ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αναφορά προβλημάτων σε Δήμους - Ανάπτυξη εφαρμογής για κινητές συσκευές

Του φοιτητή
Μπεσικιώτη Αθανάσιου
Αρ. Μητρώου: 144200

Επιβλέπων
Αδαμίδης Παναγιώτης
Καθηγητής

Θεσσαλονίκη 2023

Τίτλος Π.Ε. Αναφορά προβλημάτων σε Δήμους - Ανάπτυξη εφαρμογής για κινητές συσκευές
Κωδικός Π.Ε. 22337
Ονοματεπώνυμο φοιτητή Μπεσικιώτης Αθανάσιος
Ονοματεπώνυμο εισηγητή Αδαμίδης Παναγιώτης
Ημερομηνία ανάληψης Π.Ε. 07-11-2022
Ημερομηνία περάτωσης Π.Ε. 03-09-2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Μπεσικιώτη Αθανασίου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Τα τελευταία χρόνια οι φορητές συσκευές γίνονται όλο και πιο βασικές στην ζωή των ανθρώπων με τη χρήση τους πλέον να είναι ένα αναπόσπαστο κομμάτι της καθημερινότητας τους. Αυτό το γεγονός σε συνδυασμό με την πλέον μεγάλη άνεση αλλά και προτίμηση που έχουν οι άνθρωποι στη χρήση τους, καθώς και το γεγονός ότι πλέον ένα πολύ μεγάλο ποσοστό ανθρώπων είναι κάτοχος μιας τέτοιας συσκευής έχει οδηγήσει στο να γίνεται ένα μεγάλο μέρος παροχής υπηρεσιών μέσω εφαρμογών για φορητές συσκευές, είτε αυτό είναι ραντεβού σε κάποιο γιατρό, είτε αυτό είναι ηλεκτρονικές αγορές, είτε αυτό είναι ενημέρωση. Μια τέτοια υπηρεσία η παροχή της οποίας μπορεί να γίνει πολύ πιο εύκολη με τη χρήση φορητής συσκευής είναι και η αναφορά προβλημάτων σε ένα δήμο.

Περίληψη

Μέσω αυτής της πτυχιακής εργασίας θα περιγραφεί ο τρόπος με τον οποίο μπορεί να υλοποιηθεί μία εφαρμογή για κινητές συσκευές η οποία θα μπορεί να είναι διαθέσιμη στις δύο μεγαλύτερα λειτουργικά συστήματα, Android και iOS. Ακόμη, θα περιγραφεί το πως μπορεί να δημιουργηθεί ένα API (Application Programming Interface ή Διεπαφή Προγραμματισμού Εφαρμογών) και μία βάση ώστε η εφαρμογή να είναι πλήρως λειτουργική και να είναι μέρος ενός ολόκληρου συστήματος.

Σκοπός της πτυχιακής είναι η δημιουργία μιας εφαρμογής η οποία θα κάνει την αναφορά προβλημάτων σε ένα δήμο αρκετά ευκολότερη για τους πολίτες, κάτι που θα οδηγήσει και στην ευκολότερη επίλυσή τους. Η ανάπτυξη της εφαρμογής έγινε με τη χρήση της React Native, η δημιουργία του API με τη χρήση του Spring Boot, για βάση χρησιμοποιήθηκε η MongoDB και για την ανάπτυξη ενός Admin Panel χρησιμοποιήθηκε η React.

Issue Reporting for Municipalities - Developing a mobile application

Athanasios Besikiotis

Abstract

Through this thesis, we will describe the way we can develop a mobile application that can be available on the two largest operating systems, Android and iOS. Additionally, we will describe how an API (Application Programming Interface) and a database can be created so that the application is fully functional and part of a complete system. The purpose of the thesis is to create an application that will make reporting issues to a municipality much easier for citizens, something that will also lead to easier resolution of said issues. The development of the application was implemented using React Native, the API was created using Spring Boot, MongoDB was used for the database, and React was used for the development of an Admin Panel.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή κ. Παναγιώτη Αδαμίδη για την ευκαιρία να αναλάβω την παρούσα πτυχιακή εργασία καθώς και για τη συνεργασία και την βοήθεια που προσέφερε ώστε να τη βγάλω εις πέρας.

Τέλος, θα ήθελα να πω ένα μεγάλο ευχαριστώ στους δικούς μου ανθρώπους που με βοήθησαν και με υποστήριξαν, ο καθένας με τον τρόπο του, και μου έδωσαν κίνητρο.

Περιεχόμενα

Πρόλογος iii
Περίληψη iv
Abstract v
Ευχαριστίες vi
Περιεχόμενα vii
Κατάλογος Σχημάτων x
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Εισαγωγή 1
1.2 Δομή 1
Κεφάλαιο 2ο: Περιγραφή Προβλήματος 2
2.1 Το πρόβλημα 2
2.2 Πρόταση επίλυσης προβλήματος 2
2.3 Σύντομη αναφορά παρόμοιων εφαρμογών 3
2.3.1 Snap Send Solve. 3
2.3.2 Novoville 3
2.3.3 CityOn 3
2.3.4 Καλαμαριά (4myCity) 4
2.4 Λειτουργίες που θα υλοποιηθούν 4
Κεφάλαιο 3ο: Εργαλεία που χρησιμοποιήθηκαν 5
3.1 Εισαγωγή 5
3.2 Spring Framework - Spring Boot 5
3.2.1 Spring Framework 5
3.2.2 Πλεονεκτήματα 7
3.2.3 Μειονεκτήματα 7
3.2.4 Spring Boot 8
3.3 MongoDB - Realm 8
3.3.1 Τι είναι μία βάση εγγραφών 8
3.3.2 Διαφορές βάσεων εγγραφών με τις σχεσιακές βάσεις δεδομένων 9
3.3.3 MongoDB 9
3.3.4 Realm Web SDK 9

3.4 React Native 10
3.4.1 React Native Core Components 10
3.4.2 Βασικές έννοιες της React 10
3.4.3 Metro Bundler 13
3.4.4 Ανίχνευση σφαλμάτων 13
3.4.5 Performance 14
3.4.6 Λίγα λόγια για την TypeScript 15
3.5 Firebase 16
3.6 Amazon Web Services (Hosting του Server) 17
3.7 Επίλογος 17
Κεφάλαιο 4ο: Υλοποίηση 18
4.1 Εισαγωγή 18
4.2 Δημιουργία βάσης δεδομένων 18
4.2.1 Δημιουργία λογαριασμού Atlas 18
4.2.2 Δημιουργία ενός organization 18
4.2.3 Δημιουργία Project 18
4.2.4 Δημιουργία Cluster 18
4.2.5 Δημιουργία Database και Collections 19
4.3 Ανάπτυξη του Rest API 20
4.3.1 Δημιουργία Spring Boot project 20
4.3.2 Δομή του project 21
4.3.3 Σύνδεση με MongoDB 22
4.3.4 Διαχείριση των χρηστών 22
4.3.5 Διαχείριση των αναφορών 27
4.3.6 Spring Security 30
4.3.7 Αποστολή Push Notifications 33
4.3.8 Hosting του server 36
4.4 Ανάπτυξη της εφαρμογής για κινητά 37
4.4.1 Δημιουργία React Native Project 37
4.4.2 Δομή project 38
4.4.3 Ορισμοί βασικών τύπων 40
4.4.4 Ορισμός χρωμάτων της εφαρμογής 40
4.4.5 Εκμετάλλευση του Rest API 41

4.4.6 Βασικές έννοιες 44
4.4.7 Πλοήγηση στην εφαρμογή 56
4.4.8 Οθόνη σύνδεσης και εγγραφής 59
4.4.9 Αρχική οθόνη 65
4.4.10 Χάρτης Αναφορών 67
4.4.11 Δημιουργία Αναφοράς 72
4.4.12 Λειτουργία Push Notifications 76
4.5 Ανάπτυξη της εφαρμογής διαχειριστή 78
4.6 Επίλογος 81
Κεφάλαιο 5ο: Βελτιώσεις - Συμπεράσματα 82

ΒΙΒΛΙΟΓΡΑΦΙΑ 83

Κατάλογος Σχημάτων

Σχήμα 3.1: Αναπαράσταση Αρχιτεκτονικής του Sping Framework 5
Σχήμα 3.2: Αναπαράσταση της λειτουργίας του Web MVC Framework 7
Σχήμα 3.3: Πίνακας αντιστοίχισης των React Native UI Components με τα αντίστοιχα Views κάθε πλατφόρμας 10
Σχήμα 3.4: Παράδειγμα κώδικα ενός απλού component 11
Σχήμα 3.5: Παράδειγμα κώδικα χρήσης άγκιστρων για δυναμικό κείμενο σε component με χρήση μεταβλητής 11
Σχήμα 3.6: Παράδειγμα κώδικα χρήσης άγκιστρων για δυναμικό κείμενο σε component με χρήση μεθόδου 12
Σχήμα 3.7: Dev Menu όπως εμφανίζεται σε μία iOS συσκευή 13
Σχήμα 3.8: Παράδειγμα συντακτικού στην TypeScript 16
Σχήμα 4.1: Δημιουργία νέου Cluster 19
Σχήμα 4.2: Λίστα με τη βάση και τα collections της 20
Σχήμα 4.3: Το Spring Initializr 21
Σχήμα 4.4: Η δομή του Spring Boot project 22
Σχήμα 4.5: Η διεπαφή UserRepository 24
Σχήμα 4.6: Η κλάση UserNotFoundException 24
Σχήμα 4.7: Η μέθοδος getReportDto της κλάσης ReportDto 28
Σχήμα 4.8: Η μέθοδος login της κλάσης UserService 31
Σχήμα 4.9: Οι μέθοδοι που χρησιμοποιούνται για την παραγωγή ενός JWT και του αντίστοιχου cookie 32
Σχήμα 4.10: Η filterChain μέθοδος 32
Σχήμα 4.11: Η κλάση AuthEntryPointJwt 33
Σχήμα 4.12: Η doFilterInternal της AuthTokenFilter 33
Σχήμα 4.13: Αρχικοποίηση ενός FirebaseMessaging instance στην κλάση ThessreportApplication 34
Σχήμα 4.14: Η κλάση FirebasePushService 35
Σχήμα 4.15: Αποστολή ειδοποιήσεων όταν γίνεται αλλαγή κατάστασης μιας αναφοράς 36
Σχήμα 4.16: Παράδειγμα περιεχομένων του package.json 39
Σχήμα 4.17: Οι τύποι User και Report 40
Σχήμα 4.18: Τα χρώματα της εφαρμογής όπως έχουν οριστεί στο αρχείο colors.json 41
Σχήμα 4.19: Παράδειγμα μεθόδου που κάνει αίτημα για να πάρει όλες τις αναφορές 42
Σχήμα 4.20: Η μέθοδος createNewReport() 43
Σχήμα 4.21: Παράδειγμα χρήσης του useState hook 44
Σχήμα 4.22: Παράδειγμα χρήσης του useEffect hook 45
Σχήμα 4.23: Παράδειγμα χρήσης του useReducer hook 45
Σχήμα 4.24: Παράδειγμα χρήσης του useReducer hook 46
Σχήμα 4.25: Παράδειγμα χρήσης του useMemo hook 47
Σχήμα 4.26: Το useStorage hook 48
Σχήμα 4.27: Το αρχείο reportsSlice.ts 49
Σχήμα 4.28: Το αρχείο rootReducer.ts 50

Σχήμα 4.29: Το αρχείο index.ts του redux store 50
Σχήμα 4.30: Το αρχείο index.ts του useAuth 52
Σχήμα 4.31: Το αρχείο NewReportContext.tsx 54
Σχήμα 4.32: Η επιστροφή του SignInModalContext Provider 55
Σχήμα 4.33: App.tsx και η χρήση των Providers 56
Σχήμα 4.34: Οι δύο χρήσεις του useEffect στον MainNavigator 57
Σχήμα 4.35: Ο MainNavigator 58
Σχήμα 4.36: Ο NewReportNavigator 59
Σχήμα 4.37: Η μέθοδος δημιουργίας του CustomTextInput component 60
Σχήμα 4.38: Παράδειγμα χρήσης της μεθόδου StyleSheet.create() 61
Σχήμα 4.39: Η μέθοδος onSignInSubmit() της οθόνης σύνδεσης 62
Σχήμα 4.40: Το SignInScreen 63
Σχήμα 4.41: Η δομή του SignInActionSheet 65
Σχήμα 4.42: Η αρχική οθόνη της εφαρμογής 66
Σχήμα 4.43: Το CustomButton 67
Σχήμα 4.44: Αλλαγή των αναφορών ανάλογα με την επιλογή φίλτρων 69
Σχήμα 4.45: Η λίστα των αναφορών στην οθόνη του χάρτη 70
Σχήμα 4.46: Η λεπτομέρειες μιας επιλεγμένης αναφοράς 71
Σχήμα 4.47: Η οθόνη του χάρτη 72
Σχήμα 4.48: Επιλογή τοποθεσίας κατά τη δημιουργία νέας αναφοράς 73
Σχήμα 4.49: Η μέθοδος geocode για την εκμετάλλευση του Geocoding API της Google 74
Σχήμα 4.50: Η οθόνη επιλογής λεπτομερειών της νέας αναφοράς 75
Σχήμα 4.51: Προσθήκη εικόνας για τη νέα αναφορά 76
Σχήμα 4.52: Η διαχείριση των ειδοποιήσεων 78
Σχήμα 4.53: Χρήση του Realm SDK για παρακολούθηση της συλλογής 79
Σχήμα 4.54: Οι μέθοδοι που κάνουν αιτήματα προς το API 79
Σχήμα 4.55: Εμφάνιση των αναφορών στο αρχείο App.tsx 80
Σχήμα 4.56: Τελική εμφάνιση της εφαρμογής διαχειριστή 81

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Στις αρχές της δεκαετίας του ‘90 άρχισαν να γίνονται διαθέσιμες φορητές συσκευές τηλεπικοινωνίας με την πρώτη προσπάθεια να είναι το IBM Simon [1], το οποίο χαρακτηρίστηκε λίγο αργότερα από την πρώτη του κυκλοφορία το πρώτο “smartphone”. Οι φορητές συσκευές εξελίσσονταν με τα χρόνια τόσο στον τομέα της υπολογιστικής ισχύος τους, όσο και σε άλλους τομείς όπως το μέγεθός τους και οι επιπλέον λειτουργίες που προσέφεραν. Η μεγάλη αλλαγή ήρθε το 2008 όταν η Apple λάνσαρε το πρώτο iPhone. Η σημαντική αλλαγή που προήλθε από το πρώτο iPhone ήταν το App Store που συμπεριλαμβανόταν στην αναβάθμιση του iPhone OS 2 [2]. Αυτό επέτρεψε προγραμματιστές να αναπτύξουν για πρώτη φορά τις δικές τους εφαρμογές για κινητά και τους χρήστες να μπορούν να τις κατεβάσουν ασύρματα. Από εκείνο το σημείο και έπειτα και με τη χρήση εφαρμογών οι προγραμματιστές μπορούσαν, μεταξύ άλλων, να κάνουν ευκολότερη την παροχή υπηρεσιών. Έτσι, πλέον, οι χρήστες μπορούν να χρησιμοποιήσουν εφαρμογές εγκατεστημένες στην κινητή συσκευή τους για μία πληθώρα πραγμάτων.

1.2 Δομή

Στο πρώτο κεφάλαιο της πτυχιακή εργασίας έγινε μία μικρή εισαγωγή στη χρήση κινητών συσκευών και εφαρμογών. Στο δεύτερο κεφάλαιο θα ακολουθήσει μία περιγραφή του προβλήματος που θα προσπαθήσει να επιλυθεί και θα προταθεί μία λύση. Στη συνέχεια θα γίνει μία αναφορά σε μερικές υπάρχουσες εφαρμογές που χρησιμοποιούνται για την αναφορά προβλημάτων και, τέλος, θα περιγραφούν οι λειτουργίες που θα περιλαμβάνονται στην εφαρμογή που αναπτύχθηκε στα πλαίσια της εργασίας. Στο τρίτο κεφάλαιο θα περιγραφούν αναλυτικά τα εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν σε όλα τα στάδια της ανάπτυξης. Στο τέταρτο κεφάλαιο θα γίνει πλήρης ανάλυση όλων των σταδίων της ανάπτυξης και του πως γίνεται η συνεργασία όλων των τεχνολογιών που αναλύονται στο τρίτο κεφάλαιο. Στο πέμπτο και τελευταίο κεφάλαιο αναφέρονται τα συμπεράσματα καθώς και πιθανές βελτιώσεις.

Κεφάλαιο 2ο: Περιγραφή Προβλήματος

2.1 Το πρόβλημα

Οι πολίτες καθημερινά συναντούν ποικίλα προβλήματα που καθιστούν την καθημερινότητά τους προβληματική είτε αυτό αφορά την μετακίνηση τους, την ασφάλειά τους ή την γενικότερη λειτουργικότητά τους. Τέτοια προβλήματα μπορεί να είναι, μεταξύ πολλών άλλων, ένα παράνομα παρκαρισμένο αμάξι, μια βλάβη στην ηλεκτροδότηση ή την υδροδότηση, ζημιά σε κάποιο δρόμο, υπερχειλισμένοι κάδοι απορριμμάτων.

Παράλληλα, τα κανάλια επικοινωνίας των πολιτών ώστε να μπορούν να αναφέρουν τέτοια προβλήματα είναι περιορισμένα και μη ιδανικά κυρίως λόγω έλλειψης διαφάνειας, καθώς οι πολίτες δε μπορούν να γνωρίζουν αν όντως η αναφορά τους έχει καταγραφεί ή ποια είναι εξέλιξη της, έλλειψης χρόνου, καθώς σε πολλές περιπτώσεις η αναμονή για κλήση στην αρμόδια υπηρεσία ή η σύνταξη email προς αυτή είναι μπορεί να είναι χρονοβόρα. Αυτοί οι παράγοντες μπορεί να κάνουν δύσκολη τη διαδικασία της αναφοράς ή και να αποθαρρύνουν κάποιον από το να την κάνει.

Από τη μεριά της τοπικής αυτοδιοίκησης μπορεί να μη γίνεται σωστή συλλογή και διαχείριση αναφορών και μη σωστή διάθεση πόρων προς επίλυση των προβλημάτων. Επίσης, λόγω των καναλιών επικοινωνίας και των ενδεχόμενων υποκειμενικών περιγραφών των πολιτών μπορεί να υποτιμηθεί ή να υπερεκτιμηθεί η σημαντικότητα μιας αναφοράς. Τέλος, η συλλογή δεδομένων ενδέχεται να είναι προβληματική εώς και αδύνατη.

2.2 Πρόταση επίλυσης προβλήματος

Τα παραπάνω θα μπορούσαν να επιλυθούν με μία εφαρμογή για κινητά την οποία θα μπορούν να χρησιμοποιούν οι πολίτες για να κάνουν αναφορές και την αντίστοιχη εφαρμογή για τη διαχείριση των αναφορών από την τοπική αυτοδιοίκηση.

Για τους πολίτες θα μπορέσει να βελτιώσει τον τρόπο με τον οποίο επικοινωνούν τα προβλήματα τους σε ένα κεντρικό σημείο. Θα υπάρχει η άνεση της κινητής συσκευής καθώς πλέον είναι το νούμερο ένα εργαλείο σε πολλούς τομείς της ζωής των ανθρώπων. Οι αναφορές θα είναι πιο άμεσες και πιο λεπτομερείς καθώς οι χρήστες θα μπορούν να επιλέγουν ακριβή τοποθεσία, αλλά και να επισυνάπτουν φωτογραφία. Θα υπάρχει διαφάνεια καθώς οι χρήστες θα ενημερώνονται για την κατάσταση των αναφορών τους σε κάθε στάδιό της, οπότε θα νιώθουν ότι το γεγονός ότι ενήργησαν είχε αποτέλεσμα. Τέλος, θα υπάρχει περισσότερη διαδραστικότητα καθώς οι χρήστες θα μπορούν να βλέπουν τις αναφορές που έχουν υποβάλει άλλοι χρήστες και να δίνουν upvotes. Με αυτόν τον τρόπο θα είναι πιο ξεκάθαρη η σημαντικότητα ενός προβλήματος.

Από τη μεριά της τοπικής αυτοδιοίκησης θα υπάρξουν επίσης πολλά πλεονεκτήματα καθώς η διαχείριση των αναφορών καθώς και η επίλυση των προβλημάτων θα είναι πιο αποτελεσματική. Η πιο εύκολη, ακριβής αναφορά και τα upvotes θα έχουν σημαντικό αντίκτυπο στην εκτίμηση της σημαντικότητας των προβλημάτων και στη διαχείριση των πόρων. Θα είναι δυνατή η συλλογή δεδομένων, κάτι που θα οδηγήσει στη γρηγορότερη και πιο αποτελεσματική αντιμετώπιση των προβλημάτων. Τέλος, θα είναι πιο εύκολο να φανεί το έργο που παράγει η τοπική αυτοδιοίκηση.

2.3 Σύντομη αναφορά παρόμοιων εφαρμογών

2.3.1 Snap Send Solve

Μια πολύ απλή εφαρμογή που δίνει τη δυνατότητα επιλογής από μια ποικιλία κατηγοριών προβλημάτων, δυνατότητα επιλογής τοποθεσίας και (προεραιτικά) επισύναψης φωτογραφίας. Δυστυχώς η δοκιμή της εφαρμογής ήταν αδύνατη καθώς είναι διαθέσιμη μόνο στην Αυστραλία και τη Νέα Ζηλανδία.

2.3.2 Novoville

Από όσες εφαρμογές δοκιμάστηκαν αυτή ήταν η καλύτερη από άποψη UI. Από άποψη λειτουργικότητας είναι υπερπλήρης. Είναι διαθέσιμη για πόλεις της Ελλάδας, του Ηνωμένου Βασιλείου, της Ελβετίας, της Κύπρου και της Γαλλίας.

Αρχικά, οι χρήστες επιλέγουν την τοπική τους αυτοδιοίκηση. Οι χρήστες μπορούν να δουν τα εξής:

- Πόσα προβλήματα είναι εκκρεμή και πόσα έχουν επιλυθεί
- Λειτουργικά σημεία, όπως σημεία ανακύκλωσης
- Μηνύματα/εκδηλώσεις της περιοχής τους
- Τις δικές τους αναφορές
- Τις αναφορές άλλων χρηστών
- Στατιστικά αναφορών όπως τις πιο συχνές κατηγορίες αναφορών στην πόλη αλλά και στη χώρα, ποσοστά ανά στάδιο επίλυσης προβλήματος κ.α.
- Τα κοινωνικά δίκτυα της τοπικής αυτοδιοίκησης

Όσον αφορά την αναφορά προβλημάτων:

- Η εφαρμογή απαιτεί έναν αριθμό κινητού τηλεφώνου από τους χρήστες
- Το 1ο βήμα είναι η επιλογή τοποθεσίας από το χάρτη
- Το 2ο βήμα είναι η επιλογή κατηγορίας του προβλήματος
- Το 3ο βήμα είναι η περιγραφή του προβλήματος. Σε αυτό το στάδιο η αναφορά μπορεί να σημανθεί και ως Επείγουσα)
- Το 4ο βήμα είναι η επισύναψη φωτογραφίας στην αναφορά, είτε από τη βιβλιοθήκη της συσκευής είτε από την κάμερα

Στο Μενού της εφαρμογής οι χρήστες μπορούν να βρουν:

- Τις αναφορές τους
- Τα μηνύματα της πόλης τους
- Τον χάρτη στον οποίο φαίνονται όλες οι αναφορές της πόλης
- Χρήσιμα τηλέφωνα

2.3.3 CityOn

Το CityOn είναι μία εφαρμογή η οποία υποστηρίζει πόλεις της Ελλάδος. Αν κάποια πόλη δεν υποστηρίζεται για τη δημιουργία αναφορών τότε ο χρήστης μπορεί απλά να δει πληροφορίες, νέα και μέρη ενδιαφέροντος. Στην αρχική σελίδα υπάρχουν:

- Νέα της τοπικής αυτοδιοίκησης
- Τοπικά νέα
- Η δημιουργία αναφοράς

- Μέρη ενδιαφέροντος
- City Portal

Η αρχική σελίδα μπορεί να παραμετροποιηθεί ώστε να εμφανίζει μόνο τις επιλογές που επιθυμεί ο χρήστης.

Δημιουργία αναφορών:

- Υπάρχει η επιλογή να γίνει αναφορά προβλήματος ή προβολή ιστορικού
- Στην νέα αναφορά υπάρχουν τρεις κατηγορίες, Δημόσιοι Χώροι, Προσβασιμότητα, Φροντίδα). Κάθε κατηγορία έχει τις υποκατηγορίες της και κάθε υποκατηγορία έχει τις επιλογές της.
- Στο δεύτερο βήμα μιας αναφοράς ο χρήστης μπορεί να προσθέσει ένα σχόλιο
- Στο τρίτο βήμα ο χρήστης μπορεί να επιλέξει μία εικόνα από τη βιβλιοθήκη της συσκευής ή να βγάλει μία επί τόπου με την κάμερά του
- Στο τελευταίο βήμα μπορεί να δει τις επιλογές και να υποβάλει την αναφορά

Η εφαρμογή υποστηρίζει και αποστολή ειδοποιήσεων.

2.3.4 Καλαμαριά (4myCity)

Είναι μία απλή εφαρμογή. Στην αρχική οθόνη εμφανίζονται αναφορές από όλους τους χρήστες. Οι χρήστες μπορούν να βλέπουν τις αναφορές στο χάρτη και μπορούν να αλληλεπιδρούν με αναφορές άλλων χρηστών μέσω likes ή σχολίων. Η υποβολή νέας αναφοράς ακολουθεί τη λογική όλων των άλλων εφαρμογών.

2.4 Λειτουργίες που θα υλοποιηθούν

Η εφαρμογή η οποία θα αναπτυχθεί στα πλαίσια της πτυχιακής εργασίας θα περιλαμβάνει ένα συνδυασμό λειτουργιών των προαναφερθέντων εφαρμογών. Πιο συγκεκριμένα, οι χρήστες της εφαρμογής θα μπορούν να τη χρησιμοποιούν ως επισκέπτες αλλά και ως εγγεγραμμένοι χρήστες. Για την εγγραφή θα ζητούνται κάποια στοιχεία όπως ονοματεπώνυμο και λογαριασμός ηλεκτρονικού ταχυδρομείου. Στην αρχική οθόνη της εφαρμογής θα εμφανίζεται κάποιο δείγμα των αναφορών που έχουν γίνει στην πόλη. Αν ο χρήστης έχει συνδεθεί θα βλέπει και ένα δείγμα των δικών του αναφορών. Ο χρήστης θα μπορεί να βλέπει όλες τις αναφορές που έχουν γίνει από τον ίδιο, αλλά και γενικότερα στην πόλη, σε μία οθόνη που θα εμφανίζεται ο χάρτης καθώς και μία λίστα. Θα υπάρχουν τα απαραίτητα φίλτρα ώστε να εμφανίζονται οι αναφορές που θέλει να δει ο χρήστης. Όσον αφορά τη δημιουργία μιας νέας αναφοράς, θα ανοίγει ένα ξεχωριστό παράθυρο το οποίο θα χωρίζεται σε τρεις οθόνες. Η πρώτη θα είναι η οθόνη επιλογής τοποθεσίας, όπου είτε θα ανιχνεύεται αυτόματα η τοποθεσία του χρήστη είτε θα μπορεί να γίνει αναζήτηση. Η δεύτερη θα είναι η οθόνη λεπτομερειών όπου ο χρήστης θα επιλέγει την κατηγορία του προβλήματος και θα εισάγει ένα κείμενο με την περιγραφή του προβλήματος. Η τρίτη, και τελευταία, οθόνη θα είναι η οθόνη επιλογής φωτογραφίας όπου ο χρήστης θα μπορεί να επιλέξει μία από τις εικόνες που είναι ήδη αποθηκευμένες στη συσκευή του ή θα μπορεί να τραβήξει μία καινούργια επί τόπου. Τέλος, θα υπάρχει η δυνατότητα αποστολής ειδοποιήσεων σε κάθε στάδιο εξέλιξης κάποιας αναφοράς που έχει κάνει.

Κεφάλαιο 3ο: Εργαλεία που χρησιμοποιήθηκαν

3.1 Εισαγωγή

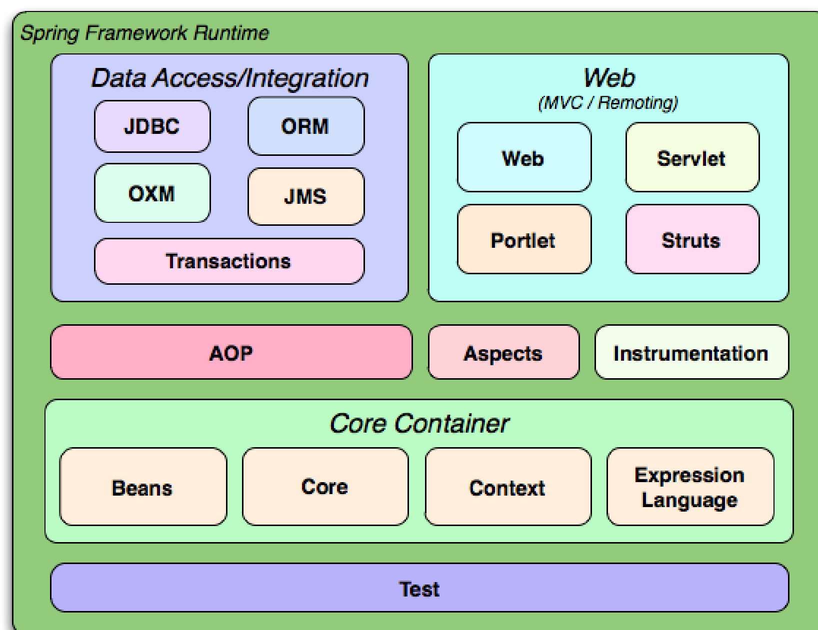
Σε αυτό το κεφάλαιο θα γίνει περιγραφή όλων των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη του API, της εφαρμογής για κινητά, καθώς και για τα περεταίρω κομμάτια όπως η βάση δεδομένων και η αποστολή ειδοποιήσεων. Αρχικά για την ανάπτυξη του API χρησιμοποιήθηκε το Spring Boot framework. Για την ανάπτυξη της mobile εφαρμογής χρησιμοποιήθηκε το React Native Framework. Για την αποθήκευση των δεδομένων των χρηστών καθώς και των αναφορών χρησιμοποιήθηκε η MongoDB, σε συνδυασμό με το Realm για παρακολούθηση της βάσης σε πραγματικό χρόνο . Για την αποστολή των ειδοποιήσεων χρησιμοποιήθηκε το Firebase. Τέλος, για το Admin Panel χρησιμοποιήθηκε η React.

3.2 Spring Framework - Spring Boot

Για την ανάπτυξη του API χρησιμοποιήθηκε το Spring Boot Framework. Σε αυτήν την ενότητα θα γίνει μια περιγραφή του Spring Framework, τις χρήσεις του, τα μέρη από τα οποία αποτελείται καθώς και τα πλεονεκτήματα και τα μειονεκτήματα. Στη συνέχεια θα γίνει το ίδιο και το Spring Boot και θα τελειώσουμε με μία σύγκριση των δύο.

3.2.1 Spring Framework

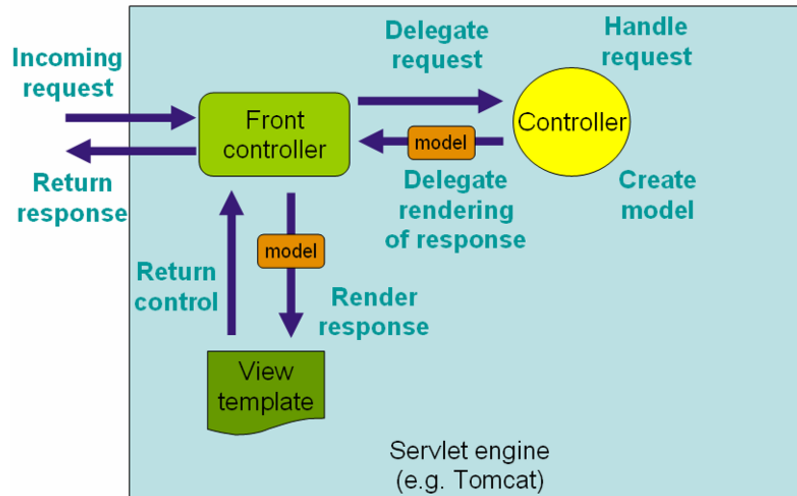
Το Spring Framework είναι μία πλατφόρμα ανάπτυξης επαγγελματικών (enterprise) εφαρμογών σε Java και είναι ανοιχτού κώδικα. Η αρχική κυκλοφορία του έγινε τον Ιούνιο του 2003 (με την πρώτη production έκδοση να κυκλοφορεί τον Μάρτιο του 2004) από τον Rod Johnson. Το framework ήταν αποτέλεσμα της ανάγκης για μια πιο ελαφριά και ευέλικτη λύση για ανάπτυξη επαγγελματικών εφαρμογών σε Java, καθώς μέχρι τότε ήταν αρκετά δύσκολη λόγω πολυπλοκότητας.



Σχήμα 3.1: Αναπαράσταση Αρχιτεκτονικής του Sping Framework

Τα βασικά μέρη/ενότητες του Spring Framework είναι τα εξής (Σχήμα 3.1):

- **IoC (Inversion of Control) Container:** Είναι ο πυρήνας, το κεντρικό κομμάτι του Spring Framework. Προσφέρει τα μέσα για τη διαχείριση και ρύθμιση Java αντικειμένων, ουσιαστικά διαχειρίζεται τον κύκλο ζωής των αντικειμένων και την επικοινωνία μεταξύ τους. Τα παραγόμενα αντικείμενα ονομάζονται beans. Πρακτικά τα αντικείμενα ορίζουν τα dependencies (άλλα αντικείμενα, δηλαδή, με τα οποία συνεργάζονται ή τα οποία χρειάζονται) τους και το IoC container κάνει inject τα dependencies όταν δημιουργεί το bean (αντικείμενο). Για αυτό το λόγο και ονομάζεται Inversion of Control (Αντιστροφή του Ελέγχου) καθώς αντί το ίδιο το αντικείμενο να ελέγχει τα dependencies, αυτό το κάνει το IoC container του Framework [3]. Υπάρχουν διάφοροι τύποι Inversion of Control, όπως Dependency Injection, Dependency Lookup και Autowiring.
- **Aspect-Oriented Programming (AOP):** Το Aspect-Oriented Programming (AOP) είναι ένα προγραμματιστικό μοντέλο που επιτρέπει στους προγραμματιστές να διαχωρίζουν τις πτυχές (aspects) ενός λογισμικού από τον κύριο κώδικά του. Με το AOP μπορεί κανείς να ορίσει πτυχές που αφορούν στην διατομεακή (crosscutting) λειτουργικότητα, όπως η καταγραφή ή η ασφάλεια, και να τις εφαρμόζει διαμέσου του προγράμματος χωρίς να αλλοιώνει τον βασικό κώδικα. Αυτό βοηθάει στον καθαρότερο και πιο επαναχρησιμοποιήσιμο κώδικα, καθώς και στην απλούστευση των διαδικασιών συντήρησης και αναβάθμισης [4][5].
- **Data Access:** ο Data Access Framework παρέχει μια αφαιρετική στρώση για τη διαχείριση των προεργασιών πρόσβασης σε δεδομένα από διάφορες πηγές. Έχει σχεδιαστεί για να απλοποιεί τον κώδικα, να τον κάνει λιγότερο επαναλήψιμο και να διαχειρίζεται εξαιρέσεις. Υποστηρίζει μια πληθώρα τεχνολογιών πρόσβασης σε δεδομένα, όπως JDBC, Hibernate, JPA και JMS, ενσωματώνοντας έναν κεντρικό μηχανισμό διαχείρισης συναλλαγών (transactions) και ένα σύστημα διαχείρισης εξαιρέσεων που είναι εύκολα διαχειρίσιμο ανεξάρτητα από το ποια τεχνολογία πρόσβασης δεδομένων χρησιμοποιείται [6][7].
- **Web MVC Framework:** Το Spring Web MVC παρέχει τα εργαλεία για τη δημιουργία web εφαρμογών με βάση το γνωστό μοντέλο Model-View-Controller (Σχήμα 3.2) (όπου το Model αναφέρεται στα δεδομένα, το View στην παρουσίαση αυτών και το Controller στη διαχείριση των αιτημάτων του χρήστη και τον συντονισμό των Model και View. Το Spring MVC, όπως και πολλά άλλα web MVC Frameworks, είναι εξαρτώμενο από αιτήματα (request-driven) που σημαίνει πως είναι σχεδιασμένο γύρω από ένα Servlet το οποίο προωθεί αιτήματα στους controllers. Το Servlet της Spring (DispatcherServlet) είναι εντελώς ενσωματωμένο με το IoC container και έτσι μπορεί να χρησιμοποιεί όλες τις υπόλοιπες λειτουργίες της Spring [8].



