5.8 Fixed Asset Model 57

Κεφάλαιο 1

Εισαγωγή

Στις μέρες μας παρατηρείται μια αυξημένη ζήτηση στο χώρο της ανάπτυξης των διαδικτυακών εφαρμογών. Ωστόσο το υλικό στο διαδίκτυο είναι χαοτικό και τα διαθέσιμα εργαλεία πολλά. Σκοπός αυτής της διπλωματικής εργασίας είναι να θέσει τα πλαίσια και τις γνώσεις που πρέπει να έχει κάποιος, που επιθυμεί να εργαστεί ως web developer. Επίσης, να εισάγει τον αναγνώστη στη νοοτροπία της σχεδίασης λογισμικού και μέσω της σχεδίασης μιας διαδικτυακή εφαρμογής να εξηγήσει τα βήματα που πρέπει να ακολουθήσει κάποιος ώστε να την ολοκληρώσει.

1.1 Αντικείμενο της διπλωματικής

Είναι γεγονός πως η σχεδίαση λογισμικού συνδέεται άρρηκτα με την επίλυση καθημερινών προβλημάτων και έχει σκοπό να δώσει λύση σε αυτά. Η τρέχουσα διπλωματική εργασία εστιάζει στην ανάγκη σχεδιασμού ενός λογισμικού, ικανό να καταγράφει τα πάγια περιουσιακά στοιχεία που έχει κάποιος στην κατοχή του, αλλά και των οικονομικών συναλλαγών στη διάρκεια της ημέρας. Ακόμη, στόχος είναι το λογισμικό αυτό να προσφέρει τη διασύνδεση των μελών μέσω ομάδων, όπου θα μπορούν να διαμοιραστούν την πληροφορία που εισάγουν. Πρόκειται λοιπόν για ένα πρόγραμμα λογιστικού χαρακτήρα, που θα μπορεί να χρησιμοποιηθεί από ιδιώτες και από εταιρείες ή ιδρύματα. Βασιζόμενοι πάνω στην ανάγκη σχεδιασμού ενός νέου λογισμικού η εργασία αυτή βρίσκει πάτημα ώστε να εξηγήσει αναλυτικά τα βήματα που πρέπει να ακολουθήσει κάποιος, καθώς και τις ενέργειες που πρέπει να κάνει, ώστε να καταφέρει να σχεδιάσει ολοκληρωμένα μία διαδικτυακή εφαρμογή αντιμετωπίζοντάς την ως ένα ολοκληρωμένο λογισμικό.

1.1.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Επεξηγήθηκε η διαφορά του front-end και του back-end.
2. Μελετήθηκαν εργαλεία ανάπτυξης εφαρμογών πελατών ή αλλιώς εργαλεία κατάλληλα για το front-end.
3. Μελετήθηκαν εργαλεία και γλώσσες κατάλληλες για την ανάπτυξη εφαρμογών εξυπηρετητή ή αλλιώς εργαλεία κατάλληλα για το back-end.
4. Παρουσιάστηκαν οι διαθέσιμες βάσεις δεδομένων και εξηγήθηκε ο τρόπος επιλογής τους.
5. Μελετήθηκαν μεθοδολογίες ανάπτυξης λογισμικού.
6. Παρουσιάστηκαν οι ενέργειες που απαιτούνται για τον ορθό σχεδιασμό λογισμικού.
7. Εξηγήθηκε ο τρόπος καθορισμού λειτουργικών και μη λειτουργικών απαιτήσεων μιας εφαρμογής.
8. Μελετήθηκαν οι υπάρχουσες εφαρμογές.
9. Ορίστηκαν οι απαιτήσεις της εφαρμογής.
10. Αναλύθηκε ο τρόπος σχεδιασμού της βάσης δεδομένων καθώς και τα ερωτήματα σε αυτή.
11. Αναλύθηκε ο τρόπος αυθεντικοποίησης των χρηστών.
12. Επεξηγήθηκε το γραφικό περιβάλλον με το οποίο αλληλεπιδρά ο χρήστης.

1.2 Οργάνωση του τόμου

Στο Κεφάλαιο 2 θα δοθεί ο ορισμός μίας διαδικτυακής εφαρμογής και θα αναλυθούν τα πιθανά εργαλεία που μπορεί να χρησιμοποιήσει κάποιος, τόσο από τη μεριά του front-end όσο και από τη μεριά του back-end. Στο Κεφάλαιο 3 θα αναλυθούν οι μεθοδολογίες και τα βήματα που πρέπει να ακολουθήσει κάποιος για τη σωστή ανάπτυξη λογισμικού. Στο κεφάλαιο

4 ορίζεται το πρόβλημα που θα επιλυθεί, μελετώνται παρόμοιες εφαρμογές και σχεδιάζονται οι απαιτήσεις για την εφαρμογή της διπλωματικής. Στο κεφάλαιο 5 γίνεται ανάλυση των υπαρκτών βάσεων δεδομένων και επιλέγεται η κατάλληλη για την εφαρμογή μας. Επίσης επεξηγούνται τα σχήματα της βάσης καθώς και τα ερωτήματα προς αυτή. Στο κεφάλαιο 6 εξηγείται το λογισμικό που χρησιμοποιήθηκε για την αυθεντικοποίηση των χρηστών. Στο κεφάλαιο 7 εξηγείται το γραφικό περιβάλλον της εφαρμογής και η επίτευξη των λειτουργικών απαιτήσεων. Στο τελευταίο κεφάλαιο 8 διατυπώνονται τα συμπεράσματα και η σύνοψη.

Κεφάλαιο 2

Ανάπτυξη διαδικτυακών εφαρμογών

Τα τελευταία χρόνια παρατηρείται μια αύξηση στην κατασκευή διαδικτυακών εφαρμογών (web application ή web app). Τι ορίζουμε όμως ως διαδικτυακή εφαρμογή; Πρόκειται για μια εφαρμογή η οποία είναι διαθέσιμη στους χρήστες μέσω του διαδικτύου (internet) ή του ενδοδικτύου(intranet) μιας εταιρείας και ο χρήστης χρειάζεται μόνο τον περιηγητή του (browser) για να τη χρησιμοποιήσει. Οι εφαρμογές αυτές συνήθως εκτελούνται σε ισχυρές υπολογιστικές μηχανές, οι οποίες έχουν τον ρόλο του σταθμού εξυπηρέτησης και παρέχουν τις υπηρεσίες τους σε περισσότερους του ενός χρήστη. Με το πέρασμα των χρόνων οι διαδικτυακές εφαρμογές ανταγωνίζονται τις τοπικές εφαρμογές υπολογιστών, οι οποίες ήταν για αρκετό καιρό οι πιο δημοφιλείς (λογικό, καθώς ήταν οι πρώτες που δημιουργήθηκαν). Πλέον λοιπόν, όταν μιλάμε για ανάπτυξη λογισμικού, συχνά απευθυνόμαστε στην ανάπτυξη και σχεδίαση μιας διαδικτυακής εφαρμογής. Η διαδικασία αυτή είναι αρκετά περίπλοκη και απαιτεί ο προγραμματιστής να κατανοεί σε βάθος τα μέρη της εφαρμογής, καθώς και τα διαθέσιμα εργαλεία, τεχνολογίες και γλώσσες προγραμματισμού, ώστε να είναι σε θέση να επιλέξει τα πιο κατάλληλα. Η διαδικασία ανάπτυξης χωρίζεται σε δύο μέρη, γνωστά και ως front-end και back-end. Στο κεφάλαιο αυτό λοιπόν θα επεξηγηθούν τα μέρη αυτά και θα αναλυθούν τα διαθέσιμα εργαλεία και οι γλώσσες ώστε να χτιστεί το υπόβαθρο της ορθής επιλογής τεχνολογιών.

2.1 Πλεονεκτήματα και μειονεκτήματα διαδικτυακών εφαρμογών

Συγκρίνοντας τις γνώριμες σε όλους τοπικές εφαρμογές υπολογιστών με τις διαδικτυακές εφαρμογές μπορεί κάποιος να ανακαλύψει τα βασικά πλεονεκτήματα καθώς και τα μειονεκτήματα αυτών.[1]

2.1.1 Πλεονεκτήματα

- **Άμεση πρόσβαση από οποιαδήποτε συσκευή**: Αρκετά θετικό είναι το γεγονός πως κάθε χρήστης έχει άμεση πρόσβαση στις εφαρμογές που θέλει να χρησιμοποιήσει, είτε από κάποιον υπολογιστή, είτε από οποιαδήποτε άλλη φορητή συσκευή που έχει ίντερνετ, χωρίς να χρειάζεται να εγκαταστήσει κάποιο επιπρόσθετο λογισμικό. Το μόνο απαραίτητο είναι ο περιηγητής διαδικτύου ο οποίος είναι προεγκατεστημένος σε όλα τα λειτουργικά συστήματα. Η ιδιότητα αυτή είναι αρκετά βοηθητική, ειδικά σε περιπτώσεις μεγάλων επιχειρήσεων, που υπό άλλες συνθήκες θα έπρεπε να εγκατασταθεί η εφαρμογή σε κάθε έναν υπολογιστή ξεχωριστά.

- **Δυνατότητα χρήσης ανεξαρτήτως τοποθεσίας**: Ως συνέχεια του παραπάνω, μια διαδικτυακή εφαρμογή είναι εύκολο οι χρήστες να τη χρησιμοποιήσουν από όπου και αν βρίσκονται. Εφόσον πρόκειται για εφαρμογή που χρησιμοποιείται στο περιβάλλον εργασίας τους, δίνεται η δυνατότητα να εργαστούν από το σπίτι τους ή από απομακρυσμένες περιοχές. Με τον τρόπο αυτό χτίζονται τα θεμέλια ώστε να είναι εφικτό το μοντέλο της απομακρυσμένης εργασίας (γνωστό και ως remote).

- **Συμβατές με όλα τα λειτουργικά συστήματα**: Ένα ακόμα πλεονέκτημα είναι ότι είναι συμβατές με όλα τα λειτουργικά συστήματα. Καθώς η εφαρμογή εκτελείται μέσω του περιηγητή του διαδικτύου, είναι ικανή να εκτελείται σε κάθε λειτουργικό σύστημα.

- **Δεν καταναλώνουν πόρους και δεν καταλαμβάνουν χώρο στον υπολογιστή του χρήστη**: Εφόσον η εφαρμογή δεν εκτελείται τοπικά στον υπολογιστή, δεν καταναλώνει πόρους από το σύστημα. Είναι λοιπόν αρκετά ελαφριά για την υπολογιστική μονάδα. Στο ίδιο πλαίσιο, η εφαρμογή δεν καταλαμβάνει καθόλου ή σχεδόν καθόλου χώρο στον δίσκο του χρήστη, αφού το σύνολο της εφαρμογής είναι αποθηκευμένο στον

εξυπηρετητή και μόνο κατά την χρήση της εφαρμογής μπορεί να υπάρξει μεταφορά δεδομένων στην υπολογιστική μονάδα του χρήστη και μόνο αν αυτός το επιθυμεί.

- **Γρήγορη αναβάθμιση και νέο βελτιωμένο περιβάλλον**: Σε αντίθεση με τις τοπικές εφαρμογές, όταν η διαδικτυακή εφαρμογή θέλει αναβάθμιση, δε χρειάζεται η αναβάθμιση αυτή να γίνει σε κάθε έναν υπολογιστή ξεχωριστά. Αντιθέτως η αναβάθμιση πραγματοποιείται μόνο στον εξυπηρετητή(server) που φιλοξενεί την εφαρμογή και ταυτόχρονα το αναβαθμισμένο πρόγραμμα είναι διαθέσιμο σε όλους τους χρήστες. Με τον τρόπο αυτό εξοικονομείται χρόνος και απαιτείται λιγότερο δυναμικό για να κάνει την αναβάθμιση, καθώς απλώς πρέπει να γίνει deploy ο νέος κώδικας της εφαρμογής. Επίσης όσο αναβαθμίζονται οι εκδώσεις της HTML και της CSS και εφόσον βγαίνουν νέα εργαλεία και βιβλιοθήκες, το γραφικό περιβάλλον είναι εύκολο να γίνεται ολοένα και πιο προσιτό στο χρήστη, καθώς και να προστίθενται νέες λειτουργίες.

- **Δυνατότητα χρήσης και εκτός διαδικτύου - ενδοδικτύου**: Οι σύγχρονες διαδικτυακές εφαρμογές (εφαρμογές με χρήση HTML5) έχουν τη δυνατότητα της εκτός διαδικτύου χρήσης μιας διαδικτυακής εφαρμογής, με την προϋπόθεση ότι η εφαρμογή έχει κατασκευαστεί με ανάλογο τρόπο. Αν λοιπόν για οποιοδήποτε λόγο διακοπεί η σύνδεση στο διαδίκτυο, ο χρήστης συνεχίζει να χρησιμοποιεί την εφαρμογή κανονικά. Αυτό επιτυγχάνεται μέσω του περιηγητή, ο οποίος κρατάει ένα αντίγραφο από τα αρχεία και τα χρησιμοποιεί όταν το κρίνει απαραίτητο. Ωστόσο είναι μια διαδικασία που δε γίνεται αυτόματα σε όλες τις διαδικτυακές εφαρμογές και προαπαιτεί να το έχει λάβει υπόψιν του ο εκάστοτε προγραμματιστής.

2.1.2 Μειονεκτήματα

- **Μη πλήρης συμβατότητα των περιηγητών**: Ένα μειονέκτημα που σχετίζεται με την τελευταία έκδοση της HTML είναι η μη πλήρης συμβατότητα των περιηγητών με την έκδοση αυτή. Ενώ η HTML5 έχει πολλές δυνατότητες, η μη συμβατότητα κάποιων περιηγητών έχει ως αποτέλεσμα να περιορίζει τους προγραμματιστές. Σε περίπτωση τώρα που κάποιος δε σκεφτεί τη μη συμβατότητα αυτό έχει ως αποτέλεσμα σε ορισμένους περιηγητές κάποιες λειτουργίες να μην είναι διαθέσιμες και έτσι η εφαρμογή είτε να έχει προβλήματα είτε να μη λειτουργεί καθόλου. Για το λόγο αυτό καλό είναι να υπάρχει ένας προτεινόμενος περιηγητής, αλλά παράλληλα να έχει προβλεφθεί και η

χρήση άλλων. Ένας καλός τρόπος, για να ελεγχθεί η συμβατότητα του περιηγητή μας με την HTML5, είναι τα διάφορα διαδικτυακά τεστ, που αξιολογούν τις δυνατότητες του.

- **Αποθηκευτικός χώρος**: Μπορεί ως πλεονέκτημα των εφαρμογών αυτών να έχουμε το ότι δε δεσμεύεται τοπικά χώρος στο μηχάνημα του χρήστη, αλλά αυτό δεν σημαίνει πως έχουμε απαλλαγεί εντελώς από το πρόβλημα του χώρου. Θα πρέπει να έχει προβλεφθεί ο κατάλληλος χώρος στον εξυπηρετητή, ώστε να αποθηκεύονται τα δεδομένα (βάσεις δεδομένων), πιθανώς κάποιες εικόνες, καθώς και χώρος στο δίσκο. Όσο η εφαρμογή βελτιώνεται και το κοινό-χρήστες της αυξάνονται, θα χρειάζεται ολοένα και περισσότερος χώρος, επομένως η εφαρμογή θα γίνεται και πιο ακριβή, ειδικά σε περίπτωση που ο χώρος αυτός δεσμεύεται στο cloud.

- **Αναγκαστική αναβάθμιση**: Ένα χαρακτηριστικό παράδειγμα πλεονεκτήματος και μειονεκτήματος ταυτόχρονα αποτελεί και η αναβάθμιση της εφαρμογής. Στην περίπτωση της τοπικής εφαρμογής ο χρήστης ή η επιχείρηση μπορούν να αναβαθμίσουν την εφαρμογή, όποτε αυτοί το κρίνουν απαραίτητο. Έχουν έτσι το δικαίωμα να κρίνουν το κόστος αναβάθμισης, την αξιοπιστία την νέας εφαρμογής αλλά και το χρόνο αναπροσαρμογής στην νέα έκδοση. Αντίθετα, στις διαδικτυακές εφαρμογές, η αναβάθμιση γίνεται χωρίς πρώτα να ερωτηθούν όλοι οι χρήστες. Για παράδειγμα, σε περίπτωση που μία αναβάθμιση περιέχει σφάλματα σε τοπικές εφαρμογές, ο χρήστης μπορεί να το αποτρέψει αυτό απλά επιλέγοντας να μην κάνει την αναβάθμιση. Αντιθέτως, στις διαδικτυακές εφαρμογές η αναβάθμιση γίνεται αυτόματα και όλοι οι χρήστες θα έχουν την έκδοση που μπορεί να εμπεριέχει κάποιο σφάλμα, μέχρι να πραγματοποιηθεί εκ νέου κάποια αλλαγή.

2.2 Front end

Η ορολογία front-end αναφέρεται στο κομμάτι της εφαρμογής που είναι ορατή στο χρήστη. Επειδή στόχος του front είναι να δημιουργήσει ένα ευχάριστο περιβάλλον στο χρήστη, οι προγραμματιστές που ασχολούνται με αυτό καλό είναι να αντιλαμβάνονται την ανθρώπινη ψυχολογία. Στην ουσία πρόκειται για το ορατό μέρος της εφαρμογής ή αλλιώς διεπαφή, μέσω του οποίου ο χρήστης αλληλεπιδρά με την εφαρμογή. Το front χτίζεται με τη βοήθεια των HTML, CSS, JavaScript. Παράλληλα υπάρχουν κάποια frameworks, τα οποία επιταχύ-

νουν τη διαδικασία σχεδιασμού. Τα πιο γνωστά front frameworks είναι τα : React, AngularJS και VueJS, λίγα λόγια για τα οποία θα πούμε παρακάτω. Ακολουθούν στην κατάταξη και τα εξής Semantic UI, Svelte, Preact, Ember τα οποία δεν θα αναλυθούν στην συγκεκριμένη διπλωματική εργασία. Το πιο γνωστό CSS framework είναι το Bootstrap και ακολουθούν τα όχι και τόσο διαδεδομένα Material UI, Bulma, Materialize, Tailwind CSS, Foundation.[2]

2.2.1 HTML, CSS και JavaScript

HTML

Η γλώσσα HTML είναι το βασικό εργαλείο που πρέπει να ξέρει να χρησιμοποιεί όποιος ασχολείται με front-end. Τα αρχικά της προέρχονται από τα ”Hyper Text Markup Language” δηλαδή γλώσσα σήμανσης υπερκειμένου και δεν είναι γλώσσα προγραμματισμού, αλλά ανήκει σε μία ομάδα γλωσσών σήμανσης. Η πρώτη της έκδοση γράφτηκε από τον Tim Berners-Lee στις αρχές της δεκαετίας του 1990 και βασικός της σκοπός ήταν να περιγράψει επιστημονικά έγγραφα. Έπειτα ακολούθησαν αρκετές βελτιωμένες εκδόσεις της έως και σήμερα. Πλέον ο βασικός της ρόλος είναι να παρέχει πληροφορίες στα προγράμματα ιστού σχετικά με τον τρόπο απόδοσης και διαχείρισης ενός εγγράφου ιστού. Οι οδηγίες δίνονται μέσω ετικετών(tags) οι οποίες χρησιμεύουν για τη μορφοποίηση κειμένου και την εμφάνιση αυτού ως επικεφαλίδα, παράγραφο, λίστα και διάφορες άλλες μορφές, αλλά και για την την προσθήκη πολυμέσων, όπως εικόνες, ήχος και βίντεο. Επιπρόσθετα δίνει τη δυνατότητα αναφοράς υπερσυνδέσμων, καθώς και προσθήκης διάφορων γραφικών και λειτουργικών στοιχείων όπως είναι τα κουμπιά, οι φόρμες συμπλήρωσης στοιχείων, τα πλαίσια εισόδου κειμένου και άλλων. Οι λεγόμενες ετικέτες, μπαίνοντας μέσα σε αγκύλες που ανοίγουν και κλείνουν αποτελούν ένα στοιχείο(element). Μια ιστοσελίδα αποτελείται από πολλά διαφορετικά στοιχεία τα οποία μπορούν να δηλωθούν είτε μόνα τους, είτε εμφωλευμένα σε άλλα στοιχεία, μέσα στο στοιχείο που αποτελεί τη ρίζα του εγγράφου, δηλαδή το <html></ html>. Κάθε στοιχείο μπορεί να έχει ένα id, το οποίο είναι μοναδικό ή μπορεί να ανήκει σε μία κλάση, στην οποία μπορούν να ανήκουν παραπάνω από ένα στοιχεία. Τα id και οι κλάσεις χρησιμεύουν, όπως θα δούμε στη συνέχεια, για τη χρήση της CSS και της JavaScript.[3] Παρακάτω φαίνεται ένα παράδειγμα σύνταξης ενός HTML εγγράφου.

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Page Title</title>
</head>
<body>

<h1>Example Heading</h1>
<p>Example paragraph.</p>

</body>
</html>
```

- Η δήλωση <!DOCTYPE html> υποδηλώνει ότι το έγγραφο είναι του τύπου HTML5
- Η δήλωση <html> αποτελεί το βασικό στοιχείο ή αλλιώς ρίζα της HTML σελίδας.
- Η δήλωση <head> περιέχει διάφορες πληροφορίες σχετικά με τη σελίδα καθώς και κάποιους συνδέσμους για το στυλ της σελίδας
- Η δήλωση <title> περιέχει τον τίτλο, ο οποίος φαίνεται στον περιηγητή στην καρτέλα όπου έχουμε ανοίξει το συγκεκριμένο έγγραφο.
- Η δήλωση <body> περιέχει το σώμα της ιστοσελίδας, δηλαδή όλα τα ορατά στο χρήστη στοιχεία. Εδώ μέσα ορίζονται τα στοιχεία που προαναφέραμε ότι μπορεί να είναι κείμενο, πολυμέσα ή λειτουργικά γραφικά.

Με την HTML δηλαδή κατασκευάζεται ο σκελετός μιας ιστοσελίδας. Ωστόσο, σχεδόν ποτέ δε θα τη δούμε να χρησιμοποιείται μόνη της. Πάντα συνδυάζεται με την CSS και τη JavaScript.[4] [5] [6]

CSS

Η γλώσσα CSS είναι υπεύθυνη για την εμφάνιση της ιστοσελίδας, δηλαδή ευθύνεται για τα χρώματα, τις γραμματοσειρές, τα μεγέθη και τις αποστάσεις. Το πλήρες όνομα της είναι Cascading Style Sheets που σημαίνει διαδοχικά φύλλα ύφους ή επάλληλα φύλλα ύφους. Θεωρείται μια γλώσσα υπολογιστή και ανήκει στις γλώσσες φύλλων ύφους, που χρησιμοποιούνται για τον έλεγχο της εμφάνισης μιας ιστοσελίδας, που έχει γραφτεί με μία γλώσσα σήμανσης. Μπορεί να χρησιμοποιηθεί με 4 διαφορετικούς τρόπους.

- Ενσωματωμένα: σε κάθε στοιχείο μέσα σε ένα HTML αρχείο.
- Εσωτερικά : στην αρχή ενός HTML αρχείου μέσα στις ετικέτες <style></style>.
- Εξωτερικά : σε ένα ξεχωριστό αρχείο με κατάληξη .css το οποίο θα συμπεριληφθεί ως σύνδεσμος στο <head> του html αρχείου με την ετικέτα <link>.
- Εισαγόμενα : όπως και στην περίπτωση του εξωτερικά, αλλά θα καλείται μέσω της δήλωσης @import.

Όταν δεν είναι εμφωλευμένη στο κάθε στοιχείο, χρησιμοποιεί την ετικέτα ή την κλάση ή το id κάποιου στοιχείου, για να ορίσει την εμφάνισή τους. Πιο συγκεκριμένα, όπως φαίνεται και παρακάτω, χρησιμοποιείται το όνομα ως εκλέκτορας(selector) και ακολουθούν οι αγκύλες, μέσα στις οποίες υπάρχουν πολλά ζευγάρια ιδιότητας και τιμής, χωρισμένα με ερωτηματικό.

