Ελληνικό Μεσογειακό Πανεπιστήμιο

Σχολή Μηχανικών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πτυχιακή

Cloud Based Σύστημα Προσφοράς και Πώλησης Μεταχειρισμένων Ρούχων

Παρασκευόπουλος Μιχαήλ (τπ4710)

Επιβλέπων εκπαιδευτικός : Αϊβαλής Κώστας

Ευχαριστίες

Η παρούσα διπλωματική εργασία, με την οποία ολοκληρώνεται η ακαδημαϊκή μου πορεία στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ελληνικού Μεσογειακού Πανεπιστημίου. Δεν θα μπορούσε να είναι εφικτή χωρίς τη στήριξη κάποιων ανθρώπων.

Θα ήθελα να ευχαριστώ την οικογένεια μου που χωρίς την στήριξη τους όλα αυτά τα χρόνια δεν θα μπορούσα να ολοκληρώσω τις σπουδές μου και να φτάσω στο σημείο να κάνω την διπλωματική μου εργασία.

Εξίσου σημαντικό ρόλο έπαιξε και ο καθηγητής του ΕΛΜΕΠΑ Κωστή Αϊβαλή οπότε θα ήθελα να του πω ένα ευχαριστώ για την ευκαιρία που έδωσε να κάνω την διπλωματική μου εργασία.

Abstract

This application was implemented to facilitate the sale and purchase of used clothes. More specifically, developing an application that specializes in searching and publishing posts about clothing makes it easier for its users to take advantage of clothes they no longer use and for others to find what they are looking for cheaper. Another advantage is that with the model applied, the sale is made between users without involving third parties with companies with the aim of profit. An application for mobile devices compatible with the most popular platforms. The main features of the application are the understandable and impressive design and the search algorithm of the posts based on the current location of the user or whatever location he wants.

The name of the application is Thrift, and it is made using React Native. For the database has been used by Firebase (Realtime, Firestore, Storage, Authentication) a NoSQL database which is used for authentication of users and for the storage of their data (Realtime, Authentication), for the storage of post data (Firestore) and for the storage of product and user photos (Storage). For the UX of the application, the Native Base library was used, which has a plethora of prebuild components for the development, mainly for the appearance of mobile applications.

Σύνοψη

Η συγκεκριμένη πτυχιακή εργασία υλοποιήθηκε με σκοπό να διευκολύνει την πώληση και αγορά μεταχειρισμένων ρούχων. Πιο συγκεκριμένα ,αναπτύσσοντας μια εφαρμογή ειδικευμένη στην αναζήτηση και δημοσιοποίηση αγγελιών με είδη ρουχισμού γίνεται πιο εύκολο για τους χρήστες της να εκμεταλλευτούν ρούχα τα οποία δεν χρησιμοποιούν πια και για άλλους να βρουν αυτό που ψάχνουν οικονομικότερα. Ένα ακόμη πλεονέκτημα είναι της είναι ότι με το μοντέλο που εφαρμόζεται η πώληση γίνεται μεταξύ χρηστών χωρίς να εμπλέκονται τρίτοι παράγοντες με επιχειρήσεις έχοντας σκοπό το κέρδος. Μια εφαρμογή για φορητές συσκευές συμβατή με τις δημοφιλέστερες πλατφόρμες. Κύρια χαρακτηριστικά της εφαρμογής είναι το κατανοητό και εντυπωσιακό design και ο αλγόριθμος αναζήτησης των αγγελιών με βάση την τωρινή τοποθεσία του χρήστη ή και οποία άλλη επιθυμεί αυτός.

Το όνομα της εφαρμογής είναι Thrift και έχει φτιαχτεί με την χρήση της React Native. Για την βάση δεδομένων έχει χρησιμοποιεί η Firebase (Realtime,Firestore,Storage,Authentication) μια NoSQL βάση η οποία χρησιμοποιείται για την αυθεντικοποίηση των χρηστών και για την αποθήκευση των δεδομένων τους (Realtime , Authentication), για την αποθήκευση των δεδομένων των αναρτήσεων (Firestore) και για την αποθήκευση των φωτογραφείων των προϊόντων και χρηστών (Storage). Για το UX της εφαρμογής χρησιμοποιήθηκε η βιβλιοθήκη Native Base η οποία έχει πληθώρα προετοιμασμένων συστατικών (prebuild components) για την ανάπτυξη κυρίως της εμφάνισης mobile εφαρμογών.

Πίνακας Περιεχομένων

Ευρετήριο

Screens

Authentication flow

- Login Screen
- Signup Screen

Main flow

- Home Screen
  - Search Screen
  - Results Screen
  - Post Detail
- Post Screen
  - Post Screen1
  - Post Screen2
  - Post Screen3
  - Post Screen4
  - Post Screen5
  - Post Screen6
- Saved List Screen
- Profile Screen
  - My posts list screen
  - Edit Screen

Components

- Bottom tab navigation
- Stack navigation
- Search Stack
- Post Stack
- Profile Stack
- Saved Stack
- App Context
- Helper
- Loadings

1

1.1 Περίληψη

Με κάθε έτος που περνάει τα κινητά τηλεφωνά γίνονται ολοένα και πιο απαραίτητα για την ευκολότερη εκπλήρωση απλών αλλά και σύνθετων στόχων στην καθημερινότητα όλων μας. Αυτό έχει ως αποτέλεσμα να υπάρχουν πολλά ήδη εφαρμογών που καλύπτουν τις επιθυμίες των χρηστών. Μια από αυτές τις επιθυμίες είναι η αγορά και πώληση μεταχειρισμένων ρούχων εύκολα γρηγορά και χωρίς την ανάμιξη τρίτων εταιριών με σκοπό το κέρδος δηλαδή το προϊόν να πηγαίνει από έναν ιδιώτη σε έναν άλλο. Υπάρχουν πολλές εφαρμογές για πώληση προϊόντων αλλά λιγότερες που ειδικεύονται και βασίζονται στον ενδυματολογικό τομέα.