Σχήμα 3.2: Αναπαράσταση της λειτουργίας του Web MVC Framework

3.2.2 Πλεονεκτήματα

- **Ευελιξία:** Το IoC (Inversion of Control) μαζί με τα Dependency Injections, η ποικιλία annotations και επιλογών ρυθμίσεων, σε συνδυασμό με πολλές έμπιστες βιβλιοθήκες θέτουν τις βάσεις για ένα μεγάλο εύρος λειτουργιών που κάνουν τη δουλειά του προγραμματιστή πολύ ευκολότερη.
- **Είναι ελαφρύ:** Το Spring Framework χρησιμοποιεί Plain Old Java Objects (POJOs), καθιστώντας το ελαφρύ. Οι προγραμματιστές δεν είναι υποχρεωμένοι να επεκτείνουν συγκεκριμένες κλάσεις ή να υλοποιούν συγκεκριμένες διεπαφές.
- **Loose Coupling:** Χάρη στο Dependency Injection του Spring Framework τα μέρη της εφαρμογής δεν γνωρίζουν την προέλευσή τους και η εξάρτηση με τα υπόλοιπα μέρη είναι χαλαρή. Έτσι, οι αλλαγές στο ένα μέρος δεν επηρεάζει τη λειτουργία των υπολοίπων.
- **Αφαιρετικότητα:** Προσφέρει δυνατά abstractions για ποικίλα πρότυπα του JEE (Java Enterprise Edition) όπως το JMS(Java Messaging Service), JDBC, JPA(Java Persistence API), JTA(Java Transaction API) και άλλα.
- **Ασφάλεια:** Υπάρχει καλός έλεγχος των τρίτων βιβλιοθηκών. Η χρήση του Spring Security framework μπορεί να κάνει τις εφαρμογές πολύ ασφαλείς.
- **Εργαλεία:** Το Spring Framework προσφέρει στον πυρήνα του πολλά εργαλεία, πολλές κλάσεις και πακέτα για ένα πολύ μεγάλο εύρος λειτουργιών, πράγμα που σημαίνει ότι δεν είναι απαραίτητη η χρήση τρίτων πακέτων ή βιβλιοθηκών για την ανάπτυξη μιας εφαρμογής

3.2.3 Μειονεκτήματα

- **Καμπύλη Εκμάθησης:** Η χρήση του Spring Framework απαιτεί κάποια προϋπάρχουσα εμπειρία. Έτσι, για έναν νέο χρήστη, η εκμάθηση και εξοικείωση με αυτό χρειάζεται πολύ χρόνο.
- **Πολυπλοκότητα:** Οι πολλές επιλογές για εργαλεία, κάτι που αναφέρθηκε ως πλεονέκτημα παραπάνω, μπορεί υπό συνθήκες να θεωρηθεί και ως μειονέκτημα καθώς μπορεί να προσθέσει μεγαλύτερη σύγχυση σε μη έμπειρους χρήστες.

3.2.4 Spring Boot

Το Spring Boot είναι ένα framework που καθιστά την ανάπτυξη Spring based εφαρμογών πολύ ευκολότερη και γρηγορότερη, με όλα τα πλεονεκτήματα του Spring Framework αλλά με ελάχιστο configuration [9].

- Το Spring Boot επιτρέπει τη δημιουργία stand-alone εφαρμογών
- Παρέχει ενσωματωμένους web servers, όπως Tomcat και Jetty
- Παρέχει αρχικά dependencies ώστε να απλουστευθεί το configuration
- Παρέχει αυτόματες ρυθμίσεις για βιβλιοθήκες του Spring αλλά και τρίτων
- Παρέχει εργαλεία που είναι έτοιμα για production εφαρμογές
- Δε χρειάζεται καθόλου XML configuration

Ένα πολύ βασικό πλεονέκτημα του Spring Boot είναι ότι αναλαμβάνει ρυθμίσεις αυτόματα, επιτρέποντας στον χρήστη να επικεντρωθεί περισσότερο στη λογική και λιγότερο στη δομή της εφαρμογής. Για παράδειγμα, αν το Spring Boot βρει το Spring MVC στο classpath θα προσπαθήσει να προσθέσει ότι χρειάζεται όπως, για παράδειγμα, ρυθμίσεις για τον Tomcat web server. Αν από την άλλη βρει το Jetty (έναν διαφορετικό web server) θα χειριστεί την κατάσταση ανάλογα και δε θα προσθέσει ρυθμίσεις για τον Tomcat, αφού δε χρειάζεται.

3.3 MongoDB - Realm

Η MongoDB είναι μία μη-σχεσιακή (non-relational) βάση εγγράφων (document database) που έχει σχεδιαστεί για να προσφέρει ευκολία στην ανάπτυξη εφαρμογών και στο scaling τους.

3.3.1 Τι είναι μία βάση εγγράφων;

Αρχικά ας δούμε τι είναι μία βάση εγγραφών. Μία βάση εγγράφων είναι, όπως γίνεται προφανές και από το όνομα, μία βάση η οποία αποθηκεύει τα δεδομένα σε έγγραφα (documents) και όχι σε γραμμές και στήλες, όπως γίνεται στις σχεσιακές βάσεις δεδομένων [10].

Τα βασικά μέρη μιας βάσης εγγράφων είναι:

- **Έγγραφα (Documents):** Το έγγραφο είναι μία καταγραφή στη βάση. Στα έγγραφα αποθηκεύονται δεδομένα σε ζεύγη πεδίου-τιμής (field-value), μπορούν να αποθηκευτούν σε μορφές JSON, BSON και XML και οι τιμές μπορούν να είναι διαφόρων τύπων, για παράδειγμα string, αριθμοί, ημερομηνίες, πίνακες και αντικείμενο.
- **Συλλογές (Collections):** Οι συλλογές είναι μία ομάδα εγγράφων και συνήθως αποθηκεύουν έγγραφα με παρόμοιο περιεχόμενο. Λόγω της ευελιξίας των βάσεων εγγράφων δεν είναι υποχρεωτικό για όλα τα έγγραφα να έχουν τα ίδια πεδία, αν και σε μερικές παρέχεται η δυνατότητα ελέγχου της δομής (schema validation). Ένα παράδειγμα θα μπορούσε να είναι μία συλλογή με τίτλο reports. Σε αυτήν την συλλογή θα μπορούσε να αποθηκευτεί ένα έγγραφο report με πεδία date, type, location, reporterFirstName, reporterLastName αλλά και ένα έγγραφο report με πεδία date, type, location, description, image χωρίς κανένα απολύτως πρόβλημα.

3.3.2 Διαφορές βάσεων εγγράφων με τις σχεσιακές βάσεις δεδομένων

Οι κυριότεροι λόγοι για τους οποίους οι βάσεις εγγράφων διαφέρουν από τις σχεσιακές βάσεις είναι οι εξής:

- Το μοντέλο των δεδομένων είναι οικείο, καθώς τα έγγραφα γίνονται αντιστοιχίζονται απευθείας σε αντικείμενα με αποτέλεσμα να απλουστεύεται η πρόσβαση των δεδομένων. Η πρόσβαση των δεδομένων γίνεται πολύ γρήγορα, καθώς αποθηκεύονται όλα μαζί, χωρίς να χρειάζεται να ανατρέξει κανείς σε πολλούς πίνακες, να βρει τα δεδομένα, να τα συγχωνεύσει ή να τα διαχωρίσει. Έτσι χρειάζεται λιγότερος κώδικας και η απόδοση αυξάνεται σημαντικά
- Τα JSON objects είναι πλέον παντού. Έχουν γίνει στάνταρ για την αποθήκευση και ανταλλαγή δεδομένων. Είναι ελαφριά, ανεξάρτητα από τη γλώσσα προγραμματισμού που χρησιμοποιείται και κυρίως είναι αναγνώσιμα από τον άνθρωπο.
- Η δομή των δεδομένων είναι ευέλικτη. Οι χρήστες δε χρειάζεται να ορίσουν εξ αρχής το σχήμα της βάσης τους και τα έγγραφα μπορούν να έχουν διαφορετικά πεδία μεταξύ τους. Αυτό δίνει την ελευθερία αλλαγής της δομής ανά πάσα στιγμή και ανάλογα με τις ανάγκες της εφαρμογής, οι οποίες μπορούν να εξελίσσονται. Έτσι, υπάρχει και κέρδος χρόνου καθώς δε χρειάζεται migration της δομής αν χρειαστεί κάποια αλλαγή.

3.3.3 MongoDB

Όπως περιγράφηκε παραπάνω η MongoDB, όντας μία βάση εγγράφων, κληρονομεί και όλα τα θετικά αυτών. Έτσι την καθιστά ικανή για μία πληθώρα χρήσεων όπως διαχείριση δεδομένων πελατών, διαδίκτυο των πραγμάτων, καταλόγους προϊόντων και διαχείριση περιεχομένου, επεξεργασία πληρωμών, εφαρμογές για κινητά, αλλά και λειτουργικά analytics και analytics πραγματικού χρόνου [11].

Η MongoDB μπορεί να τρέξει μέσω:

- Του MongoDB Atlas, το οποίο είναι μια πλατφόρμα της MongoDB Cloud Services σουίτας μέσω του οποίου η αποθήκευση και διαχείριση της βάσης γίνεται στο cloud [12].
- Της MongoDB Community, η οποία είναι η δωρεάν έκδοση της MongoDB
- Της MongoDB Enterprise, η οποία προσφέρει παραπάνω λειτουργίες όπως υποστήριξη για SNMP monitoring, LDAP authentication, Kerberos authentication και System Event Auditing

3.3.4 Realm Web SDK

Το Realm Web SDK προσφέρει τη δυνατότητα σε browser εφαρμογές να αποκτήσουν πρόσβαση σε δεδομένα που έχουν αποθηκευτεί στο Atlas και να αλληλεπιδράσουν με αυτά μέσω των App Services [13]. Χρειάζονται μόνο λίγα και απλά βήματα ώστε να γίνει η επικοινωνία της browser based εφαρμογής με τη βάση:

- Η δημιουργία μιας App Services εφαρμογής
- Η σύνδεση σε αυτήν την εφαρμογή από τον browser
- Η αυθεντικοποίηση ενός χρήστη (υπάρχει η δυνατότητα πρόσβασης των δεδομένων σε ανώνυμους χρήστες αν το επιλέξει κάποιος)
- Η αναζήτηση δεδομένων από τον browser χρησιμοποιώντας τον παραπάνω αυθεντικοποιημένο χρήστη.

3.4 React Native

Για την ανάπτυξη της εφαρμογής για κινητά χρησιμοποιήθηκε η React Native. Η React Native είναι ένα framework ανοιχτού κώδικα, το οποίο δημιουργήθηκε από την τότε Facebook (τώρα Meta Platforms Inc.) για τις ανάγκες της ανάπτυξης και αναβάθμισης της εφαρμογής Facebook για κινητά, και η πρώτη του έκδοση έγινε διαθέσιμη το 2015. Επιτρέπει την ανάπτυξη εφαρμογών για κινητά σε Android και iOS συνδυάζοντας την React με τις native λειτουργίες της εκάστοτε πλατφόρμας [14]. Για την ανάπτυξη των εφαρμογών χρησιμοποιείται JavaScript, καθώς και η TypeScript με την τελευταία να έχει ανέβει πολύ σε δημοτικότητα στις περισσότερες νέες εφαρμογές και να χρησιμοποιείται πλέον ως προεπιλογή όταν δημιουργείται ένα νέο project.

3.4.1 React Native Core Components

Η κάθε πλατφόρμα έχει τα δικά της views, όπως ViewGroup, ImageView, TextView για Android και UIView, UIImageView, UITextView για iOS αντίστοιχα, για να δείξει διάφορα στοιχεία στον χρήστη (κείμενο, εικόνες, κουμπιά). Με τη React Native μπορούν να χρησιμοποιηθούν αυτά τα views μέσω των React components, καθώς, τη στιγμή που η εφαρμογή τρέχει η React Native δημιουργεί τα αντίστοιχα native views της κάθε πλατφόρμας. Στο (Σχήμα 3.3) φαίνεται ένας πίνακας αντιστοίχισης των React Native components με τα αντίστοιχα views κάθε πλατφόρμας. Αυτό κάνει τις React Native εφαρμογές να φαίνονται και να αποδίδουν σχεδόν πανομοιότυπα με όλες τις υπόλοιπες εφαρμογές [15].

Η React Native παρέχει πολλά αρχικά components στον πυρήνα της, τα Core Components, ώστε να μπορεί να ξεκινήσει οποιοσδήποτε να αναπτύσσει την εφαρμογή του. Ακόμη, δίνει τη δυνατότητα ανάπτυξης των δικών του components. Τέλος, υπάρχει μία τεράστια κοινότητα η οποία αναπτύσσει και συντηρεί πολλά components για ποικίλες χρήσεις.

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Σχήμα 3.3: Πίνακας αντιστοίχισης των React Native UI Components με τα αντίστοιχα Views κάθε πλατφόρμας

3.4.2 Βασικές έννοιες της React

Καθώς η React Native χρησιμοποιεί τη React, θα γίνει μια αναφορά στις βασικές έννοιες της React.

- **Components:** Τα components είναι πρακτικά επαναχρησιμοποιήσιμα κομμάτια κώδικα, μπορούν να θεωρηθούν σαν blueprints που περιγράφουν τι θα φαίνεται στην οθόνη [16]. Το component ορίζεται ως μία μέθοδος η οποία επιστρέφει ένα element με κανένα εώς πολλά παιδιά. Έπειτα γίνεται export και μπορεί να χρησιμοποιηθεί απλά κάνοντάς το import σε κάποιο άλλο κομμάτι του κώδικα. Το (Σχήμα 3.4) παρέχει ένα παράδειγμα κώδικα ενός component.

```
const Cat = () => {
  return <Text>Hello, I am your cat!</Text>;
};

export default Cat;
```

Σχήμα 3.4: Παράδειγμα κώδικα ενός απλού component

- **JSX:** Καθώς με τα χρόνια άρχισε να υπάρχει μεγαλύτερη αλληλεπίδραση των χρηστών με τις web εφαρμογές δημιουργήθηκε η ανάγκη να γίνει συνδυασμός της εμφάνισης της εφαρμογής με τη λογική της, αφού πλέον η εμφάνιση ήταν άμεσα συνδεδεμένη με τη λογική. Αυτή την ανάγκη ήρθε να καλύψει το JSX, το είναι μια επέκταση στο συντακτικό της JavaScript που επιτρέπει τη χρήση markup στοιχείων μέσα σε ένα αρχείο JavaScript [17]. Έτσι δημιουργούνται, όπως αναφέρθηκε και παραπάνω, τα components. Με τη χρήση του JSX συντακτικού μπορεί κανείς επίσης να περάσει μεταβλητές, ακόμη και μεθόδους δυναμικά μέσα σε κάποιο Element χρησιμοποιώντας άγκιστρα, ανοίγοντας έτσι ένα μικρό παράθυρο στην JavaScript. Στο (Σχήμα 3.5) και (Σχήμα 3.6) φαίνονται παραδείγματα χρήσης JSX συντακτικού.

```
1  export default function TodoList() {
2    const name = 'Gregorio Y. Zara';
3    return (
4      <h1>{name}'s To Do List</h1>
5    );
6  }
7
```

Σχήμα 3.5: Παράδειγμα κώδικα χρήσης άγκιστρων για δυναμικό κείμενο σε component με χρήση μεταβλητής

```
1  const today = new Date();
2
3  function formatDate(date) {
4    return new Intl.DateTimeFormat(
5      'en-US',
6      { weekday: 'long' }
7    ).format(date);
8  }
9
10 export default function TodoList() {
11   return (
12     <h1>To Do List for {formatDate(today)}</h1>
13   );
14 }
15
```

Σχήμα 3.6: Παράδειγμα κώδικα χρήσης άγκιστρων για δυναμικό κείμενο σε component με χρήση μεθόδου

- **Properties:** Τα properties δίνουν τη δυνατότητα παραμετροποίησης των components [16]. Όλα τα Core components έχουν τα δικά τους properties. Για παράδειγμα το View component έχει το style prop που δίνει τη δυνατότητα παραμετροποίησης της εμφάνισής του. Το Image component έχει το source prop με το οποίο ορίζεται η πηγή της εικόνας που θα χρησιμοποιηθεί. Properties μπορούμε να οριστούν και σε custom components. Για παράδειγμα, αν έχουμε ένα component Car το οποίο επιστρέφει ένα Text με τα στοιχεία ενός αυτοκινήτου, μπορούν να οριστούν properties όπως manufacturer, modelName, modelNumber, dateOfManufacturing, τα οποία θα κάνουν το κείμενο που επιστρέφεται δυναμικό, ανάλογα με τις τιμές των properties.
- **State:** Το state βοηθάει στη διαχείριση δεδομένων τα οποία μπορεί να αλλάζουν είτε για να ανανεώνονται ώστε ο χρήστης να βλέπει πάντα τα σωστά, είτε ως αποτέλεσμα αλληλεπίδρασης του χρήστη με την εφαρμογή, είτε για διάφορους άλλους λόγους. Πρακτικά είναι το προσωπικό μέσο αποθήκευσης των components. Ένα παράδειγμα είναι ένα ηλεκτρονικά κατάστημα. Μπορεί να υπάρχει ένα component που δείχνει μία λίστα προϊόντων. Χρειάζεται αυτή η λίστα να είναι πάντα ανανεωμένη, ώστε ο χρήστης να βλέπει τα σωστά προϊόντα, αλλά και τα σωστά στοιχεία αυτών. Τα προϊόντα, λοιπόν, θα αποθηκευτούν στο state του component και όταν η λίστα ή τα στοιχεία κάποιων προϊόντων αλλάξουν, αυτή η αλλαγή θα αποτυπωθεί και στο component [16].

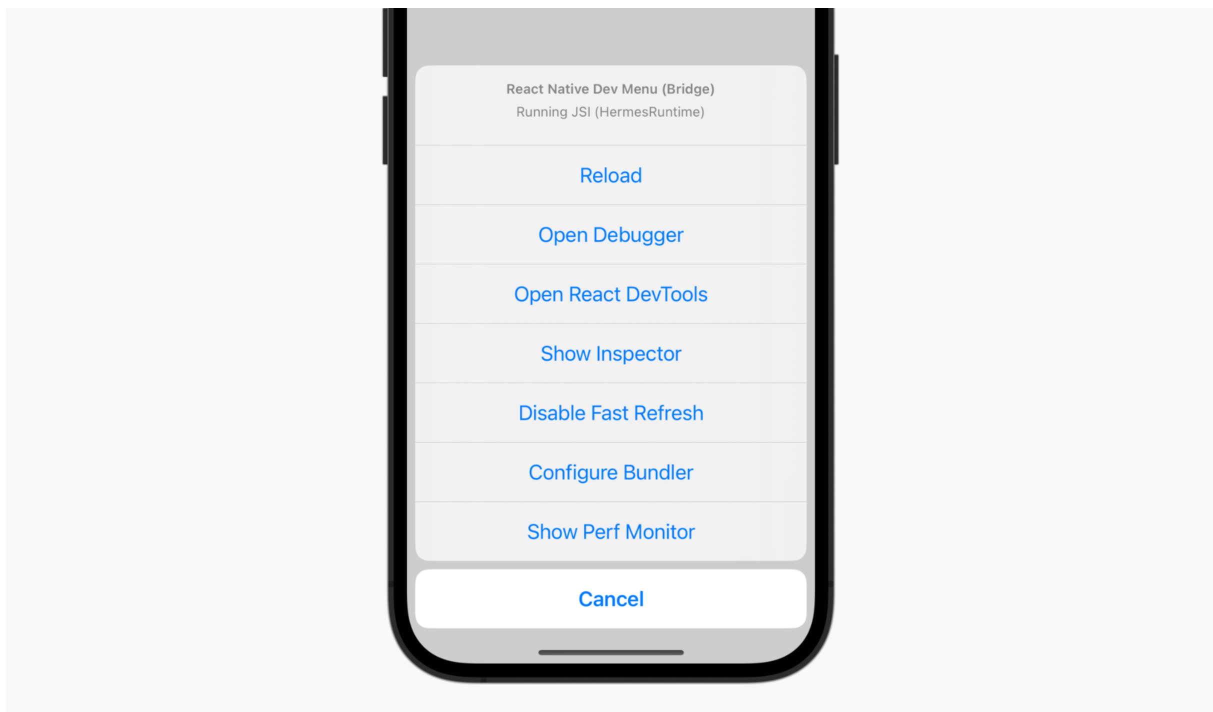
3.4.3 Metro Bundler

Το Metro είναι ένας JavaScript bundler για React Native, ο οποίος μετατρέπει τον JavaScript ES6 και JSX κώδικα χρησιμοποιώντας τη Babel σε JavaScript που μπορεί να εκτελεστεί σε κινητές συσκευές και παράγει ένα εκτελέσιμο αρχείο. Στο στάδιο ανάπτυξης μιας εφαρμογής όταν εκτελείται σε κάποια συσκευή ή simulator, το Metro είναι αυτό που τρέχει στο παρασκήνιο. Τα θετικά του Metro είναι ότι είναι αρκετά optimised, κάτι που το κάνει πολύ αποδοτικό, είναι ανεπτυγμένο αποκλειστικά για τη React Native, παρέχει Hot Reloading που κάνει τη δουλειά του προγραμματιστή απείρως γρηγορότερη και ευκολότερη καθώς δε χρειάζεται να χτιστεί από την αρχή η εφαρμογή για να γίνουν εμφανείς οι αλλαγές, κάνει bundle τα assets όπως εικόνες και fonts, παράγει source maps που είναι χρήσιμα για την αποτελεσματικότερη εύρεση σφαλμάτων και την ευκολότερη διαχείρισή τους και, τέλος, είναι αρκετά παραμετροποιήσιμο επιτρέποντας στον προγραμματιστή να ορίσει το πως θα γίνουν οι μετατροπές των αρχείων κλπ. [18] [19]

3.4.4 Ανίχνευση Σφαλμάτων

Η React Native προσφέρει διάφορα εργαλεία για την εύρεση και διαχείριση σφαλμάτων.

- **Dev Menu:** Η React Native παρέχει ένα μενού προγραμματιστή το οποίο είναι διαθέσιμο εντός της εφαρμογής και διαθέτει αρκετές επιλογές [20] (Σχήμα 3.7). Μπορεί κανείς να έχει πρόσβαση στο μενού μέσω συντομεύσεων (Cmd + D για iOS simulators, Cmd + M / Ctrl + M για Android Emulators) ή πατώντας το πλήκτρο R στο παράθυρο που εκτελείται ο Metro server.



Σχήμα 3.7: Dev Menu όπως εμφανίζεται σε μία iOS συσκευή

- **LogBox:** Τα σφάλματα κονσόλας και τα warnings εμφανίζονται σε ένα LogBox σαν ειδοποίηση πάνω από την εφαρμογή και δείχνει τον αριθμό των σφαλμάτων ή warnings. Πατώντας το LogBox επεκτείνεται και ο προγραμματιστής μπορεί να δει όλες τις

λεπτομέρειες του σφάλματος.
Τα JavaScript σφάλματα ανοίγουν απευθείας σε πλήρη οθόνη το LogBox με τις λεπτομέρειες του σφάλματος. Το LogBox μπορεί να ελαχιστοποιηθεί και να αγνοηθεί το σφάλμα, αλλά το ιδανικό θα ήταν να επιλυθεί.
Τα σφάλματα συντακτικού επίσης ανοίγουν σε πλήρη οθόνη το LogBox με το πλήρες stack trace και τη γραμμή στην οποία υπάρχει λάθος σύνταξη. Σε αυτή την περίπτωση το LogBox δε μπορεί να ελαχιστοποιηθεί καθώς πρέπει να διορθωθεί το σφάλμα ώστε να συνεχίσει η εκτέλεση της εφαρμογής [20].

- **Chrome Dev Tools:** Από το Dev Menu μπορεί κανείς να επιλέξει την επιλογή "Open Debugger". Θα ανοίξει μία καινούργια καρτέλα από την οποία μπορούν να ανοίξουν τα Developer Tools. Τα Developer Tools προσφέρουν πολλές επιλογές με τις οποίες μπορεί να γίνει η διαχείριση σφαλμάτων όπως το Sources κάτω από το Debugger Tab, το Console Tab και το Network tab [20].
- **Safari Dev Tools:** Παρομοίως στις iOS συσκευές μπορεί να γίνει η διαχείριση σφαλμάτων μέσω των Safari Dev Tools [20].
- **Flipper:** Το Flipper είναι μία εφαρμογή η οποία σε συνδυασμό με τα Android και iOS SDKs και έναν JavaScript client προσφέρει ένα σύνολο εργαλείων που βοηθούν στη διαχείριση σφαλμάτων σε εφαρμογές για κινητά αλλά και σε web εφαρμογές. Το Flipper σε συνεργασία με τις ομάδες της React Native και των Developer Tools, αλλά και με τη βοήθεια τρίτων plugins προσφέρει πολλά βοηθητικά εργαλεία που κάνουν τη διαχείριση σφαλμάτων στη React Native πολύ ευκολότερη. Προσφέρει τη δυνατότητα να βλέπει κανείς το Network traffic, το δέντρο των components όπως αλλάζουν σε πραγματικό χρόνο, τις αλλαγές στο redux store, καθώς και να κάνει Profiling στην εφαρμογή [21] [22].

3.4.5 Performance

Η React Native προσπαθεί να προσφέρει τις καλύτερες δυνατές επιδόσεις και καταφέρνει να φτάνει πολύ κοντά σε αυτές μιας αμιγώς native εφαρμογής. Βέβαια, λόγω του τρόπου λειτουργίας της η εξίσωση των επιδόσεων δεν είναι δυνατή καθώς τρέχει μία ενδιάμεση διεργασία, μία γέφυρα (bridge), η οποία στέλνει δεδομένα από την JavaScript στη native πλευρά και αντίστροφα, κάτι το οποίο έχει επίπτωση στους χρόνους [23].

Καρέ και threads

Τα καρέ ανά δευτερόλεπτο (fps) επηρεάζουν πολύ την εμπειρία των χρηστών όταν χρησιμοποιούν μία εφαρμογή καθώς όταν δε γίνεται σωστή διαχείριση των πόρων μπορεί να "χαθεί" ένα καρέ (dropped frame) κάτι που θα γίνει αντιληπτό στο μάτι. Για παράδειγμα, αν μία οθόνη δείχνει 60 καρέ το δευτερόλεπτο και η εφαρμογή κάνει πάνω από 16.67ms (όσο διαρκεί δηλαδή ένα καρέ) να κάνει τους απαραίτητους υπολογισμούς και να δείξει το τελικό αποτέλεσμα, τότε θα φανεί το κόλλημα (lag) που θα προκύψει από το χαμένο αυτό καρέ [24].

Αν ανοίξει κανείς το Performance Monitor από το Developer Menu που αναφέρθηκε παραπάνω, θα παρατηρήσει ότι υπάρχει δύο διαφορετικά framerates, αυτό του UI και αυτό της JavaScript.

JavaScript Thread

Το JavaScript thread είναι αυτό στο οποίο εκτελείται η λογική της εφαρμογής, εκεί όπου γίνονται διάφοροι υπολογισμοί, τα API calls, ο χειρισμός διάφορων events κλπ. Στην περίπτωση που οι εκάστοτε υπολογισμοί πάρουν πολύ χρόνο ή το αποτέλεσμα αυτών οδηγήσει στο να γίνουν re-render

πολύ βαριά components, τότε όσα animations ελέγχονται από την JavaScript θα σταματήσουν προσωρινά. Τέτοιες καθυστερήσεις δεν είναι σημαντικές αν κρατήσουν πολύ λίγο, αν δηλαδή χαθούν 1-2 καρέ. Αν όμως ο αριθμός των χαμένων καρέ είναι πάνω από 6 αυτό θα γίνει αντιληπτό από τον χρήστη της εφαρμογής [23].

UI Thread (main thread)

Το UI thread είναι υπεύθυνο για το rendering των views στο Android και στο iOS [25] αφού γίνουν οι απαραίτητες διεργασίες στο JS Thread.

Συνήθεις λόγοι για μειωμένο performance [23]

- **Η εκτέλεση της εφαρμογής σε development mode:** Όταν η εφαρμογή εκτελείται σε development mode, οι επιδόσεις πέφτουν καθώς τρέχουν διεργασίες στο παρασκήνιο για την καταγραφή σφαλμάτων. Οπότε, όταν χρειαστεί να δοκιμαστούν οι επιδόσεις της εφαρμογής θα πρέπει να γίνεται πάντα σε release builds
- **Το αχρείαστα re-renders:** Υπάρχουν πολλοί λόγοι για τους οποίους μπορούν να γίνονται αχρείαστα re-renders και κυρίως είναι στην ευθύνη το προγραμματιστή να κάνει τη σωστή διαχείριση. Ένας πολύ συνηθισμένος λόγος είναι ο μη έλεγχος των δεδομένων ή των properties από τα οποία εξαρτάται ένα components. Για παράδειγμα, ας γίνει η υπόθεση ότι υπάρχει μία λίστα η οποία δείχνει προϊόντα. Υπάρχει η περίπτωση αυτή η λίστα να γίνει re-render όταν γίνει ανανέωση της ακόμα και αν τα δεδομένα που πρέπει να δείξει είναι ακριβώς ίδια με τα προηγούμενα.
- **Μεγάλη απασχόληση του JS Thread:** Κύριο αποτέλεσμα του να απασχολείται για μεγάλο χρονικό διάστημα το JS Thread είναι τα αργά animations, κάτι που μπορεί να διορθωθεί σε μεγάλο βαθμό αν χρησιμοποιηθούν τα LayoutAnimations τα οποία δεν επηρεάζονται καθόλου από το JS Thread.
- **Αργά navigation transitions:** Τα animations του Navigator ελέγχονται από το JS Thread και σε κάθε ένα καρέ θα πρέπει να υπολογίζεται η επόμενο κατάσταση του animation. Η λύση για αυτό είναι να υπολογίζονται εξ αρχής όλες οι καταστάσεις του animation πριν αυτό ξεκινήσει και να στέλνονται στο main thread ώστε να εκτελείται με πολύ λιγότερες διακοπές. Αυτό το πρόβλημα λύνεται με τη χρήση της React Navigation βιβλιοθήκης.

3.4.6 Λίγα λόγια για την TypeScript

Η TypeScript είναι ένα υπερσύνολο (superset) της JavaScript, και πιο συγκεκριμένα ένα typed υπερσύνολο, το οποίο προσφέρει στατικό έλεγχο τύπων (static type checking). Η JavaScript επιτρέπει την εκτέλεση κάποιων γραμμών στις οποίες άλλες γλώσσες θα έβρισκαν κάποιο σφάλμα. Για παράδειγμα, η JavaScript επιτρέπει την πρόσβαση σε ένα ανύπαρκτο property ενός αντικειμένου. Αν δηλαδή, υπάρχει ένα αντικείμενο shape με properties width και height, η JavaScript επιτρέπει την πρόσβαση στο ανύπαρκτο property color γράφοντας “const color = shape.color;”. Η TypeScript λόγω του ελέγχου τύπου θα βρει και θα επισημάνει το σφάλμα πριν την εκτέλεση του κώδικα. Παράδειγμα κώδικα σε TypeScript φαίνεται στο (Σχήμα 3.8).

Η TypeScript λειτουργεί με τύπους που σημαίνει ότι για κάθε μεταβλητή ή μέθοδο χρειάζεται να δηλώνεται ο τύπος της μεταβλητής ή ο τύπος των παραμέτρων ή αυτού που επιστρέφει η μέθοδος. Να

σημειωθεί, ότι στις μεταβλητές δεν είναι απαραίτητο να δηλωθεί ο τύπος αν είναι κάποιος από τους βασικούς, όπως string ή number.

Τέλος, χρειάζεται να σημειωθεί ότι κάποιος γνωρίζοντας ήδη JavaScript ουσιαστικά γνωρίζει σε πολύ μεγάλο βαθμό και TypeScript καθώς μοιράζονται ένα τεράστιο σύνολο κανόνων και συντακτικού [26].

```
interface Pointlike {
  x: number;
  y: number;
}
interface Named {
  name: string;
}

function logPoint(point: Pointlike) {
  console.log("x = " + point.x + ", y = " + point.y);
}

function logName(x: Named) {
  console.log("Hello, " + x.name);
}
```

Σχήμα 3.8: Παράδειγμα συντακτικού στην TypeScript

3.5 Firebase

Για τη λειτουργία των ειδοποιήσεων (Push Notifications) χρησιμοποιήθηκε το Firebase και συγκεκριμένα το Firebase Cloud Messaging. Το Firebase είναι μία πλατφόρμα της Google με μία τεράστια ποικιλία εργαλείων που βοηθούν στην ανάπτυξη mobile και web εφαρμογών και παιχνιδιών, στη διανομή τους αλλά και στη συλλογή δεδομένων από αυτά [27]. Μπορεί να ενσωματωθεί σε Android, iOS (συμπεριλαμβανομένων και των React Native, Flutter frameworks), Web, Unity, C++ projects. Κάποιες από τις λειτουργίες που παρέχει είναι:

- **Realtime Database**, που επιτρέπει την αποθήκευση και τον συγχρονισμό JSON δεδομένων μεταξύ χρηστών και σε πραγματικό χρόνο, αλλά και offline.
- **Remote Config**, που επιτρέπει την ενεργοποίηση ή απενεργοποίηση λειτουργιών μιας εφαρμογής κάτι που βοηθάει στην παραμετροποίηση της εμπειρίας ανά χρήστη και επιτρέπει την δοκιμή λειτουργιών χωρίς την ανάγκη αναβάθμισης της εφαρμογής σε νέα έκδοση.
- **Authentication**
- **Cloud Messaging**, που επιτρέπει την αποστολή ειδοποιήσεων μεταξύ server και συσκευών χωρίς κόστος
- **Google Analytics**, ώστε να μπορεί να γίνεται έλεγχος των πηγών από τις οποίες αποκτά χρήστες μιας εφαρμογής και της χρήσης καινούργιων λειτουργιών αυτής.
- **Test Lab**, που επιτρέπει το Testing της εφαρμογής σε πραγματικές συσκευές αλλά και σε simulators
- Crashlytics, για αναφορές των crashes μιας εφαρμογής

3.6 Amazon Web Services (Hosting του Server)

To AWS (Amazon Web Services) είναι μία πλατφόρμα cloud computing της Amazon η οποία προσφέρει μία πληθώρα από services που μπορούν να χρησιμοποιηθούν για πάρα πολλά πράγμα από το να κάνουν host websites ή servers μέχρι το να αναλύουν δεδομένα. Τα μοντέλο του AWS είναι pay-as-you-go που σημαίνει ότι χρήστες πληρώνουν ανάλογα με τη χρήση που κάνουν. Έτσι οι επιχειρήσεις έχουν την ευελιξία που χρειάζονται ώστε να κάνουν scale up ή scale down ανάλογα με τις ανάγκες τους χωρίς να χρειάζεται να συντηρούν δικούς τους servers.