```
/* Code for element with tag <p></p> */
p {
  text-align: center;
  color: red;
}
/* Code for element with id="paragraph1" */
#paragraph1 {
  text-align: center;
  color: red;
}
/* Code for element with class="center" */
.center {
  text-align: center;
  color: red;
}
/* Code for element with class="center" and tag <p></p> */
p.center {
  text-align: center;
  color: red;
}
```

Έχει τη δυνατότητα επίσης να ανιχνεύει πότε κάποιο στοιχείο είναι επιλεγμένο ή πότε η οθόνη μικραίνει, ώστε να μπορεί να αλλάζει την εμφάνιση ανάλογα με την κάθε οθόνη ή το κάθε γεγονός. Μπορούμε λοιπόν να τη θεωρήσουμε ως ένα χρήσιμο σχεδιαστικό εργαλείο.

JavaScript

Η JavaScript είναι προγραμματιστική γλώσσα και μάλιστα θεωρείται η γλώσσα του διαδικτύου. Χρησιμοποιείται σε συνδυασμό με τις δύο παραπάνω γλώσσες και συγκεκριμένα είναι υπεύθυνη για τη συμπεριφορά της ιστοσελίδας. Ο αρχικός σχεδιασμός της βασίστηκε στις γλώσσες προγραμματισμού Self και Scheme. Ωστόσο η σύνταξη είναι επηρεασμένη από τη C και τα ονόματα, καθώς οι συμβάσεις ονοματοδοσίας είναι σε έναν μεγάλο βαθμό βασισμένες στη Java. Παρ' όλα αυτά δεν αποτελεί παρακλάδι κάποιων από της προαναφερόμενες γλώσσες αλλά πιο πολύ αποτελεί έναν συνδυασμό και υπηρετεί ταυτόχρονα το αντικειμενοστρεφές, το προστακτικό και το συναρτησιακό στυλ προγραμματισμού. Στη JavaScript μπορούμε να ορίσουμε μεταβλητές και συναρτήσεις. Είναι ικανή να αλλάζει δυναμικά τα ορατά στοιχεία στο χρήστη, αλλά και τις λειτουργίες τους. Αυτό το πετυχαίνει με προϋπάρχουσες συναρτήσεις, που καλούν τα στοιχεία της HTML με βάση το id ή την κλάση και έτσι προσδίδει διαφορετικό ρόλο στο καθένα τους. Για χρόνια θεωρούνταν ως client-side γλώσσα, καθώς εκτελούνταν στο πρόγραμμα περιήγησης του χρήστη αλλά τα τελευταία χρόνια με την ανάπτυξη της node.js, έχει κυρίαρχο ρόλο και στις server-side εφαρμογές. [7]

2.2.2 FRONT FRAMEWORKS

React

Η React είναι μία ανοιχτού κώδικα JavaScript βιβλιοθήκη που δημιουργήθηκε to 2013. Χρησιμοποιείται για τη δημιουργία διεπαφών χρήστη -γνωστό και ως user interface (UI)- και βασίζεται στα λεγόμενα components. Αυτό είναι και ένα από τα μεγαλύτερα πλεονεκτήματα της. Τα components μπορούν να περιέχουν κάτι απλό, για παράδειγμα ένα κουμπί, έως κάτι σύνθετο, όπως είναι ένα αναδυόμενο παράθυρο. Δέχονται παραμέτρους και αυτό τα κάνει ικανά να επαναχρησιμοποιηθούν σε πολλές περιπτώσεις απλά με διαφορετικές παραμέτρους. Αυτό γλιτώνει τον προγραμματιστή από τον επαναχρησιμοποιούμενο κώδικα και βοηθάει στο να χτιστεί μια εφαρμογή επεκτάσιμη. Επίσης κατέχει μία μεγάλη συλλογή βιβλιοθηκών. Η React είναι κατάλληλη για όσους έχουν εμπειρία στην ανάπτυξη εφαρμογών και θέλουν

να δημιουργήσουν έναν ιστότοπο ή μια εφαρμογή με πλούσια διεπαφή χρήστη. [8] [9] [10]

Angular

Η Angular είναι μια ανοιχτού κώδικα πλατφόρμα για ανάπτυξη κυρίως client-side εφαρμογών, που έχει γραφτεί σε TypeScript από την ομάδα της Google και αποτελεί μια νέα έκδοση της AngularJS, η οποία ξαναγράφτηκε από την αρχή βασισμένη στην παλιά έκδοση. Αποτελεί χρήσιμο εργαλείο για την ανάπτυξη επεκτάσιμων εφαρμογών. Περιέχει μια μεγάλη συλλογή από βιβλιοθήκες, που βοηθάνε την υλοποίηση αρκετών από τις ανάγκες που έχει μια client side, εφαρμογή. Τα θετικά της είναι ότι εξαιρετικά σύνθετες εφαρμογές ιστού μπορούν να αναπτυχθούν με την Angular και να είναι ανταγωνιστικές ως προς άλλες τοπικές εφαρμογές, καθώς δίνει τη δυνατότητα ανάπτυξής καθαρού κώδικα και παράλληλα πολύ αποδοτικού. Επίσης η Google εγγυάται πως θα υπάρξει μακροχρόνια υποστήριξη του ανοιχτού κώδικα, άρα δεν υπάρχει λόγος ανησυχίας μήπως εγκαταλειφθεί η πλατφόρμα. Ωστόσο δεν παύει να αποτελεί μία αρκετά περίπλοκη πλατφόρμα, ως προς την εκμάθηση της, οπότε δε συστήνεται σε κάποιον, που πρώτη φορά ασχολείται με front, να ξεκινήσει με την Angular. Παρόλα αυτά είναι κατάλληλη για μακροχρόνιους προγραμματιστές, που θέλουν να επενδύσουν στον τομέα και τους βοηθάει να χτίσουν εφαρμογές επεκτάσιμες. [11] [12]

View

Η ViewJS αποτελεί ένα JavaScript Framework που δημιουρήθηκε από τον Evan You το 2014. Η αρχική του ιδέα ήταν να χτίσει ένα νέο πλαίσιο ώπου θα συνδύαζε τα καλά στοιχεία από τις ήδη υπάρχουσες React και Angular. Ένα βασικό χαρακτηριστικό της Θεωρείται πως είναι αρκετά εύκολη στην εκμάθηση. Κάποιος λοιπόν που έχει ασχοληθεί με HTML, CSS και JavaScript μπορεί να ξεκινήσει να χτίζει μια εφαρμογή με View. Κατέχει λοιπόν καλά χαρακτηριστικά από τις 2 προαναφερόμενες και έχει πολύ αναλυτικό documentation, ωστόσο η μικρή ηλικία της είναι ο λόγος που υπάρχει μικρότερη υποστήριξη γι' αυτή από ότι για τους ανταγωνιστές της. Αποτελεί λοιπόν το ιδανικό πλαίσιο για κάποιον με μικρή εμπειρία στη σχεδίαση ιστοσελίδων.

2.2.3 CSS FRAMEWORKS

Bootstrap

Το Bootstrap αναπτύχθηκε από τον Mark Otto και τον Jacob Thornton και είναι μια συλλογή εργαλείων ανοιχτού κώδικα για τη σχεδίαση ιστοσελίδων και διαδικτυακών εφαρμογών. Έχει χτιστεί με HTML, CSS και JavaScript και στην ουσία δίνει τη δυνατότητα στους χρήστες της να χρησιμοποιήσουν έτοιμα μορφοποιημένα στοιχεία της html χωρίς να γράψουν από την αρχή όλη τη CSS. Πιο συγκεκριμένα, το Bootstrap έχει υλοποιήσει ένα συγκεκριμένο στυλ για όλα τα βασικά στοιχεία της html, άλλα έχει και επιπρόσθετες, πιο σύνθετες λειτουργίες, όπως είναι ένα dropdown μενού, μία μπάρα πλοήγησης, η εναλλαγή εικόνων με το σύστημα carousel, η σελιδοποίηση και άλλα. Ο χρήστης απλώς χρειάζεται να συμπεριλάβει το link για το Bootstrap ή να κάνει import την αντίστοιχη βιβλιοθήκη και έπειτα στη θέση των ετικετών βάζει τις ορισμένες από το Bootstrap αντίστοιχες ετικέτες. Ένα άλλο θετικό είναι ότι ο σχεδιασμός των στοιχείων αυτών είναι συμβατός και με τις διάφορες οθόνες, δηλαδή τα στοιχεία αυξομειώνονται, ανάλογα με το αν ο χρήστης είναι από κινητό ή υπολογιστή. Ωστόσο, παρά τα έτοιμα στυλ που παρέχει, δίνει τη δυνατότητα στο χρήστη να κάνει επέκταση του κώδικα αλλάζοντας κάποια από τα στυλ. Η δημοφιλία του οφείλεται στην ευκολία που προσφέρει σε όσους ασχολούνται με ανάπτυξη ιστοσελίδων, ιδιαίτερα σε όσους είναι αρχάριοι ή δεν τους αρέσει και τόσο το front-end.[13] [14]

2.3 Back end

Η ορολογία back-end έχει να κάνει με όσα δεν είναι ορατά στο χρήστη. Υλοποιεί τη server-side μεριά μιας εφαρμογής και είναι υπεύθυνο για τη λειτουργικότητά της. Πιο συγκεκριμένα υλοποιεί τις συναρτήσεις, τους υπολογισμούς και τις λειτουργικότητες της ιστοσελίδας και είναι η μεριά της εφαρμογής η οποία διαχειρίζεται τα δεδομένα και τις βάσεις δεδομένων. Όπως και το front, υπάρχουν αρκετά back-end frameworks τα οποία χρησιμοποιούνται για την υλοποίηση του back-end. Το κάθε ένα από αυτά βασίζεται σε διαφορετική γλώσσα προγραμματισμού. Παρακάτω θα αναλύσουμε τα βασικότερα frameworks και επιγραμματικά τις λειτουργίες της αντίστοιχής τους γλώσσας και θα προβούμε σε μία σύγκριση αυτών.[15]

2.3.1 Back-end Frameworks

Express - Node.js

- **Node.js :** Η Node αποτελεί ένα runtime περιβάλλον βασισμένο στην JavaScript που τρέχει τη JavaScript έξω από τον φυλλομετρητή. Είναι ένα εξαιρετικά ελαφρύ περιβάλλον και απίστευτα αποτελεσματικό. Σε αντίθεση με το Springboot και τη Java η Node δε βασίζεται στον πολυνηματικό προγραμματισμό αλλά σε ένα ασύγχρονο μοντέλο εισόδου-εξόδου. Είναι ικανή να επεξεργαστεί δεδομένα με ευκολία και είναι ιδανική για εφαρμογές με έντονη αλληλεπίδραση του χρήστη με το front-end. Επιπροσθέτως, η Node καταναλώνει πολύ λίγη μνήμη. Διαθέτει το εργαλείο npm που χρησιμεύει στην εγκατάσταση πακέτων και βιβλιοθηκών και του οποίου το περιεχόμενο διαρκώς αυξάνεται. Ακόμη είναι ιδανική για να χτίσει κάποιος επεκτάσιμες εφαρμογές. Ωστόσο οφείλουμε να αναφέρουμε πως δεν προτείνεται για εφαρμογές που έχουν βαρύ υπολογιστικό σύστημα. [16] [17] [18] [19] [20]

- **Express :** Το express είναι ένα πλαίσιο ανοιχτού κώδικα για διαδικτυακές εφαρμογές, ιδανικό για τη Node. Βοηθάει στον προγραμματισμό του back-end μιας εφαρμογής και είναι γνωστό ως το πλαίσιο των διακομιστών για τη Node. Παρέχει κάποιες επιπλέον λειτουργίες για τον προγραμματισμό εφαρμογών, χωρίς όμως να συγκαλύπτει τα όσα προσφέρει η Node. Δίνει επίσης τη δυνατότητα να χρησιμοποιηθούν ενδιάμεσα λογισμικά (middleware) ώστε να δημιουργηθεί ένα ισχυρό API.

Lavarel-PHP

- **PHP**: Η PHP είναι μία γλώσσα προγραμματισμού που χρησιμεύει στη δημιουργία διαδικτυακών σελίδων με δυναμικό περιεχόμενο. Είναι αρκετά εύχρηστη και κατανοητή και το χαρακτηριστικό της είναι ότι μπορεί να χρησιμοποιηθεί εμφωλευμένα μέσα στην HTML. Μία σελίδα γραμμένη σε PHP περνάει από επεξεργασία από έναν συμβατό διακομιστή, ώστε να παραχθεί είτε ένα αρχείο HTML για να γίνει το περιεχόμενο ορατό στο χρήστη, είτε να γίνει ανακατεύθυνση σε ένα άλλο PHP αρχείο για επεξεργασία δεδομένων.

- **Lavarel:** Το Laravel είναι ένα PHP framework που δημιουργήθηκε από τον Taylor Otwell και ακολουθεί το μοντέλο model-view-controller(MVC), που βοηθάει στο διαχωρισμό της λογικής και των επιπέδων παρουσίασης. Παρέχει το δικό της σύστημα

μετανάστευσης βάσεων δεδομένων, καθώς και υψηλή ασφάλεια με κωδικοποίηση σε πεδία, όπως είναι ο κωδικός πρόσβασης. Είναι χρήσιμο εργαλείο για τους PHP προγραμματιστές, ωστόσο είναι αρκετά αργό και απαιτεί άριστη χρήση της PHP. [21]

DJango-Python

- **Python:** Η Python είναι μία βασική προγραμματιστική γλώσσα υψηλού επιπέδου. Ανήκει στις αντικειμενοστραφείς γλώσσες και υποστηρίζει το μοντέλο της συλλογής σκουπιδιών(garbage collection) όντας μία δυναμική γλώσσα. Το συντακτικό της είναι αρκετά απλό καθώς ένας από τους στόχους της ήταν να πετυχαίνει με ελάχιστες γραμμές κώδικα λειτουργίες που στην Java και τη C++ γίνονται με πιο περίπλοκο τρόπο. Ωστόσο είναι πιο αργή από άλλες γλώσσες στο compile. Υποστηρίζει την ύπαρξη πολλών βιβλιοθηκών που διευκολύνουν πολλές από τις λειτουργίες που κάποιος θέλει να επιτύχει σε ένα πρόγραμμα.

- **Django:** Το Django είναι ένα ανοιχτό λογισμικό που είναι γραμμένο με Python. Κυκλοφόρησε το 2005. Έδωσε τη σκυτάλη στην Python να πρωταγωνιστήσει και αυτή σε εφαρμογές του διαδικτύου. Προτιμάται σαν πλαίσιο για μεγάλες εφαρμογές, διότι σε μικρές εφαρμογές εμπεριέχει αρκετά αρχεία χωρίς λόγο. Παρέχει στο προγραμματιστή διάφορες βιβλιοθήκες και frameworks, καθώς έχει σχεδιαστεί για να διευκολύνει τους προγραμματιστές αφήνοντάς τους να ασχοληθούν καθαρά με το κομμάτι της εφαρμογής τους. Επιπλέον παρέχει υψηλή ασφάλεια, ακόμη και σε ότι έχει να κάνει με την ταυτοποίηση του χρήστη. Είναι χρήσιμο για επεκτάσιμες εφαρμογές, όμως απαιτεί καλή χρήση της Python. Τέλος, αξίζει να αναφερθεί πως προτιμάται σαν πλαίσιο λόγω της καλής του συνδεσιμότητας πάνω από το CDN (content delivery network), που, όπως φαίνεται και από το όνομά του, πρόκειται για ένα δίκτυο αποτελούμενο από διακομιστές, οι οποίοι διαθέτουν περιεχόμενο όπως πολυμέσα, τα οποία η εκάστοτε εφαρμογή προβάλει στο χρήστη. Το θετικό είναι ότι οι διακομιστές αυτοί είναι γεωγραφικά τοποθετημένοι κοντά στον πελάτη και αυτό αυξάνει την ταχύτητα εξυπηρέτησής τους. [22]

SpringBoot-Java

- **Java:** Η Java είναι μια από της παλαιότερες γλώσσες προγραμματισμού υψηλού επιπέδου και υποστηρίζει τον αντικειμενοστραφή προγραμματισμό. Τα αρχεία της ορ-

γανώνονται με κλάσεις και αντικείμενα, μεθόδους και κατασκευαστές. Βοηθάει στην σχεδίαση ενός δομημένου κώδικα. Περιέχει τα λεγόμενα πακέτα που πρέπει να συμπεριλάβει στην αρχή του αρχείου, είναι οι αντίστοιχες βιβλιοθήκες της γλώσσας. Η χρησιμότητα της είναι ορατή σε πολλούς και διαφορετικούς τομείς του προγραμματισμού. Στη διάρκεια των χρόνων έχουν βγει πολλές νέες εκδόσεις της που περιέχουν βελτιστοποιήσεις συγκριτικά με την αρχική. Επίσης υποστηρίζει διάφορα frameworks όπως το Hibernate, το Maven, και το Apache Ant αλλά και πολλά άλλα web frameworks, εκ των οποίων το πιο γνωστό είναι το SpringBoot.

- **SpringBoot** Το SpringBoot είναι ένα λογισμικό ανοιχτού κώδικα, που έχει σχεδιαστεί βασισμένο στη Java. Οι λειτουργίες του μπορούν να χρησιμοποιηθούν από οποιαδήποτε εφαρμογή Java, αλλά υπάρχουν λειτουργίες για την κατασκευή διαδικτυακών εφαρμογών πάνω στην έκδοση της Java EE(Enterprise Edition). Βασικά του χαρακτηριστικά, που συγκαταλέγονται και στα θετικά του, είναι το ότι υποστηρίζει τον πολυνηματικό προγραμματισμό και το ότι εξασφαλίζει ύψιστη ασφάλεια. Ο πολυνηματικός προγραμματισμός είναι ιδανικός για εφαρμογές με μεγάλες υπολογιστικές εργασίες στο back end, όπως είναι η επεξεργασία βίντεο ή εικόνων. Το βασικό νόημα είναι ότι οι εργασίες μοιράζονται και γίνονται παράλληλα και έτσι μειώνεται ο υπολογιστικός χρόνος. Ένα επιπλέον θετικό είναι ότι, αφού βασίζεται στην Java, θα έχει μακροχρόνια υποστήριξη. Ωστόσο στα μειονεκτήματα της είναι ότι δεσμεύει αρκετή μνήμη.[23] [24] [25]

2.4 Full-stack

Ο όρος full-stack αναφέρεται στην ενασχόληση και με τη μεριά του πελάτη και με τη μεριά του server. Οι προγραμματιστές που ασχολούνται και με το front και με το back είναι περιζήτητοι καθώς έχουν την ικανότητα να καταλαβαίνουν πλήρως μία διαδικτυακή εφαρμογή και να προσδιορίζουν πιο στοχευμένα τις αλλαγές που πρέπει να γίνουν, ώστε η εφαρμογή να δουλεύει με το βέλτιστο δυνατό τρόπο. [26]

2.5 Η λογική Πελάτη Εξυπηρετητή και ο Παγκόσμιος Ιστός

Εφόσον αναλύθηκε σε βάθος τι είναι μια web εφαρμογή και με τι εργαλεία μπορεί να χτιστεί, για να κλείσει αυτό το κεφάλαιο, καλό είναι να αναφερθούν επιγραμματικά λίγα πράγματα για το διαδίκτυο και το μοντέλο πελάτη εξυπηρετητή, στο οποίο βασίζονται όλες οι εφαρμογές διαδικτύου.

2.5.1 Το Διαδίκτυο

Γνωστό και ως δίκτυο δικτύων, το οποίο χρησιμοποιεί το πρωτόκολλο TCP/IP για την ανταλλαγή μηνυμάτων.

2.5.2 Ο Παγκόσμιος Ιστός

Γνωστός και ως World Wide Web(www) αποτελεί ένα μοντέλο διαμοιρασμού πληροφορίας και είναι χτισμένο πάνω στο διαδίκτυο χρησιμοποιώντας το πρωτόκολλο HTTP. Το HTTP αποτελεί ένα πρωτόκολλο επικοινωνίας μεταξύ των πελατών και του εξυπηρετητή και χρησιμοποιείται ώστε να μπορούν να στέλνουν μεταξύ τους δεδομένα, πολυμέσα(για παράδειγμά εικόνες), κείμενο και άλλα.

2.5.3 Πελάτης-Εξυπηρετητής

Γνωστό ως client-server μοντέλο. Βοηθάει ώστε να διαχωριστεί η λογική εμφάνισης από τη λογική επεξεργασίας. Άρα είναι κατάλληλο για τις διαδικτυακές εφαρμογές. Το μοντέλο αυτό μπορεί να θεωρηθεί ως μια αρχιτεκτονική λογισμικού όπου ο πελάτης πάντα στέλνει ερωτήματα και ο εξυπηρετητής απαντά.

- **Εξυπηρετητής:** Ονομάζεται κάθε υπολογιστής, που τρέχει μια εφαρμογή διαδικτύου, που αντιστοιχεί στον εξυπηρετητή. Για να κάνουμε τη σύνδεση με όσα προαναφέρθηκαν οι εφαρμογές αυτές αποτελούν το back-end.

- **Πελάτης:** Είναι οι υπολογιστές που διαθέτουν πρόγραμμα περιήγησης.Αντιστοιχούν στο front-end κομμάτι της εφαρμογής.

Υπάρχουν 2 αρχιτεκτονικές πελάτη εξυπηρετητή.[27]

- **Two-Tier** Πρόκειται για την αρχιτεκτονική πελάτη εξυπηρετητή, που πραγματοποιείται σε 2 επίπεδα. Αυτή η εφαρμογή συνήθως αντιστοιχεί σε εφαρμογές εγκατεστημένες σε έναν σταθερό υπολογιστή, οι οποίες επικοινωνούν με έναν απομακρυσμένο εξυπηρετητή, που εξυπηρετεί κι άλλους πελάτες. Η λογική της εφαρμογής είναι συνυφασμένη με το γραφικό κομμάτι. Είναι εύκολο να στηθούν αυτές οι εφαρμογές, ωστόσο δεν παρέχουν τόσο μεγάλη ασφάλεια, καθώς η επικοινωνία με τη βάση γίνεται απευθείας από το χρήστη.

- **Three-Tier** Πρόκειται για την αρχιτεκτονική που πραγματοποιείται σε 3 επίπεδα και είναι επεκτάσιμη και σε παραπάνω από τρία. Τα επίπεδα είναι τα εξής : το επίπεδο του χρήστη, το επίπεδο της λογικής και το επίπεδο των δεδομένων(η βάση).Η λογική της εφαρμογής αντιστοιχεί στο μεσαίο επίπεδο. Οι εφαρμογές που χτίζονται με αυτή την αρχιτεκτονική είναι πιο δύσκολο να στηθούν, αλλά είναι επεκτάσιμες και παρέχουν πολύ μεγαλύτερη ασφάλεια, καθώς ο πελάτης δεν έρχεται ποτέ σε επαφή με τη βάση.

2.6 Συμπέρασμα

Αντιλαμβανόμαστε λοιπόν πως η επιλογή των εργαλείων τόσο για το front όσο και για το back δεν είναι εύκολη. Σε κάθε νέο project πρέπει να ορίζονται οι απαιτήσεις και η τελική επιλογή να γίνεται με βάση το δυναμικό της κάθε προγραμματιστικής ομάδας, αλλά και με βάση τις λειτουργικές απαιτήσεις του project. Δεν υπάρχει ένα μοναδικό εργαλείο που να είναι το σωστό. Το σημαντικό είναι η διερεύνηση των καινούριων εργαλείων, η καλή γνώση των παλιών και η προσαρμοστικότητα στις εκάστοτε συνθήκες.

Κεφάλαιο 3

Σχεδιασμός Λογισμικού

Η ανάπτυξη μίας διαδικτυακής εφαρμογής αντιμετωπίζεται ως ένα λογισμικό. Στο κεφάλαιο αυτό λοιπόν θα μιλήσουμε για το σχεδιασμό λογισμικού, τις μεθοδολογίες που υπάρχουν, καθώς και τις απαιτήσεις που πρέπει να οριστούν για το εκάστοτε λογισμικό.[28][29]

3.1 Τι είναι το λογισμικό

Στις μέρες μας ολοένα και περισσότερα συστήματα ελέγχονται από λογισμικά. Μερικά παραδείγματα, γνωστά στο ευρύ κοινό, είναι η ανθρώπινη επικοινωνία(μέσα κοινωνικής δικτύωσης όπως το facebook, το instagram), οι μεταφορές(taxibeat), το εμπόριο(e-shop), τα κρατικά έγγραφα(gov.gr και TAXISnet), η οικονομία (e-banking). Τι ορίζουμε όμως ως λογισμικό; Το λογισμικό αποτελείται από δομές δεδομένων, οι οποίες μπορούν να διαχειριστούν κάθε λογής δεδομένα και πληροφορίες, από προγράμματα ηλεκτρονικού υπολογιστή, τα οποία κατά την εκτέλεση τους φέρουν το επιθυμητό αποτέλεσμα, καθώς και από κείμενα, διαγράμματα και έγγραφα προδιαγραφών, τα οποία δίνουν τη συνολική εικόνα του λογισμικού και βοηθούν στην κατανόηση της χρήσης του. Ως λογισμικό λοιπόν δε νοείται μόνο ο πηγαίος κώδικας αλλά όλη η διαδικασία σχεδιασμού, αποτελούμενη από ένα δομημένο σύνολο δραστηριοτήτων όπως η σύλληψη της ιδέας, οι προδιαγραφές, οι απαιτήσεις, τα σχέδια και τα διαγράμματα, ο πηγαίος κώδικας, οι έλεγχοι και η συντήρησή του. Ένα λογισμικό μπορεί να σχεδιαστεί συγκεκριμένα για κάποιον πελάτη, ο οποίος έχει την αρχική ιδέα, είτε για έναν συγκεκριμένο τύπο χρηστών, των οποίων τις ανάγκες θέλουμε να εξυπηρετήσουμε. Σημαντικό προσόν στη σχεδιαστική σκέψη αποτελεί η παρατηρητικότητα. Παρατηρώντας τι κάνει ένας πιθανός χρήστης στην καθημερινότητά του μπορούμε να αντιληφθούμε καλύτερα

το πως θα χρησιμοποιήσει μια εφαρμογή ή τι απαιτήσεις μπορεί να έχει από αυτή. Μπαίνοντας λοιπόν στη θέση του χρήστη καταφέρνουμε να σχεδιάσουμε λογισμικά πιο κατανοητά στο χρήστη και πιο εύχρηστα.

3.2 Μεθοδολογίες ανάπτυξης λογισμικού

Οι μεθοδολογίες ανάπτυξης λογισμικού ή αλλιώς μοντέλα κύκλου ζωής του λογισμικού αναφέρονται και καθορίζονται από τις φάσεις της οποίες διέρχεται το λογισμικό από τη σύλληψη μέχρι την απόσυρσή του, καθώς και τις ενέργειες που λαμβάνουν χώρα σε καθεμιά από αυτές τις φάσεις. Τα μοντέλα ανάπτυξης χωρίζονται σε 2 βασικές κατηγορίες, τα ακολουθιακά και τα επαναληπτικά.[30] [31] [32]

- **Ακολουθιακά μοντέλα :** Η ανάπτυξη γίνεται σε διαδοχικές διακριτές φάσεις και για ολόκληρο το σύστημα του λογισμικού. Είναι λιγότερο ευέλικτη, καθώς κάθε βήμα απαιτεί την ορθότητα και την ολοκλήρωση του προηγούμενου.

- **Επαναληπτικά μοντέλα :** Η ανάπτυξη γίνεται σε τμήματα και με επαναληπτικό τρόπο. Κάθε επανάληψη φέρνει κάτι νέο και διορθώνει λάθη.

Οι διαδικασίες ανάπτυξης λογισμικού, που μπορεί να ακολουθεί ένα μοντέλο, ταξινομούνται ως ακολούθως :

- **Προδιαγραφή :** Δηλαδή καθορισμός των ενεργειών που θα εκτελεί το λογισμικό, καθώς και των περιορισμών και των προδιαγραφών που ισχύουν.

- **Ανάπτυξη :** Η κατασκευή του λογισμικού. Εδώ σε όλα τα μοντέλα μπορούμε να διακρίνουμε τρεις επιμέρους φάσεις: την ανάλυση, τη σχεδίαση και τη συγγραφή του πηγαίου κώδικα, η οποία συχνά ονομάζεται και κωδικοποίηση.

- **Επαλήθευση :** Δηλαδή η επιβεβαίωση της ικανοποίησης των προδιαγραφών και ο έλεγχος μη ύπαρξης σφαλμάτων. Είναι ένα απαραίτητο βήμα πριν την παράδοση του λογισμικού στον πελάτη.

- **Εξέλιξη :** Η τροποποίση υπαρχουσών λειτουργιών προκειμένου να εξυπηρετεί καλύτερα τον πελάτη, αλλά και η επαύξηση των λειτουργιών της εφαρμογής.

Τα διάφορα μοντέλα κύκλου ζωής, αν και βασίζονται πάνω στις προαναφερόμενες διαδικασίες, διαφοροποιούνται μεταξύ τους, ανάλογα με την εμβέλεια, δηλαδή την έκταση του υπό κατασκευή συστήματος λογισμικού, στην οποία αυτές οι διαδικασίες εφαρμόζονται, την επαναληπτικότητα των εργασιών, καθώς και τις ενδιάμεσες αποτιμήσεις του λογισμικού από τον πελάτη ή τον κατασκευαστή. Για να επιλεχτεί η κατάλληλη μεθοδολογία, ο σχεδιαστής οφείλει να γνωρίζει τις υπάρχουσες, καθώς και τα θετικά και τα αρνητικά που προκύπτουν από αυτές. Με την πάροδο των χρόνων όλο και περισσότερες μεθοδολογίες αναπτύσσονται και μάλιστα πολλές από αυτές αποτελούν συνδυασμό των ήδη προϋπάρχουσων. Στην τρέχουσα διπλωματική θα μελετήσουμε τις πιο γνωστές από αυτές.

3.2.1 Το μοντέλο του καταρράκτη

Ένα από τα πιο παλιά μοντέλα. Η κεντρική του ιδέα βασίζεται στο ότι το σύστημα λογισμικού αναπτύσσεται περνώντας ολόκληρο από διαδοχικές επιμέρους φάσεις, καθεμία από τις οποίες θεωρείται ολοκληρωμένη όταν παραχθούν ορισμένα συστατικά λογισμικού. Κάθε φάση περατώνεται με την επαλήθευση και τον έλεγχο του προϊόντος που παράχθηκε. Μόνο όταν είναι σίγουρο ότι δεν υπάρχουν λάθη προχωράμε στην επόμενη φάση. Προφανώς ανήκει στα ακολουθιακά μοντέλα ανάπτυξης. Το όνομα του προέρχεται από τους καταρράκτες οι οποίοι ρέουν προς μία κατεύθυνση. Έτσι ακριβώς συμβαίνει και με αυτό το μοντέλο. Με το που εγκριθεί μια λειτουργία και ολοκληρωθεί το στάδιο της δεν υπάρχει πισωγύρισμα. Προφανώς το μοντέλο αυτό δεν είναι κατάλληλο για λογισμικά των οποίων οι απαιτήσεις αλλάζουν συχνά . Ωστόσο η γραμμικότητα του το κάνει αρκετά εύκολο στην κατανόηση του. Στα αρνητικά του συγκαταλέγονται τα εξής: δε δέχεται τα σχόλια του πελάτη στις αρχικές φάσεις, με μεγάλο κίνδυνο να βγει έξω από τις αρχικές προδιαγραφές, ο έλεγχος γίνεται στο τελικό στάδιο, γεγονός που δυσκολεύει αρκετά τη διόρθωση λαθών και τέλος η μη αποδοχή αλλαγών δυσκολεύει πολύ την κατάσταση ειδικά σε περίπλοκα πρότζεκτ. Επομένως είναι κατάλληλο για πρότζεκτ με ξεκάθαρες απαιτήσεις.

3.2.2 Το μοντέλο της πρωτοτυποποίησης

Γνωστό και ως prototype model. Η ανάπτυξη του λογισμικού δε γίνεται εξ' ολοκλήρου αλλά σε τμήματα που ονομάζονται "πρωτότυπα". Οι διαδικασίες ανάπτυξης επαναλαμβάνονται για ένα τμήμα του συστήματος κάθε φορά και για το λόγο αυτό ανήκει στα επαναληπτικά μοντέλα. Αυτό το μοντέλο προσφέρει εικόνα για τη λειτουργία της εφαρμογής πολύ νωρί-

τερα από ότι το μοντέλο του καταρράκτη. Ο τρόπος λειτουργίας του αποφεύγει την ολική αποτυχία του λογισμικού. Επίσης στην αλληλεπίδραση με τον πελάτη συλλέγει σχόλια ώστε να βελτιωθεί το τελικό προϊόν. Ωστόσο η κατασκευή του πρωτότυπου, πάνω στο οποίο θα γίνουν οι δοκιμές και αλλαγές, έχει και το ρίσκο των χρονοβόρων προθεσμιών, καθώς μπορεί οι απαιτήσεις να αλλάξουν πολλές φορές. Επίσης το κόστος κατασκευής του πρωτοτύπου χρεώνεται στον προγραμματιστή, συνήθως με κίνδυνο να χρειαστεί και πολλαπλές αλλαγές μέχρι να καταλήξουν στο τελικό προϊόν. Είναι λοιπόν κατάλληλο για κατασκευή λογισμικού με όχι ολοκληρωμένες απαιτήσεις από της αρχή.

3.2.3 Το μοντέλο λειτουργικής επαύξησης

Το μοντέλο αυτό αποτελεί ένα συνδυασμό της ακολουθιακής ανάπτυξης του μοντέλου του καταρράκτη και της επαναληπτικής ανάπτυξης του μοντέλου της πρωτοτυποποίησης. Κεντρική ιδέα είναι ο χωρισμός του συστήματος σε ξεχωριστά τμήματα τα οποία αναπτύσσονται ανεξάρτητα ακολουθώντας την ακολουθιακή λογική. Με τον τρόπο αυτό επιτυγχάνεται μικρότερος χρόνος κατασκευής του συνολικού λογισμικού. Ωστόσο η βαρύτητα πέφτει στην αρχική κατάτμηση και σχεδίαση του λογισμικού και οι αλλαγές στις απαιτήσεις είναι ικανές να κλονίσουν ολόκληρο το λογισμικό.

3.2.4 Το σπειροειδές μοντέλο

Γνωστό και ως spiral. Προτάθηκε ως μια γενίκευση των μοντέλων της πρωτοτυποποίησης και της λειτουργικής επαύξησης, έχοντας όμως και νέα στοιχεία. Πιο συγκεκριμένα στο συγκεκριμένο μοντέλο δεν υπάρχουν προκαθορισμένες φάσεις ανάπτυξης, αλλά εξειδικεύονται στο χώρο της εφαρμογής. Η ανάπτυξη του συστήματος χωρίζεται σε πολλούς κύκλους, όπου στον καθένα προστίθενται και νέες λειτουργίες στο σύστημα. Πριν την έναρξη κάθε κύκλου μελετάται η σκοπιμότητα μιας νέας προσθήκης και το πως αυτή θα επηρεάσει το λογισμικό. Σκοπός του μοντέλου είναι η σταδιακή βελτίωση του προϊόντος.

3.2.5 Η ευέλικτη μεθοδολογία ανάπτυξης- Agile

Η δημοσιότητά της έχει ανέβει τα τελευταία χρόνια. Η προσέγγισή της είναι διαφορετική από τη συμβατική γραμμική μέθοδο και δίνει προτεραιότητα στην ικανοποίηση των χρηστών αντί να τεκμηριώνει και να παγιοποιεί κάποιο από τα βήματα και τις απαιτήσεις. Με την Agi-

le οι διαδικασίες χωρίζονται σε μέρη που χρειάζονται από μία έως τέσσερις βδομάδες για να ολοκληρωθούν. Πρόκειται για ένα επαναληπτικό μοντέλο όπου οι προγραμματιστές αναζητούν συνεχώς σχόλια από τους πελάτες. Τα θετικά που προκύπτουν από αυτή τη μεθοδολογία είναι ότι το λογισμικό έχει ελάχιστα λάθη. Επίσης ο πελάτης είναι δύσκολο να μείνει ανικανοποίητος καθώς θα βρίσκεται σε συχνή επαφή με τους προγραμματιστές. Τέλος, υπάρχει σαφήνεια σε κάθε βήμα. Από την άλλη δεν μπορεί να υπάρξει ένα ολοκληρωμένο έγγραφο προδιαγραφών από την αρχή και αυτό μπορεί να δημιουργήσει πρόβλημα, αν οι προγραμματιστές που απαρτίζουν την ομάδα δεν είναι έμπειροι και έτοιμοι να προσαρμοστούν στις μεταβλητές προδιαγραφές του πρότζεκτ. Είναι αρκετά χρήσιμη για λογισμικά που θέλουν να γίνονται πιο χρηστικά, ανάλογα με τις εκάστοτε απαιτήσεις στην αγορά.

Εκτός από τις προαναφερόμενες υπάρχουν και άλλες μεθοδολογίες, οι οποίες έχουν μικρές διαφορές από τις ήδη αναφερόμενες. Ονομαστικά αυτές είναι οι Dev Ops, Rapid application, Dynamic systems development, Extreme programming, Feature-driven development και άλλες. Μελετώντας αναλυτικότερα το τι πρεσβεύει η καθεμιά, μπορεί κάποιος να καταλήξει πως οι διαφοροποιήσεις σε πολλές από αυτές δεν είναι μεγάλες, καθώς πολλές μεθοδολογίες εμπνεύστηκαν και βασίστηκαν σε ήδη υπάρχουσες. Η επιλογή της ορθής μεθοδολογίας είναι στο χέρι του προγραμματιστή, ο οποίος αξιολογεί το έργο που έχει να φέρει εις πέρας. Δεν είναι απαγορευτικό μάλιστα κάποιος να δοκιμάσει ένα συνδυασμό μεθοδολογιών.

3.3 Διάταξη λογισμικού

Η διάταξη του λογισμικού ασχολείται με την κατάτμηση μιας εφαρμογής σε ανεξάρτητα λειτουργικά συστήματα και η ανάθεση αυτών σε περιορισμένους λειτουργικούς πόρους. Πιο συγκεκριμένα αποφασίζονται οι εργασίες που θα εκτελεί το κάθε σύστημα. Οι εργασίες αυτές χωρίζονται σε τρεις κατηγορίες. Στις εργασίες παρουσίασης, που έχουν σχέση με την επικοινωνία του συστήματος με το χρήστη. Στις εργασίες διαχείρισης δεδομένων, που έχουν να κάνουν με την ανάκτηση και αποθήκευση των δεδομένων της εφαρμογής. Και στις εργασίες επιχειρησιακής λογικής, που είναι όλες αυτές που υλοποιούν τις απαιτήσεις της εφαρμογής. Σε αυτό το σημείο γίνεται και η σύνδεση με το προηγούμενο κεφάλαιο και γίνεται κατανοητό γιατί η ανάπτυξη μιας διαδικτυακής εφαρμογής αποτελεί μέρος της ανάπτυξης λογισμικού και γιατί πρέπει να αντιμετωπιστεί ως λογισμικό.

3.4 Απαιτήσεις λογισμικού

Οι απαιτήσεις ορίζονται στο πλαίσιο του συστήματος του λογισμικού. Ως σύστημα δε θεωρείται μόνο το λογισμικό, αλλά και όσοι αλληλεπιδρούν με το σύστημα, είτε αυτοί είναι άνθρωποι είτε είναι μηχανές. Ο προσδιορισμός των απαιτήσεων αποτελεί μία χρονοβόρα, αλλά συνάμα και μία πολύ κρίσιμη διαδικασία. Πρόκειται για τον προσδιορισμό όλων των εργασιών που θα εκτελεί το σύστημα, την αλληλεπίδραση που θα έχει με το χρήστη, την εμφάνιση, αλλά και την επίδοση του. Οι απαιτήσεις του λογισμικού χωρίζονται σε 2 μεγάλες κατηγορίες: στις λειτουργικές και τις μη λειτουργικές απαιτήσεις.

- **Λειτουργικές Απαιτήσεις:** Πρόκειται για τις λειτουργίες και τους υπολογισμούς που θα εκτελεί το λογισμικό. Συνδέονται άρρικτα με το λογικό κομμάτι της εφαρμογής και έχουν τη δυνατότητα να καθορίζουν πλήρως τη συμπεριφορά του συστήματος.Πρέπει να είναι αρκετά σαφείς και είναι εύκολο να ελεγθεί και να επιβεβαιωθεί η επίτευξη τους.

- **Μη Λειτουργικές Απαιτήσεις:** Περιγράφουν τα χαρακτηριστικά που θα έχει το λογισμικό τα οποία δε συνδέονται με την εκτέλεση των λειτουργιών, αλλά έχουν να κάνουν με την εμφάνιση, την επίδοση, αλλά και με τις συνθήκες που μπορεί να υποστεί κάποιο λογισμικό. Είναι πιο γενικές και αόριστες και πολλές φορές είναι αρκετά δύσκολο να επιβεβαιωθεί ότι το σύστημα τις τηρεί ή υπάρχει περίπτωση η επιβεβαίωσή τους να πάρει χρόνο. Μάλιστα χωρίζονται στις εξής υποκατηγορίες:

  – Απαιτήσεις χρήσης: Έχουν να κάνουν με τα χαρακτηριστικά εμφάνισης του λογισμικού, δίνουν δηλαδή γενικές οδηγίες για το user interface. Επιπρόσθετα ορίζουν πως ο χρήστης θα αλληλεπιδρά με το σύστημα, για παράδειγμα με τη χρήση του πληκτρολογίου και του ποντικιού.

  – Απαιτήσεις αξιοπιστίας: Ορίζουν ποια πρέπει να είναι η συμπεριφορά του λογισμικού σε περίπτωση που προκύψουν ενδογενή ή εξωγενή σφάλματα καθώς και ποια είναι η ανοχή του σε αυτά. Ένα παράδειγμα σφάλματος είναι ο απρόσμενος τερματισμός και πως το σύστημα θα δράσει ώστε να υπάρξουν όσο το δυνατό λιγότερες απώλειες.

  – Απαιτήσεις επίδοσης: Έχουν να κάνουν με το χρόνο εκτέλεσης των λειτουργιών του συστήματος, το χρόνο που απαιτούν τα ερωτήματα προς τη βάση για να ολο-

κληρωθούν, τη μνήμη που δεσμεύει το σύστημα, αλλά και τις μονάδες επεξεργασίας.

– Απαιτήσεις σχεδίασης: Ορίζει με ποια μεθοδολογία θα γίνει η σχεδίαση του λογισμικού.

– Απαιτήσεις υλοποίησης: Καθορίζει τις γλώσσες προγραμματισμού που απαιτούνται και πρέπει να χρησιμοποιηθούν για την κωδικοποίηση του συστήματος.

– Απαιτήσεις βάσεων: Πρόκειται για τα δεδομένα που θα πρέπει να κρατάει το σύστημα αποθηκευμένα, στο επίπεδο που αυτά μπορούν να οριστούν, καθώς και τις συσχετίσεις μεταξύ τους.

– Φυσικές απαιτήσεις: Αποτελούνται από τις απαιτήσεις ενός συστήματος σε φυσικό επίπεδο, για παράδειγμα η συσκευή που θα πρέπει να χρησιμοποιηθεί ή το λειτουργικό σύστημα ή το αν θα απαιτείται σύνδεση στο διαδίκτυο.

Εφόσον αναλύθηκε το είδος των απαιτήσεων, που μπορεί να οριστούν για ένα σύστημα, ήρθε η ώρα να αναλύσουμε τη διαδικασία που αυτές καθορίζονται και αποφασίζονται. Είναι επιτακτική ανάγκη, πριν σχεδιαστεί το εκάστοτε λογισμικό να οριστεί το πρόβλημα που θέλουμε να επιλύσουμε. Μόλις το πρόβλημα εντοπιστεί, πρέπει να αναλυθεί από που προέρχεται, καθώς και το πόσο η επίλυση του με τη χρήση του λογισμικού θα συνεισφέρει και θα αλλάξει την τρέχουσα κατάσταση. Επίσης σημαντικό είναι ο κατασκευαστής να σκεφτεί την επίλυση, τόσο από τη μεριά του πελάτη, όσο και από τη μεριά του προγραμματιστή, ώστε να καταφέρει να ορίσει σωστά τις απαιτήσεις. Η βαθιά κατανόηση του προβλήματος είναι το κλειδί για το σχεδιασμό των σωστών απαιτήσεων και στη συνέχεια για την κατασκευή ενός πετυχημένου λογισμικού. Με το που συγκεντρωθούν οι απαιτήσεις του λογισμικού πρέπει να επανεξεταστούν, να αναλυθεί η χρησιμότητα τους και να εντοπιστούν πιθανές ασάφειες και συγκρούσεις. Εφόσον ολοκληρωθούν και αυτοί οι έλεγχοι, οι απαιτήσεις καταγράφονται στο λεγόμενο έγγραφο προδιαγραφών, το οποίο θα ακολουθήσουν οι προγραμματιστές για να καταλήξουν στην κατασκευή του. Στην καταγραφή των απαιτήσεων βοηθούν τα εξής τρία διαγράμματα.

- **Διαγράμματα ροής δεδομένων:** Πρόκειται για τα διαγράμματα που αποτελούν μια πρώτη περιγραφή των απαιτήσεων του πελάτη. Βασίζεται στον τρόπο που ρέουν τα δεδομένα ανάμεσα στις μονάδες λογισμικού. Κάθε μονάδα λογισμικού δέχεται τα δεδομένα σε μία μορφή και έπειτα από επεξεργασία παράγει μια έξοδο. Στα διαγράμ-

ματα ροής δεν αναλύονται όλες οι ενέργειες που γίνονται κατά την επεξεργασία, καθώς δε δίνεται και σημασία στη χρονική σειρά εκτέλεσης. Απλά αποτυπώνεται το πως τα δεδομένα μπορούν να κινηθούν στην εφαρμογή. Τα διαγράμματα αυτά είναι αρκετά κατανοητά λόγο της απλότητας τους. Επίσης επιδέχονται εύκολα αλλαγές άμα αυτό χρειαστεί. Ωστόσο αποτελούν καθοριστικό βήμα στον ορισμό των απαιτήσεων.

- **Διαγράμματα οντοτήτων-συσχετίσεων:** Πρόκειται για τα διαγράμματα που επεξηγούν τις οντότητες δεδομένων με τα πεδία που αυτές περιέχουν, καθώς και τις συσχετίσεις μεταξύ τους. Χρησιμοποιούνται πολλές φορές σε συνδυασμό με το λεξικό δεδομένων το οποίο αναλύει πως ονομάζονται τα δεδομένα, σε ποιες οντότητες χρησιμοποιούνται, ποιος είναι ο τύπος αποθήκευσης τους, καθώς και το ποιες τιμές μπορούν να λάβουν.

- **Διαγράμματα μετάβασης καταστάσεων:**Πρόκειται για το διάγραμμα που αποτυπώνει τη δυναμική συμπεριφορά του λογισμικού. Σε αντίθεση με το διάγραμμα ροής δεδομένων δίνει περισσότερη σημασία στις διακλαδώσεις που μπορεί να αποκτήσει το πρόγραμμα, ανάλογα με τις επιλογές του χρήστη. Τα διαγράμματα αυτά βασίζονται σε γεγονότα, αποκρίσεις και καταστάσεις. Πιο συγκεκριμένα ένα γεγονός θεωρείται, είτε μια επιλογή που μπορεί να κάνει ο χρήστης, είτε ένα αποτέλεσμα που μπορεί να προκύψει από άλλο πρόγραμμα. Ως απόκριση ορίζεται μια λειτουργία που εκτελεί το λογισμικό μετά από ένα γεγονός. Τέλος, η κατάσταση είναι ο χρόνος που το λογισμικό περιμένει για κάποιο γεγονός. Μέσω των γεγονότων και τον αποκρίσεων το λογισμικό μεταβαίνει σε διαφορετικές καταστάσεις. Η συνθήκη αυτή της δυναμικής περιγραφής είναι αρκετά δύσκολη να αποτυπωθεί σωστά και ολοκληρωμένα . Σχεδόν ποτέ δεν αποτυπώνονται όλα τα πιθανά σενάρια αλλά δίνεται βάση στα πιο σημαντικά.

3.5 Κωδικοποίηση

Τη διαδικασία του ορισμού των απαιτήσεων ακολουθεί η κωδικοποίηση. Μόλις προκύψει το έγγραφο προδιαγραφών, ξεκινάει ο σχεδιασμός κώδικα. Αρκετές φορές για να αποφευχθούν λάθη ή δυσκολίες στη συνεννόηση, όταν πρόκειται για ομάδα προγραμματιστών, σχεδιάζεται ο ψευδοκώδικας. Ρόλος του είναι η αποτύπωση των βασικών συναρτήσεων και αλγορίθμων του προγράμματος. Πριν ξεκινήσει η εγγραφή κώδικα των λειτουργικών απαιτήσεων, σωστό είναι να οριστούν οι βάσεις που θα χρησιμοποιηθούν και τα σχήματα αυτών,

αυτή τη φορά με όλες τις λεπτομέρειες. Έπειτα ξεκινάει το κομμάτι του προγραμματισμού. Εφόσον ολοκληρωθεί η κωδικοποίηση ακολουθεί ο έλεγχος των απαιτήσεων και η παράδοση του προϊόντος.

Κεφάλαιο 4

Η εφαρμογή Fixed Asset Management

Έχοντας αποκτήσει το θεωρητικό υπόβαθρο για τη σχεδίαση ενός λογισμικού και για την ανάπτυξη μιας διαδικτυακής εφαρμογής, έφτασε η στιγμή να ορίσουμε το πρόβλημα που θα επιλυθεί με την ανάπτυξη λογισμικού σε αυτή τη διπλωματική.

4.1 Ορισμός του προβλήματος και τρέχουσα ανάγκη

Είναι γεγονός πως στις μέρες μας ολοένα και περισσότερες λειτουργίες, που γίνονταν με το χέρι, ψηφιοποιούνται. Οτιδήποτε γραφειοκρατικό μετατρέπεται σε ψηφιακά δεδομένα. Ό,τι χρειαζόταν χαρτί για να σημειωθεί, σημειώνεται πλέον ψηφιακά με τη βοήθεια εφαρμογών ή εγγράφων. Οι υπολογισμοί γίνονται από τα συστήματα. Στους τρέχοντες λοιπόν ρυθμούς της κοινωνίας απομονώνουμε την οικονομική-περιουσιακή πτυχή της ζωής του ανθρώπου. Η ανάγκη αυτή προκύπτει από τις σύγχρονες συνθήκες, με τα έξοδα που διαρκώς αυξάνονται και τους ρυθμούς ζωής που διαρκώς γίνονται πιο εντατικοί και γρήγοροι, αλλά και από την ανάγκη ύπαρξης ελέγχου σχετικά με τα πάγια περιουσιακά στοιχεία. Το πρόβλημα προσεγγίζεται από δύο μεριές. Αρχικά θα επικεντρωθούμε στην έλλειψη διαχειριστικών συστημάτων σχετικά με τα πάγια περιουσιακά στοιχεία. Πιο συγκεκριμένα πολλές δημόσιες και ιδιωτικές επιχειρήσεις, που έχουν στην κατοχή τους αυτού του είδους τα περιουσιακά στοιχεία, παρουσιάζουν μια αδυναμία ελέγχου σχετικά με το πόσα είναι αυτά που έχουν στην κατοχή τους, πού βρίσκονται τοποθετημένα αυτή τη στιγμή, ποια ήταν η τιμή αγοράς των αποκτημάτων αυτών ή ακόμη και σε ποιους χρεώθηκαν. Η έλλειψη καταγραφής των περιουσιακών στοιχείων, ιδιαίτερα σε μεγάλες επιχειρήσεις ή ιδρύματα, ακολουθείται πολλές φορές από ιδιοποίηση των στοιχείων από μέλη των επιχειρήσεων και από μπερδέματα σε

καταστάσεις μετακομίσεων ή μετακινήσεων. Για παράδειγμα μία εταιρεία που μετακινείται σε ένα καινούριο κτίριο μπορεί αν δεν έχει καταγεγραμμένα όλα τα στοιχεία της, να αντιμετωπίσει προβλήματα με το σε ποιον ανήκε το κάθε γραφείο ή η κάθε καρέκλα, πόσα ήταν τα γραφεία ή οι υπολογιστές αρχικά και πολλά άλλα. Το πρόβλημα αυτό μπορεί να αντικατοπτριστεί και σε πιο οικεία περιβάλλοντα, όπως ένα νοικοκυριό ενός σπιτιού. Η καταγραφή των επίπλων, των ηλεκτρικών ή και των ηλεκτρονικών συσκευών, η ημερομηνία αγοράς, και η τιμή αγοράς, ακόμα και αναλυτικές περιγραφές, μπορούν να βοηθήσουν τον ιδιοκτήτη να έχει οργανωμένες σε ένα μέρος τις πληροφορίες των υπαρχόντων του, έτσι ώστε αν ποτέ χρειαστεί να τα πουλήσει, να κατέχει όλα τα δεδομένα που χρειάζεται συγκεντρωμένα. Ας σκεφτούμε μόνο πόσα χαρτιά θα χρειαζόταν κάποιος για να κρατάει όλες αυτές τις πληροφορίες. Η άλλη πτυχή του προβλήματος είναι το καθαρά οικονομικό κομμάτι των εσόδων και των εξόδων. Η ύπαρξη μιας εφαρμογής, όπου ο χρήστης θα μπορεί να οπτικοποιεί και να καταγράφει τις δαπάνες του, θα τον βοηθήσει να κάνει καλύτερη διαχείριση των χρημάτων του. Συνήθως οι άνθρωποι σημειώνουν χονδρικά τα ποσά που ξοδεύουν μέσα στο μήνα, έτσι όμως οι περισσότεροι δεν έχουν μια ξεκάθαρη εικόνα για το που τα ξοδεύουν. Επίσης, χρήσιμο θα ήταν να μπορεί μια ομάδα ανθρώπων, που για οποιονδήποτε λόγο διαχειρίζονται έναν κοινό λογαριασμό, ένα κοινό ταμείο, να μπορούν να γνωρίζουν τις κινήσεις που γίνονται στο λογαριασμό αυτό. Υπάρχει λοιπόν ανάγκη για ένα διαχειριστικό λογισμικό, που θα απευθύνεται στα έξοδα και τα έσοδα του χρήστη, καθώς και στη διαχείριση των πάγιων περιουσιακών του στοιχείων. Πιο αναλυτικά, ο χρήστης θα μπορεί να διαχειριστεί τα μηνιαία έσοδα και έξοδα του ανά πάσα στιγμή, κάτι που χωρίς εφαρμογή θα αναγκαζόταν να το κάνει στο χαρτί ή πολύ απλά να μην τα σημειώνει αναλυτικά, με αποτέλεσμα να μην έχει μια πλήρη εικόνα της κατάστασης και του πως μπορεί να τη βελτιώσει. Επίσης, στην εφαρμογή θα υπάρχουν καταγεγραμμένα τα πάγια περιουσιακά του στοιχεία και θα δίνεται η δυνατότητα έκδοσης κάποιου κωδικού QR, ώστε ανά πάσα στιγμή να μπορούν να εντοπιστούν οι πληροφορίες τους.

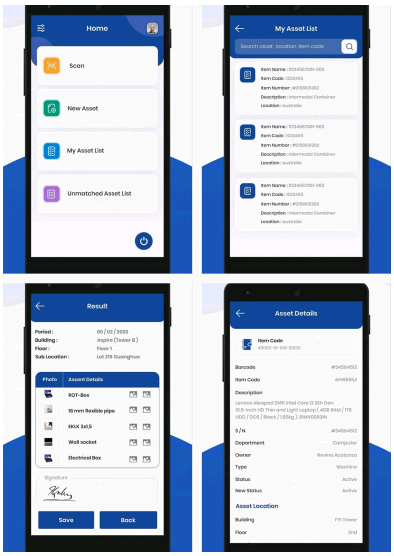
4.2 Υπάρχουσες εφαρμογές

Πριν την κατασκευή του λογισμικού και πριν το σχεδιασμό των απαιτήσεων είναι αρκετά σημαντικό να ερευνηθεί αν υπάρχουν ήδη εφαρμογές που κάνουν κάτι αντίστοιχο ή έστω που επιλύουν κάποια από τα προβλήματα. Η ύπαρξη εφαρμογών παρόμοιων με το

επιθυμητό λογισμικό δεν απαγορεύουν την ανάπτυξή του. Αντιθέτως, δημιουργούν και ένα πλαίσιο συναγωνισμού για το ποιο λογισμικό είναι πιο εύχρηστο, ελκυστικό και ποιό έχει τις καλύτερες και χρησιμότερες λειτουργίες. Κατά την έρευνα αυτή εντοπίστηκαν εφαρμογές, που είτε είναι σχεδιασμένες για την αποθήκευση δεδομένων πάγιων περιουσιακών στοιχείων, είτε διαχειρίζονται τα έξοδα του χρήστη. Ωστόσο δεν εντοπίστηκε κάποια εφαρμογή που να επιλύει και τα δύο προβλήματα ταυτόχρονα. Επίσης οι περισσότερες από αυτές τις εφαρμογές είναι σχεδιασμένες ως εφαρμογές κινητού τηλεφώνου, δηλαδή ο χρήστης θα πρέπει να εγκαταστήσει την εφαρμογή στο κινητό του για να τη χρησιμοποιήσει, καθώς και δεν έχουν όλες τη δυνατότητα ομαδικής διαχείρισης των δεδομένων. Παρακάτω επιλέχθηκαν μερικές από αυτές για να αναφερθούν.

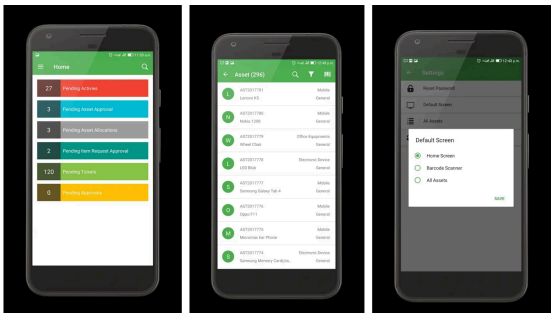
Εφαρμογές σχετικές με τη διαχείριση περιουσιακών στοιχείων:

WMSOne-Fixed Asset Management: Πρόκειται για το παλαιότερο και ευρέως χρησιμοποιηούμενο λογισμικό διαχείρισης αποθηκών και διαχείρισης πάγιων περιουσιακών στοιχείων της Ταιλάνδης. Προσφέρει τη δυνατότητα προσθήκης των διάφορων στοιχείων και την παραγωγή ενός barcode. Επίσης διαθέτει και δικό της scanner.



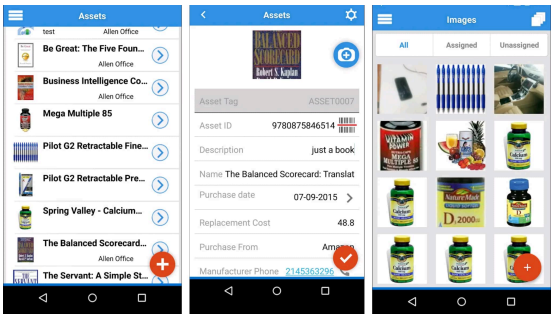
Σχήμα 4.1: Στιγμιότυπα εκτέλεσης της εφαρμογής WMSOne

Infinity Asset- Asset tracking and maintenance app: Αφορά την καταγραφή και παρακολούθηση των περιουσιακών στοιχείων. Η επιπλέον προσθήκη της είναι πως περιέχει και πληροφορίες σχετικά με τη συντήρηση των στοιχείων.



Σχήμα 4.2: Στιγμιότυπα εκτέλεσης της εφαρμογής Infinity Asset

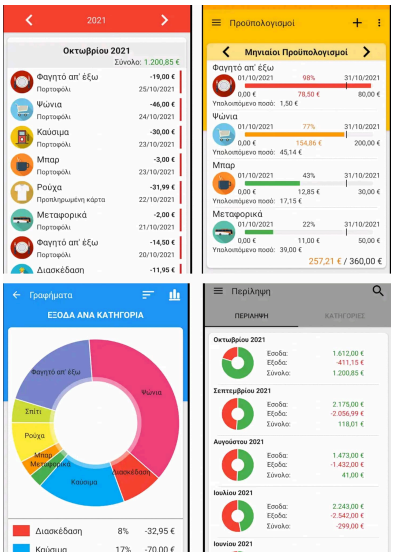
Asset panda: Πρόκειται για μία εφαρμογή που έχει σκοπό να αντικαταστίσει τα λογιστικά φύλλα για την παρακολούθηση περιουσιακών στοιχείων. Δίνει τη δυνατότητα αποθήκευσης πληροφοριών σχετικά με το κάθε περιουσιακό στοιχεία και παρέχει και έναν κωδικό barcode.



Σχήμα 4.3: Στιγμιότυπα εκτέλεσης της εφαρμογής Asset Panda

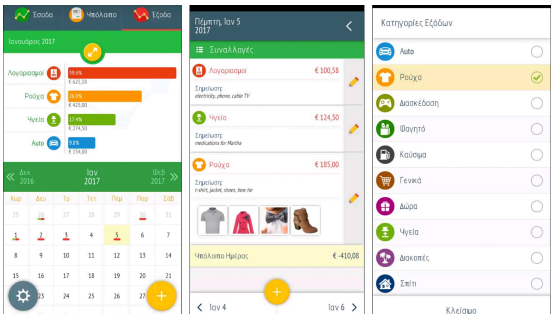
Εφαρμογές σχετικές με τη διαχείριση εξόδων:

Γρήγορος προϋπολογισμός: Διαθέτει έλεγχο των ημερήσιων εσόδων και εξόδων. Παρουσιάζει γραφήματα σχετικά με τις δαπάνες. Επίσης δίνει τη δυνατότητα συγχρονισμού των τραπεζικών λογαριασμών.



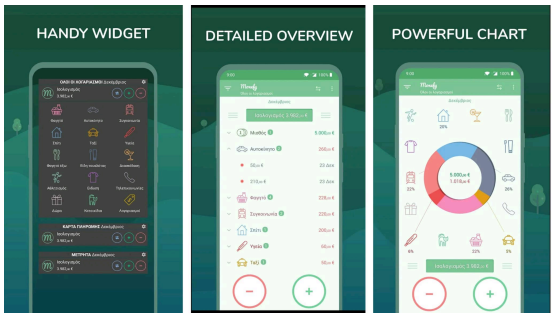
Σχήμα 4.4: Στιγμιότυπα εκτέλεσης της εφαρμογής Γρήγορος προϋπολογισμός

Διαχείριση Εξόδων: Δίνει τη δυνατότητα δημιουργίας κατηγοριών που ο χρήστης επιθυμεί. Διαθέτει ημερολόγιο για καταγραφή των εξόδων καθώς και υπολογιστική μηχανή για γρήγορους υπολογισμούς.



Σχήμα 4.5: Στιγμιότυπα εκτέλεσης της εφαρμογής Διαχείριση εξόδων

Monefy -Money Manager: Σε παρόμοιο κλίμα με τις υπόλοιπες ζητάει από το χρήστη την καταγραφή των καθημερινών του εξόδων και του προσφέρει μια οπτικοποίηση με βάση τις κατηγορίες εξόδων.



Σχήμα 4.6: Στιγμιότυπα εκτέλεσης της εφαρμογής Monefy

Παρατηρείται λοιπόν πως πολλοί έχουν προσπαθήσει να επιλύσουν το υπάρχον πρόβλημα, γεγονός που το καθιστά και επιτακτική ανάγκη. Παρατηρείται επίσης ότι οι περισσότερες εφαρμογές που έχουν να κάνουν με περιουσιακά στοιχεία είναι επι πληρωμή, ενώ αυτές που έχουν να κάνουν με έξοδα είναι δωρεάν. Ωστόσο καμία δε συνδυάζει και τις δύο λειτουργίες.

4.3 Απαιτήσεις εφαρμογής

4.3.1 Σύντομος ορισμός απαιτήσεων και ορολογία

Στην τρέχουσα εργασία θα γίνει προσπάθεια σχεδιασμού και υλοποίησης μιας διαδικτυακής εφαρμογής, που θα δίνει τη δυνατότητα στους χρήστες να καταγράφουν τα πάγια περιουσιακά τους στοιχεία, να τα κρατούν αποθηκευνένα καθώς και να εξάγουν έναν κωδικό QR που θα οδηγεί στις επιμέρους πληροφορίες του κάθε στοιχείου. Επίσης θα έχουν τη δυνατότητα να αποθηκεύουν τους λογαριασμούς που έχει στην κατοχή του και το τρέχον ποσό που αυτοί κατέχουν. Ακόμη θα τους δίνεται η ευκαιρία να καταγράφουν ημερησίως τα έσοδα και τα έξοδα τους. Τέλος, μία από τις πιο σημαντικές λειτουργίες που τη διαφοροποιεί από τις ήδη υπάρχουσες είναι η δημιουργία ομάδων και η δυνατότητα σύνδεσης σε ήδη υπάρχουσες που θα προσφέρει την ομαδική διαχείριση εξόδων και στοιχείων μιας εταιρείας ή ενός νοικοκυριού. Πριν οριστούν αναλυτικά οι λειτουργικές και μη λειτουργικές απαιτήσεις της εφαρμογής, οφείλουμε να αναφέρουμε μια βασική ορολογία σχετική με την εφαρμογή.

- **Πάγια Περιουσιακά στοιχεία:** Πρόκειται για στοιχεία μη χρηματοοικονομικής φύσεως, όπως οχήματα, εξοπλισμός γραφείου, εξοπλισμός σπιτιού, εξοπλισμός άμυνας, μηχανήματα και άλλα. Ένα πάγιο στοιχείο έχει τα εξής χαρακτηριστικά : να μπορεί η χρήση του να επιφέρει πιθανό μελλοντικό οικονομικό όφελος ή και μια υπηρεσία και

το κόστος του μπορεί να αποτιμηθεί αξιόλογα. Ακόμη και αν πληροί τα κριτήρια ενός περιουσιακού στοιχείου, η διαφοροποίηση του έγκειται στην αξία του, η οποία, αν δεν υπερβαίνει το κατώτατο όριο κεφαλαιοποίησης, το δεσμεύει, ώστε να μη μπορεί να κεφαλαιοποιηθεί, ούτε να αναφερθεί στις οικονομικές καταστάσεις.

- **Έσοδα:** Ως έσοδο ορίζεται η απόκτηση εισοδήματος. Μπορεί να προκύψει από μισθοδοσία, από πώληση, από ενοίκιο, ή από προσφορά κάποιου ατόμου.
- **Έξοδο:** Ως έξοδο ορίζεται η οικονομική δαπάνη. Μπορεί να οφείλεται στην αγορά κάποιου προιόντος ή κάποιας υπηρεσίας, στην πληρωμή ενοικίου, συνδρομών, λογαριασμών.

4.3.2 Ομάδα χρηστών

Η ομάδα χρηστών που θα απευθύνεται το λογισμικό περιλαμβάνει:

- Κάθε ενήλικος άνθρωπος που θα θέλει να καταχωρεί τα καθημερινά του έσοδα και έξοδα.
- Κάθε άνθρωπος που έχει στην κατοχή του πάγια περιουσιακά στοιχεία.
- Ιδρύματα, εταιρείες, ξενοδοχεία, πανεπιστήμια, νοικοκυριά που θα θέλουν να καταγράψουν τα υπάρχοντα τους.

Αντιλαμβανόμαστε λοιπόν πως το κοινό είναι ευρύ, για αυτό υπάρχει η απαίτηση μιας εφαρμογής που θα μπορεί να αναπροσαρμοστεί στις ανάγκες του δυναμικού κοινού της, καθώς και να χρησιμοποιηθεί από επιχειρήσεις σαν ”εσωτερική” εφαρμογή διαχείρισης υπαρχόντων.

4.3.3 Λειτουργικές απαιτήσεις

Στην υποενότητα θα γίνει ο ορισμός όλων των λειτουργικών απαιτήσεων που απαιτείται να έχει το λογισμικό.

Λειτουργικές απαιτήσεις	
Κωδικός	Περιγραφή

ΛΑ00	Οι χρήστες θα μπορούν να δημιουργούν λογαριασμό με τη χρήση του email και ενός κωδικού, συμπληρώνοντας και κάποια επιπλέον στοιχεία
ΛΑ01	Οι χρήστες θα μπορούν να συνδέονται σε έναν ήδη υπάρχοντα λογαριασμό με τη χρήση του email και του password τους
ΛΑ02	Οι χρήστες θα μπορούν να ξεσυνδεθούν οποιαδήποτε στιγμή
ΛΑ03	Το σύστημα θα ενημερώνει το χρήστη με μήνυμα λάθους κατά την είσοδο αν τα στοιχεία είναι λανθασμένα
ΛΑ04	Οι χρήστες θα μπορούν να περιηγηθούν στις διάφορες σελίδες μέσω της μπάρας πλοήγησης
ΛΑ05	Οι χρήστες θα μπορούν να δημιουργήσουν μία ομάδα
ΛΑ06	Οι χρήστες θα μπορούν να διαγράψουν μία ομάδα
ΛΑ07	Οι χρήστες θα μπορούν να συνδεθούν σε μία ομάδα με τη χρήση ενός κωδικού
ΛΑ08	Οι χρήστες θα μπορούν να λάβουν τον κωδικό για μία ομάδα
ΛΑ09	Οι χρήστες θα μπορούν να αλλάξουν το όνομα της ομάδας
ΛΑ10	Οι χρήστες θα μπορούν να επανέλθουν στη σύνδεση ως ατομικοί χρήστες
ΛΑ11	Οι χρήστες θα μπορούν να δημιουργήσουν τους δικούς τους λογαριασμούς
ΛΑ12	Οι χρήστες θα μπορούν να διαγράψουν τους λογαριασμούς τους
ΛΑ13	Οι χρήστες θα μπορούν να επεξεργαστούν τα στοιχεία των λογαριασμών τους
ΛΑ14	Οι χρήστες θα μπορούν να επεξεργαστούν τα στοιχεία τους
ΛΑ15	Οι χρήστες θα μπορούν να προσθέσουν κατηγορίες εξόδων
ΛΑ16	Οι χρήστες θα μπορούν να επεξεργαστούν τις κατηγορίες εξόδων
ΛΑ17	Οι χρήστες θα μπορούν να διαγράψουν τις κατηγορίες εξόδων αν αυτές δεν είναι επιλεγμένες σε καμία συναλλαγή
ΛΑ18	Οι χρήστες θα μπορούν να προσθέσουν κατηγορίες εσόδων
ΛΑ19	Οι χρήστες θα μπορούν να επεξεργαστούν τις κατηγορίες εσόδων

ΛΑ20	Οι χρήστες θα μπορούν να διαγράψουν τις κατηγορίες εσόδων αν αυτές δεν είναι επιλεγμένες σε καμία συναλλαγή
ΛΑ21	Οι χρήστες θα μπορούν να προσθέσουν μία συναλλαγή ως έξοδο ή ως έσοδο επιλέγοντας σε ποια κατηγορία και σε ποιον λογαριασμό αντιστοιχεί
ΛΑ22	Οι χρήστες θα μπορούν να επεξεργαστούν τις συναλλαγές τους
ΛΑ23	Οι χρήστες θα μπορούν να διαγράψουν τις συναλλαγές τους
ΛΑ24	Το λειτουργικό θα πρέπει να ανανεώνει το τρέχον ποσό σε έναν λογαριασμό όταν προστίθεται, επεξεργάζεται ή διαγράφεται μια συναλλαγή
ΛΑ25	Οι χρήστες θα μπορούν να δούνε τις συναλλαγές τους ανά ημέρα
ΛΑ26	Οι χρήστες θα μπορούν να δούνε τα συνολικά έξοδα του μήνα
ΛΑ27	Οι χρήστες θα μπορούν να δημιουργήσουν μια κατηγορία για τα πάγια περιουσιακά στοιχεία
ΛΑ28	Οι χρήστες θα μπορούν να επεξεργαστούν μια κατηγορία για τα πάγια περιουσιακά στοιχεία
ΛΑ29	Οι χρήστες θα μπορούν να τη διαγράψουν και μαζί με αυτή να διαγραφούν και όλα τα περιουσιακά στοιχεία
ΛΑ30	Οι χρήστες θα μπορούν να προσθέσουν ένα πάγιο περιουσιακό στοιχείο
ΛΑ31	Οι χρήστες θα μπορούν να επεξεργαστούν ένα πάγιο περιουσιακό στοιχείο
ΛΑ32	Οι χρήστες θα μπορούν να διαγράψουν ένα πάγιο περιουσιακό στοιχείο
ΛΑ33	Οι χρήστες θα μπορούν να ζητήσουν τον QR κωδικό του που θα αντιστοιχεί στη σελίδα επεξεργασίας του και να τον αποθηκεύσουν σαν εικόνα
ΛΑ34	Το σύστημα θα ενημερώνει το χρήστη για μια επιτυχή ενέργεια
ΛΑ35	Το σύστημα θα διαθέτει αυτόματο έλεγχο όταν ο χρήστης συμπληρώνει τα στοιχεία μιας φόρμας

Πίνακας 4.1: Λειτουργικές απαιτήσεις

4.3.4 Μη Λειτουργικές απαιτήσεις

Αφού ορίστηκαν οι λειτουργικές απαιτήσεις ήρθε η ώρα να οριστούν και οι μη λειτουργικές

Μη Λειτουργικές απαιτήσεις	
Κωδικός	Περιγραφή
ΜΛΑ00	Η εφαρμογή πρέπει να είναι εύκολη ως προς τη χρήση της
ΜΛΑ01	Οι χρήστες θα μπορούν να εισάγουν δεδομένα με τη χρήση του πληκτρολογίου τους
ΜΛΑ02	Οι χρήστες θα μπορούν να περιηγηθούν σε αυτή με τη χρήση του ποντικιού, αν πρόκειται για υπολογιστή ή μέσω της αφής, αν πρόκειται για κινητό ή τάμπλετ
ΜΛΑ03	Το σύστημα θα πρέπει να έχει μηνύματα επιτυχίας ή αποτυχίας σε περίπτωση λάθους μετά από κάθε ενέργεια
ΜΛΑ04	Το σύστημα θέλουμε να έχει γρήγορο χρόνο απόκρισης
ΜΛΑ05	Το σύστημα πρέπει να είναι εύκολα επεκτάσιμο και να έιναι σχεδιασμένο με τρόπο που νέες λειτουργίες να μπορούν να εισαχθούν σε αυτό
ΜΛΑ06	Το σύστημα θα πρέπει να διαθέτει ασφάλεια ως προς τους χρήστες και ως προς τη βάση
ΜΛΑ07	Το σύστημα θα πρέπει να τρέχει σε όλα τα λειτουργικά συστήματα

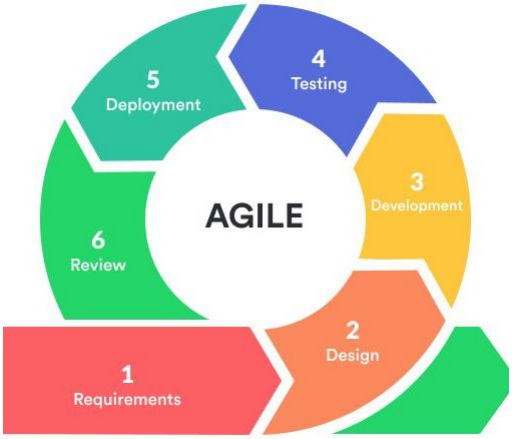
Πίνακας 4.2: Μη Λειτουργικές απαιτήσεις

4.4 Επιλογή εργαλείων και μεθοδολογίας

Εφόσον ορίστηκε το πλαίσιο στο οποίο πρέπει να σχεδιαστεί η εφαρμογή, ήρθε η ώρα να επιλέξουμε τη μεθοδολογία σχεδιασμού της, καθώς και τα εργαλεία τα οποία είναι τα πιο κατάλληλα για αυτή.

4.4.1 Μεθοδολογία

Αφού χρειαζόμαστε μία ευπροσάρμοστη και ευέλικτη εφαρμογή, όσον αφορά τις απαιτήσεις, μία καλή επιλογή για το σχεδιασμό θα ήταν η μεθοδολογία ανάπτυξης Agile, καθώς είναι ένα επαναληπτικό μοντέλο ιδανικό για την προσθήκη νέων απαιτήσεων σε κάθε κύκλο. Αποτελείται από έξι βήματα: τον προσδιορισμό των απαιτήσεων, το σχεδιασμό, την ανάπτυξη του κώδικα, τον έλεγχο, το ανέβασμα της εφαρμογής και το σχολιασμό. Μετά την ολοκλήρωση κάθε κύκλου τα βήματα ακολουθούνται από την αρχή.



Σχήμα 4.7: Βήματα του Agile μοντέλου

4.4.2 back-end

Από τα εργαλεία που μελετήθηκαν οι τελικές επιλογές πλαισίων ήταν το SpringBoot και η Node.js ως πλαίσιο ανάπτυξης του back-end. Ωστόσο το SpringBoot είναι πιο κατάλληλο για εφαρμογές με βαρύ υπολογιστικό σύστημα, ενώ η Node.js είναι πιο χρήσιμη για εφαρμογές με έντονη αλληλεπίδραση με το χρήστη. Η εφαρμογή που θα υλοποιηθεί πρόκειται να έχει συχνή επικοινωνία με το χρήστη, καθώς θα δέχεται διαρκώς εισόδους για νέα στοιχεία ή συναλλαγές. Το υπολογιστικό σύστημα δεν πρόκειται να κάνει μεγάλους υπολογισμούς, επομένως το back-end θα στηθεί με Node.js.

4.4.3 front-end

Για το front-end θα επιλεχθεί η React σε συνδυασμό με την html, css και το Bootstrap. Τα εργαλεία και οι συναρτήσεις της React θα μας προσφέρουν μια γρήγορη ανταπόκριση

στο πως ο χρήστης βλέπει τα στοιχεία ανάλογα με τις επιλογές και τις ενέργειές του. Επίσης η ανάπτυξη της εφαρμογής βασισμένη σε components θα βοηθήσει, ώστε οποτεδήποτε χρειαστεί, να επεκταθεί κάποια από τις λειτουργίες, ορισμένα κομμάτια κώδικα να μπορούν να επαναχρησιμοποιηθούν και να μη χρειάζεται να γραφούν από την αρχή.

4.4.4 Βάσεις δεδομένων

Ως βάση για το λογισμικό θα επιλεγεί μια μη σχεσιακή βάση η MongoDB. Επεξήγηση του τρόπου επιλογής της και των χαρακτηριστικών της θα γίνουν στο κεφάλαιο 5.

4.4.5 Ασφάλεια της εφαρμογής

4.4.6 4-tier

Η εφαρμογή θα αποτελείται απο το front, το keycloak, το back, και τη βάση δεδομένων.

Κεφάλαιο 5

Βάσεις Δεδομένων

Στο κεφάλαιο αυτό θα επεξηγηθεί ο τρόπος επιλογής των βάσεων δεδομένων, τα εργαλεία και οι βιβλιοθήκες που έχουν χρησιμοποιηθεί και θα αναλυθούν τα σχήματα των βάσεων και τα ερωτήματα.

5.1 Επιλογή Βάσης Δεδομένων

Οι βάσεις δεδομένων χωρίζονται σε 2 μεγάλες κατηγορίες. Τις σχεσιακές και τις μη σχεσιακές.

5.1.1 Σύγκριση σχεσιακών και μη σχεσιακών βάσεων

- **Σχεσιακές βάσεις:** Με τον όρο σχεσιακή βάση δεδομένων αναφερόμαστε στα δεδομένα που είναι αποθηκευμένα σε συσχετισμένους μεταξύ τους πίνακες. Οι σχεσιακές βάσεις δίνουν τη δυνατότητα για την προσθήκη, επεξεργασία, διαγραφή αλλά και ανάκληση των δεδομένων με συγκεκριμένα ερωτήματα και παραμέτρους. Η γλώσσα που χρησιμοποιείται κατά κύριο λόγο είναι η SQL που αποτελεί μια διαλογική γλώσσα. Ωστόσο οι σχεσιακές βάσεις είναι αυστηρές ως προς το μοντέλο των δεδομένων καθώς δε δέχονται κάποιο από τα πεδία του πίνακα να μείνει κενό. Αυτό δεν είναι πολύ ευέλικτο, ειδικά, όταν τα δεδομένα που πρόκειται να αποθηκεύσουμε υπάρχει περίπτωση να έχουν διαφορετικά μεταξύ τους πεδία, ή διαφορετικό αριθμό πεδίων.

- **Μη σχεσιακές βάσεις δεδομένων:** Με την ανάπτυξη εφαρμογών μεγάλης κλίμακας, όπου χρειαζόταν μεγάλος όγκος αποθήκευσης δεδομένων, εμφανίστηκε ένας εναλλα-

κτικός τύπος οι NoSQL ή αλλιώς μη σχεσιακές βάσεις όπου ξεπερνούν τις προσδοκίες των σχεσιακών βάσεων. Κύριο χαρακτηριστικό τους είναι ότι δεν τηρούν το σχεσιακό μοντέλο και ότι είναι σχεδιασμένες για αποθήκευση δεδομένων μεγάλης κλίμακας. Πρόκειται για κατανεμημένες βάσεις δεδομένων, οι οποίες χωρίζουν τα δεδομένα τους μεταξύ διάφορων εξυπηρετητών. Προφανώς η αποθήκευση δε γίνεται ποτέ σε πίνακες, αλλά χρησιμοποιούνται μη σχεσιακοί τρόποι οργάνωσης και ανάκτησης. Είναι σημαντικό να αναφερθεί πως οι NoSQL βάσεις δεδομένων δεν είχαν σκοπό να αντικαταστήσουν τις σχεσιακές βάσεις, αλλά ήθελαν να τις συμπληρώσουν, καθώς είναι πιο χρήσιμες όταν κάποιος δουλεύει με μεγάλο όγκο δεδομένων. Επίσης διαθέτουν πιο ”χαλαρά” μοντέλα συνέπειας για τα δεδομένα, καθώς διατίθενται να θυσιάσουν την συνοχή για την αποδοτικότητα. Οι NoSQL βάσεις δεδομένων χωρίζονται σε 4 βασικές υποκατηγορίες.[33] [34]

- Key-value stores: Πρόκειται για το πιο απλό NoSQL σύστημα διαχείρισης βάσεων δεδομένων. Τα δεδομένα αντιστοιχίζονται με ένα κλειδί και την αντίστοιχη τιμή του μέσα στη βάση. Δύο χαρακτηριστικά παραδείγματα αυτών των βάσεων είναι οι Oracle BerkleyDB και Redis.
- Document databases: Κάθε κλειδί αντιστοιχεί σε ένα ολόκληρο έγγραφο το οποίο περιέχει πολλά ζεύγη της μορφής κλειδί-τιμή ή κλειδί-πίνακας τιμών ή ακόμη και ολόκληρο εμφωλευμένο έγγραφο. Χαρακτηριστικά παραδείγματα είναι η ElasticSearch και η MongoDB.
- Graph stores: Χρησιμοποιούνται για να αποθηκεύσουν δεδομένα σχετικά με αποστάσεις, για παράδειγμα σε εφαρμογές που υπολογίζουν αποστάσεις σε χάρτη ή με διασυνδέσεις δικτύων, όπως είναι τα κοινωνικά δίκτυα. Χαρακτηριστικές αυτές της κατηγορίας είναι οι Neo4J και η Giraph.
- Column stores: Έχουν γίνει βέλτιστες για την αναζήτηση σε μεγάλα σύνολα δεδομένων και κάνουν την αποθήκευση αυτών σε στήλες και όχι σε πλειάδες. Οι πιο γνωστές είναι οι Cassandra και η HBase.

5.1.2 Η MongoDB

Σύμφωνα με τα χαρακτηριστικά που αναφέρθηκαν, η πιο κατάλληλη για την εφαρμογή είναι η MongoDB, δηλαδή μία βάση βασισμένη στην αποθήκευση πληροφορίας σε έγγραφα.

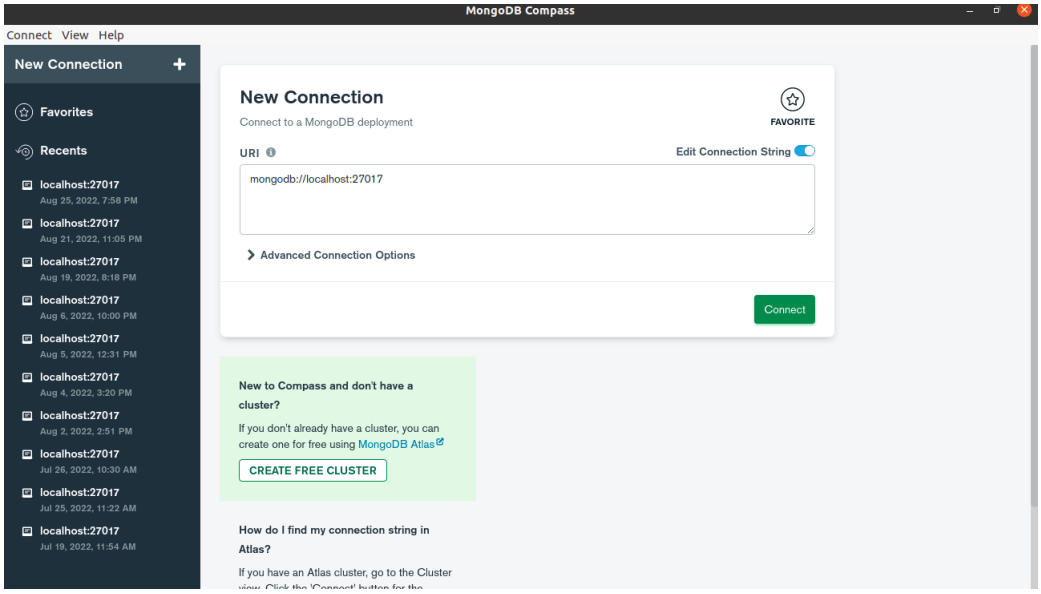
Τα θετικά της είναι ότι δε χρειάζεται εξαρχής να καθορίσουμε τη μορφή του πίνακα και τον αντίστοιχο αριθμό πεδίων, όπως στις σχεσιακές βάσεις δεδομένων. Η αντιστοιχία του πίνακα εδώ είναι η λεγόμενη συλλογή, η οποία μπορεί να περιέχει μέσα πολλά έγγραφα. Επίσης, η συγκεκριμένη βάση είναι εύκολα επεκτάσιμη με την προσθήκη νέων συλλογών ή εγγράφων που έχουν νέα πεδία. Στην περίπτωσή μας είναι χρήσιμη, καθώς δεν είναι αναγκαστικό ο χρήστης να συμπληρώνει σε κάθε προσθήκη όλα τα στοιχεία της συναλλαγής ή του πάγιου περιουσιακού στοιχείου. Στις σχεσιακές βάσεις αυτό θα αποτελούσε πρόβλημα. Επίσης, όσον αφορά τους πίνακες και τα εμφωλευμένα έγγραφα, δίνεται η δυνατότητα κάθε έγγραφο να αποθηκεύει διαφορετική ποσότητα πληροφορίας. Για παράδειγμα, όπως θα επεξηγηθεί στη συνέχεια, ο κάθε χρήστης μπορεί να προσθέσει όσες κατηγορίες θέλει για τις συναλλαγές και αυτές θα αποθηκεύονται μέσα στο έγγραφο με τις πληροφορίες του χρήστη. Αυτό δε θα ήταν εφικτό στις σχεσιακές βάσεις, καθώς κάθε κατηγορία θα έπρεπε να αποτελεί νέα εγγραφή σε πίνακα και για να βρούμε τις κατηγορίες του κάθε χρήστη, αναγκαστικά θα έπρεπε να γίνει συνένωση πινάκων(JOIN), το οποίο ως ερώτημα έχει μεγάλο κόστος. Τέλος, δίνει τη δυνατότητα να γίνει deploy online η βάση, οπότε, μόλις διαπιστώσουμε ότι τρέχει σωστά η εφαρμογή σε τοπικό επίπεδο, μπορούμε να την ανεβάσουμε online.[35]

5.2 Βιβλιοθήκες, εργαλεία και προγράμματα

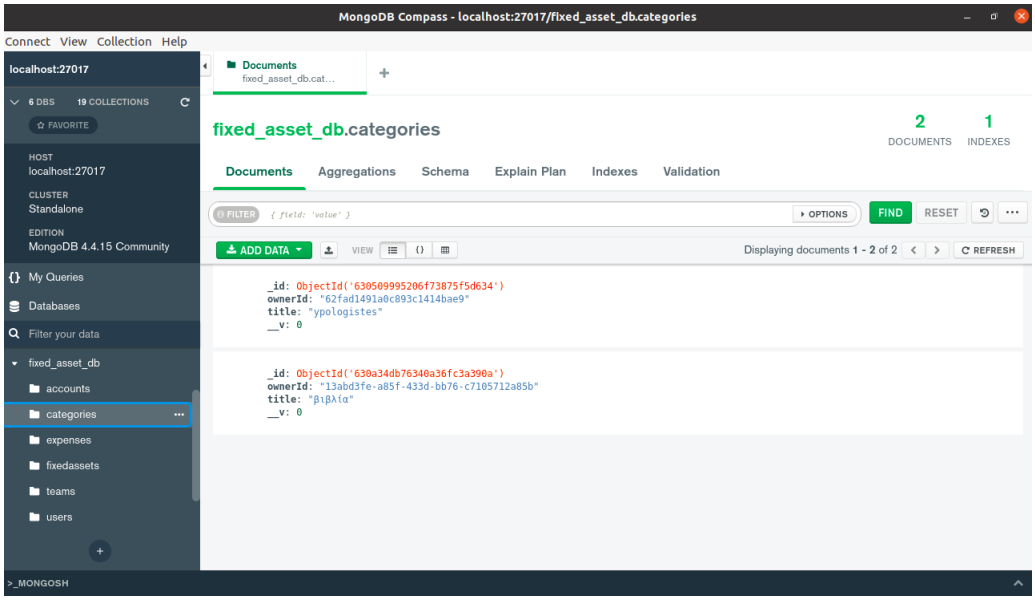
5.2.1 MongoDB Compass

Στη συγκεκριμένη εργασία, για τον έλεγχο των δεδομένων στη βάση χρησιμοποιήθηκε ένα γραφικό περιβάλλον διεπαφής χρήστη συμβατό με τη MongoDB το οποίο ονομάζεται MongoDB compass. Το πρόγραμμα αυτό δίνει τη δυνατότητα να μπορείς να βλέπεις ανά πάσα στιγμή τα δεδομένα που υπάρχουν μέσα στη βάση, να ελέγχεις άμα οι διαγραφές, οι ανανεώσεις και οι προσθήκες γίνονται σωστά, αλλά και να δοκιμάζεις τη λειτουργία διάφορων ερωτημάτων. Τα ερωτήματα αυτά μπορεί να είναι είτε το απλό find είτε τα λεγόμενα aggregation pipeline, τα οποία δίνουν τη δυνατότητα για τη δημιουργία πιο σύνθετων ερωτημάτων εύρεσης που έχουν να κάνουν με JOIN μεταξύ εγγράφων ή με την αναζήτηση σε εμφωλευμένα έγγραφα. Επίσης μπορεί κάποιος να προσθέσει δεδομένα σαν ένα JSON αρχείο, καθώς και μπορεί να εξάγει τα υπάρχοντα δεδομένα στη βάση ως ένα JSON αρχείο. Τέλος, μπορεί με το χέρι να δημιουργήσει, να διαγράψει και να επεξεργαστεί βάσεις, collections, και οποιοδήποτε έγγραφο ή πεδίο αυτού. Το γραφικό περιβάλλον αυτό μπορεί κάποιος να

το κατεβάσει από το `https://www.mongodb.com/try/download/compass`. Στα στιγμιότυπα φαίνεται τι βλέπει ο χρήστης ανοίγοντας το γραφικό περιβάλλον, όπου μπορεί να συνδεθεί είτε στην τοπική βάση μέσω του localhost:27017 είτε σε οποιαδήποτε online βάση στον Atla δίνοντας τον αντίστοιχο σύνδεσμο. Στο δεύτερο στιγμιότυπο φαίνονται στα αριστερά όλα τα collections της βάσης που επιλέξαμε καθώς και τα έγγραφα που έχει το επιλεγμένο collection.



Σχήμα 5.1: Αρχική σελίδα του compass



Σχήμα 5.2: Στιγμιότυπο από την χρήση του compass

5.2.2 Mongoose

Για την πραγματοποίηση των ερωτημάτων στη βάση από τον κώδικα, καθώς και για τον ορισμό των σχημάτων, έχει χρησιμοποιηθεί μία βιβλιοθήκη που ονομάζεται Mongoose. Αναφέρεται από πολλούς πως χρησιμοποιείται για μια κομψή μοντελοποίηση των αντικειμένων της MongoDB από τη Node.js. Η Mongoose επιλέγεται από τη Node.js για την επιβολή σχημάτων και την σταθεροποίηση αυτών στην εφαρμογή, για την επικύρωση των μοντέλων, αλλά και για τη χειραγώγηση των δεδομένων. Δίνει τη δυνατότητα να ορίσει κάποιος κενό σχήμα για μία συλλογή μέχρι να καταλήξει στο οριστικοποιημένο σχήμα. Με άλλα λόγια η βιβλιοθήκη αυτή χρησιμοποιείται για να ελέγξει σε έναν βαθμό την ελευθερία που παρέχει η MongoDB. Χρησιμοποιεί τα σχήματα, τα οποία ορίζουν τη μορφή που θα έχουν οι συλλογές. Οι δυνατοί τύποι που μπορεί να έχει ένα πεδίο είναι οι εξής :String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array, Decimal128, Map. Τα μοντέλα είναι αυτά που ευθύνονται για την εφαρμογή του σχήματος σε κάθε έγγραφο. Μέσω της κλήσης του μοντέλου μπορεί κάποιος να εφαρμόσει στη βάση όλα τα ερωτήματα που ανήκουν στης λειτουργίες CRUD, δηλαδή δημιουργία. γράψιμο, ανανέωση και διαγραφή. Η εγκατάσταση της βιβλιοθήκης αυτής γίνεται με την εντολή **npm i mongoose**. Στα παρακάτω στιγμιότυπα φαίνεται η σύνδεση με τη βάση στον κώδικα.[36]