Τα επικρατέστερα λειτουργικά συστήματα στων χώρο των κινητών είναι το Android και IOS τα οποία μέχρι και κάποια χρόνια πριν για αναπτυχθούν εφαρμογές και για τα δυο θα έπρεπε να υλοποιούν δυο πηγαίοι κώδικες για το κάθε σύστημα. Η ανάπτυξη κάποιας εφαρμογής και στα δυο συστήματα απαιτεί διάφορες γλώσσες προγραμματισμού. Όπως για παράδειγμα για μια εφαρμογή IOS χρησιμοποιούνται κυρίως οι Swift και Objective C ενώ για Android οι JAVA και Kotlin. Επίσης σημαντικό ρόλο παίζουν τα περιβάλλοντα (IDE) που μπορούν να αναπτυχθούν οι εφαρμογές καθώς η IOS εφαρμογές χρειάζονται το X-code ενώ οι Android το Android Studio. Ακόμη το κάθε λειτουργικό χρησιμοποιεί διαφορετική SDK βιβλιοθήκη και διαφορετικά API για την υλοποίηση των ίδιων λειτουργειών. Επομένως για την ανάπτυξη μιας εφαρμογής και στα δυο συστήματα είναι εξαιρετικά χρονοβόρα , κοστίζει περισσότερο και έχει ανάγκη από περισσότερους προγραμματιστές με άριστες γνώσεις πάνω σε όλα τα κομμάτια που απαιτούνται.

Λύση σε αυτό το πρόβλημα δίνει ένα open source Framework της JavaSript η React Native σχεδιασμένη κυρίως για την ανάπτυξη εφαρμογών για κινητές συσκευές με την χρήση ενός πηγαίου κώδικα (single codebase) για την ανάπτυξη εφαρμογών σε πολλαπλά συστήματα IOS , Android , Windows ,Web (crossplatform). Ακόμη για την παραγωγή του κώδικα μέσω της React Native μπορούν να χρησιμοποιούν native development αλλά και JavaScript βιβλιοθήκες καθώς και ένα οποιοδήποτε IDE που υποστηρίζει την γλώσσα. Τέλος αν συνδυαστεί με την βιβλιοθήκη React Native Expo τα SDKs για την εκάστοτε πλατφόρμα διαχειρίζονται αυτόματα.

1.2 Κίνητρο για την Διεξαγωγή της Εργασίας

Η δημιουργία μιας εφαρμογής για πώληση μεταχειρισμένων ρούχων έχει ως στόχο της την ευκολότερη πώληση ενδυμάτων που οι ιδιοκτήτες τους δεν χρησιμοποιούν πια και την αγορά τους από άλλους χρήστες της εφαρμογής.

Υπάρχουν πολλές εφαρμογές τύπου e-shop αλλά όχι αρκετές για μεταχειρισμένα ρούχα. Ακόμη λιγότερες είναι αυτές που χρησιμοποιούν τις δυνατότητες των χαρτών (google maps) αποτελεσματικά έτσι ώστε να μειώσουν τις αποστάσεις μεταξύ των χρηστών και να τους δίνουν την επιλογή να αναζητούν τις προτιμήσεις τους σε όποιο εύρος χιλιομέτρων επιθυμούν αλλά και σε όποια περιοχή ακόμη και αν αυτή δεν είναι η τωρινή του. Περαιτέρω στην εφαρμογή έχουν προστεθεί αρκετά γραφικά έτσι ώστε να είναι ευχάριστη και πλήρως κατανοητή στον χρήστη.

Ως αποτέλεσμα δημιουργήθηκε αυτή η εφαρμογή που μπορεί να αξιοποιηθεί από τον οποιοδήποτε χωρίς καμία χρέωση και σε οποιαδήποτε πλατφόρμα .

2 Μεθοδολογία

Thrift

Η αρχή του Thrift έγινε σχεδιάζοντας εμφανισιακά την εφαρμογή σε ένα γραφικό περιβάλλον (AdobeXd). Στη συνέχεια επιλέχθηκε η γλώσσα προγραμματισμού React Native η βάση δεδομένων Firebase και η βιβλιοθήκη για την εμφάνιση της εφαρμογής Native Base σύμφωνα με τα δεδομένα της εφαρμογής. Στη συνέχεια έγινε αναζήτηση βιβλιοθήκης για τους χάρτες που χρησιμοποιήθηκαν και τελικά επιλέχθηκε η react-native-maps. Έπειτα καταγράφηκαν αναλυτικά σε μια πλατφόρμα διαχείρισης έργου (Asana) τα βήματα που έπρεπε να ακολουθηθούν σε κάθε στάδιο της εφαρμογής και το χρόνο που έπρεπε να διαρκέσουν. Τέλος δημιουργήθηκε Repository στο GitHub και συνδέθηκε με την εφαρμογή στο VS-Code για το versioning του κώδικα.