Αναφορικά κάποια από τα βασικά AWS services είναι:

- **Compute:** EC2 (Elastic Compute Cloud), Lambda, Elastic Beanstalk
- **Storage:** S3 (Simple Storage Service), EBS (Elastic Block Store), Glacier
- **Databases:** RDS (Relational Database Service), DynamoDB, ElastiCache
- **Networking:** VPC (Virtual Private Cloud), CloudFront
- **Developer Tools:** CodeBuil, CodeDeploy, CodePipeline
- **Analytics:** EMR (Elastic MapReduce), Redshift, Athena
- **AI & Machine Learning:** SageMaker, Comprehend, Rekognition
- **Migration & Transfer:** Migration Hub, Datasync
- **Security & Identity:** IAM (Identity and Access Management), Cognito

To hosting του server για το συγκεκριμένο project γίνεται σε ένα EC2 Instance

3.7 Επίλογος

Σε αυτό το κεφάλαιο έγινε ανάλυση των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη όλων των μερών της εφαρμογής.

Κεφάλαιο 4ο: Υλοποίηση

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει λεπτομερής περιγραφή της υλοποίησης κάθε ξεχωριστού μέρους της πτυχιακής από το Rest API μέχρι την εφαρμογή για κινητά.

4.2 Δημιουργία βάσης δεδομένων

Αρχικά θα περιγραφεί η διαδικασία με την οποία μπορεί κανείς να δημιουργήσει ένα λογαριασμό στο MongoDB Atlas και να δημιουργήσει το δικό του Cluster και Database.

4.2.1 Δημιουργία λογαριασμού Atlas

Η δημιουργία ενός λογαριασμού για το Atlas service της MongoDB είναι πολύ εύκολη. Τα βήματα είναι τα εξής:

- Περιήγηση στο https://cloud.mongodb.com
- Στην οθόνη εγγραφής υπάρχουν τρεις επιλογές: Εγγραφή μέσω λογαριασμού Google, μέσω λογαριασμού GitHub και εγγραφή μέσω Email
- Επιλογή ενός από τους παραπάνω τρόπους εγγραφής. Και στις τρεις περιπτώσεις χρειάζεται setup 2FA.
- Αποδοχή των Terms of Service και Privacy Policy
- Ολοκλήρωση δημιουργίας λογαριασμού

4.2.2 Δημιουργία ενός organization

Στη συνέχεια χρειάζεται η δημιουργία ενός οργανισμού. Τα βήματα είναι τα εξής:

- Περιήγηση στην κατηγορία Organizations (https://cloud.mongodb.com/v2#/preferences/organizations)
- Επιλογή “Create New Organization”
- Επιλογή ονόματος για το Organization
- Επιλογή MongoDB Atlas ως Cloud Service
- Εισαγωγή μελών για τον οργανισμό. Από προεπιλογή ο δημιουργός του οργανισμού είναι και ο κάτοχος και έχει πλήρη δικαιώματα. Υπάρχει δυνατότητα πρόσκλησης παραπάνω μελών.

4.2.3 Δημιουργία Project

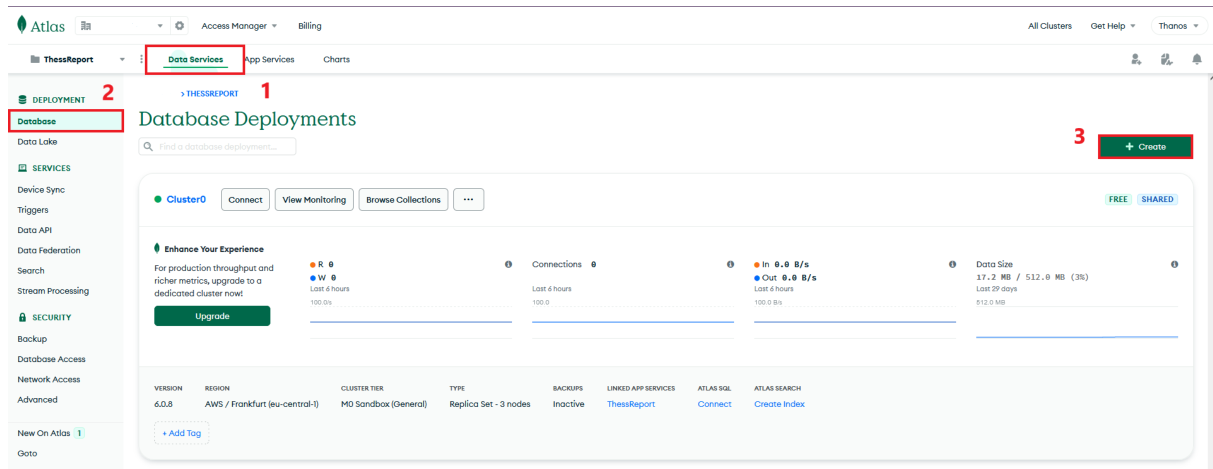
Αφού έχει δημιουργηθεί ο οργανισμός, χρειάζεται να γίνει η δημιουργία ενός project. Έχοντας την κατηγορία “Projects” επιλεγμένη, κάνουμε κλικ στο “New Project”. Επιλέγουμε ένα όνομα, τα μέλη τα οποία έχουν πρόσβαση και δημιουργούμε το project.

4.2.4 Δημιουργία Cluster

Αφού έχει δημιουργηθεί και το project σειρά έχει η δημιουργία ενός cluster στο οποίο θα γίνεται deploy η βάση. Η δημιουργία του Cluster γίνεται ως εξής:

- Επιλογή του project.
- Επιλογή του “Database” κάτω από την καρτέλα “Data Services”.
- Επιλογή “Create” (Σχήμα 4.1).

- Επιλογή τύπου Cluster. Για τη βάση που χρησιμοποιήθηκε στα πλαίσια της πτυχιακής έγινε επιλογή ενός Shared Cluster καθώς είναι η μόνη δωρεάν επιλογή. Οι υπόλοιπες ακολουθούν το pay-as-you-go μοντέλο και ενδέχεται να υπάρχουν χρεώσεις ανάλογα με τη χρήση. Σε επαγγελματικό περιβάλλον είναι πολύ πιθανόν να χρειαστεί η επιλογή ενός δυνατότερου cluster με μεγαλύτερη υπολογιστική ισχύ και χωρητικότητα.
- Επιλογή Cloud Provider ανάμεσα σε AWS, Google Cloud και Azure, και region. Για τις ανάγκες της πτυχιακής επιλέχθηκαν AWS ως provider και Frankfurt(eu-central-1) ως region.
- Ολοκλήρωση δημιουργίας cluster και deployment.



Σχήμα 4.1: Δημιουργία νέου Cluster

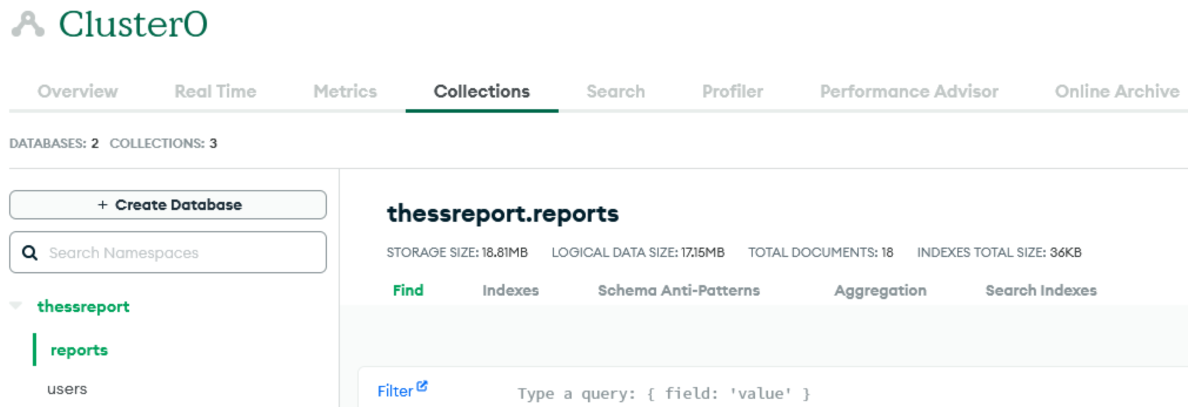
Μπορεί να οριστεί από ποιες IPs μπορεί να υπάρξει πρόσβαση στη βάση. Αυτό μπορεί να γίνει από το “Network Access” στην κατηγορία “Security”. Εκεί, στην “IP Access List”, μπορούν να προστεθούν IPs που μπορούν να έχουν πρόσβαση. Για ευκολία μπορεί να προστεθεί αυτόματα η τρέχουσα IP.

4.2.5 Δημιουργία Database και Collections

Έχοντας επιλεγμένο το Cluster και πηγαίνοντας στα Collections μπορεί κανείς να δημιουργήσει μία νέα βάση.

- Επιλογή “Create Database” στα αριστερά της σελίδας Collections
- Επιλογή ονόματος της βάσης και ονόματος του αρχικού Collection

Μετέπειτα μπορούν να προστεθούν επιπλέων collections επιλέγοντας “Create Collection” στα δεξιά της σελίδας. Στο (Σχήμα 4.2) φαίνεται η βάση και τα collections που δημιουργήθηκαν για το συγκεκριμένο project.



Σχήμα 4.2: Λίστα με τη βάση και τα collections της

Τώρα πλέον το setup είναι έτοιμο και η βάση είναι έτοιμη να δεχθεί δεδομένα.

4.3 Ανάπτυξη του Rest API

Σε αυτό το κεφάλαιο θα γίνει πλήρης περιγραφή της δημιουργίας του API που θα επικοινωνεί με τη βάση για να αποθηκεύει και να διαβάσει δεδομένα και με το οποίο θα επικοινωνεί η εφαρμογή για κινητά, αλλά και η εφαρμογή διαχειριστή.

4.3.1 Δημιουργία Spring Boot Project

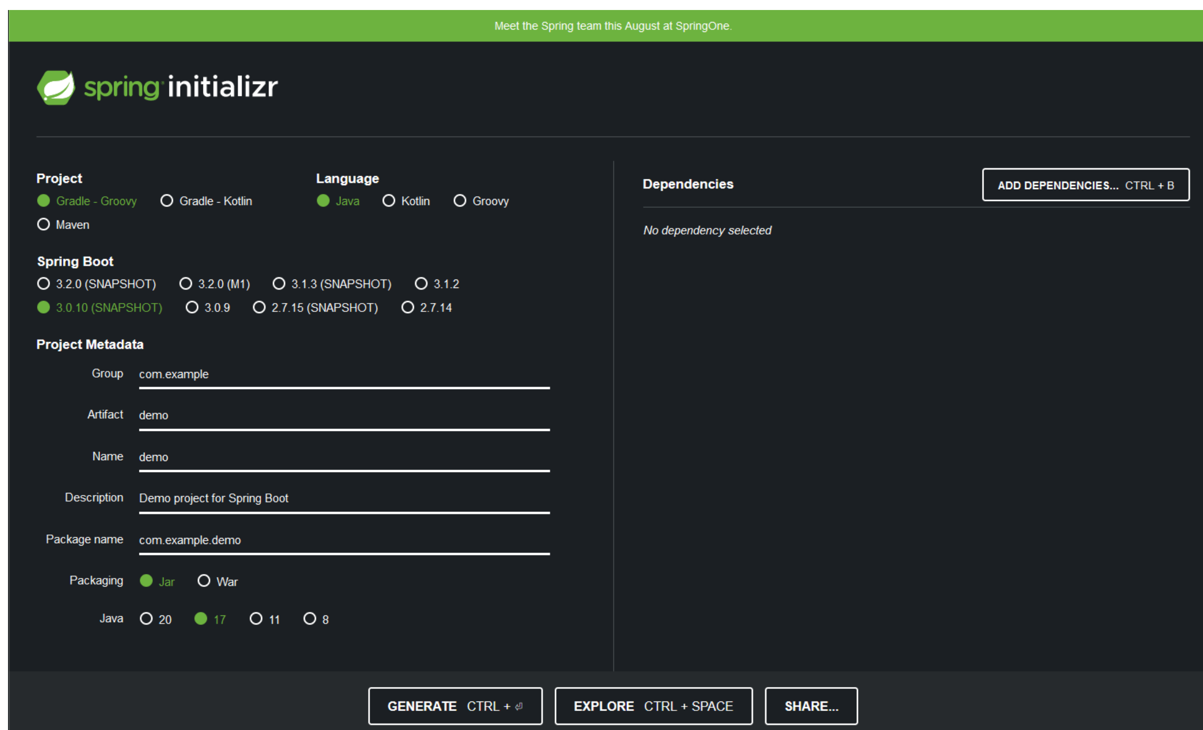
Η δημιουργία ενός Spring Boot Project είναι πάρα πολύ απλή και γρήγορη χάρη στο Spring Initializr (Σχήμα 4.3). Το Spring Initializr είναι μία σελίδα μέσω της οποίας παράγεται ένα project σε zip αρχείο με ελάχιστα κλικ. Χρειάζεται να επιλεχθεί το είδος του project (Gradle - Groovy, Gradle - Kotlin, Maven), η γλώσσα (Java, Kotlin, Groovy), η έκδοση του Spring Boot και να οριστούν τα metadata του project (Group, Name, Description, Package name, αν το Packaging θα γίνεται σε Jar ή War και η έκδοση της Java).

Το Spring Boot Project ονομάστηκε thessreport. Το setup του project έγινε ως εξής:

- Για Project επιλέχθηκε Gradle - Groovy
- Ως γλώσσα επιλέχθηκε η Java
- Ως Spring Boot version επιλέχθηκε η 3.0.0
- Για packaging επιλέχθηκε το Jar
- Ως Java version επιλέχθηκε η Java 17 (χρησιμοποιήθηκε το Open JDK 17)

Επιπλέον, μέσω του Spring Initializr μπορούν να προστεθούν εξ αρχής όσα dependencies χρειάζονται. Στο συγκεκριμένο project προστέθηκαν τα:

- Spring Security
- Spring Web
- Spring Data MongoDB
- Lombok



Σχήμα 4.3: Το Spring Initializr

4.3.2 Δομή του Project

Στο Project υπάρχει μία συγκεκριμένη δομή, ώστε όλα τα αρχεία να ομαδοποιούνται ανάλογα με τη λειτουργικότητά τους (Σχήμα 4.4). Τα αρχεία έχουν ομαδοποιηθεί ως εξής:

- **Controllers**

  Σε αυτόν τον φάκελο περιλαμβάνονται οι κλάσεις των Controllers. Στους Controllers γίνεται η διαχείριση των εισερχόμενων αιτημάτων και ανάλογα εκτελείται κάποια λογική.

- **Services**

  Σε αυτόν τον φάκελο περιλαμβάνονται οι κλάσεις των Services. Στα Services ζει ουσιαστικά το business logic και εκεί ορίζονται οι μέθοδοι που εκτελούνται από τους Controllers για να διαχειριστούν τα αιτήματα

- **Doc**

  Σε αυτόν τον φάκελο περιλαμβάνονται οι κλάσεις μέσα στις οποίες περιγράφεται η δομή των Documents που αποθηκεύονται στη βάση.

- **Dto**

  Σε αυτόν τον φάκελο περιλαμβάνονται οι κλάσεις που περιγράφουν τα DTOs, τη μορφή των αντικειμένων που επιστρέφονται στην απάντηση κάποιου αιτήματος.

- **Repositories**

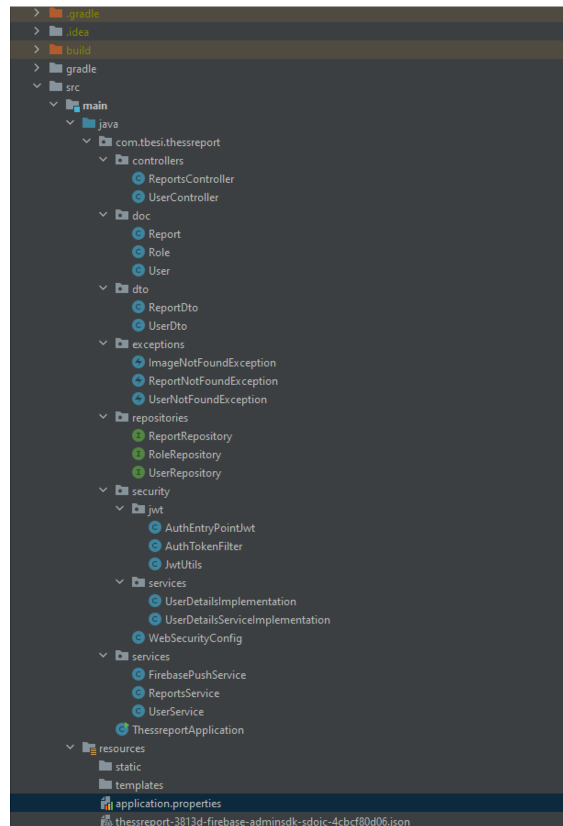
  Σε αυτόν τον φάκελο περιλαμβάνονται οι διεπαφές (interfaces) των Repositories τα οποία κάνουν επεκτείνουν το MongoRepository το οποίο προσφέρει όλες τις απαραίτητες CRUD (Create-Read-Update-Delete) μεθόδους για τη βάση δεδομένων.

- **Exceptions**

  Σε αυτόν τον φάκελο περιλαμβάνονται όλες οι κλάσεις για τα Custom Exceptions.

- **Security**

  Σε αυτόν τον φάκελο περιλαμβάνονται όλες οι κλάσεις που έχουν να κάνουν με το Security. Χωρίζεται σε δύο υπο-ομάδες, το jwt, στο οποίο υπάρχουν οι κλάσεις που σχετίζονται με τη διαχείριση των JWTs, και το services, όπου υπάρχουν οι κλάσεις στις οποίες ζει η λογική της αυθεντικοποίησης των χρηστών



Σχήμα 4.4: Η δομή του Spring Boot project

4.3.3 Σύνδεση με MongoDB

Η σύνδεση της MongoDB βάσης με το Spring Boot Project είναι πάρα πολύ απλή. Χρειάζεται απλά να προστεθεί το **spring.data.mongodb.uri** property στο application.properties αρχείο μέσα στον φάκελο resources με το uri. Το connection string για αυτό το property μπορεί να βρεθεί πηγαίνοντας στο Mongo DB UI και επιλέγοντας "Connect" στο Cluster. Εκεί εμφανίζεται το template για το connection string. Στο συγκεκριμένο project προστέθηκε και το property **spring.data.mongodb.auto-index-creation=true** ώστε να δημιουργούνται μοναδικά indexes για τα documents που έχουν το annotation **@Indexed(unique=true)**.

4.3.4 Διαχείριση των χρηστών

Η κλάση User

Αρχικά δημιουργήθηκε η κλάση User μέσα στο **com.tbesi.thessreport.doc** πακέτο. Σε αυτήν την κλάση περιγράφεται ο χρήστης. Η κλάση είναι annotated με τα εξής annotations:

- **@Document("users")**, το οποίο παρέχεται από τη MongoDB και αναγνωρίζει ένα αντικείμενο ως ικανό να αποθηκευτεί στην MongoDB. Η τιμή μέσα στις παρενθέσεις είναι το collection στο οποίο θα αποθηκευτεί το document που αντιπροσωπεύει το αντικείμενο.
- **@Data**, το οποίο παρέχεται από τη Lombok βιβλιοθήκη και δημιουργεί getter και setter μεθόδους για όλα τα πεδία της κλάσης, μία toString() μέθοδο και υλοποιήσεις για τις hashCode() και equals() μεθόδους.
- **@NoArgsConstructor**, το οποίο παράγει έναν κενό δομητή

Τα πεδία της κλάσης είναι τα:

- **id**, το id του χρήστη. Είναι annotated με το @Id annotation που κάνει το συγκεκριμένο πεδίο τον identifier του αντικειμένου
- **firstName**
- **lastName**
- **email**, το οποίο είναι annotated με το @Indexed(unique=true) annotation με τη βοήθεια του οποίου απορρίπτονται τα documents με ίδιο email
- **password**
- **phoneNumber**
- **roles**, στην περίπτωση που υπάρχουν χρήστες με διαφορετικούς ρόλους, για παράδειγμα Admin
- **firebaseRegistrationToken**, το token που παράγεται από τη mobile εφαρμογή όταν γίνεται το registration στο firebase
- **deviceId,** το id της συσκευής

Στην κλάση υπάρχει ένας δομητής που δημιουργεί ένα User αντικείμενο και δέχεται τα πεδία firstName, lastName, email, password, phoneNumber καθώς κατά την εγγραφή αυτά είναι τα πεδία που παρέχει ο χρήστης.

Η κλάση UserDto

Η κλάση UserDto πακέτο και περιγράφει το αντικείμενο το οποίο θα επιστρέφεται στην απάντηση ενός αιτήματος. Όταν, για παράδειγμα, κάποιος χρήστης κάνει σύνδεση δε χρειάζεται να επιστραφούν όλα τα πεδία του User αντικειμένου. Οπότε αυτή η κλάση έχει μόνο τα πεδία id, firstName, lastName, email και phoneNumber, ακριβώς δηλαδή τα πεδία που χρειάζεται πρακτικά ο χρήστης ώστε να μπορεί και να τα αλλάζει.

Η κλάση αυτή περιλαμβάνει και τη μέθοδο getUserDto() που δέχεται ως παράμετρο ένα αντικείμενο τύπου User και δημιουργεί ένα αντικείμενο τύπου UserDto με τα αντίστοιχα πεδία του αντικειμένου User.

Η Διεπαφή UserRepository

Η διεπαφή UserRepository (Σχήμα 4.5) είναι υπεύθυνη για τις CRUD λειτουργίες που σχετίζονται με τα User αντικείμενα. Σε αυτήν τη διεπαφή υπάρχουν όλες οι μέθοδοι για custom queries καθώς όλες οι βασικές CRUD λειτουργίες, όπως insert(), delete(), findAll(), findAllById(), saveAll(), save(), delete(), deleteById(), deleteAllById() παρέχονται απευθείας από το MongoRepository και από τα ListCrudRepository και CrudRepository τα οποία κάνει extend.

Στη συγκεκριμένη περίπτωση το μόνο που χρειάστηκε ήταν η μέθοδος findUserByEmail(), η οποία δέχεται μία παράμετρο email και κάνει αναζήτηση στο users collection για να βρει ένα αντικείμενο με το συγκεκριμένο email. Η μέθοδος επιστρέφει ένα Optional<User> αντικείμενο, που σημαίνει ότι

μπορεί να μην υπάρχει επιστρεφόμενο αντικείμενο, κάτι που ελέγχεται μετέπειτα με τη μέθοδο isPresent(). Για να αναγνωριστεί η μέθοδος σαν query για τη βάση χρειάζεται το @Query annotation. Στο παρακάτω σχήμα φαίνεται η διεπαφή και το query. Το email είναι το πεδίο βάσει του οποίου πρέπει να γίνει το query και το ?0 σημαίνει ότι πρέπει να ταιριάζει με την πρώτη παράμετρο της μεθόδου.

```
package com.tbesi.thessreport.repositories;

import com.tbesi.thessreport.doc.User;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;

import java.util.Optional;

public interface UserRepository extends MongoRepository<User, String> {

    @Query("{email:'?0'}")
    Optional<User> findUserByEmail(String email);
}
```

Σχήμα 4.5: Η διεπαφή UserRepository

Η κλάση UserNotFoundException

Η κλάση UserNotFoundException (Σχήμα 4.6) κάνει extend την ResponseStatusException. Όταν δημιουργείται το exception καλείται η super, δηλαδή ο δομητής της ResponseStatusException με παραμέτρους το status code, που είναι 404 και το μήνυμα που είναι “User Not Found”. Αυτό το exception επιστρέφεται ως απάντηση σε αιτήματα που έχουν να κάνουν με συγκεκριμένο χρήστη, αν αυτός ο χρήστης δε βρεθεί.

```
package com.tbesi.thessreport.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.web.server.ResponseStatusException;

public class UserNotFoundException extends ResponseStatusException {

    private static final String MESSAGE = "User not found";

    public UserNotFoundException() {
        super(HttpStatus.NOT_FOUND, MESSAGE);
    }
}
```

Σχήμα 4.6: Η κλάση UserNotFoundException

Η κλάση UserService

Σε αυτήν την κλάση υπάρχει η κύρια λογική για τη διαχείριση των αιτημάτων, υπάρχουν μέθοδοι, για παράδειγμα, για τη σύνδεση και την εγγραφή του χρήστη. Η κλάση είναι annotated με το @Service annotation που σημαίνει ότι αναγνωρίζεται ως κλάση που εκτελεί κάποια λογική ή διαχειρίζεται δεδομένα. Στην αρχή της κλάσης γίνονται inject τα εξής dependencies:

- **UserRepository userRepository**, για την εκτέλεση CRUD διεργασιών στο users collection
- **ReportRepository reportRepository**, για την εκτέλεση CRUD διεργασιών στο reports collection
- **AuthenticationManager authenticationManager**, για την αυθεντικοποίηση των χρηστών
- **JwtUtils jwtUtils**, για τη δημιουργία JWTs (Json Web Tokens) μετά την αυθεντικοποίηση των χρηστών
- **PasswordEncoder passwordEncoder**, για το encoding των κωδικών των χρηστών ώστε να μην αποθηκεύονται στη βάση ως απλό κείμενο

Το injection γίνεται με το @Autowired annotation. Με αυτό το annotation πρακτικά το Spring container καταλαβαίνει ότι πρέπει να δημιουργήσει ένα instance του αντίστοιχου dependency κατά τη δημιουργία του αντικειμένου, χωρίς να χρειάζονται δομητές, getters και setters [28][29]. Όταν δηλαδή αρχικοποιηθεί ένα UserService αντικείμενο, τότε θα δημιουργηθούν έτοιμα προς χρήση instances για τα παραπάνω dependencies.

Σε αυτήν την κλάση υπάρχουν οι εξής μέθοδοι:

- **login:** Η μέθοδος δέχεται ως παραμέτρος ένα email String και ένα password String και επιστρέφει ένα αντικείμενο τύπου ResponseEntity. Κάνει αυθεντικοποίηση του χρήστη μέσω του authenticationManager (η διαδικασία θα περιγραφεί σε επόμενη υποενότητα) και επιστρέφει ένα ResponseEntity αντικείμενο με ένα cookie header και ένα UserDto αντικείμενο στο σώμα.
- **register:** Δέχεται τις παραμέτρους firstName, lastName, email, password και phoneNumber. Δημιουργεί ένα αντικείμενο τύπου User με αυτές τις τιμές και σώσει το αντικείμενο στη βάση μέσω της save μεθόδου του userRepository. Η save μέθοδος, όπως αναφέρθηκε παραπάνω είναι μέθοδος που κληρονομείται από τη διεπαφή CrudRepository.
- **updateUserInfo:** Δέχεται τις παραμέτρους userId, firstName, lastName, email, password, phoneNumber. Αρχικά, γίνεται αναζήτηση του user βάσει του userId μέσω της findById μεθόδου του userRepository. Αν δε βρεθεί χρήστης με αυτό το id τότε γίνεται throw μία UserNotFoundException. Στη συνέχεια καλείται η αντίστοιχη setter μέθοδος για κάθε πεδίο που χρειάζεται να αλλάξει. Στην περίπτωση του κωδικού το πεδίο γίνεται encode με τον PasswordEncoder. Τέλος σώζεται το ανανεωμένο αντικείμενο τη βάση και επιστρέφεται το αντίστοιχο UserDto στην απάντηση του αιτήματος
- **getReportsByUser:** Δέχεται μία παράμετρο userId. Κάνει αναζήτηση του user βάσει του userId και αν τον βρει τότε καλείται η getReportsByUser() μέθοδος του reportsRepository και επιστρέφονται όλες οι αναφορές αν βρεθούν.
- **deleteUser:** Δέχεται μία παράμετρο userId. Κάνει αναζήτηση του user βάσει αυτού και, αν βρεθεί ο χρήστης, καλείται η deleteById() μέθοδος του userRepository.
- **updateFirebaseInfo:** Δέχεται τις παραμέτρους userId, token, deviceId. Αν βρεθεί ο χρήστης τότε γίνονται set τα πεδία firebaseRegistrationToken και deviceId του User αντικειμένου. Τέλος σώζεται στη βάση το ανανεωμένο User αντικείμενο.

- **deleteFirebaseInfo:** Αντίστοιχα με την παραπάνω μέθοδο αν τα registrationToken και deviceId έχουν τιμές τότε γίνονται null.

Αυτές είναι οι λειτουργίες που χρειάζονται ώστε να γίνεται η διαχείριση των χρηστών. Στην επόμενη υποκατηγορία θα γίνει περιγραφή του πότε χρησιμοποιούνται οι παραπάνω λειτουργίες.

Η κλάση UserController

Η κλάση UserController διαχειρίζεται όλα τα αιτήματα που έχουν να κάνουν με τον χρήστη. Σε αυτήν την κλάση προστέθηκαν τα @RestController και @RequestMapping annotations. Το @RestController είναι ένα annotation ευκολίας που το ίδιο είναι annotated με τα @Controller και @RequestBody annotations. Έτσι δε χρειάζεται να προστίθεται το @RequestBody annotation σε κάθε μία από τις μεθόδους που διαχειρίζονται τα αιτήματα.

Στο @RequestMapping έχει δοθεί η τιμή “/api/user” που σημαίνει ότι όλα τα endpoints του API θα είναι της μορφής “/api/user/*”.

Επίσης έχουν γίνει inject (με το @Autowired annotation) τα UserService και PasswordEncoder.

Για το κάθε endpoint υπάρχει η αντίστοιχη μέθοδος που διαχειρίζεται τα αιτήματα προς αυτό. Η κάθε μέθοδος γίνεται annotate με το κατάλληλο @RequestMapping ανάλογα με τη μέθοδο αιτήματος (request method). Για την POST μέθοδο υπάρχει το @PostMapping, το οποίο είναι συντομογραφία του “@RequestMapping(method = RequestMethod.POST”, για την GET το @GetMapping, για την PATCH το @PatchMapping και για την DELETE το @DeleteMapping. Όλες οι μέθοδοι σε αυτήν την κλάση εκτός από την login() επιστρέφουν ResponseEntity αντικείμενα. Η διαφορά με την login είναι ότι εκτελεί την αντίστοιχη login() του UserService η οποία επιστρέφει από εκείνο το σημείο ResponseEntity. Οι παράμετροι των μεθόδων που διαχειρίζονται τα αιτήματα είναι annotated με το @RequestParam annotation που σημαίνει ότι η κάθε παράμετρος αντιστοιχίζεται σε μία παράμετρο ενός αιτήματος.

Οι μέθοδοι είναι οι εξής:

- **login:** POST, “/login”, δέχεται τις παραμέτρους email και password και εκτελεί την login() μέθοδο της UserService
- **register:** POST, “/register”, δέχεται τις παραμέτρους firstName, lastName, email, password, phoneNumber και καλεί τη register() μέθοδο της UserService. Επιστρέφει ένα κενό ResponseEntity με OK status.
- **update:** PATCH, “/update”, δέχεται παραμέτρους userId, firstName, lastName, email, password, phoneNumber. Εκτός από το userId όλες οι υπόλοιπες παράμετροι είναι προαιρετικές. Καλείται η updateUserInfo() της UserService και το επιστρεφόμενο UserDto επιστρέφεται στην απάντηση.
- **getReportsByUser:** GET, “/reports”, δέχεται ως παράμετρο το userId. Καλεί την getReportsByUser() της UserService και επιστρέφει στην απάντηση μία λίστα αντικειμένων τύπου ReportDto.
- **deleteUser:** DELETE, “/delete”, δέχεται ως παράμετρο το userId, καλεί την deleteUser() της UserService και επιστρέφει μια κενή απάντηση με OK status.
- **updatePushInfo:** PATCH, “/firebaseInfo”, δέχεται τις παραμέτρους userId, token, deviceId. Καλεί την updateFirebaseInfo() της UserService και επιστρέφει μία κενή απάντηση με OK status.

- **deletePushInfo:** DELETE, "/firebaseInfo", δέχεται ως παράμετρο το userId, καλεί την deleteFirebaseInfo() της UserService και επιστρέφει μία κενή απάντηση με OK status.

Για να γίνει λίγο πιο κατανοητή η σύνδεση όλων των παραπάνω, έστω κάποιος θέλει να δει τις αναφορές ενός χρήστη. Θα γίνει ένα αίτημα στον server στο endpoint "/api/user/reports" (το "/api/user" είναι το @RequestMapping της UserController κλάσης και το "/reports" είναι το @GetMapping της getReportsByUser μεθόδου). Έτσι θα κληθεί η μέθοδος που διαχειρίζεται αυτό το endpoint, η getReportsByUser(). Η getReportsByUser() θα καλέσει με τη σειρά της την getReportsByUser() της κλάσεις UserService. Η getReportsByUser() της UserService θα κάνει ένα query στη βάση αρχικά για να βρει τον χρήστη, μέσω της findById() του UserRepository και μετά για να βρει τις αναφορές του χρήστη μέσω της findReportsByUser() του ReportsRepository. Έτσι συνεργάζονται όλες οι κλάσεις μαζί. Αυτή είναι η βασική λογική με την οποία διαχειρίζεται όλα τα αιτήματα ο server.

4.3.5 Διαχείριση των αναφορών

Σε αυτό το υποκεφάλαιο θα γίνει μία ανάλυση των κλάσεων που δημιουργήθηκαν για τη διαχείριση των CRUD διεργασιών για τις αναφορές, των αιτημάτων που είναι σχετικά με τις αναφορές και τη λογική που εκτελείται έπειτα.

Η κλάση Report

Η κλάση Report περιγράφει τα αντικείμενα των αναφορών. Όπως και η κλάση User, έχει τα @Document, @Data, @NoArgsConstructor annotations. Συγκεκριμένα στην κλάση Report το @Document annotation είναι γραμμένο ως @Document("reports") ώστε τα αντικείμενα αυτής της κλάσης να αντιστοιχίζονται με documents του reports collection. Τα πεδία της κλάσης είναι:

- **id**, που είναι ο identifier του αντικειμένου, annotated με το @Id annotation
- **user,** που είναι ένα αντικείμενο τύπου User. Είναι annotated με το @DocumentReference annotation της spring data mongodb βιβλιοθήκης που αναγνωρίζει το συγκεκριμένο πεδίο ως αναφορά σε ένα άλλο document, σε αυτήν την περίπτωση σε ένα document της users collection. Από προεπιλογή, ως reference χρησιμοποιείται το id του document.
- **date,** ένα αντικείμενο τύπου Date για να αποθηκεύει την ημερομηνία υποβολής της αναφοράς
- **category,** για την κατηγορία της αναφοράς. Στη συγκεκριμένη περίπτωση είναι ένα απλό string, αλλά θα μπορούσε να είναι και enum
- **description,** πεδίο για την περιγραφή που παρέχει ο χρήστης
- **address,** η διεύθυνση της αναφοράς
- **lat,** το γεωγραφικό πλάτος της διεύθυνσης
- **lng,** το γεωγραφικό μήκος της διεύθυνσης
- **status,** η κατάσταση της αναφοράς
- **upvotes,** ο αριθμός των upvotes της αναφοράς
- **image,** ένα πίνακας bytes για την αποθήκευση της εικόνας

Στην κλάση πέρα από τον δομητή υπάρχουν και δύο μέθοδοι, η incrementUpvotes() και η decrementUpvotes(), οι οποίες χρησιμοποιούνται για να αυξήσουν και να μειώσουν αντίστοιχα τα upvotes μιας αναφοράς κατά 1.