```
1   'use strict';
2   const mongoose = require('mongoose');
3   const config = require('../config');
4
5   module.exports = async () => {
6       //connect to fixed_asset_db
7       mongoose.connect("mongodb://localhost:27017/fixed_asset_db", { useNewUrlParser: true,
8                                                   useUnifiedTopology: true,useFindAndModify: false });
9
10      config.MONGO_DEBUG?mongoose.set('debug', true):mongoose.set('debug', false);
11  }
12
```

Σχήμα 5.3: Σύνδεση της Mongoose στη βάση δεδομένων

5.2.3 GraphQL

Η GraphQL είναι μια γλώσσα ερωτημάτων, που μπορεί να χρησιμοποιηθεί από τη μεριά του πελάτη, ώστε να ζητήσει ακριβώς τα δεδομένα και τα πεδία που χρειάζεται χωρίς να πρέπει να τρέξει στο back κάποιο εξειδικευμένο ερώτημα. Χρειάζεται να οριστεί και από τη μεριά του back και από τη μεριά του front και στην ουσία αποτελεί έναν διαμεσολαβητή κάθε φορά που ο πελάτης ζητάει δεδομένα από τον εξυπηρετητή, ώστε να του τα γυρίσει στη μορφή που θέλει. Για παράδειγμα μία μεγάλη ευκολία της grpahQL είναι ότι μπορούμε να

ορίσουμε μία μορφή επιστροφής δεδομένων, η οποία, εκτός από τα δεδομένα, θα περιέχει και τα μηνύματα επιτυχίας ή αποτυχίας. Με τον τρόπο αυτό μπορούμε πιο άμεσα στο front να ξέρουμε αν το ερώτημα στη βάση ήταν επιτυχές. Επίσης ο σωστός ορισμός των τύπων δεδομένων μειώνει τη λανθασμένη επικοινωνία ανάμεσα στο front και στο back. Ωστόσο αλλάζει τον τρόπο που ορίζονται τα ερωτήματα, ειδικά από τη μεριά του εξυπηρετητή, γεγονός που μπορεί να θεωρηθεί αρνητικό από κάποιους. Για να χρησιμοποιηθεί σωστά πρέπει να οριστούν τα σχήματα, που θα περιέχουν όλους τους πιθανούς συνδυασμούς και τύπους δεδομένων που μπορεί να ζητηθούν. Ως τύποι δεδομένων υπάρχουν οι κλασικοί τύποι από τη mongoDB αλλά επίσης και τύποι που μπορούμε να τους ονομάσουμε όπως θέλουμε και μέσα να περιέχουν τα επιθυμητά πεδία. Προσοχή χρειάζεται ο διαχωρισμός των τύπων που θα χρησιμοποιηθούν για την επιστροφή δεδομένων που πρέπει να οριστούν ως Types και σε αυτούς που πρόκειται να χρησιμοποιηθούν ως είσοδοι, οπότε θα γραφούν ως Input. Επίσης δυνατή είναι και η χρήση των Union, το νόημα των οποίων είναι πως μπορεί αυτό που θα επιστρέψει ένα ερώτημα να μην έχει πάντα την ίδια μορφή τύπου, οπότε με το union καταφέρνουμε να συμπεριλάβουμε όλα τα πιθανά σενάρια. Οι πιο συχνές λειτουργίες που χρησιμοποιούνται είναι τα queries και τα mutations. Τα queries αντιστοιχούν στις λειτουργίες διαβάσματος, δηλαδή σε ερωτήματα που δεν τροποποιούν τη βάση και απλά ανασύρουν δεδομένα από αυτή. Τα mutations αντιστοιχούν στις υπόλοιπες λειτουργίες, όπως το γράψιμο, η διαγραφή, και η ενημέρωση, άρα, όταν θέλουμε να ορίσουμε ένα τέτοιου είδους ερώτημα, το ορίζουμε στα mutation. Μία επίσης ενδιαφέρουσα λειτουργία είναι τα subscriptions, τα οποία προσφέρουν μία ζωντανή ενημέρωση κάθε φορά που η βάση τροποποιείται. Σα λειτουργία είναι χρήσιμη για εφαρμογές με πολλούς χρήστες, όπως για παράδειγμα τα μέσα κοινωνικής δικτύωσης που μόλις κάποιος δημιουργήσει μια νέα δημοσίευση θέλουμε να φαίνεται και στους υπόλοιπους. Εφόσον εξηγήσαμε τι είναι η GraphQL, ήρθε η ώρα να δούμε με ποιον τρόπο θα τη χρησιμοποιήσουμε στο λογισμικό.[37] [38]

5.2.4 Apollo

Το Apollo πρόκειται για μία πλατφόρμα της GraphQL η οποία διαθέτει βιβλιοθήκη συμβατή με το front-end το Apollo Client, καθώς και ένα πλαίσιο που υποστηρίζεται από τους διακομιστές το Apollo server.

Apollo server

Πιο συγκεκριμένα το Apollo server είναι ένας ανοιχτού κώδικα διακομιστής, συμβατός με τη GraphQL, ο οποίος, μόλις τον ενσωματώσουμε στον δικό μας διακομιστή, μας βοηθάει να στέλνουμε και να λαμβάνουμε εύκολα δεδομένα από τον πελάτη, ανεξαρτήτως από τι γλώσσα χρησιμοποιεί. Ο κώδικας αρχικοποίησης του είναι ο εξής:

```
1  'use strict';
2  const http = require('http');
3
4  const ApolloServer = require('apollo-server-express').ApolloServer;
5  const typeDefs = require('../graphql/schemas/schemas');
6  const resolvers = require('../graphql/resolvers/resolverls');
7  const {
8    ApolloServerPluginLandingPageGraphQLPlayground
9  } =require("apollo-server-core");
10 const cors = require("cors");
11
12 module.exports = async (app) => {
13
14   const httpServer = http.createServer(app);
15   const server = new ApolloServer({
16       typeDefs: typeDefs,
17       resolvers: resolvers,
18       plugins: [
19         ApolloServerPluginLandingPageGraphQLPlayground(),
20       ],
21       subscriptions: {
22         path: '/subscriptions'
23       },
24       context: async ({ req, res }) => {
25         return { req, res };
26       },
27     });
28     app.use(cors());
29
30
31     await server.start();
32     server.applyMiddleware({ app});
33
34     return httpServer;
35 };
```

Σχήμα 5.4: Σύνδεση του διακομιστή στο Apollo server

Το σημαντικό είναι στο πεδίο typeDefs να αναφέρουμε το αρχείο όπου δηλώνονται τα σχήματα και οι τύποι για τη Graph και στο πεδίο resolvers να αντιστοιχίσουμε το αρχείο όπου γίνεται η υλοποίηση των ερωτημάτων. Επίσης αξίζει να αναφερθεί πως με το πεδίο plugins μπορούμε να ενεργοποιήσουμε το playground της Graph, που μας επιτρέπει να ελέγχουμε τα mutation και τα queries που έχουμε γράψει, πριν τα συνδέσουμε με τον πελάτη.[39]

```
deleteScenario
PRETTIFY   HISTORY   http://localhost:4444/graphql

1 mutation{
2   deleteScenario(_id:"13abd3fe-a85f-433d-bb76-c7105712a85b",scenariotype:"minus",usc
3     __typename
4     ...on ScenarioMessage{
5       code
6       message
7     }
8   }
9 }
```

Σχήμα 5.5: Στιγμιότυπο από το playground της Graph

Apollo client

Πρόκειται για μία βιβλιοθήκη συμβατή με τη JavaScript και τη React, η οποία δίνει τη δυνατότητα στον πελάτη να λαμβάνει τα δεδομένα με τη μορφή που αυτός τα χρειάζεται, ζητώντας ακριβώς τα πεδία που θέλει, χωρίς όμως να εμπλέκεται με τον κώδικα που θα

εκτελέσει ο εξυπηρετητής. Παρακάτω φαίνεται ο κώδικας σύνδεσης της Graph στο front . Προσοχή πρέπει να δοθεί στο link το οποίο πρέπει να είναι το ίδιο με αυτό που θα τρέχει η Graph του εξυπηρετητή και από το οποίο θα στέλνονται τα δεδομένα.[40]

```
// graphql base api link for http requests (+ file uploads)
const httpLink = createUploadLink({
  uri: 'http://localhost:4444/graphql',
});

//subscriptions link
const wsLink = new  GraphQLWsLink ({
  uri: 'ws://localhost:4444/subscriptions',
  options: {
    reconnect: true
  }
});

// middleware for attaching the token to every graphql request
const middleware = new ApolloLink((operation, forward) => {

  let token = store.getState().token;
  console.log( "store.getState()",store.getState())

  operation.setContext({

    headers: {
      authorization: token!=="" ? `Bearer ${token}` : ""
    }
  });

return forward(operation);
});
```

Σχήμα 5.6: Σύνδεση του πελάτη στο Apollo client

```
export const client = new ApolloClient({
  link: middleware.concat(splitLink),
  cache: new InMemoryCache({
    addTypename: false
  }),
  cors:{
    origin: '*',
    credentials: true
  },
  fetchOptions: {
    mode: 'no-cors',
  },
  query: {
    fetchPolicy: 'no-cache',
    errorPolicy: 'all',
  },
  watchQuery: {
    fetchPolicy: 'no-cache',
    errorPolicy: 'all',
  },
  mutate: {
    fetchPolicy: 'no-cache',
    errorPolicy: 'all',
  },
});
console.log("httpLink",httpLink);

console.log("middleware.concat(splitLink)",middleware.concat(splitLink));
```

Σχήμα 5.7: Σύνδεση του πελάτη στο Apollo client

5.3 Σχήματα βάσης και συλλογές

Εφόσον εξηγήθηκαν τα βασικά εργαλεία, ήρθε η ώρα για το σχεδιασμό της βάσης. Ο σχεδιασμός των βάσεων μιας εφαρμογής είναι από τα πιο κρίσιμα κομμάτια, καθώς πάνω σε αυτές θα βασιστεί όλη η υπόλοιπη μεθοδολογία. Μία μεγάλη αλλαγή στη βάση, όταν ο κώδικας έχει φτάσει σε προχωρημένο σημείο, μπορεί να αποδειχτεί καταστροφική. Σύμφωνα με τα εργαλεία που χρησιμοποιούνται για κάθε διαφορετικό τύπο συλλογής, θα πρέπει να ορίζουμε το σχήμα της στη mongoose και έπειτα να ορίζουμε τον αντίστοιχο τύπο στο αρχείο της GraphQL. Πιο συγκεκριμένα, στο φάκελο models τοποθετούνται τα αρχεία που είναι σχετικά με τη mongoose ενώ μέσα στο φάκελο graphql/schemas βρίσκεται το αρχείο με τα σχετικούς τύπους και τις ενώσεις. Σύμφωνα με τις λειτουργικές απαιτήσεις της εφαρμογής

οι συλλογές δεδομένων με τα αντίστοιχα σχήματα που θα χρειαστούμε είναι οι εξής:

5.3.1 Users

Πρόκειται για τη συλλογή που θα αποθηκεύει τα δεδομένα των χρηστών την πρώτη φορά που αυτοί θα εισάγονται στην εφαρμογή. Παρακάτω φαίνεται ένας πίνακας με τα πεδία του σχήματος καθώς και τον τύπο που αυτά έχουν στην mongoose και στην Graph καθώς και η χρήση τους.

User Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο
firstname	String	String	Αποθηκεύει το μικρό όνομα του χρήστη
lastname	String	String	Αποθηκεύει το επώνυμο του χρήστη
username	String	String	Αποθηκεύει το όνομα του χρήστη κατα τη χρήση της εφαρμογής
`profile_id`	String	String	Αποθηκεύει το χαρακτηριστικό id που αντιστοιχεί στο προφίλ του χρήστη στο keycloak
scenarioPlus	[scenario]	[scenario]	Αποθηκεύει τις πληροφορίες για τις κατηγορίες εισόδων
scenarioMinus	[scenario]	[scenario]	Αποθηκεύει τις πληροφορίες για τις κατηγορίες εξόδων

Πίνακας 5.1: User Model

Όπως είναι εμφανές, χρησιμοποιείται και ένα βοηθητικό σχήμα για τον ορισμό των σεναρίων-κατηγοριών των διάφορων συναλλαγών, το οποίο φαίνεται στον επόμενο πίνακα. Το ονομάζουμε βοηθητικό, καθώς δε δημιουργείται ξεχωριστή συλλογή στη βάση που να του αντιστοιχεί, αλλά χρησιμοποιείται εμφωλευμένο μέσα στο σχήμα των Users και των Teams, όπως θα δούμε στη συνέχεια. Κάθε φορά που δημιουργείται ένας χρήστης, οι πίνακες των

σεναρίων αρχικοποιούνται με κάποιες τιμές, τις οποίες ο χρήστης στη συνέχεια μπορεί να αλλάξει εάν το επιθυμεί.

Scenario Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που δημιουργείται για το κάθε σενάριο
color	String	String	Αποθηκεύει το χρώμα που θα αντιστοιχεί σε αυτή την κατηγορία
image	String	String	Αποθηκεύει το εικονίδιο που θα αντιστοιχεί σε αυτή την κατηγορία
title	String	String	Αποθηκεύει τον τίτλο της κατηγορίας

Πίνακας 5.2: Scenario Model

Όσον αφορά το χρώμα και το εικονίδιο αποθηκεύεται ένα αλφαριθμητικό, το οποίο στο front αντιστοιχίζεται με το κατάλληλο χρώμα ή την κατάλληλη εικόνα αντίστοιχα. Επομένως για τους χρήστες ορίζονται 2 σχήματα στην mongoose και από τη μεριά της Graph έχουμε τον τύπο User καθώς και τους τύπους UserOk και UserError . Οι δύο τελευταίοι συνενώνονται σε ένα Union που ονομάζεται returnedUser . Το union δίνει τη δυνατότητα να επιστρέφουμε στο χρήστη είτε ένα έγγραφο που θα περιέχει τα στοιχεία του user και ένα μήνυμα και έναν κωδικό ή στην περίπτωση λάθους απλώς ένα μήνυμα και έναν κωδικό. Όσον αφορά τα σενάρια έχουμε δηλωμένο στη μεριά της Graph έναν τύπο και μία είσοδο, καθώς και το ScenarioMessage που χρησιμοποιείται για να επιστρέφει μήνυμα και κωδικό, όταν ο χρήστης επεξεργάζεται κάποιο σενάριο.

5.3.2 Teams

Το σχήμα αυτό χρησιμοποιείται για την αποθήκευση πληροφορίας σχετική με τις διάφορες ομάδες.

Team Model

Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο
creator	String	String	Αποθηκεύει το profile id του χρήστη που δημιούργησε την ομάδα
members	[String]	[String]	Αποθηκεύει τα profile id των χρηστών που ανήκουν στην ομάδα
title	String	String	Αποθηκεύει τον τίτλο της ομάδας
uid	String	String	Αποθηκεύει τον μοναδικό κωδικό που αντιστοιχεί στην κάθε ομάδα
scenarioPlus	[scenario]	[scenario]	Αποθηκεύει τις πληροφορίες για τις κατηγορίες εσόδων
scenarioMinus	[scenario]	[scenario]	Αποθηκεύει τις πληροφορίες για τις κατηγορίες εξόδων

Πίνακας 5.3: Team Model

Στην graph εκτός από τον τύπο Team δημιουργούνται και οι TeamsOk, TeamOk, TeamError οι οποίες ομαδοποιούνται στο union returnedTeams και χρησιμοποιούνται αντίστοιχα για την επιστροφή πίνακα ομάδων, κωδικού και μηνύματος, για την επιστροφή μεμονωμένης ομάδας κωδικού και μηνύματος, ή σε περίπτωση λάθους απλώς κωδικού και μηνύματος.

5.3.3 Accounts

Πρόκειται για τη συλλογή που θα έχει αποθηκευμένους τους λογαριασμούς του χρήστη

Account Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο

name	String	String	Αποθηκεύει το όνομα του λογαριασμού
price	Number	Float	Αποθηκεύει το ποσό των χρημάτων που υπάρχουν στο λογαριασμό
`owner_id`	String	String	Αποθηκεύει το profile id, αν ο λογαριασμός έχει γίνει από σύνδεση μεμονωμένου χρήστη ή το id της ομάδας, άμα ο λογαριασμός δημιουργηθεί από ομάδα

Πίνακας 5.4: Account Model

Από τη μεριά της graph εκτός από τον τύπο Account έχουμε και τους επιστρεφόμενους τύπους AccountError, AccountOk, AccountsOk οι οποίοι συνενώνονται στο union returnedAccounts

5.3.4 Expenses

Πρόκειται για τη συλλογή που θα αποθηκεύει τις διάφορες συναλλαγές είτε αυτές είναι έσοδα είτε είναι έξοδα.

Expense Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο
`owner_id`	String	String	Αποθηκεύει το profile id, αν η συναλλαγή έχει γίνει από σύνδεση μεμονωμένου χρήστη ή το id της ομάδας άμα η συναλλαγή δημιουργηθεί από ομάδα
account	ObjectId	ID	Αποθηκεύει το id του λογαριασμού που ανήκει η συναλλαγή

scenario	ObjectId	ID	Αποθηκεύει το id του σεναρίου που ανήκει η συναλλαγή
money	Number	Float	Αποθηκεύει το ποσό της συναλλαγής
type	String	String	Αποθηκεύει τον τύπο της συναλλαγής, μπορεί να πάρει τις τιμες ”minus” ή ”plus”
description	String	String	Αποθηκεύει μια σχετική περιγραφή
date	date	date	Αποθηκεύει την ημερομηνία της συναλλαγής
color	String	String	Αποθηκεύει το χρώμα του σεναρίου στο οποίο ανήκει η συναλλαγή
image	String	String	Αποθηκεύει το εικονίδιο του σεναρίου στο οποίο ανήκει η συναλλαγή

Πίνακας 5.5: Expense Model

Και σε αυτή την περίπτωση χρησιμοποιείται ένα βοηθητικό σχήμα, το οποίο έχει να κάνει με τις ημερομηνίες .

Date Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
year	String	String	Ο χρόνος
month	String	String	Ο μήνας
day	String	String	Η ημέρα

Πίνακας 5.6: Date Model

Αντίστοιχα στη graph ορίζεται τύπος date καθώς και τύπος Expense και το union returnedExpence που αποτελείται από τους τύπους ExpenseOk, ExpensesOk, ExpenseError.

5.3.5 Categories

Είναι η συλλογή που αποθηκεύει πληροφορία σχετική με τις κατηγορίες των πάγιων περιουσιακών στοιχείων. Τα πεδία του σχήματός της φαίνονται παρακάτω.

Category Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο
title	String	String	Αποθηκεύει το όνομα της κατηγορίας
`owner_id`	String	String	Αποθηκεύει το profile id αν η κατηγορία έχει γίνει από σύνδεση μεμονωμένου χρήστη ή το id της ομάδας, άμα η κατηγορία δημιουργηθεί από ομάδα

Πίνακας 5.7: Category Model

Στη Graph ορίζονται ο τύπος Category καθώς και οι τύποι του union returnedCategory που είναι οι CategoryOk, CategoriesOk, και CategoryError.

5.3.6 FixedAsset

Είναι η συλλογή στην οποία θα αποθηκεύονται οι πληροφορίες για τα πάγια περιουσιακά στοιχεία.

Fixed Asset Model			
Πεδίο	Τύπος mongoose	Τύπος GraphQL	Χρησιμότητα
`_id`	ObjectId	ID	Πρόκειται για το χαρακτηριστικό κωδικό που αποδίδει η βάση στο έγγραφο
title	String	String	Αποθηκεύει το όνομα του πάγιου περιουσιακού στοιχείου
date	String	String	Αποθηκεύει την ημερομηνία αγοράς σε μορφή αλφαριθμητικού
price	Number	Float	Αποθηκεύει την τιμή αγοράς
buyer	String	String	Αποθηκεύει το όνομα του αγοραστή
description	String	String	Αποθηκεύει μια σχετική περιγραφή

location	String	String	Αποθηκεύει την τοποθεσία στην οποία βρίσκεται το πάγιο περιουσιακό στοιχείο
`owner_id`	String	String	Αποθηκεύει το profile id, αν η κατηγορία έχει γίνει από σύνδεση μεμονωμένου χρήστη ή το id της ομάδας, άμα η κατηγορία δημιουργηθεί από ομάδα
`category_id`	ObjectId	ID	Αποθηκεύει το `_id` που αντιστοιχεί στην κατηγορία που ανήκει το πάγιο περιουσιακό στοιχείο

Πίνακας 5.8: Fixed Asset Model

Στην Graph δημιουργείται και τύπος και είσοδος για το FixedAsset αλλά και ένα union που ονομάζεται returnedFixedAsset που περιέχει τους εξής τύπους FixesAssetOk, FixedAssetsOk, FixedAssetError.

5.3.7 Παράδειγμα κώδικα

Ακολουθούν στιγμιότυπα κώδικα για τον ορισμό σχήματος στην mongoose και τον ορισμό types και unnions στη graph. Επιλέχθηκε το σχήμα των συναλλαγών. Τα υπόλοιπα ορίζονται με παρόμοιο τρόπο.

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

const Schema = mongoose.Schema;
const date = mongoose.Schema({
    day:String,
    year:String,
    month:String

}, { _id : true, strict: false });
// Expense model
const ExpensesSchema = new Schema({
    owner_id:String,
    account:Schema.Types.ObjectId,
    scenario:Schema.Types.ObjectId,
    money:Number,
    type:String,
    description:String,
    date:date,
    color:String,
    image:String

}, {_id : true,strict:false})

module.exports = ExpensesSchema
```