Πρώτα αναπτύχθηκε το μοντέλο δρομολόγησης των οθονών βασικός δρομολογητής που χρησιμοποιήθηκε είναι ο Bottom Tab Navigation που χώρισε την εφαρμογή σε καρτέλες. Κάθε καρτέλα περιέχει μέσα ένα δρομολογητή τύπου Stack Navigation με τις οθόνες. Στη συνέχεια αναπτύχθηκε η ροή της σύνδεσης και εγγραφής και στη συνέχεια η υπόλοιπες οθόνες. Η βασική ροή ξεκίνησε με τις οθόνες που δημιουργούν τα δεδομένα στη βάση δηλαδή ότι έχει να κάνει με την ανάρτηση της κάθε αγγελίας μετά η μηχανή αναζήτησης έπειτα ο τρόπος παρουσίασης των αναρτήσεων σε κατηγορίες όπως αποτελέσματα αναζήτησης αποθηκευμένα και πρόσφατες επισκέψεις και τέλος το προφίλ του συνδεμένου χρήστη με λειτουργίες όπως αποσύνδεση , αλλαγή σε σκούρο θέμα και επεξεργασία των αναρτήσεων. Κάθε οθόνη της οποίας υλοποιεί το πρότυπο model-view-controller πρώτα υλοποιούταν το view δηλαδή το εμφανισιακό κομμάτι της οθόνης με ενδεικτικά δεδομένα μετά ο controller δηλαδή η JavaScript που κάνει την διαχείριση των πόρων της εφαρμογής. Το model στις cloud-based-database σχηματίζεται στο διαδίκτυο σύμφωνα με τα δεδομένα που ανεβάζει ο controller.

Στο τέλος έγινε η εξαγωγή της εφαρμογής σε μορφή APK για Android και IOS. Ακολούθησε μια σειρά δοκιμών με πραγματικά δεδομένα ώστε να γίνει η επιβεβαίωση ότι όλα λειτουργούν όπως ήταν αναμενόμενα. Ακόμη στο τέλος χρησιμοποιήθηκε το ενσωματωμένο Mock Location Engine του Android studio για να γίνει περεταίρω έλεγχος στην χρήση των χαρτών και των αποστάσεων σε όλη της εφαρμογής.

Android

Το Android είναι λειτουργικό σύστημα για κινητής συσκευές το οποίο τρέχει τον πυρήνα του λειτουργικού Linux. Αρχικά αναπτύχθηκε από την Google και αργότερα από την Open Handset Alliance. Επιτρέπει στους κατασκευαστές λογισμικού να συνθέτουν κώδικα με την χρήση της γλώσσας προγραμματισμού Java, ελέγχοντας την συσκευή μέσω βιβλιοθηκών λογισμικού ανεπτυγμένων από την Google. Το Android είναι κατά κύριο λόγο σχεδιασμένο για συσκευές με οθόνη αφής, όπως τα έξυπνα τηλέφωνα και τα τάμπλετ, με διαφορετικό περιβάλλον χρήσης για τηλεοράσεις (Android TV), αυτοκίνητα (Android Auto) και ρολόγια χειρός (Android Wear). Παρόλο που έχει αναπτυχθεί για συσκευές με οθόνη αφής, έχει χρησιμοποιηθεί σε κονσόλες παιχνιδιών, ψηφιακές φωτογραφικές μηχανές, συνηθισμένους Η/Υ και σε άλλες ηλεκτρονικές συσκευές.

iOS

Το iOS είναι ένα λειτουργικό σύστημα για smartphones που δημιουργήθηκε από την Apple αποκλειστικά και μόνο για το υλικό της. Είναι λειτουργικό σύστημα που βρίσκεται εγκατεστημένο σε πολλές από τις κινητές συσκευές της Apple, συμπεριλαμβανομένων των iPod Touch και iPhone. Το συγκεκριμένο λειτουργικό σύστημα βρισκόταν εγκατεστημένο και σε συσκευές τύπου iPad έως ότου το iPadOS παρουσιάστηκε με την 13η έκδοση το 2019. Μετά το Android, είναι το δεύτερο πιο ευρέως εγκατεστημένο λειτουργικό σύστημα στον κόσμο. Τρία άλλα λειτουργικά συστήματα που κατασκευάζονται από την Apple βασίζονται σε αυτό: tvOS, watchOS και iPadOS. Είναι λογισμικό ιδιόκτητο, αν και έχει ορισμένα μέρη ανοιχτού κώδικα βάσει της άδειας χρήσης δημόσιου κώδικα Apple και άλλων αδειών χρήσης.

Android Studio

Το Android Studio είναι ένα ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE) για ανάπτυξη εφαρμογών στην πλατφόρμα Android. Ανακοινώθηκε στις 16 Μαΐου 2013 είναι διαθέσιμο ελεύθερα με την άδεια Apache License. Βασισμένο στο λογισμικό της JetBrains' IntelliJ IDEA, το Android Studio σχεδιάστηκε αποκλειστικά για προγραμματισμό Android. Είναι διαθέσιμο για Windows, Mac OS X και Linux, και αντικατέστησε τα Eclipse Android Development Tools (ADT) ως το κύριο IDE της Google για ανάπτυξη εφαρμογών Android.