Η κλάση ReportDto

Η κλάση ReportDto, παρομοίως με την κλάση UserDto, έχει τα αντίστοιχα πεδία με την κλάση Report με κάποιες μικρές διαφορές ώστε να επιστρέφονται οι πληροφορίες με ένα πιο χρήσιμο τρόπο. Η διαφορά που παρουσιάζει η ReportDto από την Report είναι το γεγονός ότι δεν επιστρέφει πεδίο User για τον χρήστη, αλλά εξάγει το όνομα και το επίθετο από το User αντικείμενο και κάνει set τα ownerFirstName και ownerLastName πεδία. Αυτό γίνεται στην getReportDto() μέθοδο (Σχήμα 4.7) η οποία μετατρέπει ένα αντικείμενο τύπου Report σε ένα αντικείμενο τύπου ReportDto.

```
public static ReportDto getReportDto(Report report) {

    ReportDto dto = new ReportDto();
    dto.setId(report.getId());
    if (report.getUser() != null) {
        dto.ownerFirstName = report.getUser().getFirstName();
        dto.ownerLastName = report.getUser().getLastName();
    }
    dto.date = report.getDate();
    dto.category = report.getCategory();
    dto.description = report.getDescription();
    dto.address = report.getAddress();
    dto.lat = report.getLat();
    dto.lng = report.getLng();
    dto.status = report.getStatus();
    dto.upvotes = report.getUpvotes();

    return dto;
}
```

Σχήμα 4.7: Η μέθοδος getReportDto της κλάσης ReportDto

Η Διεπαφή ReportRepository

Η διεπαφή ReportRepository είναι παρόμοια με τη UserRepository, κάνοντας επίσης extend τη διεπαφή MongoRepository, και μπορεί να χρησιμοποιήσει όλες τις μεθόδους για CRUD διεργασίες στο reports collection της βάσης. Στη ReportRepository προστέθηκε μία μέθοδος findReportByUser() η οποία δέχεται ένα userId String σαν παράμετρο και επιστρέφει μία λίστα με αντικείμενα τύπου Report. Η μέθοδος είναι annotated με το @Query annotation και το query προσπαθεί να αντιστοιχίσει το userId με το user._id του report document με παρόμοιο τρόπο όπως στο (Σχήμα 4.5).

Custom Exceptions για τις αναφορές

Δημιουργήθηκαν δύο κλάσεις για exceptions που μπορούν να προκύψουν κατά την ανάγνωση των αναφορών από τη βάση ή τη δημιουργία νέας αναφοράς. Αυτές είναι οι ImageNotFoundException και η ReportNotFoundException. Η πρώτη μπορεί να προκύψει κατά τη δημιουργία μιας αναφοράς στην περίπτωση που ο χρήστης δεν ανεβάσει φωτογραφία, κάτι που είναι απαραίτητο. Η δεύτερη μπορεί να

προκύψει όταν γίνεται ενημέρωση μιας αναφοράς. Οι δύο αυτές κλάσεις είναι πανομοιότυπες με την UserNotFoundException (Σχήμα 3.6) με τη μόνη διαφορά να βρίσκεται στο μήνυμα που επιστρέφεται. Η ImageNotFoundException επιστρέφει το μήνυμα "Image not found" και η ReportNotFoundException επιστρέφει το μήνυμα "Report not found".

Η κλάση ReportsService

Η κλάση ReportsService είναι αυτή που περιέχει τη λογική για τη διαχείριση των αναφορών. Όπως και η UserService, είναι και αυτή annotated με το @Service annotation. Έχουν γίνει inject με το @Autowired annotation τα ReportRepository, UserRepository και FirebasePushService (μία κλάση η οποία θα περιγραφεί σε επόμενο κεφάλαιο) dependencies.

Σε αυτήν την κλάση υπάρχουν οι εξής μέθοδοι:

- **createReport,** η οποία δέχεται τις πληροφορίες της αναφοράς και δημιουργεί ένα αντικείμενο Report. Για το πεδίο date δημιουργείται ένα αντικείμενο τύπου date και καλείται η setDate(). To status εξ αρχής είναι "Pending" και τα upvotes είναι 0. Αν δεν υπάρχει image τότε γίνεται throw ένα ImageNotFoundException.
- **getImage,** η οποία δέχεται ένα reportId, καλεί τη findById() του ReportRepositoty και αν βρεθεί η αναφορά επιστρέφει την εικόνα της.
- **getAllReports,** η οποία απλά καλεί την findAll() του ReportRepository και επιστρέφει μία λίστα αντικειμένων τύπου Reports
- **editReportStatus,** η οποία δέχεται τις παραμέτρους reportId και status. Αν βρεθεί η αναφορά βάσει του reportId τότε γίνεται set το status της αναφοράς και σώζεται στη βάση. Στη συνέχεια γίνεται προσπάθεια αποστολής push notification (κάτι που θα περιγραφεί σε επόμενο υποκεφάλαιο).
- **upvoteReport,** η οποία δέχεται μία reportId παράμετρο, καλεί τη findById() του ReportRepository και αν βρεθεί η αναφορά, τότε καλείται η incrementUpvote() της κλάσης Report και σώζεται στη βάση.
- **removeUpvoteFromReport**, η οποία αντίστοιχα καλεί την decrementUpvotes() για να μειωθούν τα upvotes της αναφοράς

Η κλάση ReportsController

Η κλάση ReportsController έχει δημιουργηθεί με την ίδια λογική που δημιουργήθηκε η κλάση UserController, οπότε δε θα επαναληφθεί η επεξήγηση των annotations. Για τα request mappings και τις παραμέτρους ισχύουν τα ίδια. Το @RequestMapping αυτής της κλάσης είναι "/api/reports", οπότε όλα τα endpoints θα είναι της μορφής "/api/reports/*". Οι μέθοδοι διαχείρισης των αιτημάτων είναι οι εξής:

- **createReport,** POST, "/new", δέχεται τις παραμέτρους userId (προαιρετική), category, description, address, lat, lng, image. Καλεί τη μέθοδο createReport() της ReportsService και επιστρέφει ένα ResponseEntity με OK status και ένα αντικείμενο ReportDto στο σώμα της απάντησης.
- **getImage,** GET, "/getImage", δέχεται μια παράμετρο reportId, καλεί τη μέθοδο getImage() της ReportsService και επιστρέφει ένα ResponseEntity με OK status και την εικόνα στο σώμα της απάντησης. Στο @GetMapping προστέθηκε και το "produces=MediaType.IMAGE_PNG_VALUE" ώστε να είναι ξεκάθαρο ότι πρέπει να επιστραφεί εικόνα.

- **getAllReports,** GET, "/all", καλεί τη μέθοδο getAllReports() της ReportsService, μετατρέπει όλα τα αντικείμενα Report σε ReportDto με τη μέθοδο getReportDto() και επιστρέφει ένα ResponseEntity με OK status και μία λίστα αντικειμένων ReportDto στο σώμα της απάντησης.
- **approveReport,** PATCH, "/inprogress", δέχεται μία παράμετρο reportId, καλεί την editReportStatus() της ReportService και ορίζει το status σε "In Progress". Έπειτα μετατρέπει το Report αντικείμενο που επιστράφηκε από αυτή τη μέθοδο σε αντικείμενο ReportDto και επιστρέφει ένα ResponseEntity με OK status και το ReportDto αντικείμενο στο σώμα της απάντησης.
- Υπάρχουν άλλες τρεις πανομοιότυπες μέθοδοι για την αλλαγή της κατάστασης της αναφοράς σε "Pending", "Rejected" και "Resolved". Τα request mappings αυτών είναι "/pending", "/reject", "/resolve".
- **upvoteReport,** PATCH, "/upvote", δέχεται μία παράμετρο reportId και καλεί την upvoteReport() της ReportService. Επιστρέφει ένα κενό ResponseEntity με OK status.
- **removeUpvote,** PATCH, "/removeUpvote", ίδια με την upvoteReport(), αλλά καλεί αντίστοιχα την removeUpvoteFromReport() της ReportService.

CORS (Cross-Origin Resource Sharing)

Στη συγκεκριμένη κλάση και ειδικότερα στις μεθόδους που αλλάζουν την κατάσταση μιας αναφοράς και στη μέθοδο getAllReports() χρειάστηκε η προσθήκη του @CrossOrigin annotation και ο ορισμός του origins σε "http://localhost:3000". Αυτό χρειάστηκε καθώς υπάρχει μία πολιτική, η SOP (Same Origin Policy), η οποία αν έχει υλοποιηθεί στον περιηγητή εμποδίζει κάποια websites από μία πηγή (origin) να στείλουν αιτήματα σε κάποια άλλη πηγή. Με τη σειρά της η CORS (Cross-Origin Resource Sharing) πολιτική ορίζει το πως μπορεί να γίνει αυτή η επικοινωνία μεταξύ των δύο πηγών [30]. Έτσι στη συγκεκριμένη περίπτωση το Admin Panel (που θα περιγραφεί σε επόμενη ενότητα) χρειάζεται να κάνει κάποια αιτήματα στον server. Από τη στιγμή που το Admin Panel τρέχει στον localhost και συγκεκριμένα στην πόρτα 3000, προσθέτοντας το "http://localhost:3000" στα origins του @CrossOrigin annotation επιτρέπεται στο Admin Panel να κάνει τα συγκεκριμένα αιτήματα χωρίς κανένα εμπόδιο.

4.3.6 Spring Security

Χρησιμοποιήθηκαν κάποιες του Spring Security ώστε να επιτρέπεται στους χρήστες να συνδέονται στην εφαρμογή, αλλά και να περιορίζονται συγκεκριμένες λειτουργίες μόνο σε χρήστες που είναι συνδεδεμένοι στην εφαρμογή. Πιο συγκεκριμένα, αιτήματα στα endpoints "/api/user/update", "/api/user/delete", "/api/user/reports", "/api/reports/upvote" μπορούν να γίνουν μόνο από χρήστες που έχουν συνδεθεί. Σε άλλη περίπτωση το api απαντάει με 401 Unauthorized.

Σύνδεση χρήστη

Στη μέθοδο login() (Σχήμα 4.8) της κλάσης UserService αρχικά καλείται η μέθοδος authenticate() της AuthenticationManager. Αυτή επιστρέφει ένα αντικείμενο τύπου Authentication. Σε αυτό αποθηκεύονται πληροφορίες για την αυθεντικοποιημένη αρχή, σε αυτήν την περίπτωση τον χρήστη. Έπειτα, εξάγονται οι λεπτομέρειες του χρήστη από το Authentication αντικείμενο σε ένα αντικείμενο τύπου UserDetailsImplementation. Στη συνέχεια παράγεται ένα JWT (JSON Web Token) βάσει του αντικειμένου UserDetailsImplementation. Τέλος επιστρέφεται ένα ResponseEntity με OK status, το οποίο έχει ένα "Set-Cookie" header στο οποίο περνάει το JWT και στο σώμα επιστρέφεται ένα UserDto αντικείμενο.

```java
public ResponseEntity<?> login(String email, String password) {
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(email, password)
    );

    SecurityContextHolder.getContext().setAuthentication(authentication);

    UserDetailsImplementation userDetails = (UserDetailsImplementation) authentication.getPrincipal();

    ResponseCookie jwtCookie = jwtUtils.generateJwtCookie(userDetails);

    List<String> roles = userDetails.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority).toList();
    return ResponseEntity.ok().header(HttpHeaders.SET_COOKIE, jwtCookie.toString())
            .body(new UserDto(userDetails.getId(),
                    userDetails.getFirstName(),
                    userDetails.getLastName(),
                    userDetails.getUsername(),
                    userDetails.getPhoneNumber()));
}
```

Σχήμα 4.8: Η μέθοδος login της κλάσης UserService

Η κλάση UserDetailsImplementation

Η μέθοδος getPrincipal() της Authentication κλάσης επιστρέφει ένα αντικείμενο UserDetail. Η διεπαφή UserDetails παρέχει πολύ βασικές μεθόδους για να εξαχθούν πληροφορίες, η getAuthorities(), η getPassword() και η getUsername(). Αν χρειάζονται παραπάνω πληροφορίες για τον χρήστη χρειάζεται να δημιουργηθεί μία ξεχωριστή κλάση η οποία να υλοποιεί τη διεπαφή UserDetails. Η UserDetailsImplementation κάνει override όλες τις μεθόδους του UserDetails, προσθέτοντας επιπλέον πεδία και τις αντίστοιχες get και set μεθόδους ώστε να μπορούν να εξαχθούν όλες οι πληροφορίες του χρήστη.

Δημιουργία ενός JWT (JSON Web Token)

Στην κλάση JwtUtils υπάρχουν διάφορες μέθοδοι για τη διαχείριση των JWTs, όπως δημιουργία καινούργιων, εξαγωγή ενός JWT από το cookie ενός αιτήματος, εξαγωγή email από ένα JWT και ο έλεγχος εγκυρότητας ενός JWT. Για τη δημιουργία ενός JWT χρειάζεται να οριστούν τρία properties στο αρχείο application.properties. Αυτά είναι τα thessreport.app.jwtCookieName, thessreport.app.jwtSecret και thessreport.app.jwtExpirationMs.

Για τη δημιουργία ενός JWT καλείται η generateJwtCookie() (Σχήμα 4.9) η οποία δέχεται ένα UserDetailsImplementation αντικείμενο και επιστρέφει ένα ResponseCookie αντικείμενο. Αρχικά καλείται η μέθοδος generateTokenFromUsername() (Σχήμα 4.9) η οποία παράγει το JWT string. Στη συνέχεια παράγεται το cookie με το JWT string και επιστρέφεται.

```java
public ResponseCookie generateJwtCookie(UserDetailsImplementation userDetails) {
    String jwt = generateTokenFromUsername(userDetails.getUsername());
    ResponseCookie cookie = ResponseCookie.from(jwtCookie, jwt).path("/api").maxAge(24*60*60).httpOnly(true).build();
    return cookie;
}

public String generateTokenFromUsername(String username) {
    return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
}
```

Σχήμα 4.9: Οι μέθοδοι που χρησιμοποιούνται για την παραγωγή ενός JWT και του αντίστοιχου cookie

Security Configuration

Όπως προαναφέρθηκε, κάποιες συγκεκριμένες λειτουργίες μπορούν να εκτελεστούν μόνο από χρήστες που έχουν συνδεθεί στην εφαρμογή για κινητά. Αυτό σημαίνει ότι κάποια αιτήματα μπορούν να εκτελεστούν μόνο αν υπάρχει ένα jwt cookie.

Αρχικά ορίζεται η κλάση WebSecurityConfig και γίνεται annotate με τα @Configuration και @EnableMethodScurity annotations. Το @Configuration annotation αναγνωρίζει την κλάση ως μία κλάση η οποία περιέχει μεθόδους που είναι annotated ως @Bean, πράγμα που σημαίνει ότι το Spring Container τα αρχικοποιεί απευθείας με την εκτέλεση της εφαρμογής.

Το βασικότερο bean σε αυτήν την κλάση είναι το filterChain() (Σχήμα 4.10). Σε αυτήν την μέθοδο ορίζεται το πως γίνεται η διαχείριση των μη αυθεντικοποιημένων αιτημάτων και το ποια endpoints είναι προστατευμένα. Η διαχείριση σφάλματος γίνεται από ένα unauthorizedHandler τύπου AuthEntryPointJwt. Τα προστατευμένα endpoints ορίζονται από τους requestMatchers.

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable().exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and() HttpSecurity
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers( ...patterns: "/api/user/update").authenticated()
            .requestMatchers( ...patterns: "/api/user/delete").authenticated()
            .requestMatchers( ...patterns: "/api/user/reports").authenticated()
            .requestMatchers( ...patterns: "/api/reports/upvote").authenticated()
            .anyRequest().permitAll();
    http.authenticationProvider(authenticationProvider());
    http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Σχήμα 4.10: Η filterChain μέθοδος

Η κλάση AuthEntryPointJwt (Σχήμα 4.11) υλοποιεί τη διεπαφή AuthenticationEntryPoint [31] του Spring Security και κάνει override τη μέθοδο commence() η οποία εκτελείται κάθε φορά που κάποιος μη αυθεντικοποιημένος χρήστης προσπαθεί να αποκτήσει πρόσβαση σε πόρους που δε θα έπρεπε. Αν λοιπόν γίνει κάποιο αίτημα στο “/api/user/update” χωρίς να υπάρχει jwt cookie τότε επιστρέφεται ένα 401 Unauthorized σφάλμα.

```java
package com.tbesi.thessreport.security.jwt;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import java.io.IOException;

@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authenticationException)
        System.out.println("Unauthorized: " + authenticationException.getMessage());
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, msg: "Unauthorized");
    }
}
```

Σχήμα 4.11: Η κλάση AuthEntryPointJwt

Τέλος, υπάρχει η κλάση AuthTokenFilter. Η AuthTokenFilter κάνει extend την κλάση OncePerRequestFilter η οποία εξασφαλίζει ότι ένα φίλτρο εκτελείται μόνο μία φορά ανά αίτημα [32]. Για να οριστεί το φίλτρο χρειάζεται να γίνει override η μέθοδος doFilterInternal() (Σχήμα 4.12). Στην doFilterInternal() εξάγεται το jwt string από τα cookies του αιτήματος ελέγχεται αν είναι null και αν είναι έγκυρο. Αν είναι έγκυρο τότε ο χρήστης αυθεντικοποιείται.

```java
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException{
    try {
        String jwt = parseJwt(request);
        if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
            String email = jwtUtils.getEmailFromJwtToken(jwt);

            UserDetails userDetails = userDetailsService.loadUserByUsername(email);
            UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
            authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }
    } catch (Exception e) {
        System.out.println("Cannot set user authentication");
    }

    filterChain.doFilter(request, response);
}

private String parseJwt(HttpServletRequest request) {
    String jwt = jwtUtils.getJwtFromCookies(request);
    return jwt;
}
```

Σχήμα 4.12: Η doFilterInternal της AuthTokenFilter

Με τη βοήθεια όλων των παραπάνω επιτυγχάνεται η σύνδεση ενός χρήστη, η πρόσβασή του σε συγκεκριμένους πόρους και αποτροπή αυτής σε μη αυθεντικοποιημένους χρήστες. Περιληπτικά λοιπόν, όταν εκτελείται η εφαρμογή στον server αρχικοποιείται ένα security configuration. Όταν ο χρήστης συνδέεται με τα σωστά στοιχεία, τότε δημιουργείται ένα JWT το οποίο στέλνεται στην απάντηση του αιτήματος. Από εκεί και πέρα κάθε αίτημα περνάει από μία σειρά φίλτρων. Αν το αίτημα γίνεται σε κάποιο από τα τέσσερα endpoints που αναφέρθηκαν παραπάνω τότε γίνεται έλεγχος για το αν περιέχει ένα jwt cookie και αν αυτό είναι. Αν όλα πάνε καλά τότε το αίτημα θα ικανοποιηθεί. Όλα τα υπόλοιπα endpoints περνάνε από τα φίλτρα χωρίς κανένα πρόβλημα.

4.3.7 Αποστολή Push Notifications

Για την αποστολή των push notifications χρειάζεται αρχικά ένα project στο Firebase. Καθώς το setup του project είναι περισσότερο σχετικό με την εφαρμογή για κινητά, θα περιγραφεί στο επόμενο

κεφάλαιο. Για την Spring Boot εφαρμογή το μόνο που χρειάστηκε από το Firebase project ήταν ένα JSON αρχείο που περιέχει κάποιο configuration και το οποίο χρειάστηκε να προστεθεί στο φάκελο resources.

Στη συνέχεια χρειάστηκε να προστεθεί η firebase-admin βιβλιοθήκη στα dependencies του project.

Στην ThessReportApplication κλάση, στην οποία εκτελείται η main μεθοδος, χρειάστηκε να προστεθεί ένα FirebaseMessaging Bean (Σχήμα 4.13) για την αρχικοποίηση του FirebaseMessaging instance. Αρχικά εξάγονται τα credentials από το JSON configuration αρχείο και με τη χρήση αυτών δημιουργείται ένα FirebaseOptions αντικείμενο. Με τη χρήση του FirebaseOptions αντικειμένου δημιουργείται ένα FirebaseApp αντικείμενο. Τέλος, δημιουργείται ένα FirebaseMessaging instance με τη χρήση του FirebaseApp αντικειμένου και επιστρέφεται. Το FirebaseMessaging instance μπορεί τώρα να χρησιμοποιηθεί για να σταλούν ειδοποιήσεις.

```
public class ThessreportApplication {
    @Bean
    FirebaseMessaging firebaseMessaging() throws IOException {
        GoogleCredentials googleCredentials = GoogleCredentials.fromStream(new ClassPathResource("thessreport-3813d-firebase-a
        FirebaseOptions firebaseOptions = FirebaseOptions.builder().setCredentials(googleCredentials).build();
        FirebaseApp firebaseApp = FirebaseApp.initializeApp(firebaseOptions);
        return FirebaseMessaging.getInstance(firebaseApp);
    }
    public static void main(String[] args) {
        SpringApplication.run(ThessreportApplication.class, args);
    }
}
```

Σχήμα 4.13: Αρχικοποίηση ενός FirebaseMessaging instance στην κλάση ThessreportApplication

Η αποστολή των ειδοποιήσεων γίνεται όταν αλλάξει η κατάσταση μιας αναφοράς και εφόσον αυτή δεν έχει καταχωρηθεί ανώνυμα. Αρχικά, έχει δημιουργηθεί η κλάση FirebasePushService. Σε αυτήν την κλάση υπάρχει η sendPushNotification() μέθοδος, η οποία δέχεται δύο παραμέτρους, body και token. Δημιουργείται ένα νέο αντικείμενο τύπου Notification που παρέχεται από τη βιβλιοθήκη του Firebase και ορίζεται ο τίτλος ως "Η αναφορά σου έχει μία ενημέρωση" και το σώμα ως η τιμή της παραμέτρου body. Έπειτα δημιουργείται ένα αντικείμενο τύπου Message που παρέχεται επίσης από τη βιβλιοθήκη του Firebase, ορίζεται το token του μηνύματος ως η τιμή της token παραμέτρου και χτίζεται το μήνυμα. Τέλος καλείται η FirebaseMessaging send(μέθοδος με παράμετρο το Message αντικείμενο που δημιουργήθηκε πριν. Ο κώδικας αυτού μπορεί να βρεθεί στο (Σχήμα 4.14).

```java
package com.tbesi.thessreport.services;

import com.google.firebase.messaging.FirebaseMessaging;
import com.google.firebase.messaging.FirebaseMessagingException;
import com.google.firebase.messaging.Message;
import com.google.firebase.messaging.Notification;
import org.springframework.stereotype.Service;

@Service
public class FirebasePushService {

    private final FirebaseMessaging firebaseMessaging;

    public FirebasePushService(FirebaseMessaging firebaseMessaging) {
        this.firebaseMessaging = firebaseMessaging;
    }

    public String sendPushNotification(String body, String token) throws FirebaseMessagingException {
        Notification notification = Notification.builder().setTitle("Η αναφορά σου έχει μία ενημέρωση").setBody(body).build();

        Message message = Message.builder().setToken(token).setNotification(notification).build();
        return firebaseMessaging.send(message);
    }
}
```

Σχήμα 4.14: Η κλάση FirebasePushService

Το μόνο που χρειάζεται τώρα είναι να καλείται η sendPushNotification() μέθοδος όταν αλλάζει η κατάσταση μιας αναφοράς. Έτσι, λοιπόν, χρειάζεται ο κατάλληλος κώδικας στην editReportStatus() μέθοδο της κλάσης ReportsService. Αφού αλλάζει η κατάσταση και σώζεται τη ενημερωμένο Report αντικείμενο στη βάση γίνεται έλεγχος για το αν η αναφορά έχει γίνει από κάποιον χρήστη. Αν το πεδίο user του αντικειμένου Report δεν είναι null τότε γίνεται έλεγχος για το αν το πεδίο firebaseRegistrationToken του αντικειμένου User είναι null. Αν δεν είναι, τότε ορίζεται μια μεταβλητή translatedStatus και ανάλογα με την κατάσταση δίνεται η αντίστοιχη τιμή. Τέλος, καλείται η sendPushNotification() μέθοδος με το translatedStatus και το token σαν παραμέτρους (Σχήμα 4.15).

```java
public Report editReportStatus(String reportId, String status) {
    Report report = reportRepository.findById(reportId).orElseThrow(ReportNotFoundException::new);
    report.setStatus(status);
    reportRepository.save(report);
    User reportUser = report.getUser();
    if (reportUser != null) {
        String token = reportUser.getFirebaseRegistrationToken();
        if (token != null) {
            System.out.println(token);
            String translatedStatus;
            switch (status) {
                case "In Progress":
                    translatedStatus = "Η επίλυση του προβλήματος που αναφέρατε είναι σε εξέλιξη";
                    break;
                case "Pending":
                    translatedStatus = "Αναμένεται η επίλυση το προβλήματος";
                    break;
                case "Rejected":
                    translatedStatus = "Η αναφορά σας έχει απορριφθεί";
                    break;
                case "Resolved":
                    translatedStatus = "Το πρόβλημα που αναφέρατε έχει επιλυθεί. Ευχαριστούμε!";
                    break;
                default: translatedStatus = "Ανοίξτε την εφαρμογή για να δείτε την εξέλιξη της αναφοράς σας";
            }
            try {
                firebasePushService.sendPushNotification(translatedStatus, token);
            } catch (FirebaseMessagingException e) {
                System.out.println(e);
            }
        }
    }
    return report;
}
```

Σχήμα 4.15: Αποστολή ειδοποιήσεων όταν γίνεται αλλαγή κατάστασης μιας αναφοράς

4.3.8 Hosting του server

Για το hosting του server επιλέχθηκε το AWS καθώς χρησιμοποιείται ευρέως για server hosting. Επιπλέον, θετικό ήταν ότι το setup ήταν πολύ εύκολο και υπάρχει η δυνατότητα δωρεάν χρήσης με μερικές υποχωρήσεις. Αρχικά χρειάστηκε η δημιουργία ενός λογαριασμού. Στη συνέχεια έπρεπε να γίνει η επιλογή περιοχής. Επιλέχθηκε η Europe (Frankfurt) eu-central-1. Στην αρχική κονσόλα ή από τα services στην αριστερά πάνω γωνία υπάρχει η επιλογή EC2. Επιλέγοντας το EC2 εμφανίζεται το EC2 Dashboard. Χρειάστηκαν τρία πράγματα:

- **Δημιουργία Key Pairs.** Η δημιουργία των key pairs χρειάζεται ώστε να είναι δυνατή η πρόσβαση στο EC2 Instance μέσω ssh. Δημιουργήθηκαν δύο key pairs καθώς χρειαζόταν πρόσβαση από δύο μηχανήματα, από αυτό στο οποίο έγινε η ανάπτυξη του API και το από αυτό στο οποίο έγινε η ανάπτυξη της εφαρμογής για κινητά.
- **Δημιουργία Security Groups.** Δημιουργώντας Security Groups ορίζονται οι τρόποι σύνδεσης και οι IPs που μπορούν να έχουν πρόσβαση στο EC2 Instance. Στα πλαίσια της πτυχιακής η πρόσβαση στο EC2 Instance είναι ελεύθερη μέσω ssh στην πόρτα 22, https στην πόρτα 443 και custom tcp στην πόρτα 8080.
- **Δημιουργία ενός EC2 Instance.** Δημιουργήθηκε ένα νέο EC2 Instance με τις προεπιλογές που επιτρέπουν τη δωρεάν χρήση.

Αφού δημιουργήθηκε το EC2 Instance χρειάστηκε η εξαγωγή του Jar αρχείου της Spring Boot εφαρμογής. Αυτό ήταν πολύ εύκολο εκτελώντας απλά το bootJar Gradle task. Έπειτα έγινε σύνδεση στο EC2 Instance μέσω SFTP με τη βοήθεια του WinSCP και έγινε upload του Jar. Για να γίνει δυνατή η εκτέλεση του Jar αρχείου χρειάστηκε η εγκατάσταση της Java στο instance. Έτσι, αφού έχει γίνει όλο το setup μπορεί να γίνει σύνδεση στο instance μέσω ssh και να εκτελεστεί το Jar μέσω της εντολής “java -jar <αρχείο .jar>“ [33].

4.4 Ανάπτυξη της εφαρμογής για κινητά

Η ανάπτυξη της εφαρμογής για κινητά έγινε με τη χρήση της React Native. Το όνομα της εφαρμογής είναι “thessreport”. Σκοπός ήταν η δημιουργία μιας εφαρμογής η οποία μπορεί να τρέξει σε Android και iOS συσκευές. Η ανάπτυξη Android εφαρμογών μπορεί να γίνει από παντού. Αντίθετα, η δυνατότητα ανάπτυξης εφαρμογών για iOS είναι περιορισμένη μόνο σε υπολογιστές με MacOS λειτουργικό σύστημα καθώς είναι αναγκαία η χρήση του Xcode, ενός IDE που είναι διαθέσιμος μόνο στο συγκεκριμένο λειτουργικό. Επιπλέον, για την διάθεση της εφαρμογής στο App Store και για την εκμετάλλευση διαφόρων πόρων της χρειάζεται μία επί πληρωμή συνδρομή στην Apple. Σε αυτήν την ενότητα θα περιγραφεί η δημιουργία του project, η δομή των αρχείων, το πως έγινε η εκμετάλλευση του API που δημιουργήθηκε στην προηγούμενη ενότητα, το πως επιτυγχάνεται η πλοήγηση στην εφαρμογή και τέλος θα περιγραφούν οι οθόνες και το πως επετεύχθει η λειτουργία αποδοχής ειδοποιήσεων.

4.4.1 Δημιουργία React Native Project

Για τη δημιουργία ενός React Native project χρειάζεται αρχικά το setup του περιβάλλοντος. Αρχικά χρειάζεται η εγκατάσταση των Node, Watchman και React Native command line interface. Έπειτα χρειάζεται κάποιο setup για Android και iOS ξεχωριστά.

Για Android χρειάζεται:

- Η εγκατάσταση ενός Java Development Kit. Για την ανάπτυξη της εφαρμογής στα πλαίσια της πτυχιακής χρησιμοποιήθηκε το openjdk 11.0.17
- Η εγκατάσταση του Android Studio. Με την εγκατάσταση του Android Studio γίνεται η εγκατάσταση του πιο πρόσφατου Android SDK (με τη δυνατότητα προσθήκης περισσότερων στη συνέχεια), κάποιων Platform tools και Android Virtual Devices

Για iOS χρειάζεται:

- Η εγκατάσταση του Xcode η οποία είναι δυνατή από το Mac App Store
- Του CocoaPods που είναι ένα σύστημα διαχείρισης dependencies για iOS

Για να δημιουργηθεί ένα project εκτελείται η εντολή “npx react-native@latest init <Όνομα του project>”. Υπάρχει η δυνατότητα επιλογής της έκδοσης της React Native με την οποία θα δημιουργηθεί το project καθώς και το template. Στα πλαίσια της πτυχιακής χρησιμοποιήθηκε η React Native 0.68 και το typescript template.

Αφού δημιουργηθεί το project μπορεί να τρέξει η εφαρμογή. Για να τρέξει το Metro χρησιμοποιείται η εντολή “npm start” ή “yarn start” ανάλογα με το package manager που είναι εγκατεστημένος (από εδώ και πέρα θα θεωρηθεί δεδομένο ότι το επιλεγμένο package manager είναι το yarn). Για να τρέξει η εφαρμογή χρησιμοποιείται η εντολή “yarn android” ή “yarn ios”. Η εφαρμογή μπορεί να τρέξει σε συσκευές Android και iOS, Android emulators και iOS simulators.

Κατά τη διάρκεια της ανάπτυξης της εφαρμογής είναι πολύ πιθανό να χρειαστεί η εγκατάσταση επιπλέον βιβλιοθηκών. Αυτό γίνεται με την εντολή “yarn add <Όνομα της βιβλιοθήκης>”.

4.4.2 Δομή project

Όταν δημιουργείται το project δημιουργούνται αυτόματα κάποια αρχεία και φάκελοι προς ευκολία του προγραμματιστή. Δε θα περιγραφεί λεπτομερώς το κάθε ένα, αλλά θα γίνει αναφορά σε κάποια βασικά. Αρχικά, υπάρχει ένας φάκελος για κάθε πλατφόρμα.

Στο φάκελο android υπάρχουν τα αρχεία που σχετίζονται αποκλειστικά με το android project. Στο αρχείο build.gradle μπορεί να παραμετροποιηθεί το ελάχιστο Android SDK(minSdkVersion), το compile Android SDK(compileSdkVersion) και το Android SDK που στοχεύει η εφαρμογή (targetSdkVersion), μπορούν να προστεθούν dependencies καθώς και τα repositories από τα οποία μπορεί να κατεβάσει αυτά τα dependencies. Το αρχείο build.gradle στο φάκελο /android/app υπάρχει configuration για το build της εφαρμογής σε android. Μπορεί να οριστεί debug configuration, release configuration καθώς και configurations άλλων flavors της εφαρμογής, αν υπάρχουν.

Στο φάκελο iOS υπάρχουν αρχεία που σχετίζονται αποκλειστικά με το iOS. Σε αυτόν τον φάκελο υπάρχει το Xcode project, το Xcode Workspace, τα assets όπως launch screens, appicons και fonts. Επίσης, υπάρχουν τα αρχεία Podfile και Podfile.lock To Podfile περιγράφει τα dependencies του project [34].