Σχήμα 5.8: Στιγμιότυπο από τον ορισμό σχήματος στην mongoose

```
type Date{
    day:String
    year:String
    month:String
}
input InputDate{
    day:String
    year:String
    month:String
}

type Expense{
    _id:ID
    owner_id:String
    account:ID
    scenario:ID
    money:Float
    type:String
    description:String
    date:Date
    color:String
    image:String
}
type ExpenseOk{
    expense:Expense
    message:String
    code:String
}
type ExpensesOk{
    expenses:[Expense]
    message:String
    code:String
}
type ExpenseError{
    message:String
    code:String
}

union returnedExpence = ExpenseOk | ExpensesOk |ExpenseError
```

Σχήμα 5.9: Στιγμιότυπο από τον ορισμό τύπων και union στην graph

```
    returnedExpence:{
        __resolveType: (obj) =>{
            if(obj.expense) return "ExpenseOk";
            if(obj.expenses) return "ExpensesOk";
            return "ExpenseError"
        }
    },
```

Σχήμα 5.10: Στιγμιότυπο από τον ορισμό union στους resolvers

5.4 Queries

Τα Queries είναι τα ερωτήματα που γίνονται στη βάση και απλά διαβάζουν δεδομένα χωρίς να την τροποποιούν [41]

5.4.1 Επεξήγηση όλων των queries

```
type Query {
    findUser: returnedUser
    getTeams: returnedTeams
    getTeam(_id:ID): returnedTeam
    getCategories(ownerId:String):returnedCategory
    getCategoryName(_id:ID):returnedCategory
    getAllFixedAsset(owner_id:String):returnedFixedAsset
    getFixedAssetByCategory(owner_id:String,category_id:ID):returnedFixedAsset
    getOneFixedAsset(_id:ID):returnedFixedAsset
    getAccounts(owner_id:String):returnedAccount
    getAllExpenses(owner_id:String,date:InputDate):returnedExpence
    getMonthExpenses(owner_id:String,year:String,month:String):returnedExpence

}
```

Σχήμα 5.11: Όλα τα queries που χρησιμοποιούνται

1. **findUser :** Είναι το ερώτημα που επιστρέφει τον user που είναι συνδεδεμένος αυτή τη στιγμή. Δε δέχεται παραμέτρους καθώς το αναγνωριστικό token από το kaycloak στέλνεται μέσα των headers κάθε φορά που γίνεται ένα ερώτημα από το front στο back . Πριν από κάθε ερώτημα το λογισμικό στέλνει ένα ερώτημα για το αν ο χρήστης

είναι έγκυρος και στην απάντηση που παίρνει βρίσκεται και το profile id που χρησιμοποιούμε ως πεδία και στη συλλογή των user. Έτσι εκτελεί μία findOne και επιστρέφει το επιθυμητό αποτέλεσμα.

2. **getTeams :** Επιστρέφει όλες τις ομάδες στις οποίες ανήκει ο συνδεδεμένος χρήστης . Αυτό το επιτυγχάνει εκτελώντας ένα aggregate πάνω στην συλλογή Teams, το οποίο ψάχνει εσωτερικά στον πίνακα των ομάδων εάν ο χρήστης είναι μέλος αυτής. Οι επιμέρους εντολές που χρησιμοποιούνται στο aggregate είναι η unwind η οποία ανοίγει τον πίνακα των members και η match η οποία ελέγχει αν ταιριάζει το id του χρήστη με τα id members.

3. **getTeam :** Επιστρέφει τις πληροφορίες μιας συγκεκριμένης ομάδας. Εκτελεί μία findOne βασισμένη στο id πάνω στη συλλογή Teams.

4. **getCategories :** Επιστρέφει όλες τις κατηγορίες που αντιστοιχούν στο χρήστη ή την ομάδα που είναι συνδεδεμένη. Χρησιμοποιεί aggregate στη συλλογή Categories και κάνει match πάνω στο πεδίο ownerid.

5. **getCategoryName :** Επιστρέφει μία μεμονωμένη κατηγορία ώστε ο χρήστης να έχει πρόσβαση στο όνομα της. Ανακτά τα δεδομένα που χρειάζεται μέσω της findOne πάνω στη συλλογή Categories.

6. **getAllFixedAsset :** Επιστρέφει όλα τα πάγια περιουσιακά στοιχεία που ανήκουν σε έναν χρήστη ή σε μία ομάδα. Χρησιμοποιεί aggregate πάνω στη συλλογή FixedAssets και match πάνω στο πεδίο του ownerid .

7. **getFixedAssetByCategory :** Επιστρέφει όλα τα πάγια περιουσιακά στοιχεία που ανήκουν σε μία κατηγορία. Χρησιμοποιεί aggregate πάνω στη συλλογή FixedAssets και match πάνω στο πεδίο του ownerid και του categoryid.

8. **getOneFixedAsset :** Επιστρέφει ένα περιουσιακό στοιχείο μεμονωμένο χρησιμοποιώντας την findOne στη συλλογή FixedAssets.

9. **getAccounts :** Επιστρέφει όλους τους λογαριασμούς που ανήκουν σε ένα χρήστη ή μία ομάδα κάνοντας aggregate πάνω στη συλλογή Accounts και match πάνω στο πεδίο του ownerid .

10. **getAllExpenses :** Επιστρέφει όλες τις συναλλαγές ενός χρήστη ή μίας ομάδας κάνοντας aggregate πάνω στη συλλογή Expenses και match πάνω στο πεδίο του ownerid .

11. **getMonthExpenses :** Ένα από τα πιο σύνθετα ερωτήματα όσον αφορά τα queries καθώς επιστρέφει τα συνολικά έξοδα ανά κατηγορία για κάποιον συγκεκριμένο χρήστη. Αυτό το πετυχαίνει εφαρμόζοντας aggregate πάνω στη συλλογή Expenses και έπειτα χρησιμοποιώντας τις εντολές match, group και project. Πιο συγκεκριμένα το match γίνεται πάνω στα πεδία year, month, ownerid και type, ώστε να συλλέξει τα έξοδα για το μήνα αυτού του χρόνου για το συγκεκριμένο χρήστη. Έπειτα κάνει group με βάση την κατηγορία-σενάριο, το χρώμα και την εικόνα και αθροίζει το πεδίο money σε μία νέα μεταβλητή. Τέλος με την project επιλέγει ποια πεδία θα γυρίσει στο χρήστη.

5.4.2 Ορισμός ενός querie στον κώδικα

Τα βήματα για να καταφέρει κάποιος να χρησιμοποιήσει ένα query είναι τα εξής:

1. Δήλωση του query στο αρχείο schemas στο back-end.

2. Ορισμός των ενεργειών του query, με άλλα λόγια γράψιμο της συνάρτησης που θα καλείται όταν καλεστεί αυτό το query στο αρχείο resolvers στο back-end.

3. Δήλωση του query και των πεδίων που αυτό θα επιστρέφει στο αρχείο Query στο front-end.

4. Κλήση του query μέσω της useQuery.

```
}
async getMonthExpenses(root,args,context){
    const {owner_id,month,year}=args;


    const check = await ValidateUser(context.req.headers.authorization);
    if(check.code===200){
        const month_expenses = await Expenses.aggregate(
            [
                {
                  '$match': {
                    'date.year': year,
                    'date.month': month,
                    'type': 'minus',
                    'owner_id': owner_id
                  }
                }, {
                  '$group': {
                    '_id': {
                      'scenario': '$scenario',
                      'color': '$color',
                      'image': '$image'
                    },
                    'totalAmount': {
                      '$sum': '$money'
                    },
                    'count': {
                      '$sum': 1
                    }
                  }
                }, {
                  '$project': {
                    'scenario': '$_id.scenario',
                    '_id': false,
                    'money': '$totalAmount',
                    'color': '$_id.color',
                    'image': '$_id.image'
                  }
                }
            ].catch(err => {console.log(err)
        console.log("Error on aggregate");
        return {message:Messages.validation.ExpencseError.msg,code:Messages.validation.ExpencseError.code}

    });
    console.log(month_expenses,"")
    if(month_expenses){
        return{expenses:month_expenses,message:Messages.validation.ExpencseData.msg,code:Messages.validation.ExpencseData.code}
    }else{
        return {expenses:[],message:Messages.validation.AcountsData.msg,code:Messages.validation.AcountsData.code}
    }
    }else{
        let code = Messages.authentication.SesionExpired.code;
        let message = Messages.authentication.SesionExpired.msg;
        return {message:message,code:code}
    }
}
```

Σχήμα 5.12: Κώδικας που εκτελεί το query και ορίζεται στους resolvers

```
export const GET_MONTH_EXPENSES = gql`
  query getMonthExpenses($owner_id:String,$year:String,$month:String) {
    getMonthExpenses(owner_id:$owner_id,year:$year,month:$month){
      __typename
      ...on ExpensesOk{
        expenses{
          color
          image
          money
          scenario
        }
        code
        message
      }
      ...on ExpenseError{
        code
        message
      }
    }
  }
  `
```

Σχήμα 5.13: Ορισμός του query στο front

```
const {data} = useQuery(GET_MONTH_EXPENSES,{
        variables:{
            owner_id: props.owner_id,
            month:JSON.stringify(props.date.getMonth()),
            year:JSON.stringify(props.date.getFullYear())
        }
    })
```

Σχήμα 5.14: Κλήση στο front μέσω της useQuery

5.4.3 Επεξήγηση find και aggregate

Σε όλα τα queries χρησιμοποιούνται τρεις συναρτήσεις .Αυτές είναι οι εξής:

- **find:** Η find καλείται συνήθως και μπορεί να λάβει τρεις παραμέτρους, τα queries, projection, options. Στη θέση του query μπαίνει με μορφή JSON το ερώτημα που καθορίζει ποια πεδία πρέπει να ταιριάζουν με ποια τιμή.Στη θέση του projection ορίζονται τα πεδία που θέλουμε να επιστραφούν. Στο options ορίζονται επιπλέον παράμετροι που τροποποιούν το ερώτημα.

- **findOne** Η κλήση της είναι παρόμοια με τη find, μόνο που, αντί για έναν πίνακα εγγράφων, γυρνάει ένα μεμονομένο έγγραφο.

- **aggregate:** Πρόκειται για το aggragation pipeline που προσφέρει η MongoDB και που αποτελείται από ένα ή περισσότερα στάδια που επεξεργάζονται τα έγγραφα. Κάθε στάδιο εκτελεί μία διαφορετική λειτουργία στα έγγραφα εισόδου, για παράδειγμα μπορεί να ομαδοποιεί τα έγγραφα ή να βρίσκει ποια ταιριάζουν σε μία συγκεκριμένη τιμή πεδίου. Τα στάδια που έχουν χρησιμοποιηθεί στην εφαρμογή είναι τα match, group, unwind,project.

5.5 Mutations

Τα Mutations είναι τα ερωτήματα που γίνονται στη βάση και τροποποιούν τα δεδομένα είτε προσθέτοντας, είτε διαγράφοντας, είτε ανανεώνοντάς τα. [42]

5.5.1 Επεξήγηση όλων των mutation

```
type Query {
    findUser: returnedUser
    getTeams: returnedTeams
    getTeam(_id:ID): returnedTeam
    getCategories(ownerId:String):returnedCategory
    getCategoryName(_id:ID):returnedCategory
    getAllFixedAsset(owner_id:String):returnedFixedAsset
    getFixedAssetByCategory(owner_id:String,category_id:ID):returnedFixedAsset
    getOneFixedAsset(_id:ID):returnedFixedAsset
    getAccounts(owner_id:String):returnedAccount
    getAllExpenses(owner_id:String,date:InputDate):returnedExpence
    getMonthExpenses(owner_id:String,year:String,month:String):returnedExpence

}
```

Σχήμα 5.15: Όλα τα mutations που χρησιμοποιούνται

1. **createUser :** Είναι το ερώτημα που επιστρέφει τον user που είναι συνδεδεμένος αυτή τη στιγμή. Δε δέχεται παραμέτρους, καθώς το αναγνωριστικό token από το keycloak στέλνεται μέσα των headers κάθε φορά που γίνεται ένα ερώτημα από το front στο back . Πριν από κάθε ερώτημα το λογισμικό στέλνει ένα ερώτημα, για το αν ο χρήστης είναι έγκυρος και στην απάντηση που παίρνει βρίσκεται και το profile id που χρησιμοποιούμε ως πεδίο και στη συλλογή των user. Έτσι εκτελεί μία findOne και επιστρέφει το επιθυμητό αποτέλεσμα.

2. **updateUser :** Μέσω της κλήσης της findOneandUpdate πάνω στη συλλογή Users ανανεώνονται τα στοιχεία του χρήστη.

3. **createTeam :** Μέσω της κλήσης της create πάνω στη συλλογή Teams δημιουργείται μία νέα ομάδα με τα στοιχεία που έχουν δοθεί από το χρήστη.

4. **unjoinTeam :** Μέσω της κλήσης της findOneandUpdate πάνω στην συλλογή Teams, σε συνδυασμό με την παράμετρο pull, διαγράφεται από τον πίνακα των members της ομάδας ο χρήστης.Η ομάδα εντοπίζεται με βάση τον χαρακτηριστικό κωδικό της.

5. **joinTeam :** Μέσω της κλήσης της findOneandUpdate πάνω στην συλλογή Teams, σε συνδυασμό με την παράμετρο push, εισάγεται στον πίνακα των members της ομάδας ο χρήστης.Η ομάδα εντοπίζεται με βάση τον χαρακτηριστικό κωδικό της.

6. **updateTeam :** Μέσω της κλήσης της findOneandUpdate πάνω στην συλλογή Teams ενημερώνονται τα στοιχεία της ομάδας.

7. **deleteTeam :** Μέσω της κλήσης της findOneAndDelete, πάνω στη συλλογή Teams, διαγράφεται η ομάδα με βάση το id, το οποίο της αποδίδει η βάση. Πριν όμως διαγραφεί γίνεται find στη συλλογή Accounts, ώστε να βρεθούν οι λογαριασμοί που ανήκαν σε αυτή την ομάδα και μετά με τη χρήση της DeleteMany στη συλλογή Expenses διαγράφονται όσες συναλλαγές αντιστοιχούσαν σε κάθε λογαριασμό της ομάδας, καθώς και ο ίδιος ο λογαριασμός με τη DeleteOne στη συλλογή Accounts. Επίσης, μέσω της find, πάνω στη συλλογή Categories, εντοπίζονται όλες οι κατηγορίες πάγιων περιουσιακών στοιχείων που ανήκαν στην ομάδα και με τη deleteMany στη συλλογή FixedAssets διαγράφονται όλα τα πάγια περιουσιακά στοιχεία της ομάδας και έπειτα με την deleteOne, πάνω στη συλλογή Categories διαγράφεται η κάθε κατηγορία.

8. **createCategory :** Με τη χρήση της create πάνω στη συλλογή Categories δημιουργείται μια νέα κατηγορία με βάση τα στοιχεία που έχει εισαγάγει ο χρήστης.

9. **updateCategory :** Με τη χρήση της findOneAndUpdate στη συλλογή Categories ανανεώνονται τα στοιχεία της εκάστοτε κατηγορίας.

10. **deleteCategory :** Με τη χρήση της findOndeandDelete στη συλλογή Categories διαγράφεται η συγκεκριμένη κατηγορία.Πριν όμως διαγραφεί εκτελείται μια DeleteMany πάνω στη συλλογή FixedAssets ώστε να διαγραφούν τα περιουσιακά στοιχεία που της αντιστοιχούν.

11. **createFixedAsset :** Με τη χρήση της create πάνω στη συλλογή FixedAssets, δημιουργείται ένα νέο πάγιο περιουσιακό στοιχείο με τα στοιχεία που έχει εισαγάγει ο χρήστης.

12. **updateFixedAsset :** Με τη χρήση της findOneAndUpdate πάνω στη συλλογή FixedAssets ανανεώνεται το εκάστοτε περιουσιακό στοιχείο.

13. **deleteFixedAsset :** Με τη χρήση της findOneAndDelete πάνω στη συλλογή FixedAssets διαγράφεται το εκάστοτε περιουσιακό στοιχείο.

14. **updateScenario :** Με τη χρήση της findOneAndUpdate πάνω στη συλλογή Users ή πάνω στη συλλογή Teams ανανεώνεται το εκάστοτε σενάριο.

15. **deleteScenario :** Με τη χρήση της findOneAndUpdate πάνω στη συλλογή Users ή πάνω στη συλλογή Teams διαγράφεται το εκάστοτε σενάριο.Πρίν διαγραφεί γίνεται έλεγχος μέσω μιας find για το αν το σενάριο χρησιμοποιείται σε κάποια συναλλαγή και αν αυτό συμβαίνει δεν τη διαγράφει και επιστρέφει μήνυμα στο χρήστη.

16. **addScenario :** Με τη χρήση της findOneAndUpdate πάνω στη συλλογή Users ή πάνω στη συλλογή Teams προστίθεται ένα νέο σενάριο.

17. **createAccount :** Με τη χρήση της create πάνω στη συλλογή Accounts δημιουργείται ένας νέος λογαριασμός.

18. **updateAccount :** Με τη χρήση της findOneAndUpdate πάνω στη συλλογή Accounts ενημερώνονται τα στοιχεία του εκάστοτε λογαριασμού.

19. **deleteAccount :** Με τη χρήση της findOneAndDelete πάνω στη συλλογή Accounts διαγράφεται ο εκάστοτε λογαριασμός.Ωστόσο πριν διαγραφεί γίνεται έλεγχος με μία find και εάν χρησιμοποιείται σε κάποια συναλλαγή δε διαγράφεται και στέλνει ενημερωτικό μήνυμα στο χρήστη.

20. **createExpense :** Με τη χρήση της create πάνω στην συλλογή Expenses δημιουργείται μία νέα συναλλαγή ανάλογα με τα στοιχεία που έχει εισαγάγει ο χρήστης.Ανάλογα με το αν είναι έσοδο ή έξοδο, με τη findOneAndUpdate σε συνδυασμό με την παράμετρο inc, αυξάνεται ή μειώνεται αντίστοιχα το ποσό του αντίστοιχου λογαριασμού.

21. **updateExpense :** Με τη χρήση της findOneAndUpdate πάνω στην συλλογή Expenses ανανεώνονται τα στοιχεία της εκάστοτε συναλλαγής.Σε περίπτωση που γίνει αλλαγή στο λογαριασμό, η παλιά τιμή προστίθεται αν ήταν έξοδο ή αφαιρείται αν ήταν έσοδο από τον παλιό λογαριασμό και η καινούρια τιμή αφαιρείται ή προστίθεται αντίστοιχα στον καινούριο λογαριασμό.

22. **deleteExpense :** Με τη χρήση της λειτουργίας findOneAndDelete πάνω στην συλλογή Expenses διαγράφεται η εκάστοτε συναλλαγή και ο λογαριαμός στον οποίο αντιστοιχούσε ανανεώνεται.

5.5.2 Ορισμός ενός mutation στον κώδικα

Τα βήματα για να καταφέρει κάποιος να χρησιμοποιήσει ένα mutation είναι τα εξής:

1. Δήλωση του mutation στο αρχείο schemas στο back-end.

2. Ορισμός των ενεργειών του mutation, με άλλα λόγια γράψιμο της συνάρτησης που θα καλείται όταν καλεστεί αυτό το mutation, στο αρχείο resolvers στο back-end.

3. Δήλωση του mutation και των πεδίων που αυτό θα επιστρέφει στο αρχείο Mutation στο front-end.

4. Κλήση του query μέσω της useMutation.

```
    //
    async createAccount(root,args,context){
        const{owner_id,name,price}=args;
        const check = await ValidateUser(context.req.headers.authorization);
        if (check.code!==200) {
            //return Error message for user
            code = Messages.authentication.SesionExpired.code;
            message = Messages.authentication.SesionExpired.msg;
            return {code:code, message:message}
        }else{
            let new_account = await Accounts.create({owner_id:owner_id,name:name,price:price}).catch(err => {
                console.log(err);
                return{
                    code:  Messages.validation.AcountsError.code,
                    message: Messages.validation.AcountsError.msg
                }
            });
            console.log("new_account",new_account);
            return{account:new_account,code: Messages.validation.AcountsOK.code,message:Messages.validation.AcountsOK.msg}

        }
    },
    async updateAccount(root,args,context){
        const{_id,name,price}=args
        const check = await ValidateUser(context.req.headers.authorization);
        if (check.code!==200) {
            //return Error message for user
            code = Messages.authentication.SesionExpired.code;
            message = Messages.authentication.SesionExpired.msg;
            return {code:code, message:message}
        }else{
            let update_account = await Accounts.findOneAndUpdate({_id:mongoose.Types.ObjectId(_id)},{name:name,price:price}).catch(err => {
                console.log(err);
                return{
                    code:  Messages.validation.AcountsError.code,
                    message: Messages.validation.AcountsError.msg
                }
            });
            console.log("update_account",update_account._doc);
            return{account:update_account._doc,code: Messages.validation.AcountsOK.code,message:Messages.validation.AcountsOK.msg}
        }
    },
    async deleteAccount(root,args,context){
```

Σχήμα 5.16: Κώδικας που εκτελούν mutation update και create

```
export const CREATE_ACCOUNT= gql`
mutation createAccount($owner_id:String,$name:String,$price:Float){
    createAccount(owner_id:$owner_id,name:$name, price:$price){
        __typename
    ...on AccountOk{
        account{
        _id
        price
        name
        owner_id
        }
        code
        message
    }
    ...on AccountError{
        code
        message
    }
  }
  }

export const UPDATE_ACCOUNT= gql`
mutation updateAccount($_id:ID,$name:String,$price:Float){
    updateAccount( _id:$_id,name:$name, price:$price){
    ...on AccountOk{
        account{
        price
        name
        _id
        owner_id
        }
    }
    ...on AccountError{
        code
        message
    }
    }

  }
`
```

Σχήμα 5.17: Ορισμός των mutation στο front

```
const [createAccount] =  useMutation(CREATE_ACCOUNT,
    {
        onCompleted(data) {

        if(data.createAccount.code=="AUTH00"){
          alert(data.createAccount.message);

        }else if(data.createAccount.code=="A01"){
          alert(data.createAccount.message);
        }


        }
});
const [updateAccount] =  useMutation(UPDATE_ACCOUNT,
    {
        onCompleted(data) {

        if(data.updateAccount.code=="AUTH00"){
          alert(data.updateAccount.message);

        }else if(data.updateAccount.code=="A03"){
          alert(data.updateAccount.message);


        }
        }
});
```

Σχήμα 5.18: Κλήση στο front μέσω της useMutation

```
const [createAccount] =  useMutation(CREATE_ACCOUNT,
    {
        onCompleted(data) {

        if(data.createAccount.code=="AUTH00"){
          alert(data.createAccount.message);

        }else if(data.createAccount.code=="A01"){
          alert(data.createAccount.message);
        }


        }
});
const [updateAccount] =  useMutation(UPDATE_ACCOUNT,
    {
        onCompleted(data) {

        if(data.updateAccount.code=="AUTH00"){
          alert(data.updateAccount.message);

        }else if(data.updateAccount.code=="A03"){
          alert(data.updateAccount.message);


        }
        }
});
```

Σχήμα 5.19: Χρήση του mutation σε συνάρτηση

5.5.3 Επεξήγηση delete, create και update

Σε όλα τα mutations χρησιμοποιούνται πέντε συναρτήσεις συμπληρωματικά με τη find που ήδη έχουμε αναλύσει. Αυτές είναι οι εξής:

- **findOneAndUpdate:** Δέχεται όπως και η find 3 παραμέτρους αλλά αυτή τη φορά η ενδιάμεση παράμετρος καθορίζει την αλλαγή που θα γίνει. Μπορεί να δίνεται μόνο

ένα καινούριο έγγραφο, το οποίο θα αντικαθιστά το παλιό, ή να δίνονται οι τιμές συγκεκριμένων πεδίων οι οποίες θα αντικαθιστούν τις παλιές ή τέλος να καλεστεί κάποια εντολή από το aggregate pipeline. Οι εντολές που χρησιμοποιήθηκαν στην εφαρμογή είναι η push και η pull οι οποίες βάζουν μία τιμή στο έγγραφο ή την αφαιρούν.

- **findOneAndDelete:** Δέχεται filter και options. Στην ουσία, με βάση το τι θα δοθεί στο filter, γίνεται η αναζήτηση του εγγράφου και έπειτα αυτό διαγράφεται.
- **DeleteMany:** Δέχεται filter όπου δίνονται οι πληροφορίες που πρέπει να έχουν τα έγγραφα που θέλουμε να διαγραφούν. Μπορεί να διαγράψει παραπάνω από ένα έγγραφο.
- **DeleteOne:** Όμοια με τη DeleteMany μόνο που διαγράφει το πρώτο έγγραφο που ταιριάζει στα filters.
- **create:** Δέχεται ως είσοδο ένα document και το προσθέτει στη βάση μαζί με το αντίστοιχο id.

5.5.4 Resolvers

Οι resolvers είναι στην ουσία συναρτήσεις, που είναι υπεύθυνες να αναθέτουν τιμές σε τύπους που έχουν οριστεί στο σχήμα. Μέσα στους resolvers αρχικά ορίζεται η συμπεριφορά των unions δηλαδή γράφεται μία συνάρτηση που καθορίζει ανάλογα με τα πεδία που έχει κάποιο επιστρεφόμενο αντικείμενο τι τύπου είναι. Επίσης γράφονται αναλυτικά οι συναρτήσεις για όλα τα mutations και όλα τα queries. Το κοινό μεταξύ σε mutations και queries είναι οι παράμετροι που λαμβάνουν. Συγκεκριμένα έχουν τα root, args, context. Το root έχει να κάνει με την κλήση κάποιου root resolver, κάτι που δεν έχει χρησιμοποιηθεί στην τρέχουσα εφαρμογή. Το context έχει να κάνει με τα δεδομένα που στέλνονται σαν headers κάθε φορά που καλείται κάποιο ερώτημα. Στη δική μας εφαρμογή από το context λαμβάνουμε το token. Τέλος τα args έχουν τις τιμές των παραμέτρων που έχουν δοθεί στο κάθε ερώτημα.

Κεφάλαιο 6

Αυθεντικοποίηση και εξουσιοδότηση

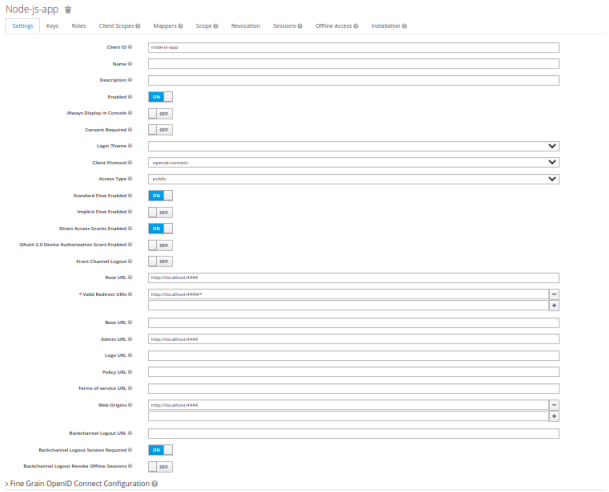
Μετά το κομμάτι των βάσεων δεδομένων ακολουθεί το κομμάτι της αυθεντικοποίησης του χρήστη αλλά και της ασφάλειας της εφαρμογής. Ένα λογισμικό δεν κινδυνεύει μόνο από επιθέσεις, όπως αυτές της διάρρηξης των κωδικών εισόδων του χρήστη, αλλά υπάρχουν πολύ πιο σοβαροί κίνδυνοι, όπως είναι οι κακόβουλοι χρήστες, που θα προσπαθήσουν να υποβάλλουν ερωτήματα στη βάση, για να εισάγουν είτε στοιχεία που δεν θα έπρεπε να υπάρχουν εκεί, είτε ακόμα και να προσπαθήσουν να διαγράψουν ολόκληρη τη βάση. Χρειαζόμαστε λοιπόν έναν ασφαλή τρόπο να διασφαλίσουμε πως τα ερωτήματα που δέχεται ο εξυπηρετητής προέρχονται αποκλειστικά από επιβεβαιωμένους χρήστες της εφαρμογής. Αυτή η ασφάλεια μπορεί να επιτευχθεί με τη χρήση ενός ενδιάμεσου λογισμικού, όπου θα κρατάει αποθηκευμένα τα στοιχεία των χρηστών και τους κωδικούς πρόσβασης και που θα επιβεβαιώνει μέσω ενός χαρακτηριστικού αλφαριθμητικού πως ο χρήστης είναι νόμιμος. Το λογισμικό αυτό ονομάζεται Keycloak.