Μέσω του Android Studio έγινε το testing της εφαρμογής στον build in emulator που παρέχει για Android συσκευές. Επίσης χρησιμοποιήθηκε το GPS Mock location που δίνεται έτσι ώστε να ελεγχθούν οι διαδικασίες που χρησιμοποιούν την τοποθεσία και τους χάρτες. Τέλος χρησιμοποιήθηκε το Android Studio για την εξαγωγή σε μορφή apk για Android συσκευές.

Xcode

Το Xcode είναι μέρος των εργαλείων προγραμματιστών της Apple, τα οποία μπορούν να ληφθούν με το iPhone SDK από την πύλη προγράμματος προγραμματιστών iPhone. Οι προγραμματιστές που είναι εγγεγραμμένοι μπορούν να κάνουν λήψη εκδόσεων προεπισκόπησης και προηγούμενων versions της σουίτας μέσω του ισότοπου του Apple Developer, μέσω του επίσημου web-site της Apple. Το Xcode περιλαμβάνει Command Line Tools (CLT), τα οποία επιτρέπουν την ανάπτυξη τύπου UNIX μέσω της εφαρμογής Terminal στο macOS. Μπορούν επίσης να ληφθούν και να εγκατασταθούν χωρίς το GUI.

Το x-code χρησιμοποιήθηκε μόνο για την εξαγωγή της εφαρμογής σε apk για IOS συσκευές. Το testing για την λειτουργικότητα σε IOS έγινε από φυσική συσκευή καθώς η εφαρμογή αναπτύχθηκε από υπολογιστή με λογισμικό Windows στο όποιο το x-code δίστιχος δεν είναι διαθέσιμο.

Cloud based server

Το Cloud-based είναι ένας όρος που αναφέρεται σε εφαρμογές, υπηρεσίες ή πόρους που διατίθενται στους χρήστες κατόπιν ζήτησης μέσω Διαδικτύου από διακομιστές παρόχου υπολογιστικού νέφους. Οι εταιρείες συνήθως χρησιμοποιούν υπολογιστές που βασίζονται σε σύννεφο ως έναν τρόπο για να αυξήσουν τη χωρητικότητα, να βελτιώσουν τη λειτουργικότητα ή να προσθέσουν πρόσθετες υπηρεσίες κατόπιν ζήτησης χωρίς να χρειάζεται να δεσμευτούν σε δυνητικά ακριβό κόστος υποδομής ή να αυξήσουν/εκπαιδεύσουν το υπάρχον προσωπικό υποστήριξης.

Firebase

Στην πληροφορική, μια βάση δεδομένων είναι μια οργανωμένη συλλογή δεδομένων που αποθηκεύονται και αποκτώνται ηλεκτρονικά από ένα σύστημα υπολογιστή. Όπου οι βάσεις δεδομένων είναι πιο περίπλοκες, συχνά αναπτύσσονται χρησιμοποιώντας επίσημες τεχνικές σχεδιασμού και μοντελοποίησης.

Η Firebase είναι μια NoSQL βάση δεδομένων που φτιάχτηκε από την την google είναι μια cloud based βάση υπηρεσία δηλαδή τρέχει στο διαδίκτυο σε κάποιον απομακρυσμένο server της google και όχι σε τοπικό server όπως οι συνηθισμένες βάσεις. Παρέχει στους προγραμματιστές μια ποικιλία εργαλείων και υπηρεσιών για να τους βοηθήσει να αναπτύξουν ποιοτικές εφαρμογές. Η Firebase χωρίζει τις υπηρεσίες τις σε κατηγορίες , για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν σχεδόν όλες οι υπηρεσίες της βάσης.

Realtime Database

Η Realtime database είναι μια υπηρεσία της Firebase η οποία είναι ο πιο γρήγορος τρόπος αποθήκευσης , ανάκτησης και ενημέρωσης δεδομένων σε πραγματικό χρόνο. Τα δεδομένα αποθηκεύονται σε μορφή JSON και είναι διαθέσιμα ακόμα και εκτός σύνδεσης. Το ελάττωμα αυτής της υπηρεσίας είναι ότι δεν είναι εύκολο να γίνει ανάκτηση των δεδομένων με φίλτρα. Αυτή η υπηρεσία χρησιμοποιήθηκε για τα προσωπικά δεδομένα των χρηστών δηλαδή για τις λίστες τους , την λίστα με τις αγγελίες που έχουν ανεβάσει , την λίστα με τις πρόσφατες αγγελίες που είδανε και την λίστα με τις αγγελίες που μάρκαραν σαν αγαπημένα.

Firestore Database

Η firestore είναι η λύση στο ελάττωμα της realtime. Δηλαδή υπάρχει η δυνατότητα ανάκτησης δεδομένων βάση κάποιων φίλτρων που δίνονται από τον χρήστη. Αυτό έχει ως αποτέλεσμα να είναι πιο αργή από την realtime. Χρησιμοποιείται κυρίως για μεγάλο όγκο δεδομένων που δεν χρειάζεται να ανακτηθούν πολλές φορές και γρήγορα. Τα δεδομένα της fierstore έχουν την μορφή φακέλων που συνδέονται μεταξύ τους (Collection>Document). Αυτή η υπηρεσία χρησιμοποιήθηκε για την αποθήκευση των αγγελιών και για την ανάκτηση τους με την χρήση φίλτρων και για την επεξεργασία τους.