Στο φάκελο node_modules στο root του project αποθηκεύονται όλες οι βιβλιοθήκες που χρησιμοποιούνται. Το αρχείο yarn.lock περιέχει μια λίστα με όλα τα dependencies του project, την έκδοση του καθενός, το url του και τα αντίστοιχα dependencies του. Αν διαγραφεί ο φάκελος node_modules μπορεί να ξαναδημιουργηθεί βάσει του yarn.lock τρέχοντας απλά την εντολή yarn. Το package.json (Σχήμα 4.16) περιέχει επίσης τη λίστα των dependencies, αλλά απλά την έκδοσή τους, χωρίς καμία άλλη πληροφορία. Επιπλέον, περιέχει τα scripts του project, στην οποία λίστα μπορούν να προστεθούν όσα scripts επιθυμεί ο προγραμματιστής για δική του ευκολία. Για παράδειγμα, το script “android” όπως φαίνεται στο (Σχήμα 4.16) θα τρέξει την εντολή “react-native run-android” και μπορεί να εκτελεστεί με την εντολή “yarn android”. Στο app.json αρχείο ορίζεται το όνομα της εφαρμογής (name και displayName). Τέλος το index.js είναι το σημείο εισαγωγής της εφαρμογής και το App.tsx είναι το αρχικό στοιχείο (element) της εφαρμογής.

```
{} package.json > {} scripts > lint
You, 2 days ago | 1 author (You)
1   {
2     "name": "thessreport",
3     "version": "0.0.1",
4     "private": true,
      ▷ Debug
5     "scripts": {
6       "android": "react-native run-android",
7       "ios": "react-native run-ios",
8       "start": "react-native start",
9       "test": "jest",
10      "lint": "eslint ."        You, 5 months ago • Initial commit …
11    },
12    "dependencies": {
13      "@babel/preset-env": "^7.20.2",
14      "@gorhom/bottom-sheet": "^4",
15      "@react-native-cookies/cookies": "react-native-cookies/cookies",
16      "@react-native-firebase/app": "^18.1.0",
17      "@react-native-firebase/messaging": "^18.1.0",
18      "@react-native-picker/picker": "^2.4.9",
19      "@react-navigation/material-top-tabs": "^6.6.2",
20      "@react-navigation/native": "^6.1.4",
21      "@react-navigation/stack": "^6.3.14",
22      "@reduxjs/toolkit": "^1.9.3",
23      "@types/react-native": "^0.71.3",
24      "@types/react-native-vector-icons": "^6.4.13",
25      "axios": "^1.3.4",
```

Σχήμα 4.16: Παράδειγμα περιεχομένων του package.json

Αφού αναφέρθηκαν κάποια από τα αρχεία που παράγονται με τη δημιουργία του project, θα γίνει μία περιγραφή της δομής των υπόλοιπων φακέλων/αρχείων που συνθέτουν την εφαρμογή. Όλοι οι φάκελοι που συνθέτουν τα κομμάτια της εφαρμογής βρίσκονται μέσα στο φάκελο src. Οι φάκελοι είναι οι εξής:

- **api,** στον οποίο βρίσκονται όλα τα TypeScript αρχεία τα οποία περιέχουν κώδικα που εκτελεί αιτήματα προς κάποιο API. Ο φάκελος περιέχει τα geocoding.ts, που κάνει αιτήματα προς την Google για να αντιστοιχίσει γεωγραφικό μήκος και πλάτος με διεύθυνση και αντίστροφα, report.ts, που στέλνει αιτήματα σχετικά με τις αναφορές στο API που δημιουργήθηκε στην προηγούμενη ενότητα, και user.ts, που στέλνει αιτήματα σχετικά με τους χρήστες στο API που δημιουργήθηκε στην προηγούμενη ενότητα.
- **components,** στον οποίο βρίσκονται components τα οποία μπορούν να επαναχρησιμοποιηθούν σε διάφορα σημεία της εφαρμογής.
- **config,** στον οποίο βρίσκονται αρχεία για configuration. Συγκεκριμένα χρειάστηκε να υπάρχει configuration για τα χρώματα τα οποία χρησιμοποιούνται από τα components της εφαρμογής.
- **hooks,** στον οποίο βρίσκονται όλοι υποφάκελοι που περιέχουν αρχεία με τον κώδικα για τα custom hooks (θα περιγραφούν σε επόμενη ενότητα).
- **navigation,** στον οποίο βρίσκονται τα αρχεία των Navigators. Υπάρχουν δύο Navigators, ο βασικός Navigator.tsx και NewReportNavigator.tsx που χρησιμοποιείται στην οθόνη δημιουργίας νέας αναφοράς.
- **screens,** στον οποίο βρίσκονται τα αρχεία για τις ξεχωριστές οθόνες της εφαρμογής.
- **store,** στον οποίο βρίσκονται αρχεία σχετικά με το redux store (θα περιγραφεί σε επόμενη ενότητα).

- **types,** στον οποίο υπάρχει ένα αρχείο index.ts στο οποίο περιγράφονται κάποιοι βασικοί τύποι αντικειμένων.

4.4.3 Ορισμοί βασικών τύπων

Οι δύο βασικοί τύποι που ορίστηκαν είναι ο User και το Report (Σχήμα 4.17). Ο User έχει τα properties id, firstName, lastName, email, phoneNumber που είναι strings και ένα property reports που είναι ένας πίνακας αντικειμένων τύπου Report και είναι προαιρετικό. Το Report έχει τα properties id, ownerFirstName, ownerLastName, date, category, description, address, lat, lng, status, upvotes. Οι τύποι είναι ουσιαστικά διεπαφές (interfaces) και γίνονται export ώστε να μπορούν να χρησιμοποιηθούν οπουδήποτε μέσα στον κώδικα, εφόσον γίνουν import.

```
src > types > TS index.ts > ...
You, 5 months ago | 1 author (You)
export interface User {
  id: string;
  firstName: string;
  lastName: string;
  email: string;
  phoneNumber: string;
  reports?: Report[];
}

You, 3 months ago | 1 author (You)
export interface Report {
  id: string;
  ownerFirstName: string;
  ownerLastName: string;
  date: string;
  category: string;
  description: string;
  address: string;
  lat: number;
  lng: number;
  status: string;
  upvotes: number;
}
```

Σχήμα 4.17: Οι τύποι User και Report

4.4.4 Ορισμός χρωμάτων της εφαρμογής

Για τα βασικά χρώματα τις εφαρμογής επιλέχθηκε μία σκοτεινή παλέτα. Υπάρχουν τέσσερα βασικά χρώματα. Το primaryBackground, που είναι το βασικό χρώμα των κεντρικών οθονών, το secondaryBackground που έχει λίγο πιο ανοιχτή απόχρωση και χρησιμοποιείται στα components που είναι πάνω στις βασικές οθόνες, το buttonsColor που χρησιμοποιείται για τα κουμπιά και είναι αρκετά πιο ανοιχτό σε απόχρωση και το primaryText που χρησιμοποιείται για όλα τα texts της εφαρμογής και είναι τελείως λευκό. Εκτός από αυτά τα βασικά χρώματα, υπάρχουν ακόμα τέσσερα, ένα για κάθε κατάσταση μιας αναφοράς ώστε να τα εικονίδια να παίρνουν το αντίστοιχο. Τα οκτώ αυτά χρώματα προστέθηκαν σε ένα colors.json αρχείο (Σχήμα 4.18) στον φάκελο config. Δημιουργήθηκε επίσης ένα αρχείο types.ts, ώστε να οριστεί ο τύπος του αντικειμένου Color. Τέλος, δημιουργήθηκε ένα αρχείο

index.ts όπου τα χρώματα περνάνε σε μία μεταβλητή. η οποία γίνεται export. Με αυτόν τον τρόπο τα χρώματα μπορούν να χρησιμοποιηθούν μέσω της μεταβλητής colors από οποιοδήποτε component της εφαρμογής χωρίς να χρειαστεί να ορίζεται ο κωδικός χρώματος σε κάθε style. Για να χρησιμοποιηθεί κάποιο χρώμα από κάποιο αρχείο το μόνο που χρειάζεται είναι να γίνει import το colors σε αυτό.

```
src > config > {} colors.json > ...
You, 3 months ago | 1 author (You)
1  {
2    "primaryBackground": "#1C1C1C",
3    "buttonsColor": "#4C565A",
4    "primaryText": "#FFFFFF",
5    "secondaryBackground": "#282828",
6    "pending": "#F3CF13",
7    "inprogress": "#F28C28",
8    "resolved": "#71B478",
9    "rejected": "#E94D4D"
10 }
```

Σχήμα 4.18: Τα χρώματα της εφαρμογής όπως έχουν οριστεί στο αρχείο colors.json

4.4.5 Εκμετάλλευση του Rest API

Για την εκμετάλλευση του Rest API δημιουργήθηκαν δύο αρχεία, το reports.ts και το user.ts. Στο reports.ts υπάρχουν μέθοδοι που κάνουν αιτήματα σε endpoints που είναι σχετικά με τις αναφορές. Πιο συγκεκριμένα υπάρχουν δύο μέθοδοι, η getAllReports που κάνει ένα αίτημα για να πάρει όλες τις αναφορές και η createNewReport που κάνει αίτημα για τη δημιουργία μιας νέας αναφοράς. Στο user.ts, υπάρχουν μέθοδοι που κάνουν αιτήματα σε endpoints που είναι σχετικά με τους χρήστες. Αυτές είναι οι login, register, getUserReports, updateFirebaseInfo, deleteFirebaseInfo.

Για την εκτέλεση των αιτημάτων χρησιμοποιήθηκε η βιβλιοθήκη Axios. "Το Axios είναι ένα πρόγραμμα πελάτης (client) το οποίο εκτελεί HTTP αιτήματα και είναι βασισμένο στα Promises" [35]. Το Promise είναι ένα JavaScript αντικείμενο το οποίο δέχεται ως παράμετρο μία μέθοδο η εκτέλεση της οποίας είναι χρονοβόρα και επιστρέφει το αποτέλεσμά της ή τον λόγω αποτυχίας της όταν είναι έτοιμη [36]. Το ίδιο κάνει και η μέθοδος fetch που είναι μέρος του πακέτου της JavaScript, όμως επιλέχθηκε το Axios λόγω των έτοιμων μεθόδων (post, get, patch, delete) και της απευθείας μετατροπής των δεδομένων της απάντησης σε μορφή JSON, σε αντίθεση με την fetch η οποία επιστρέφει τα δεδομένα όπως ακριβώς είναι [37].

Στο reports.ts αρχικά ορίζεται η σταθερά serverUrl με τιμή το url του εξυπηρετητή, ώστε να μπορεί να χρησιμοποιηθεί σε όλες τις μεθόδους. Έπειτα ακολουθούν όλες οι μέθοδοι, οι οποίες εξάγονται (δηλαδή γίνονται export) ώστε να μπορούν να χρησιμοποιηθούν από άλλα αρχεία. Οι μέθοδοι ορίζονται ως async καθώς εκτελούνται ασύγχρονα και ο τύπος του επιστρεφόμενου αντικειμένου είναι Promise. Παράδειγμα για το πως συντάσσεται μια τέτοια μέθοδος φαίνεται στο (Σχήμα 4.19). Το πρόθεμα export σημαίνει ότι αυτή η μέθοδος εξάγεται και μπορεί να χρησιμοποιηθεί από άλλα αρχεία αν εισαχθεί σε αυτά (με τη λέξη import). Το "Promise<Report[] | undefined>" σημαίνει ότι το επιστρεφόμενο αντικείμενο του Promise θα είναι είτε ένας πίνακας αντικειμένων τύπου Report ή θα είναι undefined στην περίπτωση που δεν υπάρχουν διαθέσιμες αναφορές.

```
export const getAllReports = async (): Promise<Report[] | undefined> => {
  try {
    const response: AxiosResponse<Report[]> = await axios.get(
      `${serverUrl}/api/reports/all`,
    );
    if (response) {
      return response.data;
    }
  } catch (e) {      You, 5 months ago • Many many enhancements …
    return undefined;
  }
};
```

Σχήμα 4.19: Παράδειγμα μεθόδου που κάνει αίτημα για να πάρει όλες τις αναφορές

Η μέθοδος getAllReports δε δέχεται καμία παράμετρο, κάνει ένα αίτημα στο "/api/reports/all" καλώντας την axios.get και επιστρέφει, αν υπάρχει, το αντικείμενο data της απάντησης. Αν δεν υπάρχει επιστρέφει απλά undefined. Η μέθοδος createNewReport (Σχήμα 4.20) δέχεται τις παραμέτρους userId, category, description, address, lat, lng και image, κάνει ένα αίτημα στο "/api/reports/new" καλώντας την axios.post και επιστρέφει, αν υπάρχει, το αντικείμενο data της απάντησης που είναι τύπου Report ή επιστρέφει undefined αν αποτύχει το αίτημα. Στη συγκεκριμένη περίπτωση, από τη στιγμή που χρειάστηκε να συμπεριληφθεί ένα αρχείο στο αίτημα, χρησιμοποιήθηκε το FormData. Αρχικά δημιουργήθηκε ένα FormData αντικείμενο και για κάθε μία παράμετρο έγινε append ένα ζευγάρι κελιδιού-τιμής (key-value pair). Στη συνέχεια το FormData αντικείμενο προστέθηκε στο αίτημα ως παράμετρος στη μέθοδο axios.post. Επιπλέον, προστέθηκε η κεφαλίδα "Content-Type" με τιμή "multipart/form-data".

```
export const createNewReport = async ({
  userId,
  category,
  description,
  address,
  lat,
  lng,
  image,
}: {
  userId?: string;
  category?: string;
  description?: string;
  address?: string;
  lat?: number;
  lng?: number;
  image?: Asset;
}): Promise<Report | undefined> => {
  try {
    const reportData = new FormData();
    if (userId) reportData.append('userId', userId);
    reportData.append('category', category);
    reportData.append('description', description);
    reportData.append('address', address);
    reportData.append('lat', lat);
    reportData.append('lng', lng);
    reportData.append('image', {
      uri: image?.uri,
      name: image?.fileName,
      type: image?.type,
    });
    const response: AxiosResponse<Report> = await axios.post(
      `${serverUrl}/api/reports/new`,
      reportData,
      {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      },
    );
    if (response) {
      return response.data;
    } else {
      return response;
    }
  } catch (e) {
    return undefined;
  }
};
```

Σχήμα 4.20: Η μέθοδος createNewReport()

Το αρχείο user.ts είναι δομημένο με τον ίδιο τρόπο και οι μέθοδοί του ακολουθούν την ίδια λογική. Η login δέχεται δύο παραμέτρους email και password και επιστρέφει ένα αντικείμενο τύπου User αν το αίτημα έχει απάντηση. Η register δέχεται τις παραμέτρους firstName, lastName, email, password και phoneNumber και επιστρέφει το αντικείμενο της απάντησης. Η getUserReports δέχεται μια παράμετρο userId και επιστρέφει ένα πίνακα αντικειμένων τύπου Report αν το αίτημα έχει απάντηση. Η updateFirebaseInfo δέχεται τις παραμέτρος userId, token και deviceId και επιστρέφει απλά το αντικείμενο της απάντησης, η οποία είναι κενή. Τέλος, η deleteFirebaseInfo δέχεται την παράμετρο userId και επιστρέφει επίσης απλά το αντικείμενο της απάντησης.

4.4.6 Βασικές έννοιες

Πριν γίνει η ανάλυση της λειτουργίας των οθονών καλό είναι να γίνει μία περιγραφή τριών βασικών εννοιών της React, και κατ' επέκταση της React Native, και πως χρησιμοποιήθηκαν στην εφαρμογή. Αυτές οι τρεις έννοιες είναι τα Hooks, το Context και το Redux.

Hooks

Τα hooks είναι μέθοδοι τις React και επιτρέπουν τη χρήση λειτουργιών της React από οποιοδήποτε component μέσα στην εφαρμογή. Υπάρχουν έτοιμα hooks της React, που αναγνωρίζονται εύκολα καθώς το όνομά τους ξεκινάει από "use", όμως μπορούν να δημιουργηθούν και νέα ανάλογα με τις ανάγκες της εφαρμογής [38][39]. Τα πιο συνηθισμένα hooks είναι τα useState, useEffect, useReducer, useContext, useRef, useMemo. Όταν χρειάζεται η χρήση μιας state μεταβλητής χρησιμοποιείται το useState hook. Μια μεταβλητή state είναι μία μεταβλητή η οποία έχει δύο σημαντικά χαρακτηριστικά, το πρώτο από τα οποία είναι ότι μπορεί να προκαλέσει ένα καινούριο render του component και το δεύτερο είναι ότι η τιμή της δεν αλλάζει ανάμεσα στα renders του component [40]. Για παράδειγμα, γίνεται η υπόθεση ότι υπάρχει μία οθόνη που δείχνει το όνομα ενός χρήστη, ένα πεδίο για εισαγωγή κειμένου και ένα κουμπί. Όταν κάποιος χρήστης εισάγει κάποιο κείμενο στο πεδίο και πατήσει το κουμπί η εφαρμογή θα πρέπει να δείξει το ανανεωμένο όνομα. Αν χρησιμοποιηθεί μία μεταβλητή state, όταν αλλάξει η τιμή της τότε η οθόνη θα ζωγραφιστεί από την αρχή και το όνομα θα έχει την καινούρια τιμή της μεταβλητής. Κάτι τέτοιο δε θα μπορούσε να γίνει με μία απλή μεταβλητή καθώς δεν προκαλεί νέο render της οθόνης και η τιμή της δεν παραμένει. Το useState hook δέχεται μία αρχική τιμή και επιστρέφει μία μεταβλητή και τη μέθοδο που χρησιμοποιείται για να αλλάξει την τιμή. Στο (Σχήμα 4.21) φαίνεται ένα παράδειγμα χρήσης του useState. Η αρχική τιμή είναι undefined, η μεταβλητή είναι η selectedReport και η μέθοδος για την αλλαγή της τιμής της μεταβλητής είναι η setSelectedStore.

```
const styles = createStyles();
const {status, requestPermissions} = usePermissions();
const mapRef = useRef() as React.MutableRefObject<Map>;
const [selectedReport, setSelectedReport] = useState<Report | undefined>();
const {reports} = useReports();
```

Σχήμα 4.21: Παράδειγμα χρήσης του useState hook

Το useEffect hook είναι επιτρέπει τη διαχείριση των renders ενός component, ορίζοντας μία μέθοδο η οποία εκτελείται ως αντίδραση σε αυτό το γεγονός. Το useEffect hook δέχεται ως παράμετρο μία μέθοδο, έναν πίνακα από dependencies, δηλαδή μεταβλητές από τις οποίες εξαρτάται και μία μέθοδο καθαρισμού (cleanup function) [38]. Αν το component κάνει rerender ως αποτέλεσμα αλλαγής της τιμής μιας τέτοιας μεταβλητής τότε θα εκτελεστεί η μέθοδος μέσα στην useEffect. Ο πίνακας των

dependencies μπορεί να είναι και κενός. Σε αυτήν την περίπτωση η μέθοδος μέσα στη useEffect θα εκτελεστεί μόνο την πρώτη φορά που θα γίνει render το component. Η μέθοδος καθαρισμού καλείται κάθε φορά πριν γίνει νέο render του component με τις τρέχουσες τιμές των μεταβλητών. Στο (Σχήμα 4.22) φαίνεται ένα παράδειγμα χρήσης του useEffect hook. Κάθε φορά που αλλάζει η τιμή της μεταβλητής selectedReport εκτελείται η μέθοδος μέσα στη useEffect που ουσιαστικά πηγαίνει τον χρήστη στην οθόνη με τις λεπτομέρειες της αναφοράς.

```
useEffect(() => {
  if (selectedReport) {
    bottomSheetRef.current?.snapToIndex(2);
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    navRef.navigate('ReportDetails', {report: selectedReport});
  }
}, [selectedReport]);
```

Σχήμα 4.22: Παράδειγμα χρήσης του useEffect hook

Το userReducer hook (Σχήμα 4.23) επιτρέπει μια πιο καθολική διαχείριση της κατάστασης (state), αντί για την τοπική διαχείριση με το useState hook. Αυτό είναι χρήσιμο στην περίπτωση που η τοπική διαχείριση της κατάστασης γίνεται αρκετά περίπλοκη. Έτσι οι διαφορετικές περιπτώσεις κατά την οποία μπορεί να αλλάξει η κατάσταση τοπικά μπορούν να χωριστούν σε ενέργειες και να απλοποιηθεί ο κώδικας σε τοπικό επίπεδο [41]. Για παράδειγμα, αν υπάρχει μία λίστα με αναφορές η οποία χρειάζεται να ανανεώνεται όταν γίνεται ανανέωση της οθόνης, αντί να υπάρχει μία μέθοδος τοπικά η οποία να διαχειρίζεται την κατάσταση μπορεί να υπάρχει μία ενέργεια την οποία διαχειρίζεται ένας reducer, για παράδειγμα η ενέργεια updateAllReportsData που ανανεώνει τη λίστα των αναφορών. Ο reducer είναι μία μέθοδος η οποία δέχεται δύο παραμέτρους, τη μεταβλητή κατάστασης (state) και μία ενέργεια (action). Η ενέργεια είναι ένα απλό JavaScript αντικείμενο που μπορεί να περιέχει ό,τι επιθυμεί ο προγραμματιστής αλλά καλό είναι απλά να περιέχει τις βασικές λεπτομέρειες που την περιγράφουν. Το useReducer hook δέχεται δύο παραμέτρους, την μέθοδο του reducer και τη μεταβλητή με την αρχική κατάσταση, και επιστρέφει τη νέα κατάσταση και τη μέθοδο dispatch η οποία χρησιμοποιείται για να εκτελεστεί μία ενέργεια. Βάσει όλων αυτών, όταν χρειάζεται να αλλάξει μία κατάσταση, τότε καλείται η μέθοδος dispatch με ένα αντικείμενο που προσδιορίζει την ενέργεια που πρέπει να γίνει και στη συνέχεια εκτελείται η μέθοδος του reducer με αυτήν την ενέργεια ως παράμετρο και, αφού εκτελεστεί ο κατάλληλος κώδικας, επιστρέφεται η νέα κατάσταση.

```
 5 export default function TaskApp() {
 6   const [tasks, dispatch] = useReducer(tasksReducer, initialTasks);
 7
 8   function handleAddTask(text) {
 9     dispatch({
10       type: 'added',
11       id: nextId++,
12       text: text,
13     });
14   }
15
```

Σχήμα 4.23: Παράδειγμα χρήσης του useReducer hook

"Το useRef είναι ένα hook το οποίο χρησιμοποιείται ώστε να γίνει αναφορά (reference) σε μία τιμή που δε χρησιμοποιείται για renders" [38]. Αυτό σημαίνει ότι χρησιμοποιείται απλά για να αποθηκευτούν πληροφορίες που δεν επηρεάζουν το οπτικό κομμάτι της εφαρμογής. Η πρόσβαση στην αναφορά γίνεται με την ιδιότητα current. Η πιο συνήθης χρήση είναι σε στοιχεία του DOM (Document Object Model) ώστε να μπορεί να γίνει αναφορά σε αυτά και να υπάρχει πρόσβαση σε κάποιες ιδιότητες και μεθόδους τους. Για παράδειγμα γίνεται να χρησιμοποιηθεί το useRef ώστε να κρατήσει μία αναφορά για το Map component, ώστε να δίνεται η δυνατότητα να αλλάζει η περιοχή του χάρτη (Σχήμα 4.24).

```
const ReportsMap: React.FC = () => {
  const styles = createStyles();
  const [status, requestPermissions] = usePermissions();
  const mapRef = useRef() as React.MutableRefObject<Map>;
  const [selectedReport, setSelectedReport] = useState<Report | undefined>();
  const {reports} = useReports();

  const insets = useSafeAreaInsets();

  const route = useRoute<RouteProp<{params?: {selectedReport: Report}}>>();

  const setPinColor = (status: string): string | undefined => {
    switch (status) {
      case 'Pending':
        return colors.pending;
      case 'In Progress':
        return colors.inprogress;
      case 'Resolved':
        return colors.resolved;
      case 'Rejected':
        return colors.rejected;
      default:        You, 3 months ago • App improvements
        break;
    }
  };

  const handleGrantedPermissions = () => {
    Geolocation.getCurrentPosition(
      async position => {
        mapRef.current.animateToRegion({
          latitude: position.coords.latitude,
          longitude: position.coords.longitude,
          latitudeDelta: 0.02,
          longitudeDelta: 0.02,
        });
      },
      () => undefined,
```

Σχήμα 4.24: Παράδειγμα χρήσης του useReducer hook

Το useMemo hook είναι πολύ χρήσιμο όσον αφορά τις επιδόσεις της εφαρμογής καθώς επιτρέπει την αποθήκευση τιμών που προκύπτουν από βαρείς υπολογισμούς. Έτσι δε χρειάζεται αυτοί οι υπολογισμοί να γίνονται κάθε φορά που γίνεται render ένα component αν οι μεταβλητές από τις οποίες εξαρτάται το hook δεν έχουν αλλάξει [38]. Ένα παράδειγμα ακολουθεί στο (Σχήμα 4.25). Για κάθε αναφορά (report) φαίνεται και ένα σχετικό εικονίδιο ανάλογα με την κατάστασή της. Δε χρειάζεται να υπολογίζεται σε κάθε render ποιο εικονίδιο πρέπει να φαίνεται, αν δεν αλλάξει η κατάσταση της αναφοράς.

```
const statusIcon = useMemo(() => {
  if (report.status === 'Pending')
    return <Icon name="clock" color={colors.pending} solid size={18} />;
  if (report.status === 'In Progress')
    return (
      <Icon name="ellipsis-h" color={colors.inprogress} solid size={18} />
    );
  if (report.status === 'Resolved')
    return (
      <Icon name="check-circle" color={colors.resolved} solid size={18} />
    );
  if (report.status === 'Rejected')
    return (
      <Icon name="times-circle" color={colors.rejected} solid size={18} />
    );
}, [report]);
```

Σχήμα 4.25: Παράδειγμα χρήσης του useMemo hook

Πέρα από τα hooks που έρχονται στο πακέτο της React, μπορούν να δημιουργηθούν και καινούρια ανάλογα με τις ανάγκες του καθενός ώστε να γίνει η δουλειά πολύ πιο εύκολη. Με αυτήν την αφορμή θα γίνει μια ανάλυση σε ένα από τα custom hooks που δημιουργήθηκαν για τις ανάγκες αποθήκευσης δεδομένων στο χώρο αποθήκευσης της συσκευής ώστε να παραμένουν ακόμα και αν η εφαρμογή κλείσει.

Δημιουργήθηκε λοιπόν το useStorage hook (Σχήμα 4.26). Το useStorage εκμεταλλεύεται τη βιβλιοθήκη react-native-mmkv, η οποία είναι η γρηγορότερη λύση για αποθήκευση και ανάκτηση δεδομένων από το χώρο αποθήκευσης της συσκευής [42]. Τα δεδομένα αποθηκεύονται ως ζευγάρια κλειδιού/τιμής (key/value) και οι διεργασίες αποθήκευσης και ανάκτησης γίνονται σύγχρονα που σημαίνει ότι ο χρόνος καθυστέρησης είναι πολύ μικρός.

Ο φάκελος useStorage περιέχει ένα αρχείο, το index.ts, το οποίο περιέχει όλες τις μεθόδους που θα χρειαστούν. Αρχικά δημιουργείται ένα instance της κλάσης MMKV με id "thessreport-storage" και αποθηκεύεται σε μία σταθερά με όνομα storage. Στη συνέχεια ορίζεται η useStorage η οποία μπορεί να είναι αορίστου τύπου T και επιστρέφει ένα αντικείμενο με τέσσερις μεθόδους που μπορούν να χρησιμοποιηθούν από οποιοδήποτε αρχείο χρησιμοποιήσει το hook. Η πρώτη μέθοδος είναι η setItem η οποία δέχεται μία παράμετρο key για το κλειδί και μία παράμετρο value τύπου T και καλεί τη μέθοδο set του storage, αφού μετατρέψει την τιμή της value σε string με τη μέθοδο JSON.stringify. Η δεύτερη μέθοδος είναι η getItem η οποία δέχεται μία παράμετρο key, καλεί τη μέθοδο get του storage ώστε να ανακτήσει την κατάλληλη εγγραφή, μετατρέπει την τιμή της εγγραφής σε αντικείμενο τύπου T με τη μέθοδο JSON.parse και το επιστρέφει. Αν δεν υπάρχει επιστρέφει null. Η τρίτη μέθοδος είναι η removeItem η οποία δέχεται μία παράμετρο key και καλεί τη μέθοδο delete του storage για να διαγράψει την αντίστοιχη εγγραφή. Τελευταία μέθοδος είναι η clearStoreage η οποία καλεί τη μέθοδο clearAll του storeage ώστε να διαγραφούν όλες οι εγγραφές. Με αυτές τις μεθόδους επιτυγχάνεται η αποθήκευση δεδομένων που μπορεί να χρειαστούν ακόμα και μετά από επανεκκίνηση της εφαρμογής. Τέλος, η useStoreage εξάγεται με το "export default" ώστε να μπορεί να χρησιμοποιηθεί από όποια αρχεία τη χρειάζονται.

```
src > hooks > useStorage > TS index.ts > ...
You, 5 months ago | 1 author (You)
import {MMKV} from 'react-native-mmkv';

export const storage = new MMKV({
  id: `thessreport-storage`,
});

const useStorage = <T>(): {
  setItem: (key: string, value: T) => boolean;
  getItem: (key: string) => T | null;
  removeItem: (key: string) => void;
  clearStorage: () => void;
} => {
  const setItem = (key: string, value: T) => {
    storage.set(key, JSON.stringify(value));
    return true;
  };

  const getItem = (key: string) => {
    const value = storage.getString(key);
    return value ? JSON.parse(value) : null;
  };

  const removeItem = (key: string) => storage.delete(key);

  const clearStorage = () => storage.clearAll();

  return {
    setItem,
    getItem,
    removeItem,
    clearStorage,
  };
};

export default useStorage;
```

Σχήμα 4.26: Το useStorage hook

Redux

Όπως αναφέρθηκε και παραπάνω μπορεί να χρησιμοποιηθεί το useReducer hook ώστε να γίνει μία καθολική διαχείριση κατάστασης. Αυτό εξυπηρετεί και η βιβλιοθήκη Redux. Προσφέρει έναν τρόπο ώστε να γίνει καθολική διαχείριση κατάστασης, η οποία μπορεί να χρειαστεί αν αυτή γίνει περίπλοκη ή αν χρειάζεται να χρησιμοποιηθεί σε αρκετά σημεία της εφαρμογής [43]. Αυτό επιτυγχάνεται με τη χρήση γεγονότων που λέγονται ενέργειες (actions). Η Redux προσφέρει αρκετά χρήσιμα εργαλεία. Αρχικά μπορεί να γίνει πολύ εύκολη η ρύθμιση του Redux store, το οποίο είναι υπεύθυνο για την αποθήκευση των καταστάσεων, την πρόσβαση σε αυτές και την αλλαγές σε αυτές. Επίσης, επιτρέπει την ομαδοποίηση των reducers σε έναν και την εύκολη οργάνωση των ενεργειών ώστε να είναι εύκολα κατανοητό το πότε γίνεται τι.

Στην περίπτωση της συγκεκριμένης εφαρμογής χρειάστηκε να υπάρχει πρόσβαση στις αναφορές από δύο οθόνες. Για τη χρήση της Redux χρειάστηκε η εγκατάσταση του “@reduxjs/toolkit”. Αρχικά δημιουργήθηκε ένας φάκελος store, μέσα στον οποίο δημιουργήθηκαν ένας φάκελος slices, ένα αρχείο index.ts και ένα αρχείο rootReducers.ts. Τα slices είναι ο τρόπος οργάνωσης του Redux store, ουσιαστικά είναι τα κομμάτια που το συνθέτουν [44]. Στο αρχείο reportsSlice.ts (Σχήμα 4.27) αρχικά έγινε εισαγωγή της μεθόδου createSlice του @reduxjs/toolkit και του τύπου Report. Στη συνέχεια δημιουργήθηκε μία διεπαφή ReportState που έχει δύο ιδιότητες, την allData, η οποία είναι ένας πίνακας αντικειμένων τύπου Report όπου αποθηκεύονται όλες οι αναφορές, και την userData, η οποία είναι επίσης ένας πίνακας αντικειμένων τύπου Report όπου αποθηκεύονται οι αναφορές του χρήστη που είναι συνδεδεμένος στην εφαρμογή. Έπειτα, αρχικοποιήθηκε η σταθερά initialState τύπου

ReportState με τις αρχικές τιμές των allData και userData που είναι undefined. Στη συνέχεια δημιουργήθηκε η σταθερά reportsSlice με τη χρήση της μεθόδου createSlice. Η createSlice δέχεται ένα όνομα για το slice, μία αρχική κατάσταση και ένα αντικείμενο με reducers και δημιουργεί τις κατάλληλες ενέργειες (actions). Για το reportsSlice, ως όνομα ορίστηκε το "reports", ως αρχική κατάσταση η initialState που δημιουργήθηκε παραπάνω και προστέθηκαν δύο reducers. Η πρώτη μέθοδος reducer είναι η updateAllReportsData και η δεύτερη είναι η updateUserReportsData. Η κάθε μία ανανεώνει την κατάλληλη κατάσταση, allData και userData αντίστοιχα, με την καινούρια τιμή της, το φορτίο (payload) της ενέργειας. Τέλος εξάγονται οι ενέργειες και ο reducer του reportsSlice.

```
src > store > slices > TS reportsSlice.ts > ...
You, 5 months ago | 1 author (You)
1  import {createSlice} from '@reduxjs/toolkit';
2  import {Report} from '../../types';
3
You, 5 months ago | 1 author (You)
4  interface ReportState {
5    allData?: Report[];
6    userData?: Report[];
7  }
8
9  const initialState: ReportState = {
10   allData: undefined,
11   userData: undefined,
12 };
13
14 const reportsSlice = createSlice({
15   name: 'reports',
16   initialState,
17   reducers: {
18     updateAllReportsData: (state, action) => {
19       state.allData = action.payload;
20     },
21     updateUserReportsData: (state, action) => {
22       state.userData = action.payload;
23     },
24   },
25 });
26
27 export const {
28   updateAllReportsData: updateAllReportsData,
29   updateUserReportsData: updateUserReportsData,
30 } = reportsSlice.actions;
31
32 export default reportsSlice.reducer;
```

Σχήμα 4.27: Το αρχείο reportsSlice.ts

Αφού έγινε περιγραφή του reportsSlice.ts, μπορούμε να προχωρήσουμε στην περιγραφή του rootReducer.ts αρχείου (Σχήμα 4.28). Σε αυτό το αρχείο, συνδυάζονται όλοι οι reducers σε έναν. Για να γίνει αυτό χρησιμοποιήθηκε η μέθοδος combineReducers του @reduxjs/toolkit. Ως παράμετρος χρησιμοποιήθηκε μόνο το reportsReducer. Το αποτέλεσμα αποθηκεύτηκε στη σταθερά rootReducer και έγινε εξαγωγή της ώστε να μπορεί να χρησιμοποιηθεί εκτός του συγκεκριμένου αρχείου.

```
src > store > TS rootReducer.ts > ...
      You, 5 months ago | 1 author (You)
  1   import {combineReducers} from '@reduxjs/toolkit';
  2   import reportsReducer from './slices/reportsSlice';
  3
  4   const rootReducer = combineReducers({reports: reportsReducer});
  5
  6   export default rootReducer;
  7
```

Σχήμα 4.28: Το αρχείο rootReducer.ts

Τέλος, στο index.ts (Σχήμα 4.29) του φακέλου store ενώνονται όλα αυτά ώστε να μπορεί να χρησιμοποιηθεί το store. Αρχικά καλείται η μέθοδος configureStore με reducer τον rootReducer και το αποτέλεσμά της αποθηκεύεται στη σταθερά store. Στη συνέχεια ορίζεται ο τύπος RootState ως ο τύπος επιστροφής του rootReducer και εξάγεται. Με αυτόν τον τρόπο μπορεί να ορίζεται ο τύπος της κατάστασης όταν χρειάζεται να γίνει πρόσβαση σε αυτή, ώστε αυτή να γίνεται πιο άμεσα και να περνάει τον έλεγχο τύπου. Έπειτα ορίζεται ο τύπος AppDispatch ως ο τύπος της μεθόδου dispatch του store και τέλος ορίζεται και εξάγεται η μέθοδος useAppDispatch η οποία εκτελεί τη μέθοδο useDispatch της βιβλιοθήκης react-redux με τύπο επιστροφής AppDispach. Έτσι μπορεί να χρησιμοποιηθεί η dispatch μέθοδος με τον σωστό τρόπο και τον σωστό τύπο ώστε να εκτελούνται οι ενέργειες αλλαγής της κατάστασης.

```
src > store > TS index.ts > ...
      You, 5 months ago | 1 author (You)
  1   import {configureStore} from '@reduxjs/toolkit';
  2   import {useDispatch} from 'react-redux';
  3   import reduxFlipper from 'redux-flipper';
  4   import rootReducer from './rootReducer';
  5
  6   const store = configureStore({
  7     reducer: rootReducer,
  8     middleware: getDefaultMiddleware =>
  9       __DEV__
 10         ? getDefaultMiddleware().concat(reduxFlipper())
 11         : getDefaultMiddleware(),
 12   });
 13
 14   export type RootState = ReturnType<typeof rootReducer>;
 15
 16   export type AppDispatch = typeof store.dispatch;
 17   export const useAppDispatch = () => useDispatch<AppDispatch>();
 18
 19   export default store;
```

Σχήμα 4.29: Το αρχείο index.ts του redux store

Context

Από την προηγούμενη υποενότητα των Hooks παραλήφθηκε η επεξήγηση του useContext hook ώστε να περιγραφεί ξεχωριστά μαζί με τη λογική του Context. Το useContext hook χρησιμοποιείται από τα components που χρειάζεται ώστε να έχουν πρόσβαση σε κάποιο Context [38]. Το Context

χρησιμοποιείται ώστε τα components που είναι υψηλότερα σε ιεραρχία να μπορούν να περνάνε δεδομένα στα components που είναι χαμηλότερα σε ιεραρχία χωρίς να χρειάζεται να περνάνε ως ιδιότητες (properties) [45]. Συνήθως, αν κάποιο component θέλει να περάσει κάποια δεδομένα στο component παιδί του, τότε τα περνάει ως ιδιότητα. Αυτό γίνεται αρκετά περίπλοκο όταν το δέντρο των components βαθαίνει. Έτσι, αντί ο κάθε γονέας να περνάει την ιδιότητα στο κάθε ξεχωριστό παιδί, χρησιμοποιεί το Context. Το Context μπορεί να χρησιμοποιεί επίσης τα προαναφερθέντα hooks, για παράδειγμα το useState, ώστε να μπορεί να αλλάζει όποτε χρειάζεται.