6.1 Keycloak

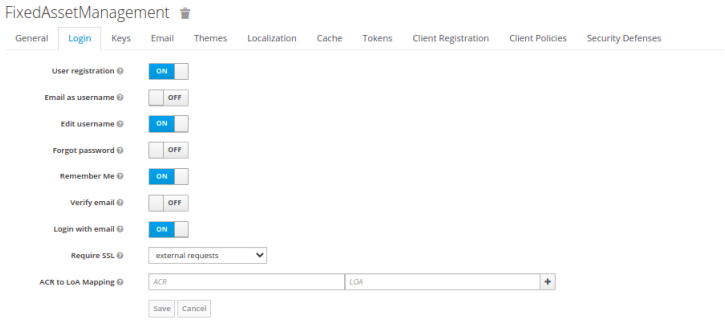
Πρόκειται για ένα εργαλείο διαχείρισης και πρόσβασης των χρηστών σε διάφορες εφαρμογές. Προσφέρει λειτουργίες εγγραφής(sign up), σύνδεσης (login) και αποσύνδεσης. Δίνει τη δυνατότητα να ορίσει κάποιος ομάδες χρηστών με διαφορετικά δικαιώματα η κάθε μία. Για παράδειγμα είναι δυνατόν κάποιοι χρήστες να οριστούν ως διαχειριστές και άλλοι ως απλοί χρήστες. Τα δικαιώματα αυτά μπορούμε να τα ελέγχουμε αργότερα, ακριβώς πριν γίνει το ερώτημα στη βάση δεδομένων και με τον τρόπο αυτό να εξασφαλίσουμε πως χρήστες που δεν επιθυμούμε δεν θα διαπεράσουν τη βάση. Επίσης προσφέρει λειτουργίες σύνδεσης

με το μέσα κοινωνικής δικτύωσης, αλλά και επιβεβαίωση μέσω αποστελλόμενου email. Επίσης προσφέρει έτοιμα templates για την εγγραφή του χρήστη δίνοντας μας όμως παράλληλα και τη δυνατότητα να τα τροποποιήσουμε. Με άλλα λόγια προσφέρει έναν τρόπο να ασφαλίσει κάποιος την εφαρμογή του δωρεάν και έχει έτοιμες κάποιες λειτουργίες όπως είναι τα cookies, τα session ή ακόμη και η κρυπτογράφηση του κωδικού που για να τα φτιάξει κάποιος από το μηδέν και να βεβαιωθεί πως προσφέρουν ασφάλεια θέλει αρκετό κόπο. Για να χρησιμοποιήσει κάποιος το keycloak πρέπει να τρέξει την κονσόλα του και να συνδεθεί ως admin ώστε να μπορέσει να κάνει τις κατάλληλες ρυθμίσεις. Πιο συγκεκριμένα, αφού συνδεθούμε στο Keycloak (τώρα που η εφαρμογή τρέχει σε τοπικό δίκτυο είναι διαθέσιμο μέσω του `http://localhost:8080/admin/`) πρέπει να δημιουργήσουμε ένα Realm με το όνομα που επιθυμούμε για να το χρησιμοποιήσουμε στην εφαρμογή μας. Έπειτα, στη σελίδα Login, μπορεί κάποιος να επιλέξει τις λειτουργίες που θα προσφέρει κατά τη σύνδεση του χρήστη η εφαρμογή. Στην καρτέλα email γίνονται οι κατάλληλες ρυθμίσεις σε περίπτωση που θέλουμε η εφαρμογή να στέλνει email στο χρήστη. Στη σελίδα Themes μπορεί όποιος θέλει να συμπεριλάβει το δικό του θέμα για τα γραφικά κατά τη διάρκεια της σύνδεσης. Στην τρέχουσα εφαρμογή επιλέχθηκε η χρήση του default θέματος. Ακόμη, από την καρτέλα tokens μπορούμε να ελέγξουμε κάθε πόσο θα λήγει το token που αντιστοιχεί στον κάθε χρήστη. Αυτό το token είναι στην ουσία που αυξάνει την ασφάλεια, διότι πρόκειται για ένα τυχαίως παραγόμενο αλφαριθμητικό το οποίο παράγεται για τον κάθε χρήστη και έχει χρονικό όριο λήξης. Μέσω αυτού του αλφαριθμητικού μπορούμε να ελέγξουμε κάθε φορά αν ο χρήστης είναι επιθυμητός, καθώς είναι αρκετά δύσκολο ακόμα και κάποιος κακόβουλος χρήστης να καταφέρει να βρει και να αντιγράψει κάποια από τα ισχύοντα token κάθε χρονική στιγμή. Στη συνέχεια ακολουθούν οι ρυθμίσεις για τους clients. Για το keycloak ως πελάτες θεωρούνται όσοι του θέτουν ερωτήματα, δηλαδή στη δική μας περίπτωση και ο εξυπηρετητής του back και ο πελάτης του front . Θα ορίσουμε λοιπόν 2 πελάτες, τους node-js-app και react-web-app . Οφείλουμε στις πληροφορίες να εισάγουμε και τους συνδέσμους στους οποίους είναι διαθέσιμη η εφαρμογή, αντίστοιχα για το front και το back. Στην καρτέλα Roles μπορούμε να ορίσουμε ρόλους, όπως το αν κάποιος θα είναι admin και μετά να γυρίσουμε πίσω στους clients και να κάνουμε την αντιστοιχία που χρειάζεται. Στην καρτέλα Authentication γίνονται οι ρυθμίσεις σχετικά με τα cookies και τα session. Επίσης από την καρτέλα users μπορούμε να δούμε ποιοι χρήστες έχουν κάνει εγγραφή. Οι ρυθμίσεις που μπορεί κάποιος να κάνει είναι πολλές και απαιτούν ώρες αφιερωμένες στο documentation του keycloak. Στις

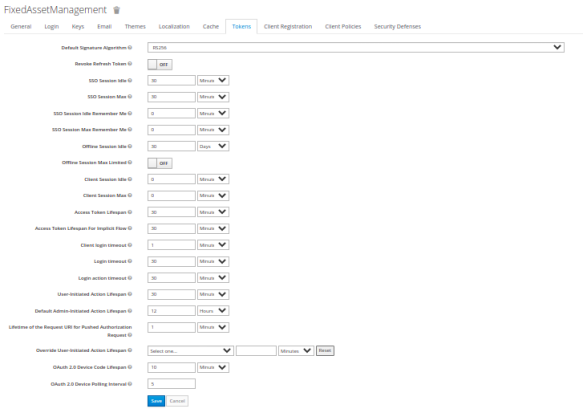
εικόνες παρακάτω φαίνονται κάποιες από τις βασικές ρυθμίσεις που έγιναν στην τρέχουσα εφαρμογή.[43] [44]



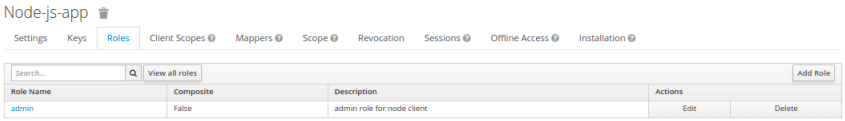
Σχήμα 6.1: Γενικές ρυθμίσεις για το Realm



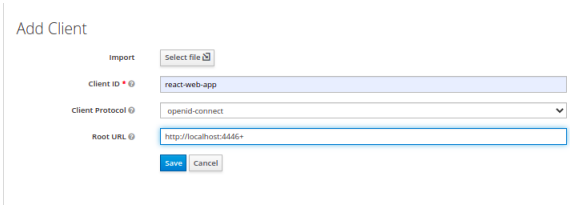
Σχήμα 6.2: Ρυθμίσεις για το Login



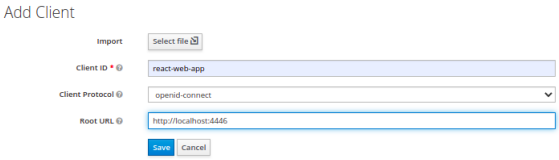
Σχήμα 6.3: Ρυθμίσεις για τα Tokens



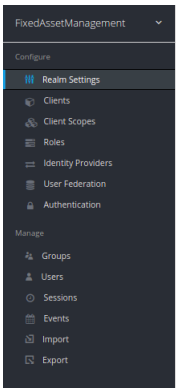
Σχήμα 6.4: Ρυθμίσεις για τους ρόλους



Σχήμα 6.5: Ρυθμίσεις για τον client του back



Σχήμα 6.6: Ρυθμίσεις για τον client του front



Σχήμα 6.7: Keycloak sidebar

6.1.1 Authentication,Cookies,Session

Πριν προχωρήσουμε στην επεξήγηση του κώδικα και των λειτουργικών απαιτήσεων που επιτεύχθηκαν με το keycloak καλό είναι να κάνουμε μια αναφορά στη βασική ορολογία που σχετίζεται με την αυθεντικοποίηση του χρήστη.

- **Authentication:** Στα ελληνικά μεταφράζεται ως αυθεντικοποίηση. Πρόκειται για τη διαδικασία κατά την οποία το σύστημα ελέγχει την ταυτότητα του χρήστη. Ο έλεγχος αυτός πραγματοποιείται συνήθως με την αντιστοίχιση στοιχείων που ο χρήστης έχει παραχωρήσει στο λογισμικό. Η πιο γνωστή μέθοδος είναι αυτή του συνδυασμού ενός user name με το password.

- **HTTP Cookies:** Τα cookies χρησιμοποιούνται για να αποθηκεύουν πληροφορίες σχετικά με την κίνηση του χρήστη στον παγκόσμιο ίστο. Συγκρατούν πληροφορίες σχετικά με τα ενδιαφέροντα του, ώστε να του προωθούν κατάλληλες διαφημίσεις ή και προτεινόμενες ιστοσελίδες. Ωστόσο τα HTTP Cookies είναι μια υποκατηγορία που εξειδικεύεται στη μέθοδο του Authentication. Χάρη στα cookies οι χρήστες μπορούν να κρατάν αποθηκευμένους τους κωδικούς τους για τη σύνδεση τους σε διάφορες ιστοσελίδες.

- **Session:** Ως session αναφέρεται ο χρόνος που περνάει ο χρήστης σε μία ιστοσελίδα. Μπορεί να ρυθμιστεί να κρατάει συγκεκριμένο χρονικό διάστημα, στο πέρας του οποίου θα ζητάει ξανά από το χρήστη την επιβεβαίωση των στοιχείων του.

- **Authorization:** Είναι η διαδικασία παραχωρήσης δικαιωμάτων σε κάποιον. Στο χώρο του διαδικτύου έχει να κάνει με το περιεχόμενο που επιτρέπεται να βλέπει ο χρήστης μετά την αυθεντικοποίηση του.

6.2 Σύνδεση στο Keycloak

Όπως φαίνεται και στις εικόνες παρακάτω είναι σημαντικό για τη σύνδεση του Keyckloak με την εφαρμογή να χρησιμοποιηθεί ο σύνδεσμος στον οποίο είναι διαπεράσιμο το Keycloak, καθώς και να δοθεί το όνομα του πελάτη που δημιουργήσαμε στο Realm, στο οποίο αντιστοιχεί ο εξυπηρετητής μας. Αντίστοιχη είναι και η διαδικασία από τη μεριά του πελάτη. Στο δεύτερο στιγμιότυπο φαίνεται η διαδικασία υποβολής ερωτήματος στο keycloak μέσω του axios. Η παράμετρος που χρειάζεται να λάβει αντιστοιχεί στο token, το οποίο έχει δημιουργηθεί για το συγκεκριμένο session και η οποία στέλνεται μαζί με τα δεδομένα, όταν ο χρήστης υποβάλει ένα ερώτημα στη βάση. Πριν λοιπόν εκτελεστεί το οποιοδήποτε ερώτημα στη βάση, γίνεται το ερώτημα στο keycloak, όπου επιστρέφει διάφορες πληροφορίες σχετικά με τα στοιχεία του χρήστη, του ρόλου που του αντιστοιχεί αλλά και έναν κωδικό

επιτυχίας, ο οποίος χρησιμοποιείται για να ελεγχθεί η εγκυρότητά του. Σε περίπτωση που το απεσταλμένο token έχει λήξει, η εφαρμογή ειδοποιεί το χρήστη ώστε να ξανασυνδεθεί, σε αντίθετη περίπτωση συνεχίζει με την εκτέλεση του ερωτήματος.

```
var session = require('express-session');
var Keycloak =require('keycloak-connect');
const chalk =require('chalk');

let keycloak;
var keycloakConfig={
    url: "http://localhost:8080/",
    realm: "FixedAssetManagement",
    clientId: "node-js-app"

};
function initKeycloak(){
    if(keycloak){
        console.log('Returning existing keycloak instance');
        return keycloak;
    }else{
        console.log("Initializing Keycloak");
        var memoryStore=new session.MemoryStore();

        keycloak= new Keycloak({
            store:memoryStore,
            secret:'any_key',
            resave :false,
            saveUnitialized:true
        },keycloakConfig);
        return keycloak
    }
}
module.exports={
    initKeycloak
};
```

Σχήμα 6.8: Σύνδεση στο Keycloak από τη μεριά του Server

```
var return_data ={
    data:null,
    code:""
};
module.exports =  async function(token){
    console.log("TOKEN",token);
    const  data  = await axios.get(
        'http://localhost:8080/realms/FixedAssetManagement/protocol/openid-connect/userinfo',{
        headers: {
            Authorization: token,
        }
        }
    ).catch(err=>{
        return_data={
            data:null,
            code:err.response.status
        }
        return(return_data)
    }
    );
    return_data={
        data:data,
        code:data.status
    }
      return(return_data)
}
```

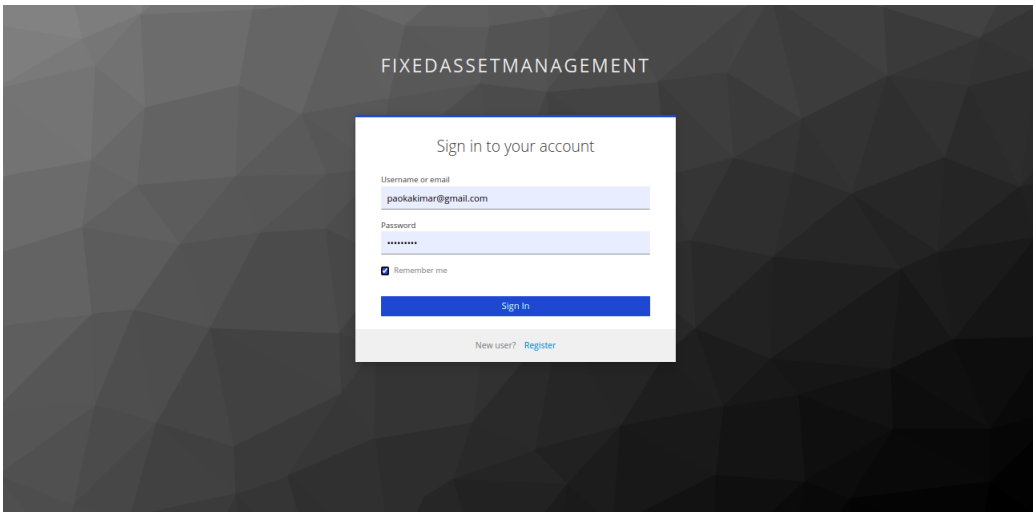
Σχήμα 6.9: Στιγμιότυπο κώδικα από επικοινωνία με το Keycloak

6.3 Σύνδεση χρήστη-LOGIN

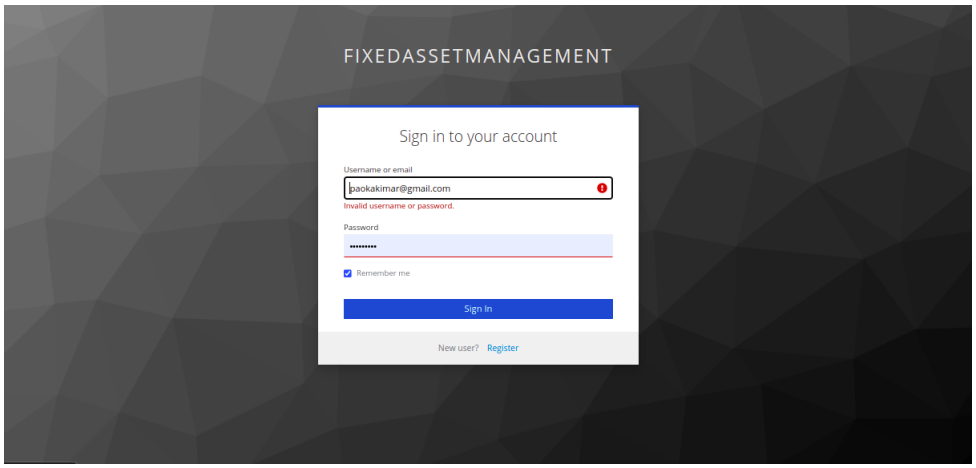
Ο χρήστης μόλις επιλέξει login από τη μπάρα πλοήγησης της εφαρμογής, ανακατευθύνεται στο keycloak, όπου έχει τη δυνατότητα να εισάγει το email και τον κωδικό πρόσβασης. Εάν αλλάξει γνώμη υπάρχει και η επιλογή να ανακατευθυνθεί στη σελίδα εγγραφής χρήστη. Άμα εισάγει λανθασμένα στοιχεία του εμφανίζεται μήνυμα λάθους. Σε περίπτωση που καταφέρει να βάλει τα σωστά στοιχεία, κατευθύνεται στη σελίδα που είχε δοθεί ως παράμετρος, όταν καλέστηκε η συνάρτηση για το login. Συγκεκριμένα η συνάρτηση αυτή ανήκει στις διαθέσιμες βιβλιοθήκες που διαθέτει η React και συσχετίζονται με το Keycloak και καλείται ως εξής

```
const { keycloak } = useKeycloak();
```

```
keycloak.login({redirectUri:"http://localhost:4446/"});
```



Σχήμα 6.10: Σελίδα για είσοδο του χρήστη στην εφαρμογή



Σχήμα 6.11: Περίπτωση εισόδου λανθασμένων στοιχείων

Όσον αφορά τη μεριά του client με το που συνδεθεί ο χρήστης χρησιμοποιεί μια συνάρτηση η οποία αποθηκεύει δεδομένα στην τοπική μνήμη του διακομιστή, αποθηκεύει εκεί το token, που αντιστοιχεί στην τρέχουσα σύνδεση, για να μπορεί να το στέλνει στον εξυπηρετητή, αλλά και καλεί το αντίστοιχο ερώτημα για να ανακτήσει τα δεδομένα του χρήστη .

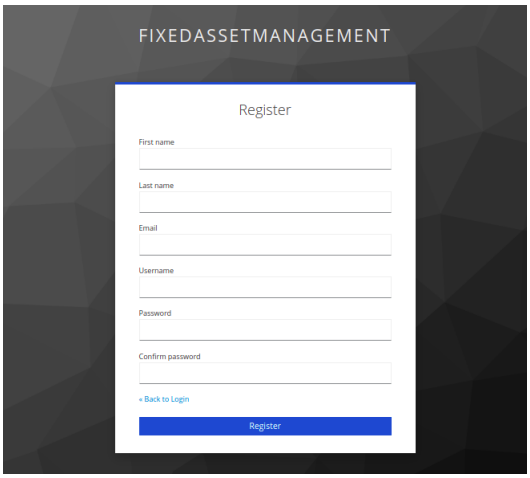
6.4 Εγγραφή χρήστη - SIGNUP

Αντίστοιχα με τη διαδικασία σύνδεσης ο χρήστης εισάγει κάποιες βασικές πληροφορίες για αυτόν. Η εγγραφή γίνεται βασισμένη στο email, οπότε δεν μπορεί ένας χρήστης με το

ίδιο email να κάνει εγγραφή δύο διαφορετικές φορές. Η συνάρτηση που καλείται για να οδηγηθούμε στη σελίδα της εγγραφής είναι η εξής

```
keycloak.register();
```

Μετά την εγγραφή γίνεται το αντίστοιχο ερώτημα στη βάση, ώστε να καταχωρηθούν και εκεί τα στοιχεία του .



Σχήμα 6.12: Σελίδα εγγραφής του χρήστη στην εφαρμογή

6.5 Αποσύνδεση χρήστη - LOGOUT

Ανά πάσα στιγμή ο χρήστης μπορεί να επιλέξει από το μενού να κάνει αποσύνδεση. Τότε οι διαδικασίες που γίνονται στην εφαρμογή είναι οι εξής. Καθαρίζει η τοπική μνήμη ώστε να μην μείνουν τα στοιχεία της προηγούμενης σύνδεσης. Επίσης καλείται η συνάρτηση

```
keycloak.logout({redirectUri: 'http://localhost:4446'});
```

η οποία ενημερώνει και το keycloak πως ο χρήστης αποσυνδέθηκε ώστε να απενεργοποιηθεί η ισχύς του token και να διακοπεί το session.

Κεφάλαιο 7

Το γραφικό περιβάλλον και οι λειτουργικές απαιτήσεις

Εφόσον λοιπόν εξηγήθηκαν αναλυτικά οι βάσεις δεδομένων και στη ουσία η δουλειά που γίνεται στο back-end, και αφού αναλύθηκε και το ενδιάμεσο λογισμικό που χρησιμοποιήθηκε για την αυθεντικοποίηση, έφτασε η ώρα να δώσουμε βάση στον τρόπο λειτουργίας του front-end, στο πως μοιάζει το γραφικό περιβάλλον της εφαρμογής και ποιες οι δυνατότητες που έχει ο χρήστης αλληλεπιδρώντας με αυτό. Κλείνοντας με αυτό το κεφάλαιο θα έχει ολοκληρωθεί η παρουσίαση του λογισμικού από όλες τις μεριές.

7.1 React και HTML

Αρχικά, πριν παρουσιαστούν όλες οι λειτουργίες, θα γίνει αναφορά στις βασικές συναρτήσεις που προσφέρει η React ή που χρησιμοποιούνται από την html και κάνει εφικτή την επικοινωνία και την αλληλεπίδραση με το χρήστη.

- **useState** : Είναι ένα hook της React όπως ονομάζεται και το οποίο μας δίνει τη δυνατότητα να χρησιμοποιούμε τοπικές μεταβλητές σε λειτουργικά στοιχεία . Πιο συγκεκριμένα η σύνταξη της γίνεται ως εξής

  ```
  const[new_value, Setnewvalue]=useState("example");
  ```

  Με τον τρόπο αυτό δίνεται η δυνατότητα, όταν θέλουμε να αλλάξουμε την τιμή του, να καλούμε τη δεύτερη παράμετρο του πίνακα και μέσα στις παρενθέσεις να βάζουμε την επιθυμητή τιμή για την πρώτη παράμετρο. Η ανάθεση τιμής μπορεί να καλεστεί

είτε από το ίδιο component, στο οποίο ορίζεται η μεταβλητή, είτε από κάποιο άλλο component . Αυτό αποδεικνύεται χρήσιμο καθώς διευκολύνει τη χρήση των λεγόμενων pop up παραθύρων, είτε όσον αφορά την εμφάνιση και εξαφάνιση τους, είτε όσον αφορά τις ενέργειες που θα εκτελέσουν.[45]

- **useEffect** : Είναι ένα αρκετά χρήσιμο hook καθώς χρησιμοποιείται για την ανάκτηση δεδομένων, την αυτόματη ενημέρωση του component καθώς και για την χρήση χρονομέτρων. Βοηθάει για παράδειγμα όταν θέλουμε να εμφανίσουμε ένα στοιχείο μόνο για ορισμένο χρονικό διάστημα. [46]

- **onClick** : Πρόκειται για τη συνάρτηση που ανιχνεύει πότε ο χρήστης επιλέγει κάποιο στοιχείο ή συγκεκριμένα πότε πατάει πάνω του. Μέσα σε αυτή τη συνάρτηση καλούμε κάθε φορά τη συνάρτηση που επιθυμούμε να εκτελεστεί, αν ο χρήστης επιλέξει το συγκεκριμένο στοιχείο.

- **onChange** : Πρόκειται για τη συνάρτηση που ανιχνεύει πότε αλλάζει η τιμή σε ένα πεδίο εισόδου ή σε ένα πεδίο εισαγωγής κειμένου. Είναι αρκετά χρήσιμο σε φόρμες συμπλήρωσης στοιχείων καθώς και κάθε φορά που ο χρήστης θέλει να επεξεργαστεί κάποια πληροφορία. Χρησιμοποιείται πολύ συχνά σε συνδυασμό με την useState, έτσι ώστε να κρατάμε σε κάποια μεταβλητή την τρέχουσα ισχύουσα τιμή.

- **useNavigate** : Πρόκειται για το hook που μας βοηθάει να ανακατευθυνθούμε σε μία διαφορετική σελίδα. Συνεργάζεται με τα Routes.

- **Routes** :Πρόκειται για ένα βασικό κομμάτι της εφαρμογής, μέσω του οποίου ορίζονται όλα τα πιθανά pathos στα οποία μπορεί να οδηγηθεί κάποιος και ορίζουν ποια σελίδα θα φορτώνεται κάθε φορά.

```
<Route path="/addFixedAsset/:category/:id"
element={<AddFixedAsset />} />
<Route path="/updateFixedAsset/:id"
element={<UpdateFixedAsset />} />
<Route path="/expences" element={<Expenses/>} />
<Route path="/categories" element={ <Categories/>}/>
<Route path="/error" element={<Error />} />
<Route path="/settings" element={<Settings />} />
```

```
<Route path="/accounts" element={<Accounts />} />
<Route path ="/fixedAsset/:category/:id"
element = {<FixedAsset />} />
<Route path="/teams" element={<Teams/>}></Route>
<Route path="/" element={<MainPage />} />
```

Παραπάνω φαίνεται το κομμάτι της δρομολόγησης για την τρέχουσα εφαρμογή. Οι τιμές που έχουν μπροστά τους άνω κάτω τελεία σημαίνει πώς θα λαμβάνουν τιμή κάθε φορά διαφορετική και στη σελίδα που έχει καλεστεί θα μπορούμε να λάβουμε την τιμή τους με τη χρήση της useParams.[47]

- **useParams** : Είναι το hook που χρησιμοποιείται μετά την δρομολόγηση και ταιριάζει τις τιμές με τα αντίστοιχα κλειδιά τους. Έτσι μπορούμε να λάβουμε κάθε φορά τη διαφορετική τιμή που δίνεται στο σύνδεσμο. Αυτό είναι πολύ χρήσιμο ιδίως όταν για παράδειγμα μία σελίδα αλλάζει τις τιμές της ανάλογα με το προϊόν που αντιστοιχεί. Με το να μπορούμε να πάρουμε κάθε φορά το αναγνωριστικό id της βάσης, ώστε να φορτώσουμε τα αντίστοιχα δεδομένα καλώντας το ερώτημα, είναι πολύ βοηθητικό.

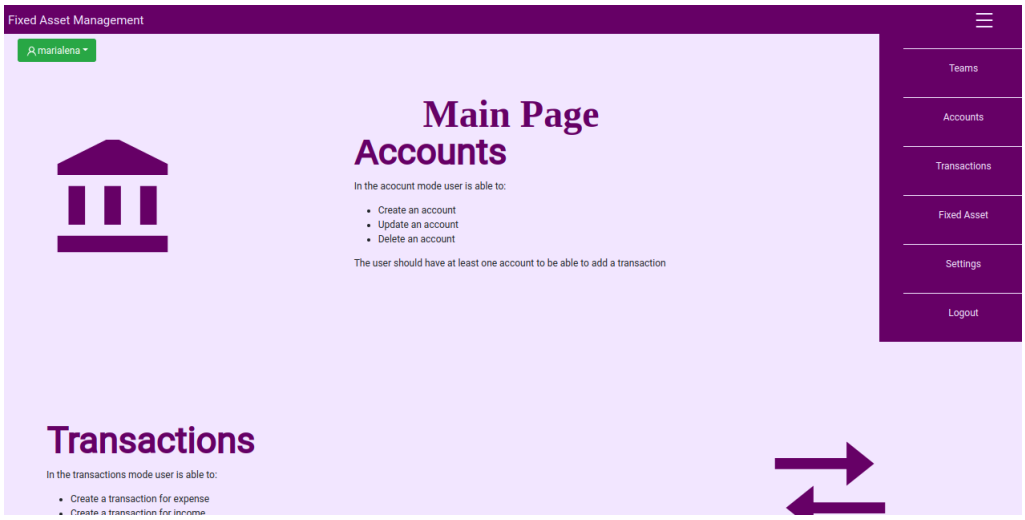
- **props** : Δεν πρόκειται για συνάρτηση αλλά για τη δυνατότητα να περνάμε παραμέτρους από το ένα component στο άλλο.

Αυτές είναι οι βασικές συναρτήσεις και τρόποι ανάκτησης μεταβλητών που χρησιμοποιήθηκαν για να στηθεί ένα γραφικό περιβάλλον με αλληλεπίδραση. Επιμέρους βιβλιοθήκες για πιο εξειδανικευμένες λειτουργίες θα αναφερθούν στη συνέχεια.

7.2 Κεντρική σελίδα

Σε αυτή τη σελίδα δίνεται μια σύντομη περιγραφή των λειτουργιών της εφαρμογής. Από εδώ ο χρήστης, όταν ακόμα δεν είναι συνδεδεμένος, έχει τη δυνατότητα να συνδεθεί επιλέγοντας το login ή το singup κουμπί πάνω δεξιά στη μπάρα πλοήγησης. Με το που καταφέρει, να συνδεθεί μπορεί πλέον να έχει πρόσβαση στο dropdown μενού που εμφανίζεται στα δεξιά εάν πατήσει το κουμπί με τις 3 μπάρες. Εκεί του δίνεται η δυνατότητα να επιλέξει σε ποια από τις λειτουργίες θέλει να πλοηγηθεί ή απλώς να κάνει αποσύνδεση.

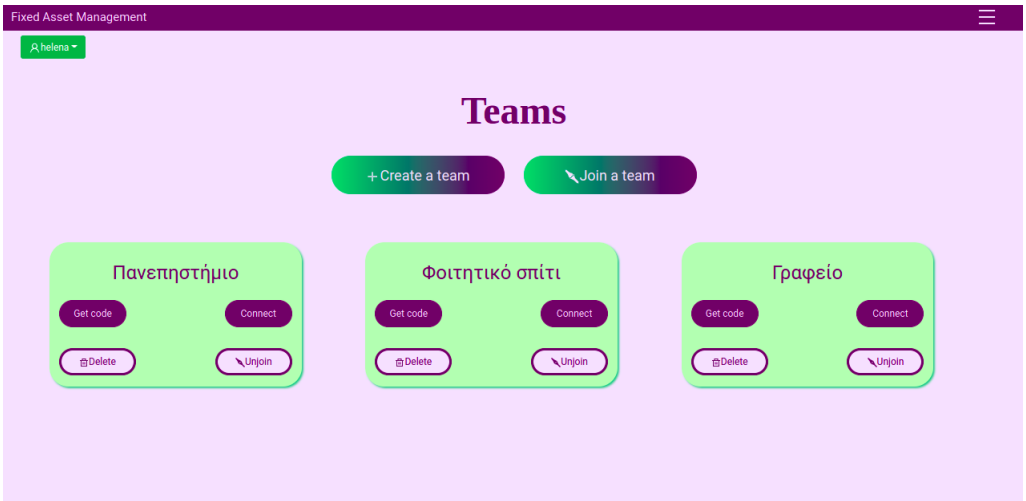
Σχήμα 7.1: Main page πριν συνδεθεί ο χρήστης



Σχήμα 7.2: Dropdown μενού για την περιήγηση στην εφαρμογή

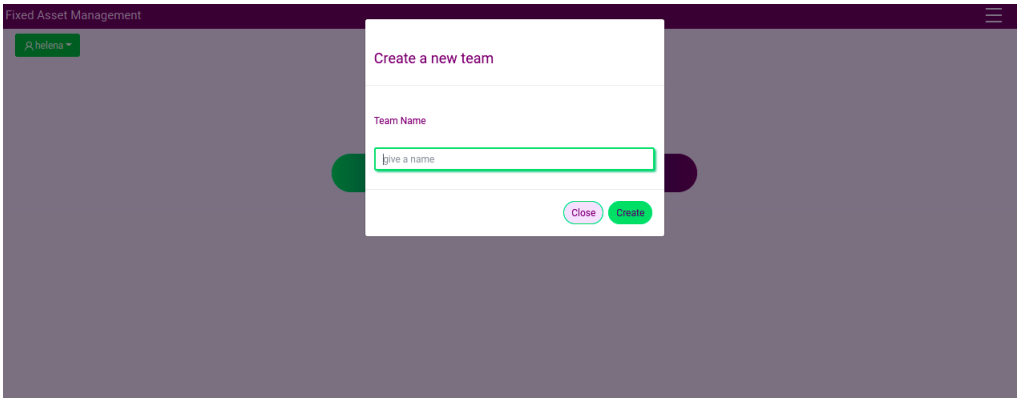
7.3 Η λειτουργία Teams

Ο χρήστης μόλις επιλέξει να κατευθυνθεί στα Teams αντικρίζει μία σελίδα όπου υπάρχουν όλες οι ομάδες που είναι εγγεγραμένος ή έχει δημιουργήσει ο ίδιος.



Σχήμα 7.3: Παράσταση όλων των ομάδων που είναι εγγεγραμένος ο χρήστης

Του δίνεται η επιλογή να δημιουργήσει μία νέα ομάδα, δίνοντας το όνομα που αυτή θα έχει.