Authentication

Το Authentication είναι μια υπηρεσία της Firebase η οποία προσφέρει ένα έτοιμο σύστημα back-end αυθεντικοποίησης ενός χρήστη. Δηλαδή από την στιγμή που χρησιμοποιηθεί, όλες οι λειτουργίες τύπου έγκυρη φόρμα ηλεκτρονικής διεύθυνσης η έγκυρος κωδικός έχουν ληφθεί υπόψη και δεν χρειάζεται να γίνουν επιπλέον ενέργειες πάρα μόνο η υλοποίηση του front-end. Ακόμη το Authentication προσφέρει εναλλακτικούς τρόπους σύνδεσης όπως Facebook, Google και AppleID. Χρησιμοποιήθηκε αυτή η υπηρεσία για την εφαρμογή για την σύνδεση και εγγραφή των χρηστών με τον τρόπο σύνδεσης Email & Password.

Storage

Η storage είναι μια υπηρεσία στην οποία ο χρήστης της βάσης μπορεί να αποθηκεύσει πολυμέσα δηλαδή εικόνες και βίντεο και έπειτα να τα χρησιμοποιεί οπού θέλει στην εφαρμογή.

JavaScript

Η JavaScript (JS) είναι γλώσσα προγραμματισμού για Η/Υ. Η JavaScript αποτελεί μέρος της υλοποίησης των browsers, ώστε τα client-side scripts να μην μπορούν να επικοινωνήσουν με τον χρήστη, τα δεδομένα να αλλάζουν ασύγχρονα και το περιεχόμενο του εγγράφου που εμφανίζεται να αλλάζει δυναμικά.

Η JavaScript είναι μια δυναμική prototype-based γλώσσα σεναρίων, με ασθενείς τύπους και συναρτήσεις ως αντικείμενα πρώτης τάξης. Η C έχει επηρεάσει τη σύνταξη της JavaScript. Η Java δανείζει πολλά ονόματα και συμβάσεις στην JavaScript, αλλά έχουν πολύ διαφορετική σημασιολογία και δεν σχετίζονται μεταξύ τους. Από τις γλώσσες προγραμματισμού Self και Scheme προέρχονται οι βασικές αρχές σχεδιασμού της JavaScript. Είναι multi-paradigm γλώσσα, υποστηρίζοντας αντικειμενοστραφές, συναρτησιακό και προστακτικό στυλ προγραμματισμού.

Το JavaScript χρησιμοποιείται και σε web applications υπάρχουν παραδείγματα όπως τα PDF, οι εξειδικευμένοι browsers και τα desktop widgets. Οι καινούργιες εικονικές μηχανές και τα πλαίσια ανάπτυξης για JavaScript (όπως το Node.js) έχουν επίσης τη JavaScript πιο δημοφιλή για την ανάπτυξη εφαρμογών Ιστού στην πλευρά του διακομιστή (server-side). [4]

React Native

Το React Native είναι μια Javascript βιβλιοθήκη για τη δημιουργία UI. Με ανοιχτό κώδικα από το Facebook το 2013, δημιουργήθηκε από μια ευρεία κοινότητα προγραμματιστών. Οι χρήστες επαινούν το React Native για την απόδοση και την ευελιξία του. Όπως θα περίμενε κανείς, το React Native σχεδιάστηκε για τις ανάγκες της ανάπτυξης του Facebook και, ως εκ τούτου, ταιριάζει ιδιαίτερα σε πολύπλοκες διαδικτυακές εφαρμογές που ασχολούνται σε μεγάλο βαθμό με την αλληλεπίδραση των χρηστών και την ανταλλαγή δεδομένων.

Το React Native χρησιμοποιεί το React για να στοχεύσει πλατφόρμες εκτός του προγράμματος περιήγησης, όπως το iOS και το Android, υλοποιώντας μια «γέφυρα» μεταξύ Javascript και πλατφόρμας host.

Expo

Το Expo είναι μια εργαλειοθήκη που δημιουργήθηκαν γύρω από το React Native η οποία βοηθάει στο να ξεκινήσει μια εφαρμογή πολύ γρήγορα. Παρέχει μια λίστα εργαλείων που απλοποιούν τη δημιουργία και τη δοκιμή της οποιαδήποτε εφαρμογής. Εξοπλίζει τον προγραμματιστή με τα στοιχεία της διεσπάρης χρήστη και των υπηρεσιών. Συνήθως, είναι διαθέσιμα σε στοιχεία React Native τρίτων.

Σχέδιο Δράσης για την εκπόνηση της Πτυχιακής Εργασίας State of the Art

Για την δημιουργία του Thrift χρησιμοποιήθηκαν εργαλεία που χρησιμοποιούνται κυρίως για την ανάπτυξη mobile εφαρμογών React Native , Expo , Firebase , Native Base. Η εφαρμογή χτίστηκε έτσι ώστε να είναι κατανοητή , εμφανίσιμη και γρήγορη παρόλο τον μεγάλο όγκο των δεδομένων.

Η εφαρμογή έχει δομηθεί έτσι ώστε να είναι όλα μπροστά στον χρήστη σε απόσταση δυο η τριών πατημάτων. Για αυτόν τον σκοπό χρησιμοποιήθηκε ο Navigator της React Native Bottom-Tab-Navigation σε συνδυασμό με τον Stack Navigator. Έτσι η εφαρμογή χωρίστηκε σε τομείς (Αναζήτηση , Αγαπημένα , Ανάρτηση , Προφίλ χρήστη) που έπειτα μπήκαν στις καρτέλες.