Για να είναι διαθέσιμο το Context προς χρήση στα υπόλοιπα components χρησιμοποιείται η ιδιότητα Provider του Context. Ο Provider είναι επίσης ένα component το οποίο μπορεί να έχει πολλά παιδιά. Όλα τα παιδιά του Provider μπορούν να έχουν πρόσβαση στα δεδομένα του Context. Συγκεκριμένα στην εφαρμογή που αναπτύχθηκε έχουν χρησιμοποιηθεί μερικά Contexts, ένα για την αυθεντικοποίηση και γενικότερη διαχείριση του χρήστη με όνομα AuthContext, ένα για τα δικαιώματα της εφαρμογής με όνομα PermissionsContext, ένα για την οθόνη της σύνδεσης χρήστη ώστε να μπορεί να εμφανίζεται από οποιαδήποτε οθόνη, με όνομα SignInModalContex, και ένα για context που χρησιμοποιείται μόνο από τις οθόνες της νέας αναφοράς, με όνομα NewReportContext. Αυτά χρησιμοποιούνται στην εφαρμογή ως custom hooks όπως και το useStorage.

Ανάλυση των Contexts της εφαρμογής

Προς ευκολία κατανόησης της λειτουργίας της εφαρμογής και του πως χρησιμοποιούνται τα Contexts θα γίνει μία ανάλυση τους καθενός. Έτσι θα είναι πιο εύκολη η εξήγηση της λογικής του κώδικα σε κάθε οθόνη. Τα τρία contexts χρησιμοποιούνται με την λογική των custom hooks. Δηλαδή για το κάθε ένα υπάρχει ένας ξεχωριστός φάκελος στο φάκελο hooks. Για το AuthContext υπάρχει ο useAuth, για το PermissionsContext υπάρχει ο usePermissions, για το SingInModalContext υπάρχει ο useSignInModal και για το NewReportContext υπάρχει ο useNewReport.

Στο AuthContext αρχικά ορίζεται ο τύπος του Context, η διεπαφή AuthContextType. Ο τύπος AuthContextType έχει ως ιδιότητες τις user, που είναι ένα αντικείμενο User ή είναι null, τη μέθοδο initializeAuth η οποία επιστρέφει ένα αντικείμενο τύπου UserObject (το οποίο με τη σειρά του περιέχει ένα αντικείμενο τύπου User), τη μέθοδο loginUser και τη μέθοδο logoutUser. Έπειτα, αρχικοποιείται το AuthContext με τη βοήθεια της μεθόδου createContext της React. Συνεχίζοντας, αρχικοποιείται ο AuthProvider ο οποίος επιστρέφει τον Provider του AuthContext. Μέσα στον AuthProvider εισάγονται οι setItem, getItem και removeItem από το useStorage hook. Στη συνέχεια, ορίζεται μια τοπική μεταβλητή κατάστασης, η user, μαζί με τη μέθοδό της, τη setUser, με τη βοήθεια του useState hook. Επιπλέον, ορίζεται και η dispatch που χρησιμοποιεί τη useAppDispatch του redux store. Μετά ορίζονται τρεις μέθοδοι, η initializeAuth, η loginUser και η logoutUser. Στην initializeAuth ανακτάται η εγγραφή με κλειδί “TRUser” με την getItem του useStorage και επιστρέφεται μέσα σε ένα αντικείμενο Promise. Η loginUser δέχεται μία παράμετρο τύπου UserObject. Η μεταβλητή κατάστασης user παίρνει την τιμή της ιδιότητας user του αντικειμένου UserObject και στη συνέχεια καλεί την setItem του useStorage ώστε να αποθηκευτεί το αντικείμενο σε μια εγγραφή με κλειδί “TRUser”. Μετά γίνεται ένα αίτημα ώστε να βρεθούν οι αναφορές του χρήστη και αποθηκεύονται στην τοπική μεταβλητή reports. Αν υπάρχουν αναφορές καλείται η dispatch και εκτελεί την ενέργεια updateUserReportsData ώστε να αποθηκευτούν οι αναφορές στο store. Στην logoutUser αρχικά γίνεται ένα αίτημα ώστε να διαγραφούν οι πληροφορίες του χρήστη σχετικά με τις ειδοποιήσεις, καλείται λοιπόν η deleteFirebaseInfo. Μετά η τιμή της μεταβλητής κατάστασης user γίνεται null και καλείται η removeItem του useStorage ώστε να διαγραφή η εγγραφή με κλειδί “TRUser”. Τέλος καλείται η μέθοδος clearAll του CookieManager, ώστε να μην παραμείνει

το jwt του χρήστη αποθηκευμένο με την αποσύνδεσή του. Εν τέλει, επιστρέφεται ο Provider του AuthContext. Αφού έχει δημιουργηθεί το AuthContext κάπως πρέπει να γίνει και διαθέσιμο προς χρήση. Μπορεί να χρησιμοποιηθεί ως hook. Έτσι λοιπόν στο αρχείο index.ts (Σχήμα 4.30) στο φάκελο useAuth ορίζεται η μέθοδος useAuth η οποία επιστρέφει ένα αντικείμενο AuthContextType το οποίο το παίρνει από το useContext hook της React το οποίο το context. Εισάγοντας, λοιπόν, το useAuth οπουδήποτε μέσα στην εφαρμογή υπάρχει πρόσβαση στο AuthContext.

```
src > hooks > useAuth > TS index.ts > ...
You, 5 months ago | 1 author (You)
1  import React from 'react';
2  import {AuthContext, AuthContextType} from './AuthContext';
3
4  const useAuth = (): AuthContextType => {
5    const context = React.useContext(AuthContext);
6
7    if (context === undefined) {
8      throw new Error('useAuth must be used within a AuthProvider');
9    }
10
11   return context;
12 };
13
14 export default useAuth;
15
```

Σχήμα 4.30: Το αρχείο index.ts του useAuth

Το PermissionsContext είναι υπεύθυνο για την αποθήκευση και την πρόσβαση στα δικαιώματα που έχει δώσει ο χρήστης στην εφαρμογή. Κυρίως ασχολείται με τα δικαιώματα τοποθεσίας. Αρχικά χρειάζεται η βιβλιοθήκη react-native-permissions, καθώς παρέχει τις απαραίτητες μεθόδους για έλεγχο και αιτήματα δικαιωμάτων αλλά και τους απαραίτητους τύπους. Αρχικά, χρειάζεται να γίνει ένας διαχωρισμός των δικαιωμάτων τοποθεσίας που χρειάζονται στο iOS και των δικαιωμάτων τοποθεσίας που χρειάζονται στο Android. Στο iOS χρειάζεται να ζητηθεί άδεια για πρόσβαση στην τοποθεσία όταν η εφαρμογή είναι σε χρήση αφού δε χρειάζεται πρόσβαση όσο η εφαρμογή είναι κλειστή. Στις καινούργιες εκδόσεις του Android συμβαίνει το ίδιο, με τη διαφορά ότι τα δικαιώματα χωρίζονται σε ακριβή τοποθεσία και κατά προσέγγιση τοποθεσία. Πριν ζητηθούν τα δικαιώματα τοποθεσίας χρειάζεται να οριστούν σε κάποια αρχεία ότι χρειάζονται αυτά τα δικαιώματα. Για το iOS χρειάζεται να προστεθεί στο αρχείο Info.plist μία εγγραφή με κλειδί “NSLocationWhenInUsageDescription” [46] συνοδευόμενη από την επεξήγηση που εμφανίζεται στο χρήστη όταν ζητούνται τα δικαιώματα. Για το Android χρειάζεται να προστεθούν στο αρχείο AndroidManifest.xml δύο εγγραφές, μία για την τοποθεσία κατά προσέγγιση “android.permission.ACCESS_COARSE_LOCATION” και μία για την ακριβή τοποθεσία “android.permission.ACCESS_FINE_LOCATION” [47]. Επίσης καθώς ζητείται και δικαίωμα για τη χρήση κάμερας, για το iOS χρειάζεται να προστεθεί και η εγγραφή με κλειδί “NSCameraUsageDescription”. Η δημιουργία του context ακολουθεί την παραπάνω λογική του AuthContext. Αρχικά υπάρχει το αρχείο PermissionsContext.ts. Σε αυτό ορίζεται ο τύπος του Context ως μια διεπαφή PermissionsContextType. Οι ιδιότητες του PermissionsContextType είναι μία μέθοδος requestPermissions, μία μέθοδος checkPermissions και ένα αντικείμενο status με τέσσερις ιδιότητες, preciseLocation, approximateLocation, media και camera. Στο status θα αποθηκεύονται οι τιμές των δικαιωμάτων που έχουν δοθεί. Οι τέσσερις ιδιότητες του αντικειμένου status είναι τύπου

PermissionsStatus το οποίο μπορεί να έχει τέσσερις πιθανές τιμές, unavailable δηλαδή μη διαθέσιμη λειτουργία, blocked που σημαίνει ότι ο χρήστης απέρριψε το συγκεκριμένο δικαίωμα, denied που σημαίνει ότι ακόμα δεν έχει ζητηθεί, granted που σημαίνει ότι ο χρήστης αποδέχθηκε το συγκεκριμένο δικαίωμα και limited που σημαίνει ότι η λειτουργία για την οποία ζητήθηκε το δικαίωμα είναι περιορισμένη. Στη συνέχεια δημιουργείται το context με τη μέθοδο createContext της React και δίνονται οι αρχικές τιμές στις ιδιότητες του context. Η κατάσταση όλων των ιδιοτήτων του αντικειμένου status είναι “denied” ώστε να μπορούν να ζητηθούν από τον χρήστη. Μετά αρχικοποιείται ο PermissionsProvide. Μέσα ορίζεται η μεταβλητή κατάστασης status με την αντίστοιχη μέθοδό της, setStatus ώστε να αποθηκεύονται οι τιμές των δικαιωμάτων. Έπειτα ορίζονται οι μέθοδοι requestPermisisons και checkPermisisons.

Η requestPermissions είναι async και επιστρέφει ένα Promise. Δέχεται μία παράμετρο type για τον τύπο του δικαιώματος που ζητείται. Αν ο τύπος είναι “location” τότε γίνεται έλεγχος της τιμής των preciseLocation και approximateLocation και αν είναι blocked ή unavailable επιστρέφεται μία απορριφθείσα Promise με τη μέθοδο Promise.reject(), άρα δε ζητούνται δικαιώματα. Αν περάσει από αυτόν τον έλεγχο τότε γίνεται ακόμη ένας έλεγχος για το αν η εφαρμογή τρέχει σε συσκευή iOS ή Android. Αν η τρέχουσα πλατφόρμα είναι iOS τότε ζητείται δικαίωμα χρήσης της τοποθεσίας όταν η εφαρμογή είναι ενεργή. Αυτό γίνεται με τη μέθοδο request() της βιβλιοθήκης react-native-permissions με παράμετρο “ios.permission.LOCATION_WHEN_IN_USE”. Το αποτέλεσμα της μεθόδου αποθηκεύεται στη μεταβλητή result. Αν η τιμή της result είναι “granted” ή “limited” τότε ανανεώνεται η τιμή της ιδιότητας preciseLocation με τη βοήθεια της setStatus και παίρνει την τιμή της result και επιστρέφεται μία επιλυμένη Promise με τη μέθοδο Promise.resolve(). Αν η τρέχουσα πλατφόρμα είναι Android τότε ζητούνται δύο δικαιώματα με τη χρήση της μεθόδου requestMultiple() της βιβλιοθήκης react-native-permissions. Η παράμετρος της requestMutliple είναι ένας πίνακας με strings δικαιωμάτων και στη συγκεκριμένη περίπτωση είναι android.permission.ACCESS_FINE_LOCATION και android.permission.ACCESS_COARSE_LOCATION. Το αποτέλεσμα της μεθόδου αποθηκεύεται σε μία μεταβλητή result. Αν η τιμή της result για ένα από τα δύο δικαιώματα είναι “limited” ή “granted” τότε ανανεώνεται η μεταβλητή κατάστασης status και στην ιδιότητα preciseLocation αποθηκεύεται η τιμή της android.permission.ACCESS_FINE_LOCATION της result και στην ιδιότητα approximateLocation αποθηκεύεται η τιμή της android.permission.ACCESS_COARSE_LOCATION της result. Τέλος επιστρέφεται ένα επιλυμένο Promise με τη μέθοδο Promise.resolve().

Η checkPermissions κάνει περίπου την ίδια δουλειά χωρίς όμως να επιστρέφει κάποιο Promise.

Τέλος γίνεται ο ίδιος έλεγχος που γίνεται και στην checkPermissions μέσα σε μία useEffect ώστε να αρχικοποιούνται σωστά οι καταστάσεις των δικαιωμάτων όταν ανοίγει η εφαρμογή και επιστρέφεται ο Provider του PermissionsContext. Στο index.tsx του φακέλου usePermissions επιστρέφεται το PermissionsContext με τον ίδιο τρόπο όπως στο (Σχήμα 4.29) ώστε να χρησιμοποιηθεί όπου είναι απαραίτητο.

Το NewReportContext χρησιμοποιείται από την οθόνη της νέας αναφοράς ώστε να κρατάει την κατάσταση και τις ιδιότητες της μέχρι να έρθει η ώρα να υποβληθεί. Χρησιμοποιείται πιο πολύ για ευκολία ώστε να χρειάζεται να περνάει η κάθε οθόνη την αναφορά στην επόμενη. Στο αρχείο NewReportContext.tsx (Σχήμα 4.31) στο φάκελο useNewReport αρχικά ορίζεται η διεπαφή NewReport με ιδιότητες userId, category, description, address, lat, lng, image. Στη συνέχεια ορίζεται μία διεπαφή για τον τύπο του context, η NewReportContextType με ιδιότητες newReport που είναι αντικείμενο τύπου NewReport και setNewReport που είναι μία μέθοδος που δέχεται σαν παράμετρο

ένα αντικείμενο τύπου NewReport. Έπειτα αρχικοποιείται το context με τη μέθοδο createContext της React. Μετά αρχικοποιείται ο Provider του context, ο NewReportProvider στον οποίο απλά ορίζεται μία μεταβλητή κατάστασης newReport με την αντίστοιχη μέθοδο setNewReport και επιστρέφεται. Στο index.tsx στο φάκελο useNewReport επιστρέφεται το NewReportContext με παρόμοιο τρόπο όπως στο (Σχήμα 4.30)

```
src > hooks > useNewReport > TS NewReportContext.tsx > ...

You, 3 months ago | 1 author (You)
interface NewReport {
  userId?: string;
  category?: string;
  description?: string;
  address?: string;
  lat?: number;
  lng?: number;
  image?: Asset;
}

You, 3 months ago | 1 author (You)
export interface NewReportContextType {
  newReport: NewReport | undefined;
  setNewReport: (obj: NewReport | undefined) => void;
}
      You, 3 months ago • App improvements
export const NewReportContext = React.createContext<NewReportContextType>({
  newReport: undefined,
  setNewReport: () => undefined,
});

export const NewReportProvider = ({
  children,
}: {
  children: ReactNode;
}): JSX.Element => {
  const [newReport, setNewReport] = useState<NewReport | undefined>();

  return (
    <NewReportContext.Provider value={{newReport, setNewReport}}>
      {children}
    </NewReportContext.Provider>
  );
};
```

Σχήμα 4.31: Το αρχείο NewReportContext.tsx

Το SignInModalContext χρησιμοποιείται ώστε να μπορεί να ανοίγει η οθόνη της σύνδεσης/εγγραφής από οποιοδήποτε σημείο. Το SignInModalContext.tsx είναι γραμμένο με την ίδια λογική όπως και τα υπόλοιπα contexts. Οι ιδιότητες του SignInModalContextType είναι μία μέθοδος openSignInModal(), μία μέθοδος closeSignInModal(), μία μέθοδος getPage() και μία μέθοδος setActivePage() η οποία δέχεται μία παράμετρο page τύπου string. Μέσα στον SignInModalProvider καλείται το useDisclose hook το οποίο διαχειρίζεται το άνοιγμα και το κλείσιμο κάποιον στοιχείων, όπως ενός Alert ή, στην περίπτωση της συγκεκριμένης εφαρμογής, ενός modal και αλλάζουν τα ονόματα των ιδιοτήτων isOpen, onOpen() και onClose() σε isSignInModalOpen, onSignInModalOpen() και onSignInModalClose() ώστε να είναι πιο ξεκάθαρη η χρήση τους. Στη συνέχεια ορίζεται μία μεταβλητή κατάστασης page και η αντίστοιχη μέθοδος setPage με τη χρήση του useState hook με αρχική κατάσταση ‘login’, ώστε η πρώτη οθόνη που είναι ενεργή όταν δημιουργείται το Context να είναι η οθόνη σύνδεσης. Η μέθοδος openSignInModal(), αλλάζει την κατάσταση page σε ‘login’ και καλεί την onSignInOpen() ώστε να ανοίξει το modal. Η μέθοδος closeSignInModal() αλλάζει επίσης την κατάσταση page σε ‘login’ και καλεί την onSignInClose() ώστε να κλείσει το modal. Η μέθοδος

getPage() επιστρέφει απλά την τιμή της μεταβλητής κατάστασης page. Η μέθοδος setActivePage() δέχεται μία παράμετρο και αλλάζει την μεταβλητή κατάστασης page καλώντας την setPage() αλλάζοντας την ενεργή σελίδα του modal βάσει της παραμέτρου. Τέλος επιστρέφεται ο Provider του SignInModalContext (Σχήμα 4.32), όμως μέσα του επιστρέφεται και το SignInActionSheet το οποίο έχει τις ιδιότητες isOpen, onClose, getPage και setActivePage. Το component SignInActionSheet είναι πρακτικά αυτό που ανοίγει και που περιέχει όλες τις οθόνες. Οι ιδιότητές του ελέγχουν το πότε ανοίγει, πότε κλείνει και ποια οθόνη θα είναι ενεργή κάθε στιγμή.

```
46
47      return (
48        <SignInModalContext.Provider
49          value={{
50            openSignInModal: openSignInModal,
51            closeSignInModal: closeSignInModal,
52            getPage,
53            setActivePage,
54          }}>
55          {children}
56          <SignInActionSheet
57            isOpen={isSignInOpen}
58            onClose={closeSignInModal}
59            getPage={getPage}
60            setActivePage={setActivePage}
61          />
62        </SignInModalContext.Provider>
63      );
64    };
```

Σχήμα 4.32: Η επιστροφή του SignInModalContext Provider

Στο αρχείο index.ts επιστρέφεται το context ώστε να μπορεί να χρησιμοποιηθεί από οποιοδήποτε σημείο στον κώδικα.

Πηγαίνοντας στο App.tsx αρχείο (Σχήμα 4.33), δηλαδή το αρχικό στοιχείο της εφαρμογής μέσα στο οποίο εμφανίζονται όλα τα components της, φαίνεται πως χρησιμοποιούνται οι Providers των contexts. Σε αυτό χρησιμοποιούνται μεταξύ άλλων, ο AuthProvider, ο PermissionsProvider και ο SignInModalProvider. Κατώτερα στην ιεραρχία βρίσκεται ο κεντρικός Navigator της εφαρμογής (ακολουθεί επεξήγηση στην επόμενη ενότητα), οπότε όλα τα παιδιά του θα έχουν πρόσβαση στα αντίστοιχα contexts.

```
TS App.tsx > ...
11  import 'react-native-gesture-handler';
12  import Navigator from './src/navigation/Navigator';
13  import {DefaultTheme, NavigationContainer} from '@react-navigation/native';
14  import {Provider} from 'react-redux';
15  import store from './src/store';
16  import {SignInModalProvider} from './src/hooks/useSignInModal/SignInModalContext';
17  import {NativeBaseProvider} from 'native-base';
18  import {AuthProvider} from './src/hooks/useAuth/AuthContext';
19  import {colors} from './src/config';
20  import {PermissionsProvider} from './src/hooks/usePermissions/PermissionsContext';
21
22  const App = (): JSX.Element => {
23    const navTheme = {
24      ...DefaultTheme,
25      colors: {...DefaultTheme.colors, background: colors.primaryBackground},
26    };
27
28    return (
29      <Provider store={store}>
30        <AuthProvider>
31          <NativeBaseProvider>
32            <PermissionsProvider>
33              <SignInModalProvider>
34                <NavigationContainer theme={navTheme}>
35                  <Navigator />
36                </NavigationContainer>
37              </SignInModalProvider>
38            </PermissionsProvider>
39          </NativeBaseProvider>
40        </AuthProvider>
41      </Provider>
42    );
43  };
44
45  export default App;
46
```

Σχήμα 4.33: App.tsx και η χρήση των Providers

4.4.7 Πλοήγηση στην εφαρμογή

Η πλοήγηση στην εφαρμογή επιτυγχάνεται με τη χρήση ενός Navigator. Ο Navigator είναι ένα component που παρέχεται από τη βιβλιοθήκη "@react-navigation". Συγκεκριμένα, χρησιμοποιήθηκε ένας Stack Navigator, που σημαίνει ότι οι οθόνες είναι σε μία στοίβα (stack) και εμφανίζονται η μία πάνω από την άλλη [48]. Η πρώτη οθόνη είναι και η αρχική που βλέπει ο χρήστης όταν ανοίγει την εφαρμογή και σε αυτή φαίνονται όλες οι αναφορές, καθώς και το κουμπί για τη δημιουργία μιας αναφοράς. Η δεύτερη οθόνη είναι ο χάρτης και ανοίγει όταν ο χρήστης επιλέξει μία αναφορά ή επιλέξει να δει όλες τις αναφορές. Η τρίτη οθόνη είναι η οθόνη δημιουργίας αναφοράς και στην πραγματικότητα είναι και αυτή ένας navigator. Σε αυτήν την οθόνη επιλέχθηκε ένας Top Tab Navigator, που σημαίνει ότι οι οθόνες είναι χωρισμένες σε καρτέλες και η αλλαγή οθονών γίνεται είτε επιλέγοντας την καρτέλα, είτε σέρνοντας το δάχτυλο δεξιά ή αριστερά, είτε προγραμματιστικά [48]. Η οθόνη δημιουργίας αναφοράς είναι χωρισμένη σε τρεις καρτέλες. Στην πρώτη επιλέγεται η τοποθεσία του χρήστη, στη δεύτερη εισάγονται οι λεπτομέρειες, δηλαδή η κατηγορία και η περιγραφή της αναφοράς, και στην τρίτη επιλέγεται μία φωτογραφία. Τέλος, υπάρχει ένα ξεχωριστό παράθυρο στο οποίο υπάρχουν οι οθόνες για τη σύνδεση, εγγραφή και αποσύνδεση του χρήστη.

Ο Κύριος Navigator

Η βασική πλοήγηση υλοποιείται στο αρχείο Navigator.tsx. Αρχικά, καλείται η μέθοδος createStackNavigator() της βιβλιοθήκης @react-naviation/stack η οποία δημιουργεί ένα ζευγάρι Navigator και Screen components. Αυτά είναι τα δύο βασικά συστατικά που χρησιμοποιούνται για την πλοήγηση. Στη συνέχεια δημιουργείται ο MainNavigator όπως ένα οποιοδήποτε component. Μέσα ορίζεται η μέθοδος initializeApp(). Μέσα στην initializeApp() καλείται η μέθοδος getReports() ώστε

να γίνει ένα αίτημα στο server να φέρει όλες τις αναφορές. Αν υπάρχουν τότε γίνεται dispatch η ενέργεια updateAllReportsData() με παράμετρο τις αναφορές που ήρθαν στην απάντηση. Τέλος καλείται η μέθοδος initializeAuth(). Μετά χρησιμοποιείται ένα useEffect hook (Σχήμα 4.34) που στον πίνακα εξαρτήσεων έχει τον user του useAuth hook, ώστε να γίνεται διαχείριση του render που προκαλείται από την αλλαγή του user. Μέσα σε αυτήν την useEffect αν υπάρχει ο user τότε καλείται η μέθοδος getUserReports() με παράμετρο το id του user και αν υπάρχουν αναφορές τότε γίνεται dispatch η ενέργεια updateUserReportData() με τις αναφορές του χρήστη. Στη συνέχεια καλείται η μέθοδος updateFirebaseInfo() που χρησιμοποιείται για την αποστολή ειδοποιήσεων. Στη συνέχεια χρησιμοποιείται ξανά το useEffect hook το οποία καλεί τη μέθοδο initializeApp() και στη συνέχεια εκτελείται κάποιος κώδικας για την διαχείριση ειδοποιήσεων. Αυτό εκτελείται μόνο την πρώτη φορά που εμφανίζεται ο MainNavigator.

Ο MainNavigator επιστρέφει ένα Navigator component (Σχήμα 4.35) που εμφωλεύει τρία Screen components, ένα με όνομα Home, ένα με όνομα ReportsMap και ένα με όνομα NewReportNav. Το πρώτο Screen χρησιμοποιεί το component Home, το δεύτερο χρησιμοποιεί το component ReportsMap και το τρίτο χρησιμοποιεί το component NewReportNavigator. Το Screen component δέχεται αρκετές παραμέτρους και έχει επιλογές όπως την εμφάνιση και τον τίτλο της κεφαλίδας καθώς. Μία επιλογή που υπάρχει είναι ο ορισμός του component που εμφανίζεται στη δεξιά περιοχή της κεφαλίδας. Για αυτό δημιουργήθηκε ένα καινούριο component, το AccountRightHeader. Αυτό το μόνο που κάνει είναι να εμφανίζει ένα εικονίδιο για το λογαριασμό του χρήστη και να ανοίγει το παράθυρο σύνδεσης και εγγραφής το οποίο θα περιγραφεί παρακάτω. Αυτό χρησιμοποιήθηκε στα Home και ReportsMap Screens.

```
useEffect(() => {
  if (user) {
    getUserReports(user.id).then(user_reports => {
      if (user_reports != null) {
        dispatch(updateUserReportsData(user_reports));
      }
    });
    messaging()
      .getToken()
      .then(token => {
        updateFirebaseInfo(user.id, token, 'Android device id');
      });
  }
}, [user]);

useEffect(() => {
  intializeApp().then(() => {
    messaging().requestPermission();
    messaging().onMessage(() => {
      if (user)
        getUserReports(user.id).then(user_reports => {
          if (user_reports != null) {
            dispatch(updateUserReportsData(user_reports));
          }
        });
    });
    messaging().onNotificationOpenedApp(remoteMessage => {
      console.log('Notification when in background:', remoteMessage);
    });
    messaging()
      .getInitialNotification()
      .then(remoteMessage => {
        console.log('Notification when quit:', remoteMessage);
      });
  });
}, []);
```

Σχήμα 4.34: Οι δύο χρήσεις του useEffect στον MainNavigator

```
return (
  <Navigator>
    <Screen
      name="Home"
      component={HomeScreen}
      options={{
        title: '',
        headerShadowVisible: false,
        cardStyle: {backgroundColor: colors.primaryBackground},
        headerStyle: {
          backgroundColor: colors.primaryBackground,
        },
        headerTransparent: true,
        headerRight: () => <AccountRightHeader />,
      }}
    />

    <Screen
      name="ReportsMap"
      options={{
        title: '',
        headerShadowVisible: false,
        cardStyle: {backgroundColor: colors.primaryBackground},
        headerStyle: {
          backgroundColor: colors.primaryBackground,
        },
        headerTransparent: true,
        headerRight: () => <AccountRightHeader />,
      }}>
      {() => (
        <ReportsProvider>
          <FiltersProvider>
            <ReportsMap />
          </FiltersProvider>
        </ReportsProvider>
      )}
    </Screen>
    <Screen
      name="NewReportNav"
      component={NewReportNavigator}
      options={{
        title: '',
        headerShown: false,
        cardStyle: {backgroundColor: colors.primaryBackground},
        presentation: 'modal',
      }}
    />
  </Navigator>
);
```

Σχήμα 4.35: Ο MainNavigator

Ο Navigator της Νέας Αναφοράς

Όπως αναφέρθηκε παραπάνω ο MainNavigator μπορεί να ανοίξει μία οθόνη με component NewReportNavigator (Σχήμα 4.36). Όπως επίσης αναφέρθηκε παραπάνω ο συγκεκριμένος Navigator είναι Top Bar Navigator. Αυτό σημαίνει ότι υπάρχουν διαφορετικές επιλογές στο πως μπορεί να αλλάξει η ενεργή οθόνη. Η πρώτη επιλογή είναι να γίνει κλικ πάνω στην αντίστοιχη καρτέλα, η δεύτερη επιλογή είναι να γίνει κύλιση προς τα δεξιά ή προς τα αριστερά και η τρίτη είναι να γίνει προγραμματιστικά. Στη συγκεκριμένη περίπτωση ήταν αναγκαίο να ακολουθείται κάποια συγκεκριμένη σειρά στην πλοήγηση ώστε να μην υπάρχει περίπτωση να υπάρχουν ελλιπείς πληροφορίες στη νέα αναφορά. Έτσι η δύο πρώτοι τρόποι αλλαγής οθόνης απενεργοποιήθηκαν. Αυτό έγινε αλλάζοντας την ιδιότητα screenOptions.swipeEnabled του Navigator σε false και βάζοντας έναν listener στους screenListeners του Navigator ώστε να αποτρέπεται η πλοήγηση στις οθόνες του όταν γίνεται κλικ στην αντίστοιχη καρτέλα. Οι τρεις οθόνες είναι οι LocationScreen, ReportInfoScreen και PhotoSelectionScreen που χρησιμοποιούν τα components με τα αντίστοιχα ονόματα.

```
src > navigation > TS NewReportNavigator.tsx > ...
 3   import {colors} from '../config';
 4   import ReportInfoScreen from '../screens/newReport/ReportInfoScreen';
 5   import {createMaterialTopTabNavigator} from '@react-navigation/material-top-tabs';
 6   import PhotoSelectionScreen from '../screens/newReport/PhotoSelectionScreen';
 7   import {NewReportProvider} from '../hooks/useNewReport/NewReportContext';
 8
 9   const {Navigator, Screen} = createMaterialTopTabNavigator();
10
11   const NewReportNavigator = () => {
12     return (
13       <NewReportProvider>
14         <Navigator
15           backBehavior="order"
16           screenListeners={() => ({
17             tabPress: e => {
18               e.preventDefault();
19             },
20           })}
21           screenOptions={{
22             swipeEnabled: false,
23             tabBarActiveTintColor: colors.primaryText,
24             tabBarStyle: {
25               backgroundColor: colors.secondaryBackground,
26             },
27             tabBarIndicatorStyle: {backgroundColor: colors.primaryText},
28           }}>
29           <Screen
30             name="LocationScreen"
31             component={LocationScreen}
32             options={{title: 'ΤΟΠΟΘΕΣΙΑ'}}
33           />
34           <Screen
35             name="ReportInfoScreen"
36             component={ReportInfoScreen}
37             options={{title: 'ΛΕΠΤΟΜΕΡΕΙΕΣ'}}
38           />
39           <Screen
40             name="PhotoSelectionScreen"
41             component={PhotoSelectionScreen}
42             options={{title: 'ΕΙΚΟΝΑ'}}
43           />
44         </Navigator>
45       </NewReportProvider>
46     );
47   };
48
49   export default NewReportNavigator;
```

Σχήμα 4.36: Ο NewReportNavigator

4.4.8 Οθόνη Σύνδεσης και Εγγραφής

Σε αυτή την ενότητα θα γίνει μία περιγραφή της παραθύρου που εμφανίζει τις οθόνες σύνδεσης και εγγραφής. Όπως προαναφέρθηκε σε προηγούμενη ενότητα, στο φάκελο useSignInModal υπάρχουν αρχεία για τα components που χρησιμοποιούνται για να συνθέσουν αυτό το παράθυρο.