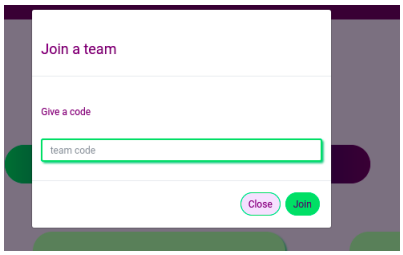
Σχήμα 7.4: Αναδυόμενο παράθυρο για την δημιουργία ομάδας

Επίσης, αν έχει τον κωδικό κάποιας προϋπάρχουσας ομάδας, μπορεί να συνδεθεί σε αυτή. Ο κωδικός έχει τη λογική πως τα ήδη υπάρχοντα μέλη σε μία ομάδα μπορούν να προσκαλέσουν κάποιον άλλον να ενταχτεί γνωστοποιώντας του τον κωδικό. Επίσης, στα κουτιά όπου φαίνεται ξεχωριστά η κάθε ομάδα, του δίνεται επιλογή είτε να απεγγραφεί από την ομάδα μέσω του κουμπιού unjoin, είτε να διαγράψει την ομάδα και μαζί με αυτή και όλα τα περιουσιακά στοιχεία, τους λογαριασμούς, τις κατηγορίες και τις συναλλαγές που είχαν καταχωρηθεί. Μπορεί ακόμη να επιλέξει να πάρει τον κωδικό της ομάδας. Ο μοναδικός κωδικός δημιουργείται μόλις δημιουργηθεί η ομάδα και χρησιμοποιεί τον εξής κώδικα παραγωγής μοναδικών αλφαριθμητικών

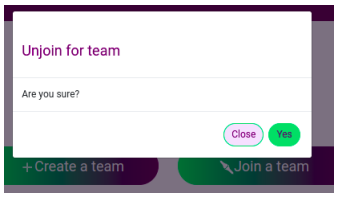
```
const uid = new ShortUniqueId({ length: 10 });
```

```
const unique_code = uid();
```

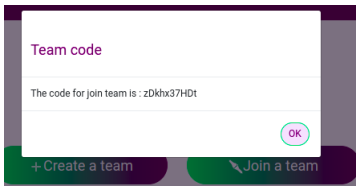
Τέλος μέσω του connect να συνδεθεί στην ομάδα και να περιηγηθεί στην εφαρμογή ως μέλος αυτής.



Σχήμα 7.5: Αναδυόμενο παράθυρο για την εγγραφή σε κάποια ομάδα με βάση τον κωδικό της

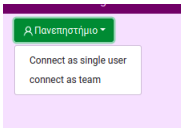


Σχήμα 7.6: Αναδυόμενο παράθυρο για επιβεβαίωση απεγγραφής



Σχήμα 7.7: Αναδυόμενο παράθυρο για τον κωδικό της ομάδας

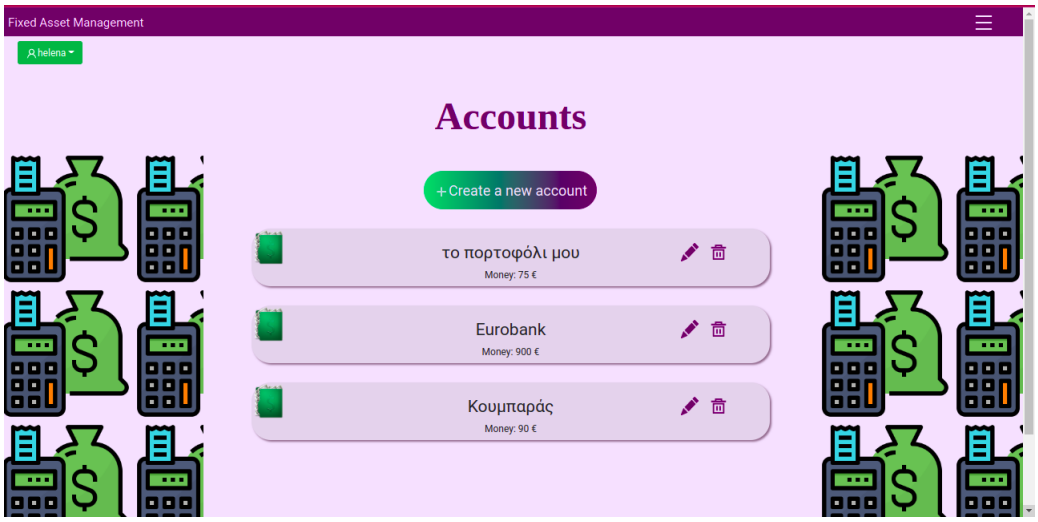
Από τη στιγμή που κάποιος αλλάξει τη σύνδεση του σε ομαδική, στο πράσινο σύμβολο πάνω αριστερά, όπου μέχρι τώρα φαινόταν το username του, πλέον φαίνεται το όνομα της ομάδας. Πατώντας πάνω σε αυτό το κουμπί μπορεί ανά πάσα στιγμή να διαλέξει να ξαναγίνει ατομική η σύνδεσή του ή από οποιαδήποτε άλλη σελίδα, μπορεί να επιλέξει το connect as team για να πλοηγηθεί πίσω στη σελίδα με τις ομάδες και να επιλέξει όποια θέλει.



Σχήμα 7.8: Κουμπί επιλογής σύνδεσης

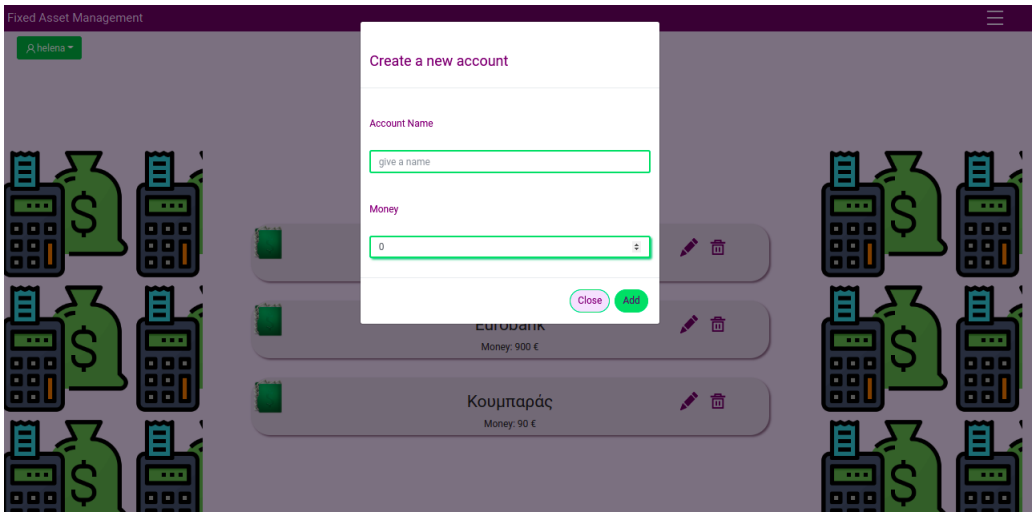
7.4 Η λειτουργία accounts

Όταν κάποιος κατευθύνεται στη σελίδα accounts μπορεί να δεί τους λογαριασμούς του ή της ομάδας του αν είναι συνδεδεμένος ως ομάδα.

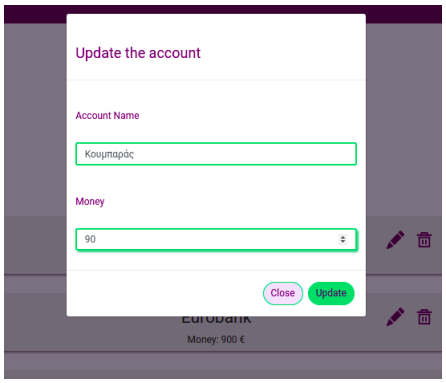


Σχήμα 7.9: Εμφάνιση όλων των λογαριασμών

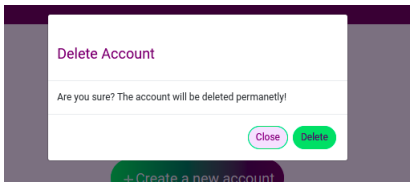
Οι λογαριασμοί αρχικοποιούνται με ένα όνομα και ένα ποσό και στη συνέχεια χρησιμεύουν όταν κάποιος προσθέτει μία νέα συναλλαγή ώστε ανάλογα άμα πρόκειται για έξοδο η τιμή τους μειώνεται, εάν πρόκειται για έσοδο η τιμή τους αυξάνεται. Και πάλι, όπως και στις ομάδες, υπάρχουν αναδυόμενα παράθυρα για την επιβεβαίωση διαγραφής ή την ενημέρωση του λογαριασμού, που ενεργοποιούνται από τα αντίστοιχα εικονίδια πάνω στο component του λογαριασμού, αλλά και ένα ανδυώμενο παράθυρο για την προσθήκη που ενεργοποιείται από το κουμπί προσθήκης. Είναι σημαντικό να αναφερθεί πως, αν ο χρήστης δοκιμάσει να διαγράψει κάποιο λογαριασμό που χρησιμοποιείται ήδη σε συναλλαγή, η εφαρμογή δεν του το επιτρέπει.



Σχήμα 7.10: Αναδυόμενο παράθυρο για τη δημιουργία νέου λογαριασμού



Σχήμα 7.11: Αναδυόμενο παράθυρο για την επεξεργασία λογαριασμού

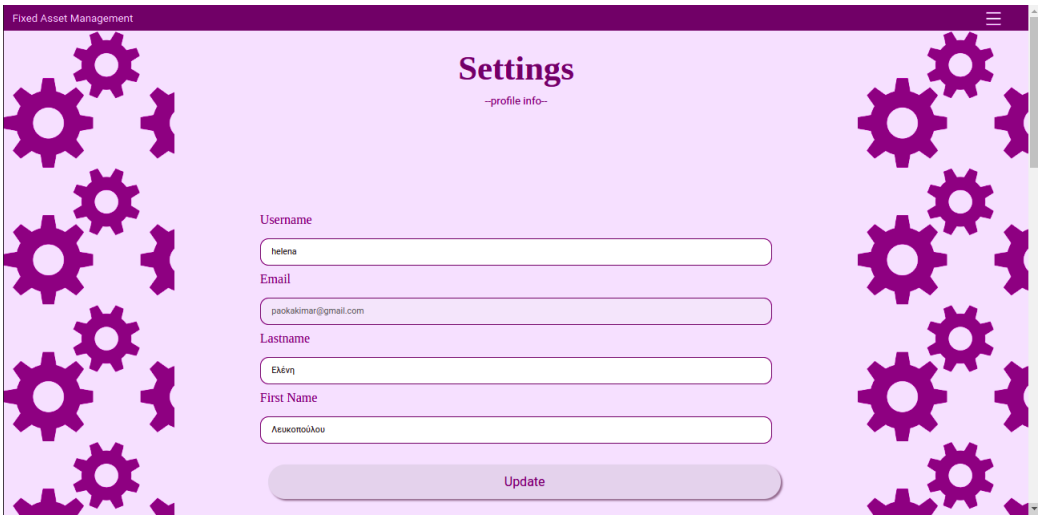


Σχήμα 7.12: Αναδυόμενο παράθυρο για την διαγραφή λογαριασμού

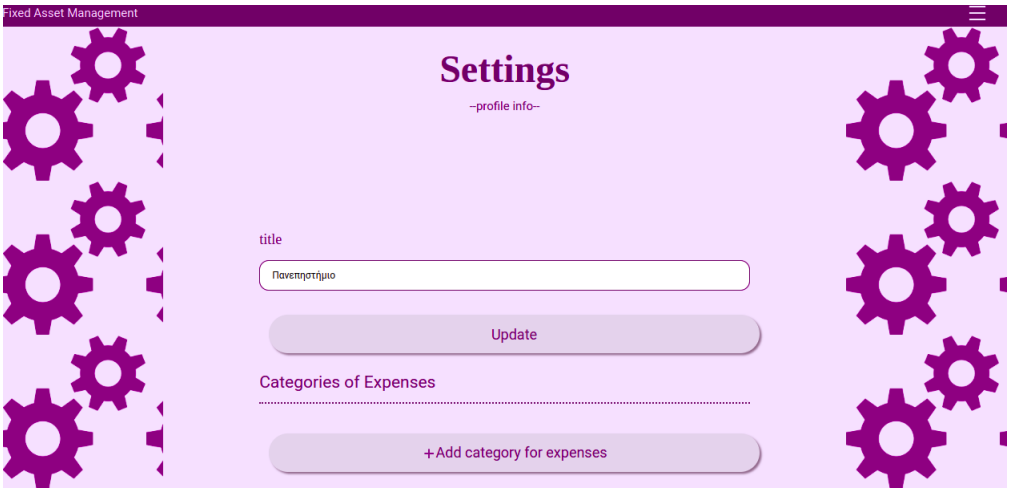
7.5 Η λειτουργία settings

Όταν κάποιος οδηγείται στη σελίδα των ρυθμίσεων, έχει τη δυνατότητα να δει τα στοιχεία με τα οποία εγγράφηκε και να τα το τροποποιήσει. Μόνο το πεδίο του email παραμένει απενεργοποιημένο. Αν είναι συνδεδεμένος ως ομάδα αντίστοιχα μπορεί να δει το όνομα της

ομάδας και να το επεξεργαστεί. Για την επεξεργασία δεν υπάρχουν αναδυόμενα παράθυρα απλώς αρκεί να πατήσει κάποιος το κουμπί update.

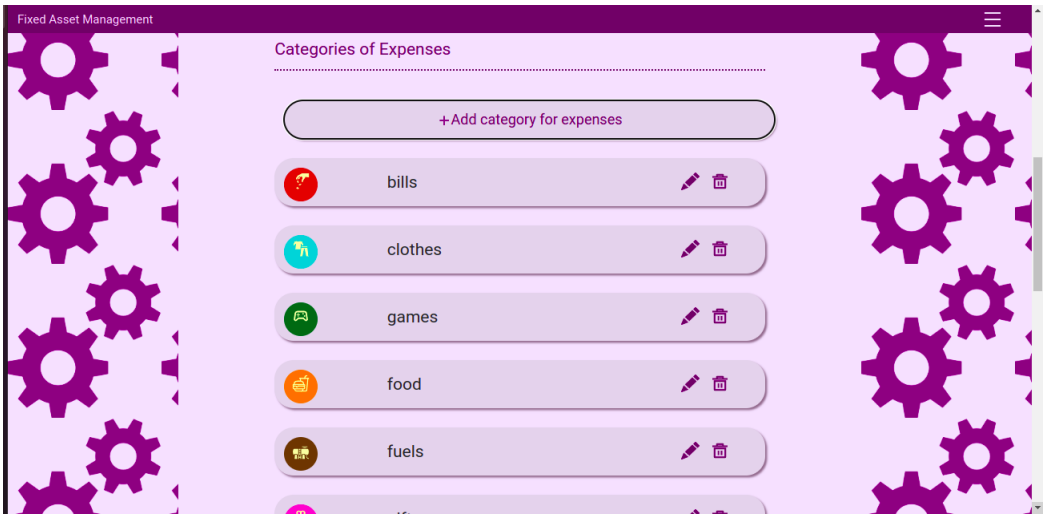


Σχήμα 7.13: Προβολή στοιχείων χρήστη με δυνατότητα επεξεργασίας

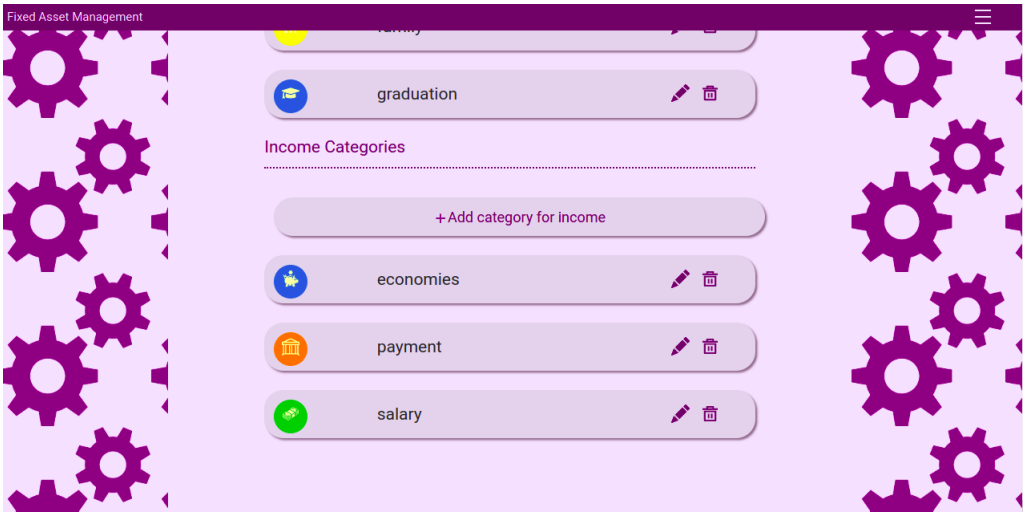


Σχήμα 7.14: Προβολή στοιχείων χρήστη που είναι συνδεδεμένος ως ομάδα

Κάτω από τα στοιχεία του χρήστη βρίσκεται μία λίστα με τις διάφορες κατηγορίες εσόδων και εξόδων. Κάθε φορά που ένας χρήστης ή μία ομάδα δημιουργείται, παρέχεται έτοιμος ένας συνδυασμός από τις κατηγορίες τις οποίες στη συνέχεια ο χρήστης μπορεί να τροποποιήσει, να διαγράψει ή και να εμπλουτίσει προσθέτοντας καινούριες. Οι κατηγορίες αυτές στη συνέχεια θα χρησιμοποιηθούν, ώστε να μπορεί κάποιος να κατατάξει την συναλλαγή του. Όταν κάποιος δοκιμάσει να διαγράψει μία κατηγορία δεν του δίνεται αυτή η δυνατότητα εάν χρησιμοποιείται σε κάποια συναλλαγή και επιπλέον ενημερώνεται με το αντίστοιχο μήνυμα.

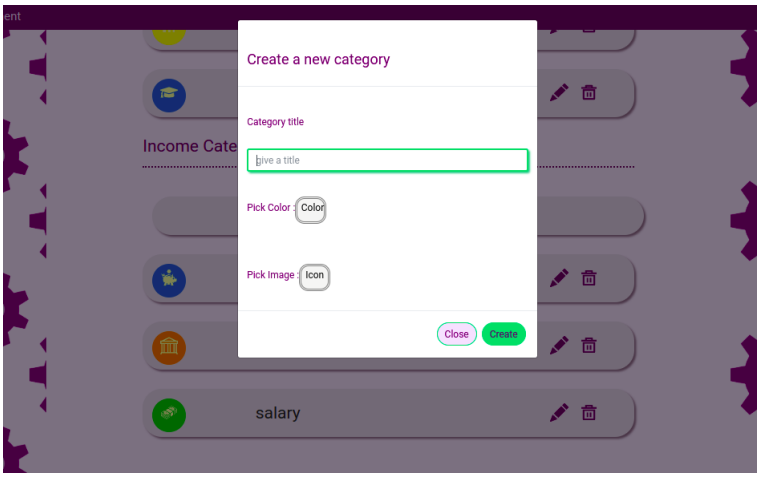


Σχήμα 7.15: Κατηγορίες εξόδων

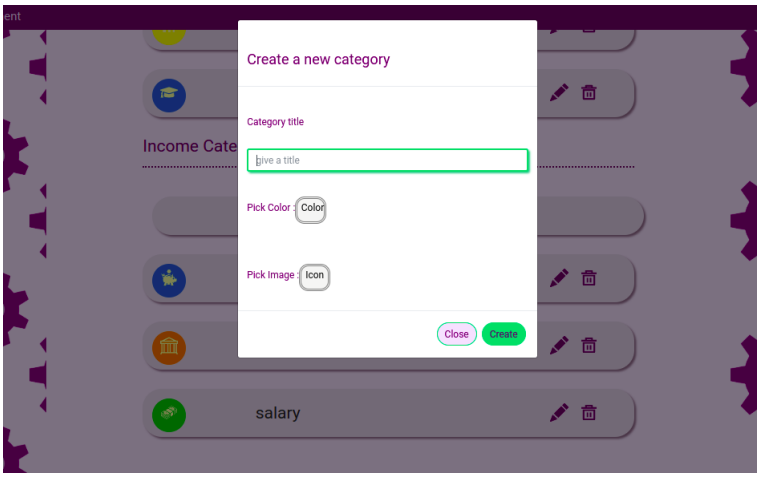


Σχήμα 7.16: Κατηγορίες εσόδων

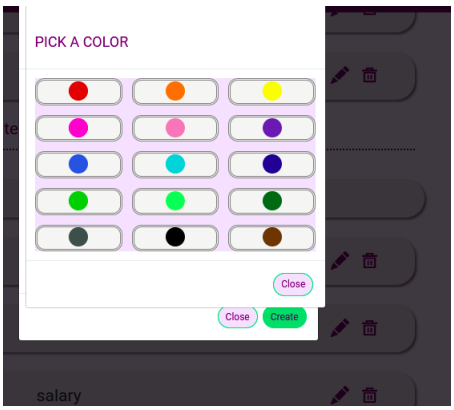
Κάθε κατηγορία χαρακτηρίζεται από ένα χρώμα, ένα εικονίδιο και ένα όνομα. Κατά την προσθήκη ή την επεξεργασία τους ο χρήστης καλείται να διαλέξει το επιθυμητό χρώμα και το επιθυμητό εικονίδιο μέσα από τις επιλογές που του προβάλλει η εφαρμογή.



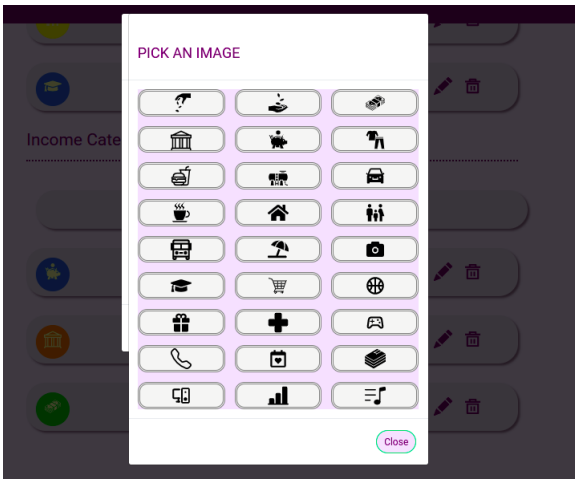
Σχήμα 7.17: Αναδυόμενο παράθυρο για προσθήκη κατηγορίας



Σχήμα 7.18: Αναδυόμενο παράθυρο για επεξεργασία κατηγορίας



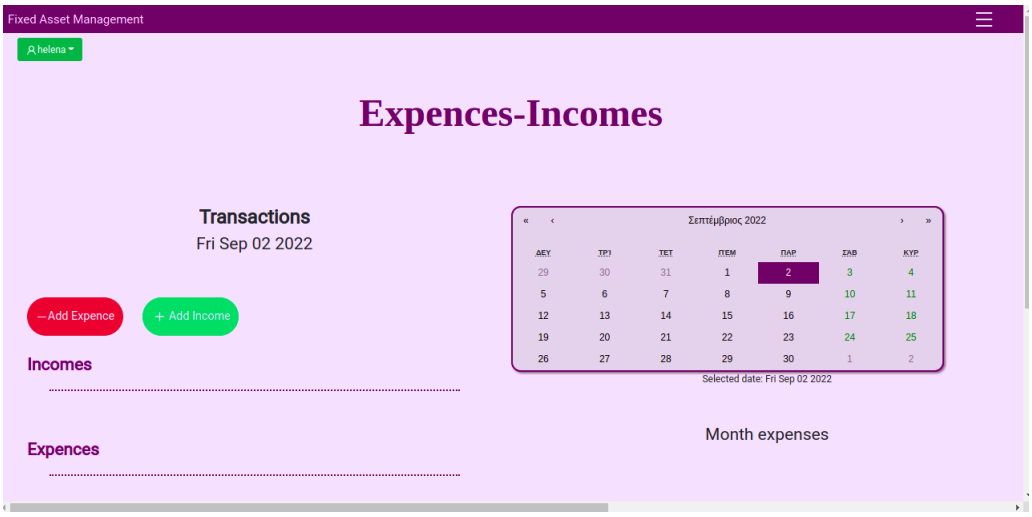
Σχήμα 7.19: Αναδυόμενο παράθυρο για επιλογή χρώματος



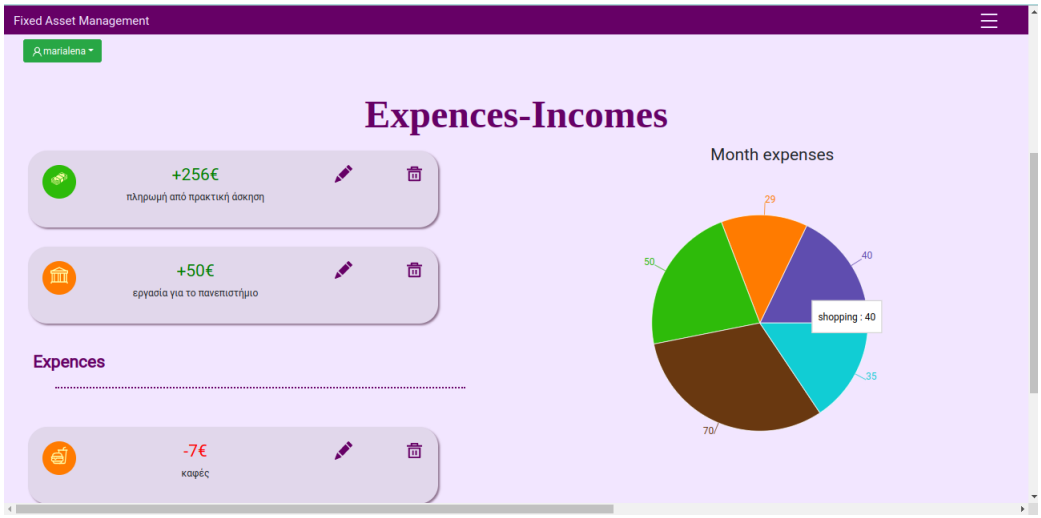
Σχήμα 7.20: Αναδυόμενο παράθυρο για επιλογή εικονιδίου

7.6 Η λειτουργία transactions

Πρόκειται για τη σελίδα όπου ο χρήστης θα μπορεί να προσθέτει τα καθημερινά έσοδα και έξοδά του. Στη σελίδα φαίνεται ένα ημερολόγιο όπου είναι αρχικοποιημένο με την τρέχουσα μέρα, αλλά δίνει τη δυνατότητα για περιήγηση και σε διαφορετικές μέρες. Κάθε φορά που μία ημέρα επιλέγεται, στα δεξιά της σελίδας και κάτω από το ημερολόγιο φαίνονται σε ένα κυκλικό διάγραμμα τα συνολικά έξοδα του μήνα ανά κατηγορία. Η ιδέα ήταν να μπορεί ο χρήστης να οπτικοποιεί τις δαπάνες του και το που ξοδεύει το μεγαλύτερο ποσό των χρημάτων του. Ακουμπώντας ο χρήστης το ποντίκι πάνω στο διάγραμμα βλέπει το όνομα της κατηγορίας. Στα αριστερά φαίνονται ημερησίως οι συναλλαγές του χρήστη διαχωρισμένες σε έσοδα και έξοδα. Η καθεμία διαθέτει κατάλληλο κουμπί για επεξεργασία και για διαγραφή, που ανοίγουν το αντίστοιχο παράθυρο. Επίσης πάνω αριστερά υπάρχουν τα κουμπιά που δίνουν τη δυνατότητα δημιουργίας μιας νέας συναλλαγής. Τα στοιχεία που μπορεί να εισάγει ο χρήστης είναι η τιμή, κάποιο σχόλιο ή περιγραφή σχετική, η ημερομηνία της συναλλαγής και μετά πρέπει να επιλέξει την κατάλληλη κατηγορία και τον αντίστοιχο λογαριασμό. Με το που προσθέσει τη συναλλαγή αυτόματα ενημερώνεται και ο λογαριασμός. Επιπρόσθετα εάν σε περίπτωση επεξεργασίας αλλάξει ο λογαριασμός, εφόσον είναι έξοδο η παλιά τιμή προστίθεται στον παλιό λογαριασμό και η νέα τιμή αφαιρείται από τον καινούριο. Το ακριβώς ανάποδο συμβαίνει στην επεξεργασία των εσόδων, όταν παρατηρείται αλλαγή του λογαριασμού. Τέλος αν στην επεξεργασία αλλάξει το ποσό, υπολογίζεται η διαφορά, ώστε να γίνει η σωστή ενημέρωση του λογαριασμού.



Σχήμα 7.21: Η εμφάνιση της σελίδας όταν δεν έχουν προστεθεί συναλλαγες



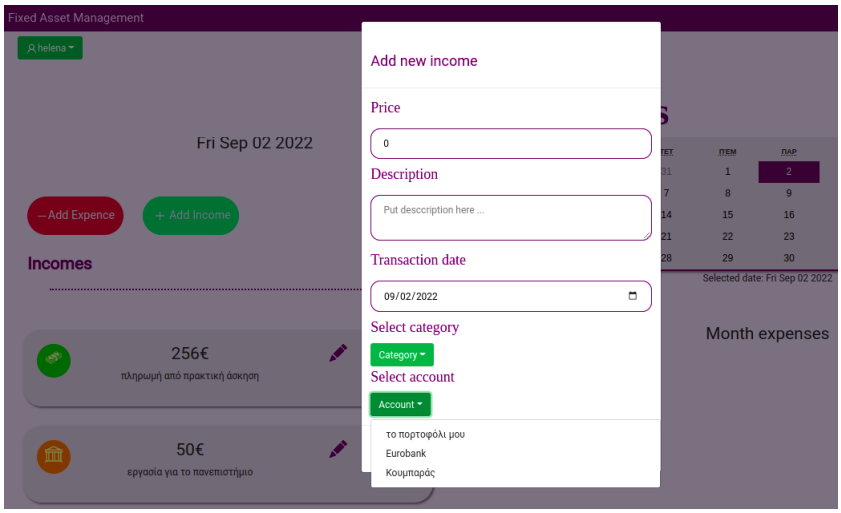
Σχήμα 7.22: Διάγραμμα εξόδων για ένα μήνα

Αξίζει σε αυτό το σημείο να αναφερθούν οι βιβλιοθήκες που χρησιμοποιήθηκαν για την κατασκευή του ημερολογίου και του γραφήματος. Για το ημερολόγιο χρησιμοποιήθηκε η react-calendar, η οποία μπορεί να εγκατασταθεί με την εντολή npm i react-calendar. Διαθέτει το βασικό σχήμα του ημερολογίου, του οποίου τα γραφικά μπορεί κάποιος να τροποποιήσει μέσω της css και κάποιες συναρτήσεις, με τις οποίες μπορεί κάποιος να δει την επιλεγμένη μέρα, μήνα ή χρόνο. Επίσης δίνει τη δυνατότητα να ενεργοποιήσει κάποιος μόνο ένα μέρος του ημερολογίου έτσι ώστε ο χρήστης να μην μπορεί να δει πέρα από αυτή την ημερομηνία. Για τα διαγράμματα χρησιμοποιήθηκε βιβλιοθήκη recharts που μπορεί κάποιος να την εγκαταστήσει με την εντολή npm i recharts. Ο κώδικας που καλέστηκε είναι ο εξής:

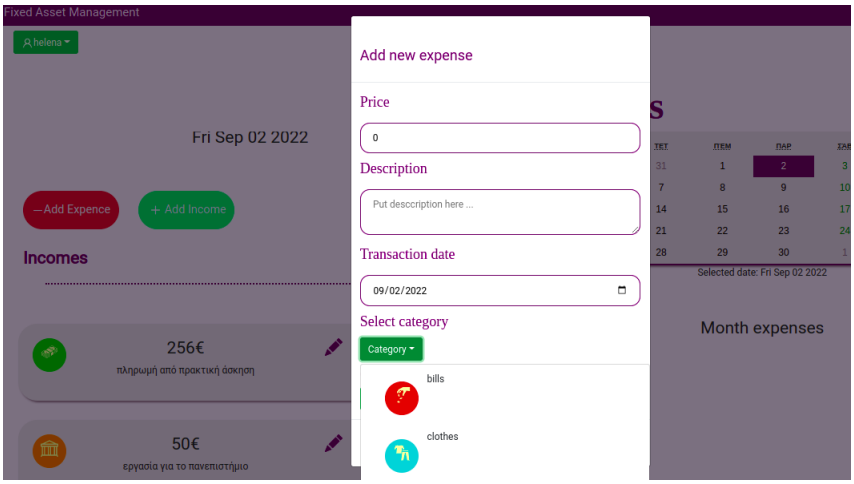
```
<PieChart width={2000} height={600}>
```

```
        <Pie
        dataKey="value"
        isAnimationActive={true}
        data={data01}
        cx={400}
        cy={200}
        outerRadius={180}
        label
        />
        <Tooltip />
    </PieChart>
    </>
```

Με το outerRadius ορίζεται το μέγεθος του κύκλου και με τα cx cy η θέση του στον οριζόντιο και κάθετο άξονα. Τα δεδομένα κατασκευάστηκαν ώστε να έχουν τη μορφή πίνακα από JSON όπου μέσα περιέχουν ένα πεδίο name για το όνομα που θα εμφανίζεται όταν κάποιος επιλέγει το συγκεκριμένο πεδίο, value για την τιμή και fill για το χρώμα εμφάνισης.