Ένα από τα ξεχωριστά χαρακτηριστικά του Thrift είναι ότι χρησιμοποιεί με μοναδικό τρόπο τις λειτουργίες τις βιβλιοθήκης react native maps. Καθώς επιτρέπει στον χρήστη να τοποθέτηση την τοποθεσία του ή μια άλλη κάνοντας επαφή στο σημείο του χάρτη που επιθυμεί. Επίσης του επιτρέπετε να μεγαλώσει και να μικρύνει την εμβέλεια που θα ψάχνει ή θα δημοσιεύει αγγελίες , με την χρήση ενός slider που αλλάζει την χιλιομετρική απόσταση από το σημείο που διάλεξε ο χρήστης. Ακόμη όταν ένας χρήστης βρίσκει αυτό που θέλει οι χάρτες του δείχνουν την εμβέλεια που ψάχνει αλλά και την εμβέλεια που έχει δημοσιευτεί η αγγελία με την χρήση κύκλων πάνω στους χάρτες που δείχνουν την απόκλιση των ακτινών αγοραστή και πωλητή. Στο κομμάτι των φίλτρων , για την απόσταση μεταξύ πωλητή και αγοραστή χρησιμοποιήθηκε ένας απλός αλγόριθμος υπολογισμού της ευθείας απόστασης μεταξύ των συντεταγμένων δυο σημείων.

Τέλος η εφαρμογή έχει χτιστεί έτσι ώστε να αποφεύγεται η χρήση επαναλαμβανομένου κώδικα με την χρήση βοηθητικών συναρτήσεων η και βοηθητικών γραφικών στοιχείων, Το UI χτίστηκε με την βιβλιοθήκη Native Base η οποία δίνει έτοιμα στοιχεία που μπορούν να χρησιμοποιηθούν για της ανάγκες της εφαρμογής, ελάχιστες φορές χρειάστηκε να χρησιμοποιηθούν γραφικά στοιχεία που φτιάχτηκαν μόνο για αυτή την εφαρμογή. Ακόμη μια προσθήκη στο καλύτερο user experience είναι το Dark Mode που μπορεί να επιλεγεί από τον χρήστη το οποίο υλοποιήθηκε με τον Theme Provider της Native Base.

3.2 Σημαντικοί στόχοι για την ολοκλήρωση της Πτυχιακής Εργασίας

Πρωταρχικός στόχος ήταν η υλοποίηση μιας εφαρμογής για την πώληση και αγορά μεταχειρισμένων ρούχων ευκολότερα από ποτέ. Το συγκεκριμένο σύστημα θα έπρεπε να είναι πλήρως κατανοητό και εύκολο στην χρήση από οποιονδήποτε χρήστη. Επίσης θα έπρεπε να είναι γρήγορο και να έχει αναλυτικές πληροφορίες για τα ρούχα με φωτογραφίες χωρίς να απαιτεί από τον χρήστη πολλά πατήματα. Έτσι δημιουργήθηκε η ανάγκη να φτιαχτούν λίστες που δίνουν πληροφορίες στον χρήστη χωρίς αυτός να χρειαστεί να κάνει πάτημα σε κάθε προϊόν που εμφανιζόταν.

Ένας σημαντικός προβληματισμός πριν από όλα ήταν το πως θα δοθεί στον χρήστη η πληροφορία για την απόσταση του από κάποιον άλλο χρήστη έτσι χρησιμοποιήθηκαν οι χάρτες με γραφικά στοιχεία μέσα έτσι ώστε να κάνουν κατανοητό στον χρήστη τι συμβαίνει.

Ακόμη στόχος ήταν η εύκολη αναζήτηση και ανάρτηση για αυτό και χρησιμοποιήθηκαν στοιχεία που επέτρεπαν στον χρήστη να επιλέξουν τις λεπτομερείς των ρούχων από λίστες επιλογών μειώνοντας έτσι την περίπτωση λάθους επιπλέων λήφθηκαν υπόψη οι πιθανότητες λάθους των χρηστών , για αυτό και δημιουργήθηκε σύστημα ελέγχου των εισαγωγών των χρηστών πριν την καταχώριση τους στο σύστημα. Αυτό είχε ως αποτέλεσμα το να είναι αδύνατο να δημοσιευθεί μια αγγελία η να γίνει μια αναζήτηση χωρίς ο χρήστης να έχει καταχώρηση τις απαραίτητες και σωστές πληροφορίες.

4 Κύριο μέρος Πτυχιακής

4.1 Ανάλυση Προβλήματος

Πριν την άνθηση της τεχνολογίας των smartphones υπήρχαν δύο τρόποι για κάποιον να πουλήσει η να αγοράσει ένα μεταχειρισμένο ρούχο. Ο πρώτος ήταν να βρει κάποιον γνωστό του που να ενδιαφέρεται και να γίνει συναλλαγή και ο δεύτερος ήταν να πάει σε ένα φυσικό κατάστημα που αγοράζει και πουλάει μεταχειρισμένα είδη ρουχισμού. Η πρώτη περίπτωση είχε μικρή πιθανότητα να συμβεί ενώ στην δεύτερη η τιμή για ένα μεταχειρισμένο ρούχο δεν ήταν και πολύ χαμηλότερη από ένα καινούριο αφού έμπαινε στην μέση της συναλλαγής το κατάστημα με σκοπό το κέρδος.