Αρχικό component που χρησιμοποιείται στις δύο βασικότερες οθόνες, αυτές τις σύνδεσης και της εγγραφής, είναι το CustomTextInput. Ο βασικός λόγος που χρειάστηκε ένα ειδικό component αντί να χρησιμοποιηθεί το απλό TextInput της React Native ήταν η διαχείριση των σφαλμάτων και η αλλαγή της εμφάνισης βάσει αυτών. Στο αρχείο CustomTextInput.tsx λοιπόν αρχικά ορίζεται μία διεπαφή TextInputProps με ιδιότητες invalid, error, placeholder, keyboardType, secureTextEntry. Αυτές οι ιδιότητες θα μπορούν να παραμετροποιηθούν κάθε φορά που θα χρησιμοποιείται το CustomTextInput. Στη συνέχεια ορίζεται το CustomTextInput component ως ένα FunctionComponent. Αυτό σημαίνει πως πρακτικά είναι μία μέθοδος η οποία μπορεί να δέχεται κάποιες παραμέτρους μπορεί να εκτελεί κώδικα εσωτερικά και επιστρέφει ένα JSX στοιχείο [49]. Από εδώ και πέρα όταν αναφέρεται ότι

δημιουργείται ένα component θα εννοείται ότι είναι FunctionComponent. Το CustomTextInput (Σχήμα 4.37) δέχεται μερικές ιδιότητες κατά τη δημιουργία του ως παραμέτρους. Αυτές είναι οι πέντε της διεπαφής TextInputProps και οι value και onChangeText() από τη διεπαφή IInputProps της βιβλιοθήκης Native Base. Οπότε πρακτικά υπάρχουν επτά ιδιότητες που μπορούν να παραμετροποιήσουν το component. Έπειτα επιστρέφεται ένα fragment το οποίο μπορεί να εμφολεύσει πολλά components χωρίς να έχει κάποιο δικό του tag [38]. Μέσα σε αυτό χρησιμοποιείται ένα TextInput component με όλες τις ιδιότητες που χρειάζεται και τις οποίες παίρνει από τις παραμέτρους του CustomTextInput. Εδώ υπάρχει η λογική ότι το πλαίσιο του TextInput αλλάζει χρώμα και γίνεται κόκκινο όταν υπάρχει κάποια σφάλμα στο πεδίο. Το TextInput component ακολουθείται από ένα TextComponent το οποίο εμφανίζεται υπό προϋποθέσεις (conditional rendering). Εμφανίζεται μόνο όταν η ιδιότητα invalid είναι αληθής.

```
const CustomTextInput: React.FC<TextInputProps & IInputProps> = props => {
  const {
    invalid,
    error,
    placeholder,
    value,
    onChangeText,
    keyboardType,
    secureTextEntry,
  } = props;
  const styles = createStyles();
  return (
    <>
      <TextInput
        autoCapitalize="none"
        placeholder={placeholder}
        placeholderTextColor={colors.primaryText}
        value={value}
        onChangeText={onChangeText}       You, 5 months ago • Many many enhancements
        keyboardType={keyboardType}
        secureTextEntry={secureTextEntry}
        selectionColor={colors.primaryText}
        style={[
          styles.textInput,
          {
            borderColor: invalid || error ? '#ff3333' : colors.primaryText,
          },
        ]}
      />
      {invalid && (
        <Text
          style={{
            textAlign: 'left',
            width: '100%',
            color: '#ff3333',
          }}>
          {error?.message}
        </Text>
      )}
    </>
  );
};
```

Σχήμα 4.37: Η μέθοδος δημιουργίας του CustomTextInput component

Τέλος, μετά τον ορισμό του CustomTextInput component, ορίζεται η μέθοδος createStyles (Σχήμα 4.38) η οποία καλεί τη μέθοδο StyleSheet.create() ώστε να υπάρχουν κάπου έτοιμα ορισμένα τα styles με σκοπό να χρησιμοποιηθούν έτοιμα από τα components. Αυτό χρησιμοποιείται περισσότερο για θέμα οργάνωσης ώστε να είναι πιο απλός ο κώδικας του component. Η ίδια λογική για τα styles

χρησιμοποιείται από όλα τα components οπότε δε θα γίνει περαιτέρω αναφορά στην περιγραφή των υπολοίπων.

```
58  const createStyles = () =>
59    StyleSheet.create({
60      textInput: {
61        width: '100%',
62        borderRadius: 15,
63        borderWidth: 1,
64        height: 50,
65        color: colors.primaryText,
66        paddingHorizontal: '5%',
67        marginVertical: 15,
68      },
69    });
70
```

Σχήμα 4.38: Παράδειγμα χρήσης της μεθόδου StyleSheet.create()

Επόμενο component είναι το SignInScreen, δηλαδή η οθόνη σύνδεσης χρήστη. Ένα βασικό hook που χρησιμοποιήθηκε σε αυτό το component είναι το useForm της βιβλιοθήκης react-hook-form. Αυτό το hook είναι πολύ χρήσιμο στη διαχείριση των πεδίων μιας φόρμας καθώς υπάρχουν έτοιμες μέθοδοι οι οποίες είτε μπορούν να παρακολουθούν αλλαγές σε συγκεκριμένα πεδία, να διαβάσουν τις τιμές τους να διαγράψουν τις τιμές τους και συνολικά όλης της φόρμας και πολλά άλλα. Αρχικά ορίζεται η διεπαφή SignInForm με ιδιότητες email και password ώστε να χρησιμοποιηθεί ως τύπος του useForm hook. Έπειτα δημιουργείται το SignInScreen component με ιδιότητες μία μέθοδο onClose() και μία μέθοδο onClose(). Μέσα στη μέθοδο δημιουργίας του component αρχικοποιείται το useForm hook με τύπο SignInForm και εξάγονται οι ιδιότητες του, control, handleSubmit() και reset(). Στη συνέχεια ορίζονται δύο μεταβλητές κατάστασης, η loginError με τη μέθοδό της setLoginError() και η loading με τη μέθοδό της setLoading(). Η πρώτη χρησιμοποιείται ώστε να γίνεται γνωστό το πότε προκύπτει ένα σφάλμα και η δεύτερη ώστε να γίνεται γνωστό το αν υπάρχει αναμονή για τη σύνδεση του χρήστη. Έπειτα ορίζεται η μέθοδος onSignInSubmit() (Σχήμα 4.39) η οποία δέχεται μία παράμετρο data τύπου SignInForm. Η μέθοδος αυτή καλεί τη μέθοδο reset() του useForm hook, ώστε να αδειάσουν όλα τα πεδία, αλλάζει την τιμή της loading σε true με τη χρήση της setLoading() και καλεί τη μέθοδο login() του αρχείου user.ts ώστε να γίνει το αίτημα για σύνδεση. Αν υπάρξει απάντηση τότε καλείται η μέθοδος loginUser() του useAuth hook και απευθείας η onClose() ώστε να κλείσει όλο το ActionSheet. Αν δεν υπάρξει απάντηση τότε καλείται η setLoginError() με τιμή true.

```
const onSignInSubmit: SubmitHandler<SignInForm> = async data => {
  reset();
  setLoading(true);
  const response = await login({email: data.email, password: data.password});
  setLoading(false);
  if (response) {
    await loginUser({user: response?.data});
    onClose();
  } else {
    setLoginError(true);
  }
};
```

Σχήμα 4.39: Η μέθοδος onSignInSubmit() της οθόνης σύνδεσης

Το component επιστρέφει ένα View component (Σχήμα 4.40) μέσα στο οποίο υπάρχουν δύο Controller components, ένα για το κάθε πεδίο Email και Κωδικός, ένα View το ενδεχόμενο σφάλμα το οποίο εμφανίζεται μόνο όταν η μεταβλητή loginError είναι αληθής, ένα κουμπί το οποίο υποβάλει τη φόρμα όταν πατιέται και ένα κείμενο ώστε να παραπέμψει το χρήστη να κάνει εγγραφή αν δεν έχει λογαριασμό.

Το Controller component είναι μέρος της βιβλιοθήκης react-hook-form. Έχει διάφορες ιδιότητες προς χρήση, μερικές από τις οποίες είναι:

- rules: ένα αντικείμενο στο οποίο ορίζονται οι κανόνες βάσει των οποίων η τιμή ενός πεδίου είναι αποδεκτή. Στα συγκεκριμένα πεδία προστέθηκε ο κανόνας “required” καθώς τα πεδία είναι υποχρεωτικά και το μήνυμα που επιστρέφει αν δεν ικανοποιείται ο κανόνας είναι “Υποχρεωτικό πεδίο”
- render: Υποχρεωτική ιδιότητα η οποία ορίζει ποιο component η στοιχείο θα εμφανίσει ο Controller. Στη συγκεκριμένη περίπτωση εμφανίζει ένα CustomTextInput
- control: Που ορίζει το από που ελέγχεται ο συγκεκριμένος Controller. Και στα δύο πεδία το control είναι αυτό που εξήχθη από το useForm hook. Έτσι είναι ξεκάθαρο ότι και τα δύο πεδία ελέγχονται από ένα σημείο. Με αυτόν τον τρόπο όταν, για παράδειγμα, κληθεί η μέθοδος reset() θα καθαρίσουν όλα τα πεδία που ελέγχονται από το ίδιο control.

Το κουμπί για την υποβολή της φόρμας είναι ένα custom component που θα περιγραφεί αργότερα. Πρακτικά, επειδή σε αρκετά σημεία χρειάστηκε να υπάρχει ένα κουμπί που να ακολουθεί το θέμα της εφαρμογής, δημιουργήθηκε ένα component που χρησιμοποιείται όπου χρειάζεται. Σε αυτήν την περίπτωση όταν κάποιος χρήστης κάνει κλικ σε αυτό το κουμπί καλείται η handleSubmit του useForm hook και έπειτα η onSignInSubmit().

Το κείμενο που παραπέμπει τον χρήστη στο να κάνει εγγραφή περικλείεται σε ένα Pressable component, το οποίο είναι παρόμοιο με το View με τη διαφορά ότι δέχεται κλικ τα οποία διαχειρίζεται με τη μέθοδο onPress(). Η onPress στη συγκεκριμένη περίπτωση καλεί τη μέθοδο setActivePage() με τιμή ‘signup’. Έτσι αλλάζει η οθόνη και εμφανίζεται η οθόνη εγγραφής.

```
return (
  <View style={{width: '80%', alignItems: 'center'}}>
    <Controller
      name="email"
      defaultValue=""
      control={control}
      rules={{required: 'Υποχρεωτικό πεδίο'}}
      render={({field: {onChange, value}, fieldState: {invalid, error}}) => (
        <CustomTextInput
          keyboardType="email-address"
          placeholder="E-mail *"
          value={value}
          onChangeText={onChange}
          invalid={invalid}
          error={error}
        />
      )}
    />
    <Controller
      name="password"
      defaultValue=""
      control={control}
      rules={{required: 'Υποχρεωτικό πεδίο'}}
      render={({field: {onChange, value}, fieldState: {invalid, error}}) => (
        <CustomTextInput
          value={value}
          onChangeText={onChange}
          invalid={invalid}
          error={error}
          secureTextEntry
          placeholder="Κωδικός *"
        />
      )}
    />
    {loginError && (
      <View style={styles.errorContainer}>
        <Text style={styles.errorText}>Η σύνδεση απέτυχε</Text>
      </View>
    )}
    <View style={{marginVertical: 15}}>
      <CustomButton
        onPress={handleSubmit(data => {
          setLoginError(false);
          onSignInSubmit(data);
        })}       You, 3 months ago • App improvements
        buttonText={loading ? 'ΣΥΝΔΕΣΗ...' : 'ΣΥΝΔΕΣΗ'}
      />
    </View>
```

Σχήμα 4.40: Το SignInScreen

Η ίδια ακριβώς λογική ακολουθείται και στο component SignUpScreen. Αξίζει να σημειωθεί ότι υπάρχει η μέθοδος doLogin() η οποία εκτελείται όταν γίνεται μία επιτυχής εγγραφή. Οι υπόλοιπες λειτουργίες και η δομή της οθόνης είναι παρόμοιες.

Τέλος, υπάρχει ένα component UserOptions το οποίο μέχρι στιγμής δίνει την επιλογή στο χρήστη να αποσυνδεθεί. Το component επιστρέφει ένα View που περιέχει ένα Pressable. Η onPress() καλεί την onClose() ώστε να κλείσει το ActionSheet και μετά καλεί την logoutUser() του useAuth() hook ώστε να εκτελεστεί η αποσύνδεση.

Το βασικό αρχείο που πρακτικά περιγράφει όλο το ActionSheet και στο οποίο χρησιμοποιούνται όλα τα παραπάνω components είναι το αρχείο index.tsx του φακέλου components. Αρχικά, κάποια βασικά components που εισάγονται είναι τα εξής:

- ActionSheet: Είναι ένα component της βιβλιοθήκης native-base η οποία παρέχει πολλά έτοιμα προς χρήση components και αρκετά που δεν προσφέρει η React Native στο πακέτο της. Το ActionSheet είναι ένα component διαλόγου το οποίο εμφανίζεται πάνω από την τρέχουσα οθόνη [50].
- KeyboardAvoidingView: Είναι ένα component του πακέτου της React Native και βοηθάει στην αυτόματη προσαρμογή μιας οθόνης κατά την εμφάνιση ή απόκρυψη του πληκτρολογίου.
- ScrollView: Είναι ένα component του πακέτου της React Native και επιτρέπει την κύλιση του περιεχομένου που βρίσκεται εντός του.

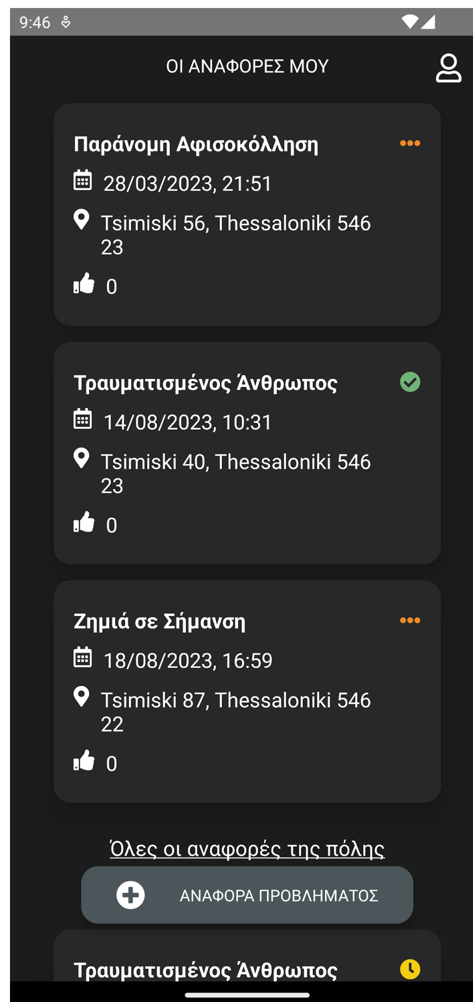
Στο index.tsx αρχείο δημιουργείται το component SignInActionSheet (Σχήμα 4.41). Επιστρέφει ένα Actionsheet με ένα KeyboardAvoidingView εσωτερικά. Μέσα στο KeyboardAvoidingView υπάρχει ένα Actionsheet.Content. Έτσι αναγνωρίζεται το περιεχόμενο του Actionsheet. Μέσα υπάρχει ένα ScrollView. Εντός του ScrollView εμφανίζονται δυναμικά ένα από τα τρία components, το UserOptions, το SignInScreen ή το SignUpScreen. Το UserOptions εμφανίζεται όταν υπάρχει το αντικείμενο user του useAuth hook. Το SignInScreen εμφανίζεται όταν η τιμή της μεταβλητής page του SignInModalContext είναι ‘login’. Το SignUpScreen εμφανίζεται όταν η τιμή της μεταβλητής page του SignInModalContext είναι ‘signup’.

```
const SignInActionSheet: React.FC<SignInActionSheetProps> = ({
  isOpen,
  onClose,
  getPage,
  setActivePage,
}) => {
  const page = getPage();
  const {user} = useAuth();

  return (
    <Actionsheet isOpen={isOpen} onClose={onClose} hideDragIndicator>
      <KeyboardAvoidingView
        behavior={'padding'}
        enabled={true}
        style={{width: '100%'}}>
        <Actionsheet.Content
          style={{backgroundColor: colors.secondaryBackground}}>
          <ScrollView
            stickyHeaderIndices={[0]}
            nestedScrollEnabled={true}
            style={{width: '100%'}}
            contentContainerStyle={{alignItems: 'center'}}>
            <Pressable
              onPress={onClose}
              style={{margin: 15, alignSelf: 'flex-end', height: 50}}>
              <Icon
                name="times-circle"
                color={colors.primaryText}
                size={25}
                solid
              />
            </Pressable>
            {user && (
              <UserOptions onClose={onClose} setActivePage={setActivePage} />
            )}
            {page === 'login' && !user && (
              <SignInScreen onClose={onClose} setActivePage={setActivePage} />
            )}
            {page === 'signup' && (
              <SignUpScreen onClose={onClose} setActivePage={setActivePage} />
            )}
          </ScrollView>
        </Actionsheet.Content>
      </KeyboardAvoidingView>
    </Actionsheet>
  );
};

export default SignInActionSheet;
```

Σχήμα 4.41: Η δομή του SignInActionSheet

4.4.9 Αρχική οθόνη

Η αρχική οθόνη (Σχήμα 4.42) είναι η οθόνη που βλέπει ο χρήστης όταν ανοίγει την εφαρμογή. Σε αυτή αρχικά εμφανίζεται ένα δείγμα αναφορών της πόλης. Αν ο χρήστης είναι συνδεδεμένος τότε θα δει και κάποιο δείγμα των δικών του αναφορών. Τέλος, υπάρχει ένα κουμπί στο κάτω μέρος της οθόνης ώστε να κάνει μια νέα αναφορά.



Σχήμα 4.42: Η αρχική οθόνη της εφαρμογής

Για την εμφάνιση των αναφορών και των κουμπιών της εφαρμογής, καθώς αυτά θα ήταν αρκετά επαναχρησιμοποιήσιμα, δημιουργήθηκαν δύο components. Το πρώτο είναι το ReportComponent και το δεύτερο είναι το CustomButton.

ReportComponent

Το ReportComponent είναι υπεύθυνο για να δείχνει τις πληροφορίες της αναφοράς. Δέχεται ένα αντικείμενο τύπου Report και μία μέθοδο onPress() ώστε να ορίζεται η ενέργεια που θα εκτελείται όταν κάποιος κάνει κλικ πάνω σε μία αναφορά. Αρχικά ορίζονται μεταβλητές για κάποιες πληροφορίες της αναφοράς που χρειάζονται επεξεργασία πριν παρουσιαστούν. Η πρώτη είναι η shortAddress στην οποία αποθηκεύεται μία μικρότερη μορφή της διεύθυνση της αναφοράς ώστε να παραλείπεται το “Thessaloniki, Greece” καθώς αυτό εννοείται. Η δεύτερη είναι η formattedDate στην οποία αποθηκεύεται η επεξεργασμένη μορφή της ημερομηνίας καθώς αυτή αποθηκεύεται σε UTC (Universal Time Coordinated) και πρέπει να μετατραπεί ώστε να ταιριάζει στη ζώνη ώρας της Ελλάφος. Για αυτή τη μετατροπή χρησιμοποιείται η βιβλιοθήκη moment-timezone. Στη συνέχεια, με τη χρήση του useMemo hook, αποφασίζεται το εικονίδιο που θα εμφανίζεται ανάλογα με την κατάσταση της αναφοράς (Σχήμα 4.25)

CustomButton

Το CustomButton (Σχήμα 4.43) χρησιμοποιείται ώστε να υπάρχει ένα καθολικό στυλ στα κουμπιά της εφαρμογής και ώστε να είναι επαναχρησιμοποιήσιμα καθώς χρειάζονται σε αρκετές οθόνες. Το CustomButton έχει τέσσερις ιδιότητες, μία μέθοδος onPress() η οποία ορίζει τι θα εκτελεστεί όταν το κουμπί πατηθεί, το buttonText που ορίζει το κείμενο που θα εμφανίζεται στο κουμπί, το iconName για την περίπτωση που το κουμπί χρειάζεται να εμφανίζει και ένα εικονίδιο (όπως το κουμπί της νέας αναφοράς) και το enabled που καθορίζει αν το κουμπί είναι ενεργό. Αν το enabled είναι false τότε αλλάζει το χρώμα του κουμπιού ώστε να είναι ξεκάθαρο ότι δεν είναι ενεργό και αποτρέπεται η εκτέλεση της μεθόδου onPress().

```
You, 2 days ago | 1 author (You)
interface CustomButtonProps {
  onPress: () => void;
  buttonText: string;
  iconName?: string;
  enabled?: boolean;
}

const CustomButton: React.FC<CustomButtonProps> = (
  props: CustomButtonProps,
) => {
  const buttonEnabled = props.enabled === true || props.enabled === undefined;
  const styles = createStyles(buttonEnabled);
  return (
    <Pressable
      onPress={() => {
        buttonEnabled && props.onPress();
      }}>
      <View style={styles.container}>
        {props.iconName && (
          <Icon name={props.iconName} size={25} color={colors.primaryText} />
        )}
        <Text style={{color: colors.primaryText}}>{props.buttonText}</Text>
      </View>
    </Pressable>
  );
};

const createStyles = (enabled?: boolean) =>
  StyleSheet.create({
    container: {
      backgroundColor: enabled
        ? colors.buttonsColor
        : colors.secondaryBackground,
      flexDirection: 'row',
      justifyContent: 'space-evenly',
```

Σχήμα 4.43: Το CustomButton

Στην περίπτωση του κουμπιού της νεάς αναφοράς η onPress() εκτελεί την μέθοδο navigate του useNavigation hook με παράμετρο “NewReportNav” που επιτρέπει την αλλαγή οθόνης στον Navigator. Η παράμετρος πρέπει να έχει την ίδια τιμή με την ιδιότητα name κάποιου από τα Screen components του Navigator ώστε να μπορέσει να γίνει η πλοήγηση.

Αξίζει να σημειωθεί ο τρόπος που γίνεται η χρήση του redux store ώστε να υπάρξει πρόσβαση στις αναφορές. Αυτό γίνεται με τη χρήση του useSelector hook της βιβλιοθήκης react-redux περνώντας ως παράμετρο μία μέθοδο που ορίζει πιο κομμάτι του state χρειάζεται. Στη συγκεκριμένη περίπτωση χρειάστηκε το state.reports. Με αυτόν τον τρόπο υπάρχει πρόσβαση στο allData και στο userData.

4.4.10 Χάρτης Αναφορών

Όταν κάποιος κάνει κλικ πάνω σε μία αναφορά ή στον κείμενο “Δες όλες τις αναφορές της πόλης” τότε πηγαίνει στην οθόνη με τον χάρτη τον αναφορών. Στο πάνω μέρος της οθόνης εμφανίζονται τα φίλτρα, από κάτω φαίνεται ο χάρτης με τις πινέζες που αντιστοιχούν στις αναφορές και από κάτω

υπάρχει μία λίστα, η οποία αρχικά είναι ελαχιστοποιημένη, αλλά είναι συρόμενη ώστε να μπορεί να εμφανιστεί στο πλήρες μέγεθός της.

Τοπικά Contexts

Για την οθόνη του χάρτη χρειάστηκαν δύο καινούρια contexts, αυτό των φίλτρων, δηλαδή το FiltersContext, και ένα context που κρατάει τοπικά την κατάσταση των αναφορών καθώς η λίστα αυτών μπορεί να αλλάξει ανάλογα με τα φίλτρα.

Το FiltersContext κρατάει απλά μία μεταβλητή κατάστασης filters με την αντίστοιχη μέθοδό της setFilters χρησιμοποιώντας το useState hook. Το αντικείμενο filters έχει τις ιδιότητες filters και showAll. Το filters με τη σειρά του περιέχει τις πιθανές καταστάσεις μιας αναφοράς, δηλαδή pending, inprogress, resolved και rejected. Οι αρχικές τιμές όλων είναι αληθής καθώς εξ αρχής χρειάζεται να φαίνονται όλες οι αναφορές. Η ιδιότητα showAll επηρεάζει το αν θα φαίνονται όλες οι αναφορές ή μόνο του συγκεκριμένου χρήστη που έχει συνδεθεί.

Αντίστοιχα το ReportsContext κρατάει μία μεταβλητή κατάστασης reports, που είναι ένα πίνακας αντικειμένων τύπου Report, μαζί με την αντίστοιχη μέθοδό της, setReports().

Τοπικά components

Για τα φίλτρα χρειάστηκαν δύο components, το είναι είναι το FilterButton και το άλλο είναι το Filters. Το FilterButton είναι ένα κουμπί που έχει δύο καταστάσεις, επιλεγμένο και μη επιλεγμένο, ανάλογα με τις οποίες αλλάζει η εμφάνισή του. Η μόνη λειτουργία του είναι να εκτελεί την μέθοδο onPress() που δέχεται σαν ιδιότητα. Το Filters είναι component στο οποίο υπάρχει μία λίστα από FilterButton components. Συνολικά είναι έξι και χωρίζονται σε δύο κατηγορίες, αυτά που είναι σχετικά με την κατάσταση της αναφοράς, “Εκκρεμείς”, “Σε Εξέλιξη”, “Επιλυμένες”, “Απορριφθείσες”, και αυτά που είναι σχετικά με τον χρήστη, “Όλες” και “Οι δικές μου”. Το κάθε ένα FilterButton έχει τη δική του onPress() η οποία καλεί τη setFilters() του FiltersContext αλλάζοντας την κατάσταση των φίλτρων. Η μεταβλητή κατάστασης filters του FiltersContext είναι στον πίνακα εξάρτησης ενός useEffect (Σχήμα 4.44). Με αυτόν τον τρόπο κάθε φορά που αλλάζει κάποιο φίλτρο απευθείας ανανεώνεται η λίστα των αναφορών.

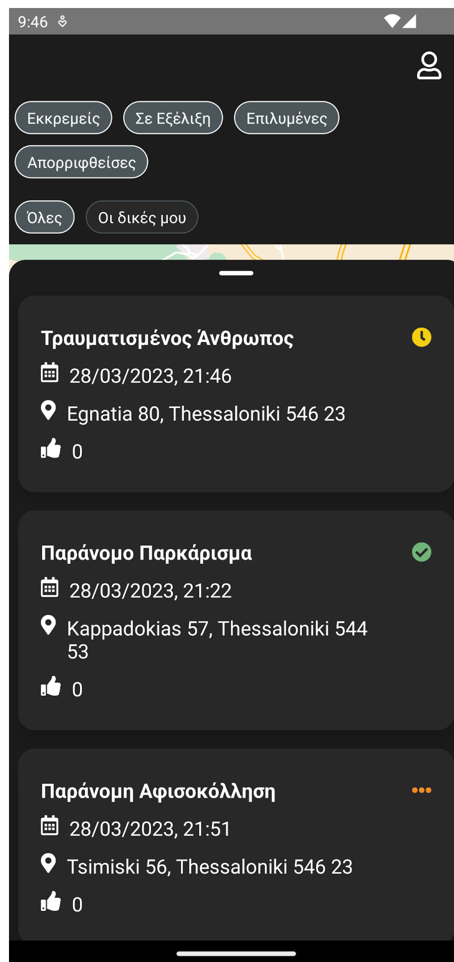
```
useEffect(() => {
  const activeFilters: string[] = [];
  if (filters?.statuses?.pending) activeFilters.push('Pending');
  if (filters?.statuses?.inprogress) activeFilters.push('In Progress');
  if (filters?.statuses?.resolved) activeFilters.push('Resolved');
  if (filters?.statuses?.rejected) activeFilters.push('Rejected');

  if (filters?.showAll) {
    setReports(
      allData?.filter(report => activeFilters.includes(report.status)),
    );
  } else {
    setReports(
      userData?.filter(report => activeFilters.includes(report.status)),
    );
  }

  setPending(filters?.statuses?.pending ?? false);
  setInProgress(filters?.statuses?.inprogress ?? false);
  setResolved(filters?.statuses?.resolved ?? false);
  setRejected(filters?.statuses?.rejected ?? false);
  setShowAll(filters?.showAll ?? false);
  setShowUser(!filters?.showAll);
}, [filters]);
```

Σχήμα 4.44: Αλλαγή των αναφορών ανάλογα με την επιλογή φίλτρων

Λίστα Αναφορών

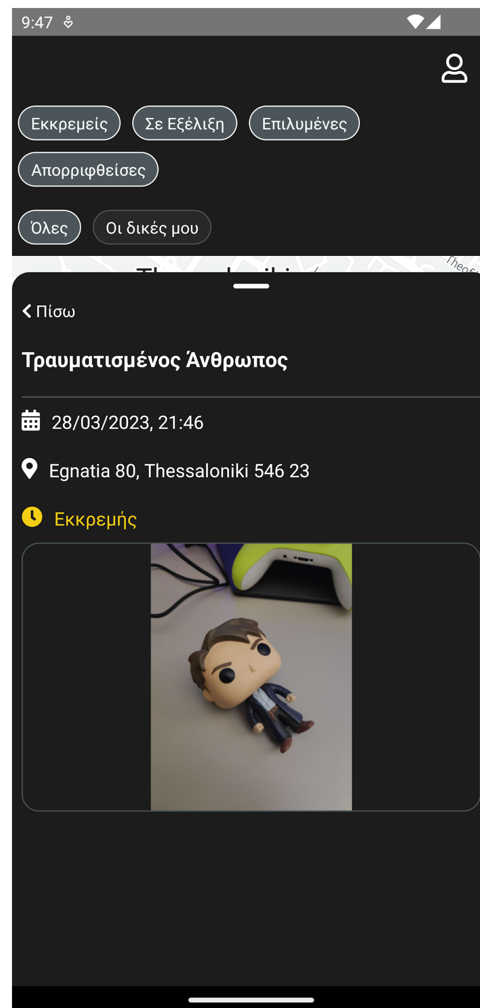
Η λίστα αναφορών είναι ένα συρόμενο παράθυρο στο οποίο μπορούν να οριστούν σημεία που κλειδώνει η θέση του και ο χρήστης μπορεί να το σύρει ώστε να δει όλο το περιεχόμενο του ή να το κρύψει. Αυτό εμφανίζεται πάνω από το χάρτη και δείχνει τη λίστα των αναφορών ανάλογα και με τα φίλτρα που έχουν επιλεχθεί. Αυτό το παράθυρο δημιουργείται με τη βοήθεια της βιβλιοθήκης @gorhom/bottom-sheet [51]. Τα αρχεία που απαρτίζουν αυτό το παράθυρο είναι στο φάκελο ReportsActionSheet. Στο αρχείο index.tsx ορίζεται το ReportsActionSheet component που αποτελείται από ένα Stack Navigator με δύο Screen components, ReportsList (Σχήμα 4.45) και ReportDetails (Σχήμα 4.46). Το πρώτο εμφανίζει τη λίστα των αναφορών και το δεύτερο εμφανίζει όλες τις λεπτομέρειες μιας επιλεγμένης αναφοράς. Δέχεται δύο παραμέτρους, μία μεταβλητή selectedReport και μία μέθοδο setSelectedReport(). Με τη χρήση του useEffect hook και τη μεταβλητή selectedReport στον πίνακα εξάρτησης, όταν αλλάζει η επιλεγμένη αναφορά αλλάζει και η οθόνη του Navigator στο ReportDetails και παράλληλα η θέση του παραθύρου ώστε να φαίνεται το περιεχόμενο. Ο Navigator εμφωλεύεται μέσα σε ένα BottomSheet component του @gorhom/bottom-sheet.

Το ReportsList component είναι πολύ απλό. Περιέχει ένα BottomSheetScrollView της βιβλιοθήκης @gorhom/bottom-sheet το οποίο είναι απλά ένα ScrollView που υποστηρίζει χειρονομίες του BottomSheet [51]. Μέσα στο BottomSheetScrollView εμφανίζεται μία λίστα από ReportComponent components. Η onPress() του κάθε ReportComponent εκτελεί τη μέθοδο navigation.navigate() με πρώτη παράμετρο “ReportDetails” και ως δεύτερη παράμετρο το αντικείμενο της αναφοράς.



Σχήμα 4.45: Η λίστα των αναφορών στην οθόνη του χάρτη

Το ReportDetails component κάνει αρκετά ίδια πράγματα με το ReportComponent με τη διαφορά ότι δείχνει τις πληροφορίες με μεγαλύτερη λεπτομέρεια. Αρχικά δείχνει την κατάσταση της αναφοράς και σε κείμενο πέρα από απλό εικονίδιο. Ακόμη, εμφανίζει το όνομα του χρήστη που έκανε την υποβολή της αναφοράς. Επίσης, εμφανίζει την εικόνα της αναφοράς. Αυτό γίνεται με τη χρήση του FastImage component, που είναι μία αναβάθμιση του Image component της React Native όσον αφορά τις επιδόσεις [53]. Βάζοντας ένα url στην ιδιότητα source κατεβαίνει η εικόνα. Σε αυτήν την περίπτωση γίνεται εκμετάλλευση ενός endpoint του API, του “/api/reports/getImage?reportId=”. Βάζοντας το αντίστοιχο id εμφανίζεται η κατάλληλη εικόνα.



Σχήμα 4.46: Η λεπτομέρειες μιας επιλεγμένης αναφοράς

Χάρτης

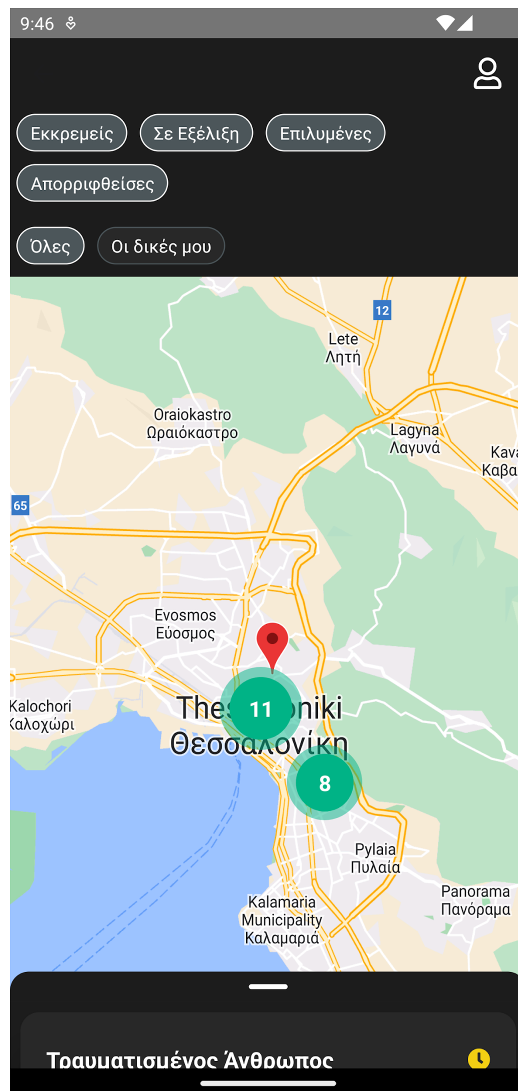
Ο χάρτης καθώς και το παράθυρο με τη λίστα των αναφορών και τα φίλτρα συνδυάζονται στο ReportsMap component (Σχήμα 4.47). Σε αυτό ορίζεται μία μεταβλητή κατάστασης selectedReport με τη μέθοδο setSelectedReport(). Έπειτα ορίζεται η μέθοδος setPinColor() η οποία ανάλογα με την κατάσταση της αναφοράς δίνει το αντίστοιχο χρώμα στην πινέζα της αναφοράς στο χάρτη.

Στη συνέχεια ορίζεται η μέθοδος handleGrantedPermissions() η οποία, από τη στιγμή που ο χρήστης έχει δώσει δικαιώματα για πρόσβαση της τοποθεσίας του, ανακτά την τοποθεσία με τη χρήση της μεθόδου getCurrentPosition() της βιβλιοθήκης react-native-geolocation-service. Η getCurrentPosition() επιστρέφει ένα αντικείμενο GeoPosition με ιδιότητες latitude, longitude, latitudeDelta, longitudeDelta και βάσει αυτών αλλάζει το κέντρο του χάρτη.

Έπειτα, χρησιμοποιείται το useEffect hook και στον πίνακα εξαρτήσεων υπάρχουν τα status.preciseLocation, status.approximateLocation και route.params.selectedReport. Το route.params.selectedReport είναι η παράμετρος selectedReport που περνάει στο ReportsMap όταν εκτελείται η μέθοδος navigation.navigate() όταν επιλέγεται μία αναφορά από την αρχική οθόνη. Αν λοιπόν έχει επιλεχθεί κάποια αναφορά τότε καλείται η setSelectedReport() και αλλάζει το κέντρο του χάρτη στις συντεταγμένες της αναφοράς. Αν δεν υπάρχει επιλεγμένη αναφορά γίνεται έλεγχος των δικαιωμάτων τοποθεσίας. Αν δεν έχουν ζητηθεί τότε καλείται η μέθοδος requestPermissions() του

PermissionsContext. Αν έχουν απορριφθεί τότε δεν εκτελείται τίποτα. Αν ο χρήστης έχει δώσει δικαιώματα για χρήση τοποθεσίας τότε καλείται η handleGrantedPermissions().

Το Component που χρησιμοποιείται για τον χάρτη είναι το MapView της βιβλιοθήκης react-native-map-clustering. Η συγκεκριμένη βιβλιοθήκη επιλέχθηκε γιατί το συγκεκριμένο MapView components ομαδοποιεί τις πινέζες στο χάρτη, σε αντίθεση με το MapView component της βιβλιοθήκης react-native-maps. Μέσα στο MapView υπάρχει μία λίστα από MapMarker components που αντιστοιχούν σε κάθε αναφορά. Όταν γίνεται κλικ σε κάποια πινέζα εκτελείται η setSelectedReport(). Έτσι ενεργοποιείται το useEffect στο ReportsActionSheet ώστε να εμφανιστούν οι λεπτομέρειες της αναφοράς.



Σχήμα 4.47: Η οθόνη του χάρτη

4.4.11 Δημιουργία αναφοράς

Σε προηγούμενη ενότητα έγινε περιγραφή του NewReportNavigator και του πως λειτουργεί. Σε αυτή την ενότητα θα αναλυθούν οι τρεις επιμέρους οθόνες που χρησιμοποιεί.

Οθόνη επιλογής τοποθεσίας

Στην οθόνη επιλογής τοποθεσίας (Σχήμα 4.48) αρχικά υπάρχει μία μπάρα αναζήτησης ώστε να μπορεί ο χρήστης να αναζητήσει την τοποθεσία του. Το μεγαλύτερο μέρος της οθόνης καλύπτεται από τον χάρτη και τέλος υπάρχουν δύο κουμπιά, ένα ώστε ο χρήστης να μπορεί να πάει πίσω και να ακυρώσει τη διαδικασία και ένα για να πάει στο επόμενο βήμα.