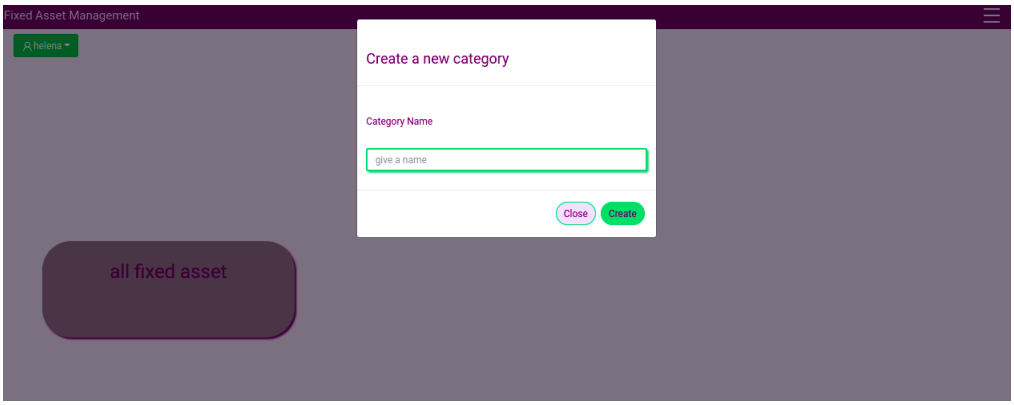
Σχήμα 7.23: Αναδυόμενο παράθυρο για προσθήκη, την ώρα που ο χρήστης επιλέγει λογαριασμό



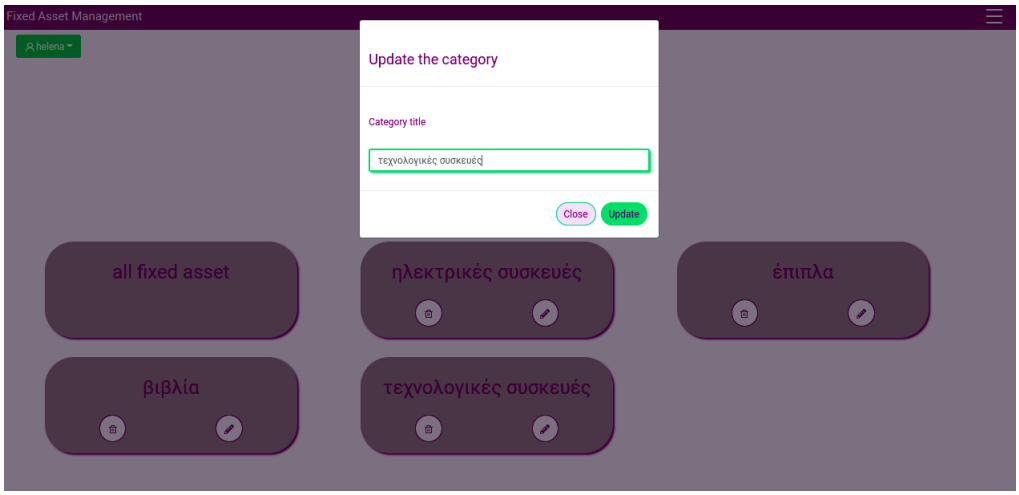
Σχήμα 7.24: Αναδυόμενο παράθυρο για προσθήκη όταν ο χρήστης επιλέγει κατηγορία

7.7 Η λειτουργία fixed asset

Όταν ο χρήστης επιλέξει αυτή τη λειτουργία από το μενού κατευθύνεται σε μία σελίδα, όπου του εμφανίζονται οι κατηγορίες για τα πάγια περιουσιακά στοιχεία. Μπορεί ο ίδιος να προσθέσει όσες κατηγορίες θέλει, να τις τροποποιήσει, δηλαδή να τις μετονομάσει, καθώς και να επιλέξει να τις διαγράψει. Σε περίπτωση διαγραφής διαγράφονται και όλα τα περιουσιακά στοιχεία που ανήκουν σε αυτή την κατηγορία. Επιλέγοντας μία συγκεκριμένη κατηγορία κατευθύνεται σε μία σελίδα, όπου μπορεί να δεί όλα τα πάγια περιουσιακά στοιχεία της συγκεκριμένης κατηγορίας, καθώς και να προσθέσει νέα. Επιλέγοντας το all fixed assets μπορεί να δει όλα τα περιουσιακά στοιχεία που ανήκουν σε αυτόν ή στην ομάδα αν είναι συνδεδεμένος ως ομάδα.



Σχήμα 7.25: Αναδυόμενο παράθυρο για δημιουργία κατηγορίας

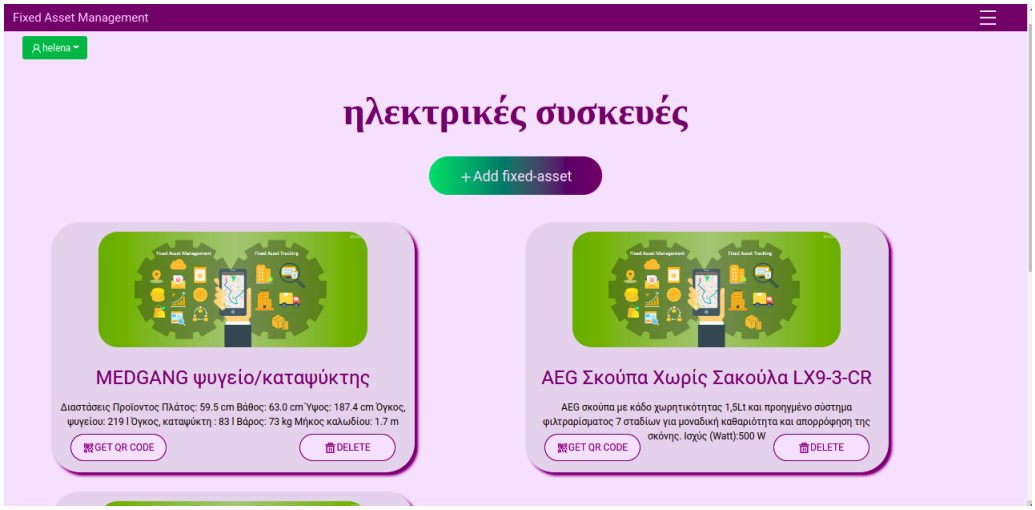


Σχήμα 7.26: Αναδυόμενο παράθυρο για επεξεργασία κατηγορίας

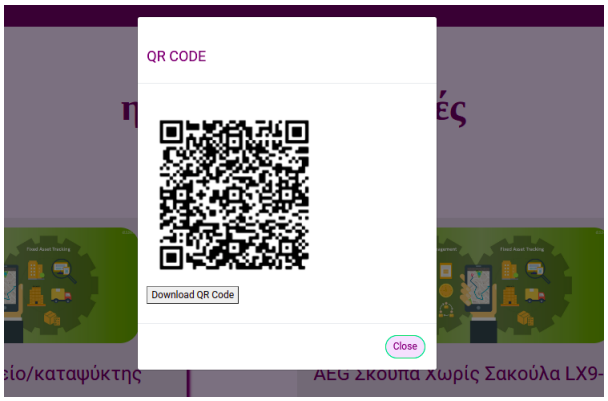
Σχήμα 7.27: Όλες οι κατηγορίες

Μόλις επιλέξει μία κατηγορία και του εμφανιστούν τα πάγια περιουσιακά στοιχεία, έχει τη δυνατότητα να επιλέξει να διαγράψει κάποιο από αυτά. Στη διαγραφή εμφανίζεται ένα αναδυόμενο παράθυρο, ώστε ο χρήστης να επιβεβαιώσει τη διαγραφή. Επίσης μπορεί να επιλέξει την δημιουργία QR κωδικού. Ο κωδικός αυτός αντιστοιχεί σε σύνδεσμο που τον κατευθύνει στη σελίδα επεξεργασίας του πάγιου περιουσιακού στοιχείου. Δίνεται επίσης η επιλογή να αποθηκεύσει τον QR κωδικό σαν εικόνα, ώστε να μπορεί να τον κολλήσει στο αντίστοιχο περιουσιακό στοιχείο και ανά πάσα στιγμή να έχει πρόσβαση στις πληροφορίες που του αντιστοιχούν. Πατώντας πάνω στο περιουσιακό στοιχείο οδηγείται σε μία φόρμα επεξεργασίας των πληροφοριών του. Αρχικά όλα τα πεδία είναι απενεργοποιημένα, ώστε απλά να μπορεί κάποιος να δει τις πληροφοφρίες. Αν ο χρήστης επιλέξει Activate, ενεργοποιεί την

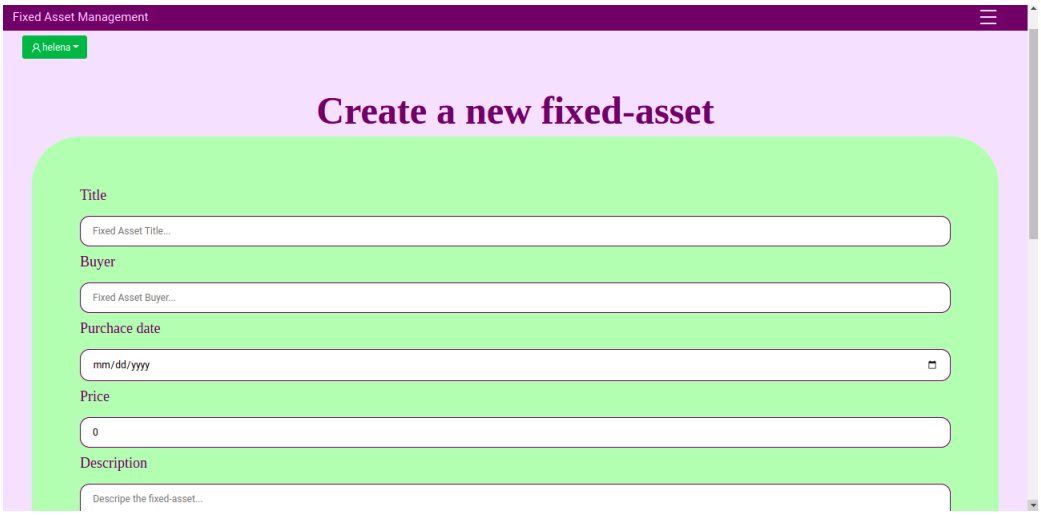
επεξεργασία και τα πεδία που μπορεί να επεξεργαστεί είναι διαθέσιμα. Πατώντας το update τα πάγιο περιουσιακό στοιχείο ενημερώνεται. Οποιαδήποτε στιγμή μπορεί να επιλέξει deactivate, ώστε να απενεργοποιήσει την επεξεργασία ή go back ώστε να γυρίσει στη σελίδα όπου μπορούσε να δει όλα τα περιουσιακά στοιχεία. Επίσης πατώντας το κουμπί της προσθήκης κατευθύνεται σε μια σελίδα συμπλήρωσης στοιχείων για το νέο περιουσιακό στοιχείο. Το στοιχείο αυτό αυτόματα κατατάσσεται στην επιλεγμένη κατηγορία. Τα στοιχεία που ζητείται να βάλει ο χρήστης κατά την προσθήκη είναι ένας τίτλος, ο αγοραστής του στοιχείου, το ποσό αγοράς, η ημερομηνία αγοράς και η τρέχουσα τοποθεσία του.



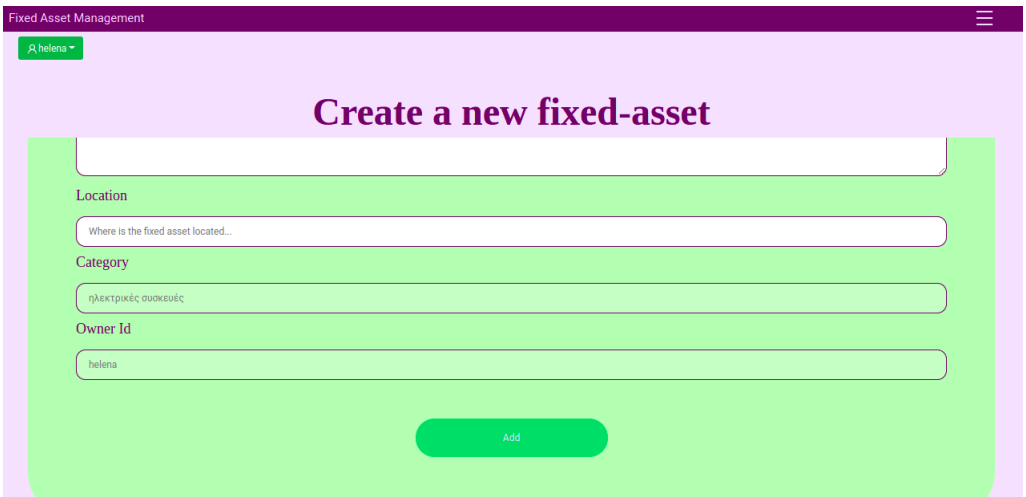
Σχήμα 7.28: Όλα τα περιουσιακά στοιχεία μίας κατηγορίας



Σχήμα 7.29: QR code



Σχήμα 7.30: Φόρμα προσθήκης πάγιου περιουσιακού στοιχείου



Σχήμα 7.31: Φόρμα προσθήκης πάγιου περιουσιακού στοιχείου συνέχεια

Σχήμα 7.32: Επεξεργασία πάγιου περιουσιακού στοιχείου

7.7.1 QR code

Πριν ολοκληρωθεί και η ανάλυση της λειτουργίας των πάγιων περιουσιακών στοιχείων έχει ενδιαφέρον να αναλύσουμε τι είναι ένας QR κωδικός, καθώς και το πως επιλέγουμε την πληροφορία που θα αποθηκευτεί σε αυτόν. Ο κωδικός QR είναι ένα κώδικας τετραγώνων ή δισδιάστατος γραμμωτός κώδικας ο οποίος εφευρέθηκε το 1994 από την ιαπωνική αυτοκινητοβιομηχανία Denso Wave. Είναι ικανό να αποθηκεύσει πληροφορίες σχετικές με το αντικείμενο στο οποίο επισυνάπτεται. Χωρίζεται σε 2 κατηγορίες το στατικό και το δυναμικό. Οι στατικοί κωδικοί περιέχουν πληροφορίες που δεν μπορούν να τροποποιηθούν, γι'αυτό συνήθως επιλέγονται για δεδομένα που δεν είναι πιθανό να αλλάξουν, αλλιώς σε περίπτωση τροποποίησης τους ο κωδικός αχρηστεύεται εντελώς και πρέπει να εκδοθεί από την αρχή. Οι δυναμικοί κωδικοί QR δίνουν τη δυνατότητα να τροποποιεί κάποιος όσες φορές θέλει τα δεδομένα, καθώς δεν κρατούν αποθηκευμένα τα ίδια τα δεδομένα, αλλά ένα σύνδεσμο στον οποίο θα ανακατευθύνεται ο χρήστης. Στην τρέχουσα εφαρμογή θεωρήθηκε πιο σωστό να αποθηκεύεται στον κωδικό ο σύνδεσμος που θα ανακατευθύνει στη σελίδα με τα στοιχεία του προϊόντος, αφού σα λειτουργία παρέχεται η τροποποίηση των προϊόντων, οπότε ανά πάσα στιγμή μπορεί κάποιος να τα αλλάξει. Έτσι είναι πιο ασφαλές να αποθηκεύσουμε τον σύνδεσμο, δηλαδή η δημιουργία ενός δυναμικού κωδικού. Η βιβλιοθήκη που χρησιμοποιήθηκε για την κατασκευή του κωδικού είναι η qrcode.react και ο κώδικας που καλέστηκε είναι ο εξής:

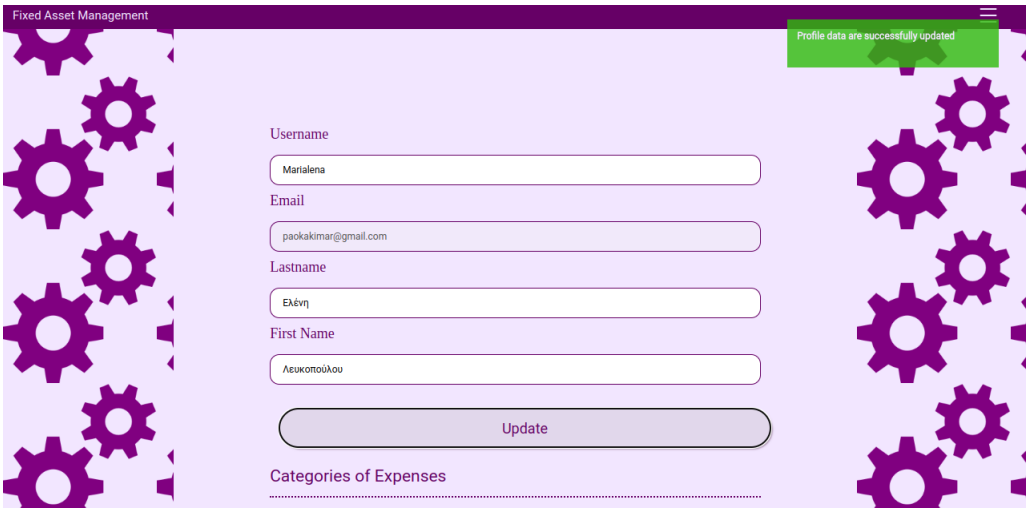
```
<QRCode
```

```
        id="qr-gen"
        value={qrValue}
        size={290}
        level={"H"}
        includeMargin={true}
    />
```

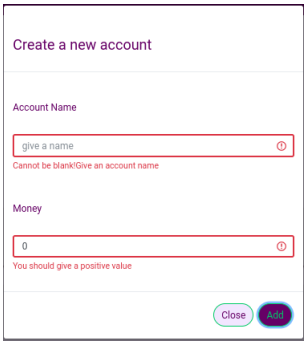
όπου στην θέση της τιμής δίνουμε κάθε φορά τον επιθυμητό σύνδεσμο.[48]

7.8 Επεξεργασία λαθών

Για το τέλος έμεινε να εξηγήσουμε πως το σύστημα διαχειρίζεται τα λάθη. Στη μεριά του back-end έχει φτιαχτεί ένα αρχείο JSON, το οποίο ονομάζεται validation.json. Στο αρχείο αυτό υπάρχουν αντιστοιχισμένοι κωδικοί και το αντίστοιχο γραπτό μήνυμα τους. Κάθε φορά λοιπόν που εκτελείται ένα ερώτημα παίρνουμε από το αρχείο τον κατάλληλο κωδικό και το κατάλληλο μήνυμα, ώστε να ξέρει και το front βασιζόμενο στον κωδικό σε ποια περίπτωση βρισκόμαστε. Μόλις επιστραφούν τα δεδομένα του ερωτήματος, στο front ελέγχεται ο εκάστοτε κωδικός και εμφανίζεται μια ειδοποίηση στο χρήστη για λίγα δευτερόλεπτα με το αντίστοιχο μήνυμα. Εάν το ερώτημα έχει επιτύχει η ειδοποίηση έχει πράσινο χρώμα, ενώ εάν έχει προκύψει κάποιο σφάλμα η ειδοποίηση έχει κόκκινο χρώμα. Επίσης σε κάθε φόρμα που καλείται να συμπληρώσει ο χρήστης, με τη χρήση κατάλληλων validators ελέγχεται, ώστε να μην υπάρχουν κενά πεδία ή κάποια λάθος πληροφορία και να αποφευχθούν τα πιθανά λάθη στο ερώτημα που θα γινόταν στη βάση.



Σχήμα 7.33: Ειδοποίηση επιτυχούς ενημέρωσης



Σχήμα 7.34: Παράδειγμα χρήσης validators

Κεφάλαιο 8

Συμπεράσματα

Ολοκληρώνοντας λοιπόν αυτή την εργασία αξίζει να κάνουμε μια αναδρομή στα όσα αναφέρθηκαν και να καταλήξουμε σε ένα συμπέρασμα.

8.1 Σύνοψη και συμπεράσματα

Έπειτα από αυτή την πολύμηνη μελέτη πάνω στο αντικείμενο της ανάπτυξης διαδικτυακών εφαρμογών καταλήγουμε στο συμπέρασμα πως ο σχεδιασμός ενός λογισμικού είναι μία διαδικασία που χρειάζεται χρόνο και που δεν σταματά ποτέ. Η τεχνολογία γύρω μας αλλάζει και αναπτύσσεται ραγδαία, τα εργαλεία και τα πλαίσια προγραμματισμού διαρκώς πληθαίνουν και εξελίσσονται απαιτούν λοιπόν άτυπα από τον προγραμματιστή να βρίσκεται διαρκώς σε εγρήγορση και ενημερωμένος σχετικά με τις εξελίξεις. Όσον αφορά το λογισμικό κι αυτό παράλληλα με το τεχνολογικό του περιβάλλον πρέπει να αναβαθμίζεται και να βελτιώνεται διαρκώς. Είναι λοιπόν λάθος κάποιος να θεωρήσει πως το λογισμικό του έχει φτάσει στην ολοκλήρωση του καθώς πάντα υπάρχει η δυνατότητα βελτίωσης. Η ραγδαία αυτή εξέλιξη δικαιολογεί και την έντονη δημοφιλία των επαναληπτικών μοντέλων ανάπτυξης λογισμικού όπως προαναφέρθηκε. Η εργασία ολοκλήρωσε το στόχο της καθώς κατάφερε να θέσει ένα πλαίσιο θεωρητικής γνώσης σχετικά με τη διαδικτυακή ανάπτυξη εφαρμογών αλλά και σχετικό με τον σχεδιασμό λογισμικού γενικότερα. Έγινε μια λεπτομερής μελέτη διάφορων εργαλείων και επιλέγχθηκαν τα πιο κατάλληλα για την εφαρμογή που αναπτύχθηκε και κατάφερε να θέσει ένα τρόπο σκέψης ώστε ο αναγνώστης να μπορεί σε μία αντίστοιχη περίπτωση να κάνει τη σωστή επιλογή. Ακόμη ακολουθήθηκε με τη σειρά ο τρόπος σκέψης και σχεδιασμού από τις βάσεις δεδομένων μέχρι και τον γραφικό περιβάλλον που βλέπει ο

πελάτης. Από τη μεριά της εφαρμογής από την άλλη επιτεύχθηκε η κατασκευή ενός αξιόλογου λογισμικού διαχείρισης πάγιων περιουσιακών στοιχείων και συναλλαγών που πληροί τις λειτουργικές απαιτήσεις που τέθηκαν στην αρχή και που μπορεί να εξυπηρετήσει τις ομάδες των πιθανών χρηστών.

8.2 Μελλοντικές επεκτάσεις

Όπως προαναφέρθηκε ένα λογισμικό ποτέ δεν φτάνει στο τέλος του, έχοντας δημιουργήσει λοιπόν μια αξιόλογη βάση μερικές από τις μελλοντικές επεκτάσεις που έχω σκοπό να κάνω είναι οι εξής:

- Προσθήκη εικόνων και αποδείξεων στα πάγια περιουσιακά στοιχεία.
- Προσθήκη αποδείξεων στις συναλλαγές.
- Μπάρα αναζήτησης με φίλτρα για να πάγια περιουσιακά στοιχεία αλλά και επιλογές κατάταξης αυτών με βάση τη χρονολογία ή την αλφαβητική σειρά των τίτλων.
- Προσθήκη ειδοποιήσεων σε όλα τα μέλη όταν κάποιος τροποποιεί ή προσθέτει κάτι στην ομάδα.
- Προσθήκη κάποιου αλγορίθμου πρόβλεψης βασιζόμενου στα ήδη υπάρχοντα έσοδα και έξοδα του χρήστη που θα προβλέπει τις δαπάνες του επόμενου μήνα.
- Η εφαρμογή να γίνει deploy και να αποκτήσει κάποιον δημόσιο σύνδεσμο καθώς στα πλαίσια της εργασίας ελέγχθηκε και χρησιμοποιήθηκε στο localhost.

Βιβλιογραφία

[1] Διαδικτυακή εφαρμογή. https://el.wikipedia.org/wiki/%CE%94%CE%B9%CE%B1%CE%B4%CE%B9%CE%BA%CF%84%CF%85%CE%B1%CE%BA%CE%AE_%CE%B5%CF%86%CE%B1%CF%81%CE%BC%CE%BF%CE%B3%CE%AE/.

[2] Τεχνολογίες front-end. https://www.keycdn.com/blog/frontend-frameworks.

[3] Html. https://www.w3schools.com/html/.

[4] Thomas A. Powell. *HTML CSS:The Complete Reference*. Mc Graw Hill, 5 edition, 2010.

[5] Dave Taylor. *Creating Cool Web Sites with HTML, XHTML,and CSS*. Wiley Publishing, Inc., 1 edition, 2004.

[6] Shay Howe. *Learn to Code HTML CSS Develop Style Websites*. New Riders, 1 edition, 2014.

[7] Azat Mardan. *Full Stack JavaScript: Learn Backbone.js, Node.js and MongoDB*. Apress, 2 edition, 2015.

[8] Συγκριση angular και react. https://www.interviewbit.com/blog/angular-vs-react/.

[9] React. https://reactjs.org/.

[10] Joe Morgan. *How To Code in React.js*. DigitalOcean, 1 edition, 2021.

[11] angular. https://angular.io/.

[12] Angular js. https://angularjs.org/.

[13] Bootstrap original. https://getbootstrap.com/.

[14] Bootstrap. https://www.w3schools.com/bootstrap/default.asp.

[15] Τεχνολογίες back-end. keycdn.com/blog/best-backend-frameworks.

[16] Node. https://www.w3schools.com/nodejs/nodejs_intro.asp.

[17] Θετικά και αρνητικα της node. https://www.simform.com/blog/nodejs-advantages-disadvantages/.

[18] Node original. https://nodejs.org/en/about/.

[19] David Guttman. *Fullstack Node.js The Complete Guide to Building Production Apps with Node.js*. Fullstack.io., 1 edition, 2019.

[20] Ethan Brown. *Web Development with Node and Express*. O'Reilly Media, Inc, 1 edition, 2014.

[21] Lavarel. https://ddi-dev.com/blog/programming/pros-and-cons-of-laravel-framework-for-web-app-development/.

[22] Django advantages. https://data-flair.training/blogs/django-advantages-and-disadvantages/.

[23] Springboot. https://spring.io/projects/spring-boot/.

[24] Σύγκριση node και springboot 1. https://www.sayonetech.com/blog/nodejs-vs-java-spring-boot-microservice/.

[25] Σύγκριση node και springboot 2. https://www.inexture.com/nodejs-vs-spring-boot-choosing-the-best-technology/.

[26] Συγκριση front-end και back-end. https://www.keycdn.com/blog/backend-vs-frontend.

[27] Haroon Shakirat Oluwatosin. Client-server model. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 16(1):167–71, Feb. 2014.

[28] Ian Sommerville. *SOFTWARE ENGINEERING*. Addison-Wesley, 9 edition, 2011.

[29] Βασίλειος Βεσκούκης. *Στοιχεία Τεχνολογίας Λογισμικού*. Kallipos, Αθήνα, 2015.

[30] 8 μεθοδολογίες ανάπτυξης λογισμικού. https://www.uptech.team/blog/software-development-methodologies.

[31] 11 μεθοδολογίες ανάπτυξης λογισμικού. https://www.indeed.com/career-advice/career-development/software-development-methodologies.

[32] 4 μεθοδολογίες ανάπτυξης λογισμικού. https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/.

[33] Nosql βάσεις. https://www.mongodb.com/nosql-explained.

[34] Jennifer Widom Hector Garcia-Molina, Jeffrey D. Ullman. *DATABASE SYSTEMS The Complete Book*. Pearson Education Inc., 2 edition, 2009.

[35] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2 edition, 2013.

[36] mongoose. https://mongoosejs.com/docs/guides.html.

[37] graphql. https://graphql.org/.

[38] Robin Wieruch. *The Road to GraphQL*. Independently published, 1 edition, 2018.

[39] Apollo server. https://www.apollographql.com/docs/apollo-server/.

[40] Apollo client. https://www.apollographql.com/docs/react/.

[41] queries. https://www.apollographql.com/docs/react/data/queries/.

[42] mutations. https://www.apollographql.com/docs/react/data/mutations/.

[43] keycloak. https://www.keycloak.org/.

[44] Pedro Igor Silva Stian Thorgersen. *Keycloak - Identity and Access Management for Modern Applications*. Packt Publishing, 1 edition, 2021.

[45] usestate. `https://reactjs.org/docs/hooks-state.html`.

[46] useeffect. `https://reactjs.org/docs/hooks-effect.html`.

[47] Route. `https://www.w3schools.com/react/react_router.asp`.

[48] qr-codes. `https://blog.hubspot.com/blog/tabid/6307/bid/16088/everything-a-marketer-should-know-about-qr-codes.aspx`.

ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα Α

Η χρήση της εφαρμογής

Στο παράρτημα αυτό δίνονται σύντομες οδηγίες για το τι πρέπει να κάνει κάποιος ώστε να τρέξει την εφαρμογή σε τοπικό επίπεδο.Αρχικά ο κώδικας της εφαρμογής μπορεί να βρεθεί στο `https://github.com/MLefkopoulou/Fixed-Asset-Management`. Στο repository αυτό θα βρει κάποιος 3 φακέλους. Ο φάκελος back_management αντιστοιχεί στον εξυπηρετητή της εφαρμογής που είναι γραμμένος σε Node.js. Στο φάκελο φάκελο front_management υπάρχει ο κώδικας του πελάτη που είναι γραμμένος σε React.Τέλος στον φάκελο keycloak-18.0.2 υπάρχουν οι ρυθμίσεις του keycloak για την τρέχουσα εφαρμογή. Πιο αναλυτικά στον φάκελο front_management μπορεί κάποιος στους υποφακέλους του src να βρεί τα εξής:

- components : Είναι ο φάκελος που περιέχει όλα τα js αρχεία που αντιστοιχούν σε components
- css : Είναι ο φάκελος που περιέχει css αρχεία για τα προηγούμενα components και ευθύνονται για την ωραιοποίηση της εμφάνισης.
- graphql : Περιέχει 2 αρχεία , τα Queries και τα Mutation όπου βρίσκονται ορισμένα τα ερωτήματα όπως εξηγήσαμε.
- helpers : Περιέχει βοηθητικές συναρτήσεις
- redux : Περιέχει τον κώδικα που καλείται για να αποθηκευτεί ή να ανασυρθεί κάτι από την τοπική μνήμη.
- pages : Περιέχει το σκελετό των των σελίδων μέσα στα αντίστοιχα js αρχεία.

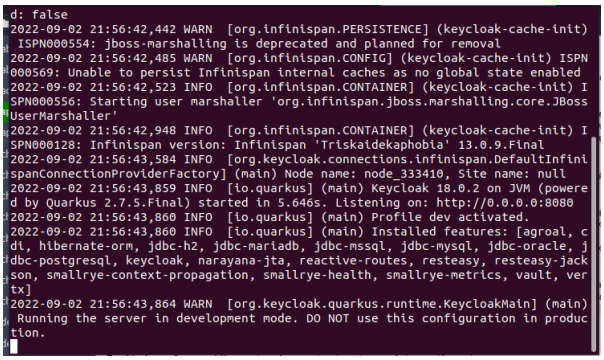
- App.js: Περιέχει όλα τα Routes
- index.js: Περιέχει τις αρχικοποιήσεις του client.

Πιο αναλυτικά στον φάκελο back_management μπορεί να βρει κάποιος τα εξής:

- graphql :Περιέχει δύο υποφακέλους τον resolvers και τον schemas όπου ορίζονται αντίστοιχα οι συναρτήσεις και οι τύποι για τη graphql.
- helpers :Περιέχει βοηθητικές συναρτήσεις
- loaders : Περιέχει αρχεία τα οποία κάνουν τις διάφορες αρχικοποιήσεις για τον εξυπηρετητή όπως η σύνδεση με τη graphql, η ενεργοποίηση του express server και άλλα.
- models : Περιέχει όλα τα σχήματα της mongoose.
- app.js : Πρόκειται για τον κώδικα που ξεκινάει την εφαρμογή-εξυπηρετητή.

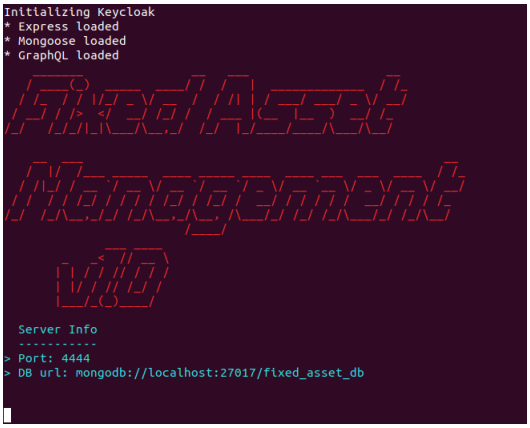
Συνίσταται αν κάποιος θέλει να επεξεργαστεί τον κώδικα να χρησιμοποιήσει το Visual Studio Code . Πρόκειται για ένα πρόγραμμα επεξεργασίας κώδικα το οποίο δίνει τη δυνατότητα στο χρήστη να βλέπει τον κώδικα με χρώματα ώστε να είναι πιο ευνόητη η σύνταξη , αλλά του προσφέρει και ένα τερματικό από όπου μπορεί να τρέχει τον κώδικα. Επίσης προσφέρεται και η λειτουργία αποσφαφλμάτωσης.Το θετικό αυτού του προγράμματος είναι ότι δίνει τη δυνατότητα στο χρήστη να κατεβάσει διάφορες επεκτάσεις οι οποίες το κάνουν συμβατό με πολλές γλώσσες προγραμματισμού. Επίσης θα χρειαστεί να έχει εγκατεστημένη στον υπολογιστή του την mongoDB ώστε να μπορέσει να δημιουργηθεί η βάση.

Μόλις κάποιος κατεβάσει τα αρχεία και έχει εγκαταστήσει και τη MongoDB μπορεί να ανοίξει 3 τερματικά , ένα για καθέναν από τους 3 φακέλους. Στο τερματικό που αντιστοιχεί στο keycloak πρέπει κάποιος να τρέξει την εντολή bin/kc.sh start-dev.Αν όλα πάνε καλά θα δει την αντίστοιχη έξοδο στο τερματικό και θα έχει ενεργοποιήσει επιτυχώς το keycloak.



Σχήμα Α.1: Στιγμιότυπο από την επιτυχή εκκίνηση του Keycloak

Στο φάκελο back_management αρχικά πρέπει να τρέξει την εντολή npm i ώστε να εγκατασταθούν όλα τα κατάλληλα πακέτα. Έπειτα πρέπει να τρέξει την εντολή npm run dev και πρόκειται να δει την αντίστοιχη έξοδο.



Σχήμα Α.2: Στιγμιότυπο από την επιτυχή εκκίνηση του back-end

Στο τερματικό που αντιστοιχεί στο front_management αρκεί απλώς να τρέξει npm i για να εγκατασταθούν τα πακέτα και στη συνέχεια να τρέξει npm start.Τότε η εφαρμογή θα είναι διαπεράσιμη στο http://localhost:4446/