Πλέον υπάρχουν πολλές πλατφόρμες τύπου e-shop για την πώληση ρούχων αλλά οι περισσότερες είναι αποκλειστικά για καινούρια ρούχα και απευθύνονται σε μεγάλους ομίλους. Ακόμη υπάρχουν εφαρμογές που είναι αποκλειστικά για μεταχειρισμένα ρούχα αλλά συνηθώς από πίσω βρίσκετε ένας τρίτος παροχής που μόνο αυτός μπορεί να δημοσιεύει αναρτήσεις το οποίο είναι σχεδόν το ίδιο με την περίπτωση του φυσικού καταστήματος που αναφέραμε πριν.

4.1.2 Απαιτήσεις

Smartphone

Η σημαντικότερη απαίτηση για την λειτουργία της εφαρμογής από θέμα υλικού είναι το smartphone δηλαδή μια φορητή συσκευή που συνδυάζει το κινητό τηλέφωνο και τις λειτουργίες υπολογιστών σε μια μονάδα. Διακρίνονται από τα τηλέφωνα χαρακτηριστικών από τα εκτεταμένα λειτουργικά συστήματα για κινητά και τις ισχυρότερες δυνατότητες υλικού, τα οποία διευκολύνουν το ευρύτερο λογισμικό, το Διαδίκτυο και τη λειτουργικότητα πολυμέσων (συμπεριλαμβανομένης μουσικής, βίντεο, καμερών και παιχνιδιών), παράλληλα με το βασικό τηλέφωνο, υπάρχουν λειτουργίες όπως κλήσεις και SMS. Η εφαρμογή επίσης μπορεί να λειτουργήσει κανονικά σε ταμπλέτα με παροιμία χαρακτηριστικά.

Πιο συγκεκριμένα για την καλύτερη λειτουργία της εφαρμογής είναι απαραίτητο είναι ένα smartphone με σύνδεση στο διαδίκτυο , την εφαρμογή εγκατεστημένη , και ενεργοποιημένη την τοποθεσία. Η τοποθεσία πρέπει να είναι ενεργοποιημένη για να μπορεί ο χρήστης να αναζητά ρούχα στην περιοχή του σε ακτίνα που επιλέγει ο ίδιος και η σύνδεση στο διαδίκτυο πρέπει να υπάρχει καθώς χωρίς αυτήν ο χρήστης δε μπορεί να τραβήξει δεδομένα άλλων χρηστών με αποτέλεσμα η εφαρμογή να μην έχει σκοπό. Προαιρετικά χαρακτηριστικά είναι η κάμερα και η σύνδεση με κάρτα SIM. Η κάμερα χρειάζεται για την ανάρτηση φωτογραφιών ως πωλητής και η άκρατα SIM γιατί η εφαρμογή υποστηρίζει απευθείας κλήσεις σε πωλητές.

4.2 Σχεδιασμός

Ο σχεδιασμός της εφαρμογής ξεκίνησε σχεδιάζοντας τον τρόπο με τον οποίο ο χρήστης θα μετακινητέ από την μια οθόνη στην άλλη για αυτόν τον σκοπό χρησιμοποιήθηκε ένα ενσωματωμένος μηχανισμός της React Native το React Native Navigation πιο συγκεκριμένα χρησιμοποιήθηκαν πέντε Stack Navigation και ένας BottomTabNavigation. Με αυτόν τρόπο η εφαρμογή χωρίστηκε σε διαφορετικά απαιτούμενα.

Έπειτα συνδέθηκε η βάση δεδομένων και σχεδιαστικέ η ροή αυθεντικοποίησης με την οποία ο χρήστης μπορεί να συνδεθεί η να εγγραφεί στην εφαρμογή. Μετά έπρεπε να φτιαχτεί η ροή των αναρτήσεων έτσι ώστε να γεμίσει τη βάση (Firestore) με τα σωστά δομημένα δεδομένα και στη συνέχεια η μηχανή αναζήτησης για την ανάκτηση των δεδομένων με τα φίλτρα που μπορούν να εφαρμοστούν έτσι ώστε ο χρήστης να βλέπει ακριβώς αυτό που θέλει χωρίς περιττές πληροφορίες. Στην ίδια ροή της εφαρμογής φτιάχτηκε η οθόνη με την λίστα αποτελεσμάτων και η οθόνη με τις λεπτομέρειες της κάθε αγγελίας.

Αφού ο βασικός κορμός της εφαρμογής χτίστηκε έπρεπε να μπουν επιπλέον λειτουργίες για να γίνει πιο φιλική προς τον χρήστη. Έτσι φτιάχτηκε η οθόνη με τη λίστα με τις αγαπημένες αναρτήσεις του συνδεδεμένου χρήστη ώστε να μην χρειάζεται να κάνει κάθε φορά την ίδια αναζήτηση με τα ίδια φίλτρα.

Τέλος φτιάχτηκε η οθόνη με το προφίλ του χρήστη , αυτή η οθόνη εκτός από τα στοιχεία του χρήστη περιλαμβάνει την λίστα με τις αναρτήσεις του χρήστη. Για να ολοκληρωθεί η εφαρμογή το μόνο που έλειπε ήταν ένας τρόπος για τους χρήστες να μπορούν να επεξεργαστούν της ήδη δημοσιευμένες αναρτήσεις τους και φτιάχτηκε με μια ξεχωριστή οθόνη που ο χρήστης φτάνει σε αυτή μέσω της λίστας των αναρτήσεων του.