Σχήμα 4.48: Επιλογή τοποθεσίας κατά τη δημιουργία νέας αναφοράς

Λειτουργικά, αρχικά χρησιμοποιείται το useEffect hook με τα status.preciseLocation και status.approximateLocation του PermissionsContext στον πινακα εξαρτήσεων ώστε να ελέγχεται αν έχουν δοθεί δικαιώματα πρόσβασης τοποθεσίας. Αν έχουν ήδη δοθεί τότε, ανακτάται η τοποθεσία του χρήστη, το κέντρο του χάρτη αλλάζει βάσει αυτής και το κείμενο της μπάρας αναζήτησης αλλάζει αντίστοιχα ώστε να εμφανίσει τη διεύθυνση που αντιστοιχεί στις συντεταγμένες. Αν δεν έχουν ζητηθεί δικαιώματα τότε εμφανίζεται ο κατάλληλος διάλογος συστήματος. Αν έχουν απορριφθεί τότε δε γίνεται τίποτα. Το κουμπί "Αναζήτηση" εκτελεί μία μέθοδο findCoordinates() όταν πατηθεί ώστε να αντιστοιχιστεί η διεύθυνση σε συντεταγμένες.

Για να γίνει αυτό αλλά και για να αλλάξει η τιμή της μπάρας αναζήτησης ανάλογα με την τοποθεσία του χρήστη χρειάζεται να γίνει ένα αίτημα στο Geocoding API της Google. Για τη χρήση αυτού χρειάζεται ένα API key. Αυτό γίνεται πολύ απλά κάνοντας σύνδεση στο console.cloud.google.com δημιουργώντας ένα νέο project και ενεργοποιώντας τα Geocoding API services. Έπειτα μπορεί να γίνει η δημιουργία ενός κλειδιού [53]. Για την εκμετάλλευση του Geocoding API της Google δημιουργήθηκε ένα αρχείο geocoding.ts το φάκελο api. Σε αυτό υπάρχει μία μόνο μέθοδος geocode() (Σχήμα 4.49) η οποία δέχεται είτε μία παράμετρο address με τη διεύθυνση του χρήστη, είτε μία παράμετρο latlng που είναι ένα string με το γεωγραφικό μήκος και πλάτος της τοποθεσίας του χρήστη. Ανάλογα με την παράμετρο της μεθόδου κατά την κλήση της επιστρέφει το ανάλογο αποτέλεσμα. Δηλαδή αν περάσει η παράμετρος address τότε κάνει την αντιστοίχιση της τιμής της σε συντεταγμένες. Αν περάσει η παράμετρος latlng γίνεται αντιστοίχιση των συντεταγμένων σε διεύθυνση. Το αίτημα γίνεται στο “https://maps.googleapis.com/maps/api/geocode/json”.

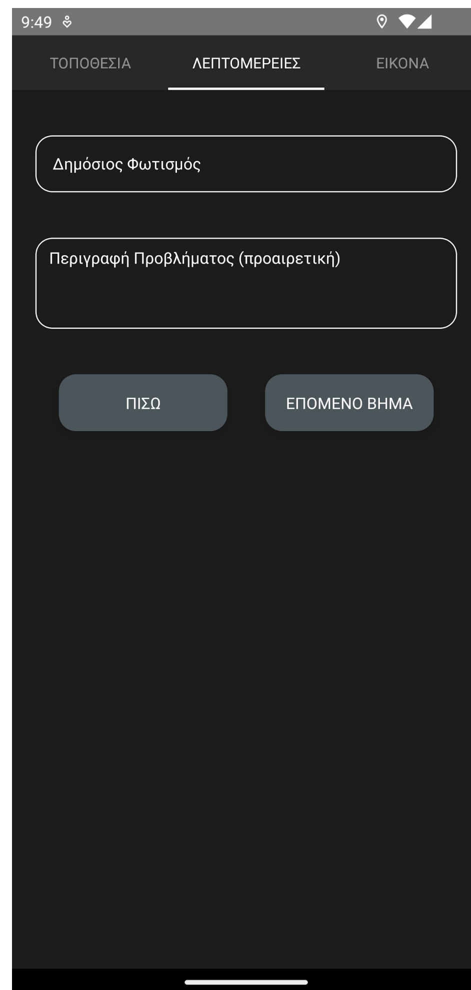
```
6   export const geocode = async ({
7     address,
8     latlng,
9   }: {
10    address?: string;
11    latlng?: string;
12  }): Promise<any | undefined> => {
13    try {
14      const type = address ? 'address' : 'latlng';
15      const queryParams = address ? `${address}, Θεσσαλονίκη` : latlng;
16      const response: AxiosResponse<any> = await axios.get(
17        `${mapsUrl}?${type}=${queryParams}&key=${apiKey}`,
18      );
19      return response.data;
20    } catch {
21      return undefined;
22    }
23  };
```

Σχήμα 4.49: Η μέθοδος geocode για την εκμετάλλευση του Geocoding API της Google

Τέλος, ο χρήστης μπορεί να προχωρήσει στο επόμενο βήμα μόνο όταν επιλεχθεί κάποια τοποθεσία. Όταν το κουμπί “Επόμενο Βήμα” πατηθεί τότε καλείται η μέθοδος setNewReport() του NewReportContext ώστε να αποθηκευτεί στη μεταβλητή κατάστασης newReport το address, το lat και το lng.

Οθόνη επιλογής λεπτομερειών αναφοράς

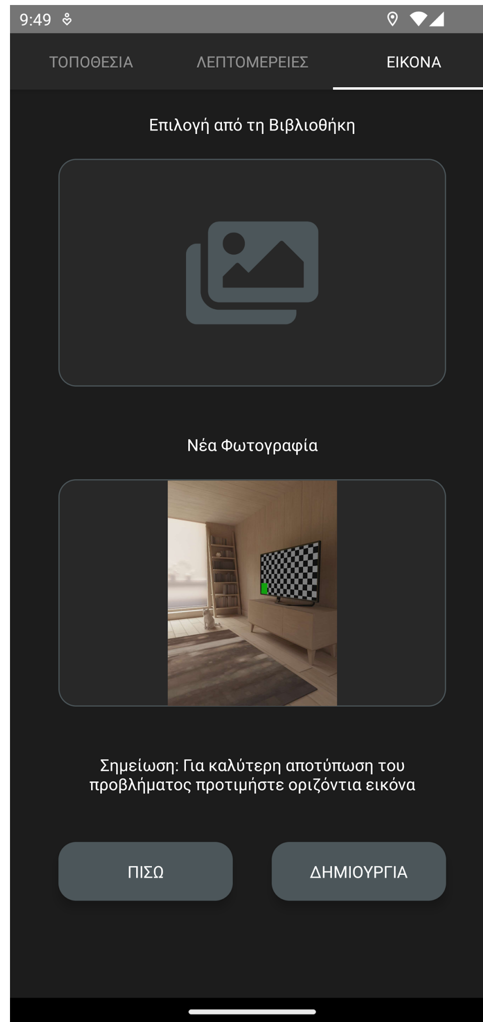
Η οθόνη επιλογής λεπτομερειών (Σχήμα 4.50) είναι πολύ απλή. Αρχικά υπάρχει μία λίστα επιλογών για την κατηγορία της αναφοράς. Οι πιθανές κατηγορίες είναι Παράνομο Παρκάρισμα, Ζημιά σε Σήμανση, Αποχέτευση, Δημόσιος Φωτισμός, Κάδοι Απορριμάτων - Ανακύκλωσης, Ζημιά Οδοστρώματος, Ζημιά Πεζοδρομίου, Τραυματισμένος Άνθρωπος, Τραυματισμένο Ζώο, Παράνομη Αφισοκόλληση, Παράνομο Γκραφίτι, Ζημιά σε παιδική χαρά, Ηχορύπανση, Πτώση Δέντρου και Βλάβη Ηλεκτροδότησης. Από κάτω υπάρχει ένα πεδίο εισαγωγής κειμένου ώστε να μπορέσει ο χρήστης να περιγράψει το πρόβλημα με μεγαλύτερη λεπτομέρεια.



Σχήμα 4.50: Η οθόνη επιλογής λεπτομερειών της νέας αναφοράς

Οθόνη επιλογής εικόνας

Στην οθόνη επιλογής εικόνας (Σχήμα 4.51) υπάρχουν δύο περιοχές. Αν γίνει κλικ στην πρώτη τότε ο χρήστης μπορεί να επιλέξει μία εικόνα από τη βιβλιοθήκη της συσκευής. Αν γίνει κλικ στη δεύτερη τότε ανοίγει η κάμερα της συσκευής ώστε να αποτυπωθεί μία νέα φωτογραφία. Για να γίνει αυτό χρησιμοποιήθηκε η βιβλιοθήκη react-native-image-picker. Στην πρώτη περίπτωση καλείται η μέθοδος launchImageLibrary(). Ο χρήστης μπορεί να επιλέξει μία εικόνα από τη βιβλιοθήκη και το αποτέλεσμα της μεθόδου αποθηκεύεται στη μεταβλητή κατάστασης newReport στην ιδιότητα image. Στη δεύτερη περίπτωση καλείται η μέθοδος launchCamera(), το αποτέλεσμα της οποίας αποθηκεύεται επίσης στην ιδιότητα image της μεταβλητής κατάστασης newReport. Από τη στιγμή που έχει επιλεχθεί μία εικόνα με έναν από τους δύο τρόπους ενεργοποιείται το κουμπί “Δημιουργία”. Όταν γίνει κλικ σε αυτό τότε καλείται η μέθοδος createNewReport() ώστε να γίνει το αίτημα για δημιουργία νέας αναφοράς.



Σχήμα 4.51: Προσθήκη εικόνας για τη νέα αναφορά

4.4.12 Λειτουργία Push Notifications

Για τη λειτουργία των ειδοποιήσεων χρειάζεται ένα project στο Firebase. Για να γίνει αυτό χρειάζονται τα εξής βήματα:

- Επίσκεψη στη σελίδα https://console.firebase.google.com
- Σύνδεση με έναν λογαριασμό Gmail
- Επιλογή “Add Project” στην αρχική σελίδα
- Στο πρώτο βήμα της δημιουργίας χρειάζεται ένα όνομα για το Project
- Στο επόμενο βήμα δίνεται η επιλογή ενεργοποίησης των Google Analytics. Είναι προαιρετικό για τη χρήση των ειδοποιήσεων.
- Αν επιλεχθεί η ενεργοποίησή τους τότε υπάρχει ένα ακόμα βήμα για την επιλογή λογαριασμού Google Analytics
- Αν όχι τότε πατώντας “Create Project” δημιουργείται το νέο project

Στην αρχική σελίδα του νέου project χρειάζεται να δημιουργηθούν δύο εφαρμογές, μία για κάθε πλατφόρμα. Και στις δύο περιπτώσεις χρειάζεται απλά το package name της εφαρμογής. Όταν δημιουργείται μία εφαρμογή στο Firebase project παράγονται ένα configuration αρχείο. Για android είναι το αρχείο google-services.json και για iOS είναι GoogleService-Info.plist. Αυτά τα δύο αρχεία πρέπει να προστεθούν στο φάκελο του React Native project. To google-service.json πρέπει να

προστεθεί στο φάκελο /android/app και το GoogleService-Info.plist στο φάκελο /ios/<project_name>. Για την ενεργοποίηση των ειδοποιήσεων στο Android δε χρειάζεται κάποια επιπλέον παραμετροποίηση. Εδώ πρέπει να σημειωθεί πως από την έκδοση 13 του Android και έπειτα χρειάζεται να ζητούνται δικαιώματα για ειδοποιήσεις από το χρήστη. Για να γίνει αυτό χρειάζεται το project να χρησιμοποιεί έκδοση της React Native 0.71+. Επειδή το project χρησιμοποιεί την έκδοση 0.68, δεν είναι δυνατό να ζητούνται δικαιώματα σε συσκευές με Android 13. Στις προηγούμενες εκδόσεις του Android αυτό είναι κάτι που δε χρειάζεται. Για το iOS χρειάζεται κάποια επιπλέον παραμετροποίηση η οποία είναι αναγκαία σε γενικότερο πλαίσιο ώστε η εφαρμογή να μπορεί να τρέχει και να διατίθεται σε iOS συσκευές. Για αυτό χρειάζεται ένας λογαριασμός προγραμματιστή στην Apple (ο οποίος έρχεται και με κάποιο κόστος, των $99 το χρόνο). Έπειτα χρειάζονται τέσσερα πράγματα :

- Η δημιουργία ενός App ID [46]. Κάνοντας σύνδεση στο Apple Developer Account, https://developer.apple.com/account , και επιλέγοντας Identifiers εμφανίζεται η επιλογή δημιουργίας νέου Identifier. Πρέπει να γίνει επιλογή “App IDs” και στο επόμενο βήμα επιλογή “App”. Στη συνεχεια χρειάζεται να δοθεί μία περιγραφή και ένα bundleId. Το bundleId είναι το package name της εφαρμογής. Παρακάτω υπάρχει μία λίστα με λειτουργίες [46]. Για να μπορέσει η εφαρμογή να δεχθεί ειδοποιήσεις χρειάζεται η επιλογή “Push Notifications”. Έπειτα μπορεί να ολοκληρωθεί η δημιουργία του App ID.
- Η δημιουργία ενός Certificate. Το Certificate χρησιμοποιείται ως υπογραφή της εφαρμογής ώστε να πιστοποιείται. Αρχικά από έναν υπολογιστή με MacOS χρειάζεται η δημιουργία ενός Certificate Signing Request. Αυτό γίνεται ανοίγοντας την εφαρμογή Keychain Access, επιλέγοντας από το μενού Keychain Access -> Certificate Assistane -> Request A Certificate From A Certificate Authority. Βάζοντας τα κατάλληλα στοιχεία δημιουργείται το Certificate Signing Request. Στη συνέχεια πηγαίνοντας και πάλι στο Apple Developer Account και επιλέγοντας Certificates μπορεί να γίνει δημιουργία ενός νέου. Ανάλογα με τις ανάγκες μπορεί να χρειαστεί ένα Development, που χρησιμοποιείται αποκλειστικά για να υπογράψει debug builds της εφαρμογής, και ένα για Distribution, που χρησιμοποιείται αποκλειστικά για την υπογραφή του build που θα διατεθεί στο App Store. Το μόνο που χρειάζεται είναι η επιλογή του τύπου και του Certificate Signing Request που δημιουργήθηκε.
- Η δημιουργία ενός Provisioning Profile [46]. Για αυτό χρειάζεται η επιλογή ενός App ID και ενός Certificate. Αν το Provisioning Profile είναι για Development χρειάζεται να επιλεχθούν και οι συσκευές στις οποίες θα μπορεί να εγκατασταθεί η εφαρμογή.
- Τέλος, χρειάζεται η δημιουργία ενός κλειδιού από την κατηγορία Keys του Apple Developer Account με την λειτουργία “Apple Push Notifications service (APNs)” ενεργοποιημένη [54]. Αυτό το κλειδί χρειάζεται να προστεθεί στην iOS εφαρμογή στο Firebase Project πηγαίνοντας στην κατηγορία Cloud Messaging και επιλέγοντας “Upload” στο “APNs Auth Key” στο iOS app.

Αφού έχουν γίνει όλα αυτά, μπορούν να ζητηθούν δικαιώματα για λήψη ειδοποιήσεων (Σχήμα 4.52). Προστέθηκαν τρεις listeners, ένας για την περίπτωση που η ειδοποίηση λαμβάνεται ενώ η εφαρμογή είναι ανοιχτή, ένας για την περίπτωση που η εφαρμογή είναι στο παρασκήνιο και ένας για την περίπτωση που η εφαρμογή είναι στο προσκήνιο.

```
useEffect(() => {
  intializeApp().then(() => {
    messaging().requestPermission();
    messaging().onMessage(() => {
      if (user)
        getUserReports(user.id).then(user_reports => {
          if (user_reports != null) {
            dispatch(updateUserReportsData(user_reports));
          }
        });
    });
    messaging().onNotificationOpenedApp(remoteMessage => {
      console.log('Notification when in background:', remoteMessage);
    });
    messaging()
      .getInitialNotification()
      .then(remoteMessage => {
        console.log('Notification when quit:', remoteMessage);
      });
  });
}, []);
```

Σχήμα 4.52: Η διαχείριση των ειδοποιήσεων

4.5 Ανάπτυξη της εφαρμογής διαχειριστή

Σε αυτήν την ενότητα θα γίνει μία περιγραφή της web εφαρμογής διαχειριστή όπου και θα εμφανίζονται οι αναφορές που υποβάλουν οι χρήστες και θα είναι δυνατή η διαχείρισή τους από κάποιον υπεύθυνο. Αρχικά θα γίνει μια περιγραφή της δημιουργίας του project και των βασικών του αρχείων. Έπειτα θα γίνει μια περιγραφή του πως χρησιμοποιείται το Realm Web SDK από μία React εφαρμογή και, τέλος, θα περιγραφεί η λειτουργία.

Δημιουργία project

Η δημιουργία της εφαρμογής έγινε με τη χρήση της React, για την οποία δε χρειάζεται κάποια ανάλυση καθώς οι έννοιες είναι ίδιες με τη React Native. Η δημιουργία του project έγινε με τη χρήση του typescript template εκτελώντας την εντολή "yarn create react-app thessreport-admin –template typescript" [55]. Τα βασικά αρχεία ενδιαφέροντος είναι τα index.tsx, index.css, App.tsx και App.css. Το index.tsx είναι το αρχείο εισαγωγής της εφαρμογής και σε αυτό γίνεται απλά render ό,τι υπάρχει μέσα στο App.tsx. Το App.tsx είναι το αρχείο το οποίο περιέχει το βασικά component (και τα παιδιά του) που εμφανίζεται όταν τρέχει η εφαρμογή. Στο App.css ορίζονται τα styles των components της εφαρμογής.

Realm Web SDK

Όπως αναφέρθηκε στην ενότητα 3.3.4 χρειάστηκε να δημιουργηθεί μία App Service εφαρμογή στο MongoDB Atlas περιβάλλον. Για τη σύνδεση της React εφαρμογής με την App Service εφαρμογή του Atlas χρειάστηκε η χρήση της βιβλιοθήκης realm-web. Η σύνδεση με την εφαρμογή γίνεται στο αρχείο App.tsx. Αρχικά δημιουργείται ένα realm app με την εντολή "Realm.App({"app-service-id"})" και αποθηκεύεται στη μεταβλητή realm_app. Στη συνέχεια μέσα σε ένα useEffect hook (Σχήμα 4.53)

καλείται η μέθοδος initdb() η οποία κάνει σύνδεση ανώνυμα στο realm app. Στη συνέχεια εξάγεται η επιθυμητή συλλογή και καλείται η μέθοδος watch() έτσι ώστε να ελέγχονται τυχόν αλλαγές σε αυτή και να γίνεται κάποια ενέργεια. Στη συγκεκριμένη περίπτωση, κάθε φορά που γίνεται κάποια αλλαγή στη συλλογή καλείται η μέθοδος getAllReports() η οποία κάνει αίτημα στο server για να πάρει όλες τις αναφορές. Έτσι, ανανεώνεται η λίστα των αναφορών.

```
const initdb = async () => {
  realm_app.logIn(Realm.Credentials.anonymous());
  const mongodb = realm_app.currentUser?.mongoClient("mongodb-atlas");
  const reports_collection = mongodb
    ?.db("thessreport")
    .collection("reports");

  if (reports_collection)
    for await (const change of reports_collection.watch()) {
      getAllReports().then((data) => setReports(data));
    }
};
```

Σχήμα 4.53: Χρήση του Realm SDK για παρακολούθηση της συλλογής

Λειτουργία

Αρχικά, όπως και στο React Native project χρειάστηκε να δημιουργηθεί ένα αρχείο για τις μεθόδους που κάνουν αιτήματα στο server. Δημιουργήθηκε λοιπόν ένα αρχείο reports.ts στο οποίο υπάρχουν δύο μέθοδοι (Σχήμα 4.54), η getAllReports() που κάνει αίτημα στο "/api/reports/all" και η μέθοδος updateReportsStatus() η οποία κάνει αίτημα στα "/api/reports/pending/", "/api/reports/inprogress/", "/api/reports/reject/", "/api/reports/resolve/", ανάλογα με την αλλαγή που χρειάζεται. Για να γίνουν τα αιτήματα χρησιμοποιείται η βιβλιοθήκη Axios όπως και στο React Native project.

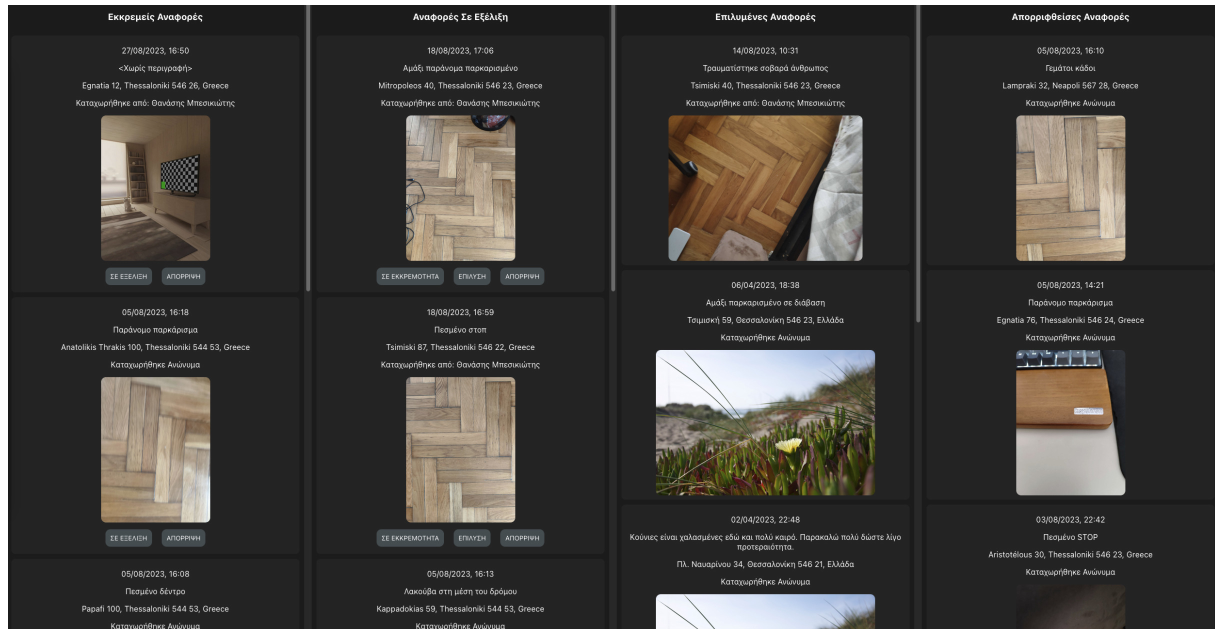
```
export const getAllReports = async (): Promise<Report[] | undefined> => {
  try {
    const response: AxiosResponse<Report[]> = await axios.get(
      `${serverUrl}/api/reports/all`
    );
    if (response) {
      return response.data;
    }
  } catch (e) {
    return undefined;
  }
};

export const updateReportStatus = async (
  reportId: string,
  newStatus: string
): Promise<Report | undefined> => {
  try {
    const response: AxiosResponse<Report> = await axios.patch(
      `${serverUrl}/api/reports/${newStatus}?reportId=${reportId}`
    );
    if (response) {
      return response.data;
    }
  } catch (e) {
    return undefined;
  }
};
```

Σχήμα 4.54: Οι μέθοδοι που κάνουν αιτήματα προς το API

Στο αρχείο App.tsx αρχικά ορίζεται μία μεταβλητή κατάστασης reports με την αντίστοιχη μέθοδό της setReports(). Στη συνέχεια με τη χρήση του useMemo hook οι αναφορές χωρίζονται ανάλογα με την κατάστασή του. Στη μεταβλητή pendingReport αποθηκεύονται οι αναφορές που είναι σε εκκρεμότητα, στην inProgressReports αποθηκεύονται οι μεταβλητές που είναι σε εξέλιξη, στη resolvedReports αποθηκεύονται οι αναφορές που έχουν επιλυθεί και στη rejectedReports αποθηκεύονται οι αναφορές που έχουν απορριφθεί. Έπειτα στο useEffect hook, που εκτελείται όταν ξεκινάει η εφαρμογή, καλείται η getAllReports() και έπειτα η setReports() με τα αποτελέσματά της. Έπειτα γίνεται η σύνδεση με το Realm App Service όπως περιγράφηκε παραπάνω. Δομικά, υπάρχει ένα div γονέας μέσα στο οποίο υπάρχουν τέσσερα ακόμα divs (Σχήμα 4.55), ένα για κάθε πιθανή κατάσταση μιας αναφοράς. Μέσα στο κάθε div από τα τέσσερα εμφανίζονται οι αναφορές, με τη χρήση του ReportDetails component, που έχουν την αντίστοιχη κατάσταση, ταξινομημένες βάσει της ημερομηνίας υποβολής σε μορφή στηλών. Το ReportDetails component δέχεται μία παράμετρο report τύπου Report και εμφανίζει τις πληροφορίες της αναφοράς. Ανάλογα με την κατάσταση μιας αναφοράς εμφανίζονται και τα κατάλληλα κουμπιά για την αλλαγή κατάστασης. Αν η αναφορά είναι εκκρεμής τότε μπορεί να αλλάξη σε εξέλιξη ή να απορριφθεί. Αν η αναφορά είναι σε εξέλιξη τότε μπορεί να επιστρέψει σε εκκρεμότητα, να επιλυθεί ή να απορριφθεί. Αν η αναφορά είναι επιλυμένη ή έχει απορριφθεί τότε δε μπορεί να αλλάξει η κατάστασή της Τέλος, δημιουργήθηκαν δύο νέες κλάσεις, report_container και reportsColumn ώστε να παραμετροποιηθεί η εμφάνιση των στηλών και των αναφορών. Στο (Σχήμα 4.56) παρουσιάζεται το πως φαίνεται τελικά η εφαρμογή διαχειριστή.

```
return (
  <div className="App">
    <div id="pendingReports" className="reportsColumn">
      <h3>Εκκρεμείς Αναφορές</h3>
      {pendingReports &&
        pendingReports
          .sort(
            (reportA, reportB) =>
              moment(reportB.date).unix() - moment(reportA.date).unix()
          )
          ?.map((r) => {
            return <ReportDetails key={r.id} report={r} />;
          })}
    </div>
    <div id="inProgressReports" className="reportsColumn">
      <h3>Αναφορές Σε Εξέλιξη</h3>
      {inProgressReports &&
        inProgressReports
          .sort(
            (reportA, reportB) =>
              moment(reportB.date).unix() - moment(reportA.date).unix()
          )
          ?.map((r) => {
            return <ReportDetails key={r.id} report={r} />;
          })}
    </div>
```

Σχήμα 4.55: Εμφάνιση των αναφορών στο αρχείο App.tsx



Σχήμα 4.56: Τελική εμφάνιση της εφαρμογής διαχειριστή

4.6 Επίλογος

Σε αυτό το κεφάλαιο έγινε αναλυτική περιγραφή της υλοποίησης όλης της ιδέας. Από τη δημιουργία και το στήσιμο της MongoDB βάσης, στη δημιουργία του API με τη χρήση Spring Boot, στη δημιουργία της εφαρμογής και κινητά αλλά και της εφαρμογής διαχειριστή.

Κεφάλαιο 5ο: Βελτιώσεις - Συμπεράσματα

Στα πλαίσια αυτής της πτυχιακής εργασίας αναπτύχθηκε μία εφαρμογή για κινητά ώστε να γίνει πιο εύκολη η αναφορά προβλημάτων σε έναν δήμο. Η ανάπτυξη του API μέσω Spring Boot αποδείχτηκε λίγο μεγαλύτερη πρόκληση από το αναμενόμενο καθώς το Documentation της ήταν λίγο δυσανάγνωστο. Η ανάπτυξη της εφαρμογής ήταν αρκετά ευκολότερη. Το τελικό αποτέλεσμα ήταν το επιθυμητό καθώς οι χρήστες μπορούν να υποβάλουν αναφορές στην τοποθεσία τους μέσα από μία ευρεία επιλογή κατηγοριών. Έτσι μπορούν να μεταφέρουν λεπτομερώς το πρόβλημα στην τοπική τους αυτοδιοίκηση. Από τη μεριά της τοπικής αυτοδιοίκησης υπάρχει ένας εύκολος τρόπος να δει κάποιος διαχειριστής τις αναφορές και να τις διαχειριστεί. Έτσι οι χρήστες μπορούν και να ενημερώνονται για την εξέλιξή τους.

Η εφαρμογή μπορεί να δεχτεί μερικές βελτιώσεις εμφάνιση αλλά και στην εμπειρία χρήστη. Αρχικά, θα μπορούσε να γίνει μία πιο σωστή διαρρύθμιση της αρχικής οθόνης ώστε το κουμπί της δημιουργίας νέας αναφοράς να μην εμποδίζει άλλα στοιχεία από πίσω. Επιπλέον, θα μπορούσαν να εμπλουτιστούν λίγο οι επιλογές των φίλτρων στην οθόνη του χάρτη ώστε οι χρήστες να μπορούν να φιλτράρουν τις αναφορές και ανάλογα με την κατηγορία της. Αυτό θα χρειαζόταν και κάποια αλλαγή στο σχεδιασμό της οθόνης. Επίσης, μία καλή προσθήκη, από τη στιγμή που υποστηρίζεται και από το API που δημιουργήθηκε, θα ήταν και η υποστήριξη των upvotes αναφορών από χρήστες. Ένα ακόμα κομμάτι που θα μπορούσε να βελτιωθεί είναι η ανατροφοδότηση του χρήστη, ώστε να είναι πιο ξεκάθαρο το αν υπήρξε κάποιο σφάλμα. Τέλος, η αποστολή επιπλέον πληροφοριών στις ειδοποιήσεις ώστε να γίνεται και καλύτερη διαχείριση από την εφαρμογή θα βελτίωνε αρκετά την εμπειρία των χρηστών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Simon History. [Online]. Available: https://simoneer.github.io/history/

[2] Apple's App Store launches with more then 500 apps. [Online]. Available: https://appleinsider.com/articles/08/07/10/apples_app_store_launches_with_more_than_500_apps

[3] The IoC container, Spring Framework Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html

[4] Aspect Oriented Programming with Spring, Spring Framework Reference. [Online]. Available: https://docs.spring.io/spring-framework/reference/core/aop.html

[5] Introduction to Spring AOP, baeldung.com. [Online]. Available: https://www.baeldung.com/spring-aop

[6] Part IV. Data Access, Spring Framework Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/spring-data-tier.html

[7] DAO Supoprt, Spring Framework Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/dao.html

[8] Web MVC framework, Spring Framework Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[9] Spring Boot Overview. [Online]. Available: https://spring.io/projects/spring-boot

[10] What is a Document Database?, MongoDB. [Online]. Available: https://www.mongodb.com/document-databases

[11] MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/

[12] MongoDB Cloud Services. [Online]. Available: https://www.mongodb.com/products/platform/cloud

[13] MongoDB Realm Documentation. [Online]. Available: https://www.mongodb.com/docs/realm/

[14] Chapter 1. What is React Native, oriely.com. [Online]. Available: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[15] Core Components and Native Components, React Native Documentation. [Online]. Available: https://reactnative.dev/docs/intro-react-native-components

[16] React Fundamentals, React Native Documentation. [Online]. Available: https://reactnative.dev/docs/intro-react#your-first-component

[17] Writing Markup with JSX, React Documentation. [Online]. Available: https://react.dev/learn/writing-markup-with-jsx

[18] Metro, React Native Documentation. [Online]. Available: https://reactnative.dev/docs/metro

[19] Metro Documentation, facebook.github.io. [Online]. Available: https://facebook.github.io/metro/docs/getting-started

[20] Debugging Basics, React Native Documentation. [Online]. Available: https://reactnative.dev/docs/debugging

[21] Flipper Documentation. [Online]. Available: https://fbflipper.com/docs/getting-started/

[22] React Native Support, Flipper Documentation. [Online]. Available: https://fbflipper.com/docs/features/react-native/

[23] Performance Overview, React Native Documentation. [Online]. Available: https://reactnative.dev/docs/performance

[24] Frame Rate (iOS and tvOS), Apple Developer archive documentation. [Online]. Available: https://developer.apple.com/library/archive/documentation/3DDrawing/Conceptual/MTLBestPracticesGuide/FrameRate.html

[25] React Native: Understanding Threads, Medium.com. [Online]. Available: https://brooklinmyers.medium.com/react-native-understanding-threads-e026c7d62bb2

[26] TypeScript for the New Programmer. [Online]. Available: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[27] Firebase Documentation. [Online]. Available: https://firebase.google.com/docs

[28] Understanding Spring Boot @Autowired Annotation. [Online]. Available: https://gustavopeiretti.com/spring-boot-autowired-annotation/

[29] Guide to Spring @Autowired. [Online]. Available: https://www.baeldung.com/spring-autowire

[30] Cross-Origin Resource Sharing and Why We Need Preflight Requeests, baeldung.com. [Online]. Available: https://www.baeldung.com/cs/cors-preflight-requests

[31] Interface AuthenticationEntryPoint, Spring Security Documentation. [Online]. Available: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/AuthenticationEntryPoint.html

[32] Class OncePreRequestFilter, Spring Security Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html

[33] Tutorial: Get started with Amazon EC2 Linux instances, Amazon AWS Dcoumentation. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

[34] The Podfile, Cocoapods Guides. [Online]. Available: https://guides.cocoapods.org/using/the-podfile.html

[35] Getting Started, Axios Documentation. [Online]. Available: https://axios-http.com/docs/intro

[36] Promise, javascript.info. [Online]. Available: https://javascript.info/promise-basics

[37] Axios vs. fetch(): Which is best for making HTTP requests?

[38] React Reference. [Online]. Available: https://react.dev/reference/react

[39] Using Hooks, React Documentation. [Online]. Available: https://react.dev/learn#using-hooks

[40] State: A Component's Memory, React Documentation. [Online]. Available: https://react.dev/learn/state-a-components-memory

[41] Extracting State Logic into a Reducer, React Documentation. [Online]. Available: https://react.dev/learn/extracting-state-logic-into-a-reducer#

[42] React Native MMKV Github page. [Online]. Available: https://github.com/mrousavy/react-native-mmkv

[43] Redux Overview and Concepts, Redux Tutorials. [Online]. Available: https://redux.js.org/tutorials/essentials/part-1-overview-concepts#why-should-i-use-redux

[44] Creating Slices of State, Redux Toolkit User Guide. [Online]. Available: https://redux-toolkit.js.org/usage/usage-guide#creating-slices-of-state

[45] Passing Data Deeply with Context, React Documentation. [Online]. Available: https://react.dev/learn/passing-data-deeply-with-context

[46] Apple Developer Documentation. [Online]. Available: https://developer.apple.com/documentation

[47] Foreground Location, Google Developer Documentation. [Online]. Available: https://developer.android.com/training/location/permissions#foreground

[48] Navigators, React Navigation. [Online]. Available: https://reactnavigation.org/docs

[49] ReactJS Functional Components, geeksforgeeks.org. [Online]. Available: https://www.geeksforgeeks.org/reactjs-functional-components/

[50] Native Base Documentation. [Online]. Available: https://docs.nativebase.io/3.1.x/action-sheet

[51] React Native Bottom Sheet. [Online]. Available: https://gorhom.github.io/react-native-bottom-sheet/

[52] React Native Fast Image Github. [Online]. Available: https://github.com/DylanVann/react-native-fast-image

[53] Use API Keys, Google Developer Documentation. [Online]. Available: https://developers.google.com/maps/documentation/javascript/get-api-key

[54] Registering a key, iOS Messaging Setup, React Native Firebase. [Online]. Available: https://rnfirebase.io/messaging/usage/ios-setup#1-registering-a-key

[55] Adding TypeScript, Create React App Documentation. [Online]. Available: https://create-react-app.dev/docs/adding-typescript/