Αφού όλη η λογική και γραφικός σχεδιασμός ήταν έτοιμος για να γίνει η εφαρμογή πιο προσιτή στον χρήστη και πιο εντυπωσιακή φτιάχτηκε εάν επαναχρησιμοποιούμενο αρχείο που είχε μέσα διαφορετικού τύπου φόρτωσης σελίδας όπου εφαρμόστηκαν ανάλογα. Για αυτόν τον σκοπό χρησιμοποιήθηκε το Skeleton Loading από την Native Base όπως και στα σχεδόν όλα τα γραφικά στοιχεία.

4.3 Υλοποίηση

4.3.1 Navigation

1. StackNavigation.js

Το αρχείο StackNavigation.js περιέχει μέσα δυο οθόνες και το Bottom tab navigator. Είναι υπεύθυνο για την προβολή της οθόνης Σύνδεσης (LoginScreen,js) και της οθόνης Εγγραφής (SignUpScreen.js) αλλά ο βασικός του σκοπός είναι να δρομολογεί τον χρήστη από την οθόνη σύνδεσης στην οθόνη εγράφης και ανάποδα και όταν γίνει σύνδεση η εγράφη δρομολογεί τον χρήστη στο Bottom Tab Navigation απενεργοποιώντας το κουμπί 'πίσω' ή χειρονομία (swipe back gesture).

```
<Stack.Navigator >
  <Stack.Screen name="SignUp" component={SignUp}
    options={{
      headerShown: false
    }}
  />
  <Stack.Screen name="Login" component={Login}
    options={{
      headerShown: false
    }}
  />
  <Stack.Screen component={BottomTabNavigation} name="BottomTabNavigation"
    options={{
      title: 'BottomTabNavigation',
      headerShown: false,
      gesturesEnabled: false,
    }}
  />
</Stack.Navigator>
```

2. BottomTabNavigator.js

Το αρχείο BottomTabNavigator.js είναι το επακόλουθο του StackNavigation.js , είναι υπεύθυνο για την δημιουργία των τεσσάρων καρτελών (Tabs) με τους τίτλους και τα σχετικά εικονίδια που αποτελούν την βασική δομή της εφαρμογής. Κάθε μια καρτέλα περιέχει μέσα ένα στοιχείο τύπου Stack.

```
<Tab.Screen name="HomeScreen" component={SearchStack}
  options={{
    headerShown: false,
    tabBarLabel: 'Search',
    tabBarIcon: ({ color, size }) => (
      <Ionicons name="search" size={24} color={color} />
    )
  }} />
<Tab.Screen name="SavedScreen" component={SavedStack}
  options={{
    headerShown: false,
    tabBarLabel: 'Saved',
    tabBarIcon: ({ color, size }) => (
      <Entypo name="heart-outlined" size={30} color={color} />
    )
  }} />
```

```
<Tab.Screen name="PostScreen" component={PostStack}
  options={{
    headerShown: false,
    tabBarLabel: 'Post',
    tabBarIcon: ({ color, size }) => (
      <Ionicons name="add-circle-sharp" size={28} color={color} />
    )
  }} />
<Tab.Screen name="ProfileScreen" component={ProfileStack}
  options={{
    headerShown: false,
    tabBarLabel: 'Profile',
    tabBarIcon: ({ color, size }) => (
      <Ionicons name="md-person" size={24} color={color} />
    )
  }} />
```

a. SearchStack.js
Το αρχείο SearchStack.js περιέχει την βασική οθόνη που βρίσκεται ο χρήστης όταν ανοίγει την εφαρμογή (HomeScreen.js) , την οθόνη αναζήτησης (SearchScreen.js) , την οθόνη αποτελεσμάτων (ResultsScreen.js) και την οθόνη με επιπλέον λεπτομέρειες (PostDetailScreen.js) για κάποιο προϊόν.

```
const Stack = createNativeStackNavigator();
const SearchStack = () => {
  return (
    <Stack.Navigator >
      <Stack.Screen name="Home" component={HomeScreen}
        options={{
          headerShown: false
        }}
      />
      <Stack.Screen name="Search" component={SearchScreen}
        options={{
          headerShown: false
        }}
      />
      <Stack.Screen name="Results" component={ResultsScreen}
        options={{
          headerShown: false
        }}
      />
      <Stack.Screen name="PostDetailScreen" component={PostDetailScreen}
        options={{
          headerShown: false
        }}
      />
    </Stack.Navigator >
  )
}
export default SearchStack;
```

b. SavedStack.js
Το αρχείο SavedStack.js περιέχει μέσα την οθόνη με τις αποθηκευμένες αναρτήσεις του χρήστη (SavedScreen.js) και την οθόνη με επιπλέον λεπτομέρειες (PostDetailScreen.js).

```
<Stack.Navigator >
  <Stack.Screen name="SavedScreen" component={SavedScreen}
    options={{
      headerShown: false
    }}
  />
  <Stack.Screen name="PostDetailScreen" component={PostDetailScreen}
    options={{
      headerShown: false
    }}
  />
</Stack.Navigator >
```

c. PostStack.js
Το αρχείο PostStack.js το οποίο περιλαμβάνει μέσα στην ουσία έναν wizard με βήματα για το ανέβασμα μιας αγγελίας. Ο wizard αποτελείται από τις οθόνες PostScreen μέχρι PostScreen6.

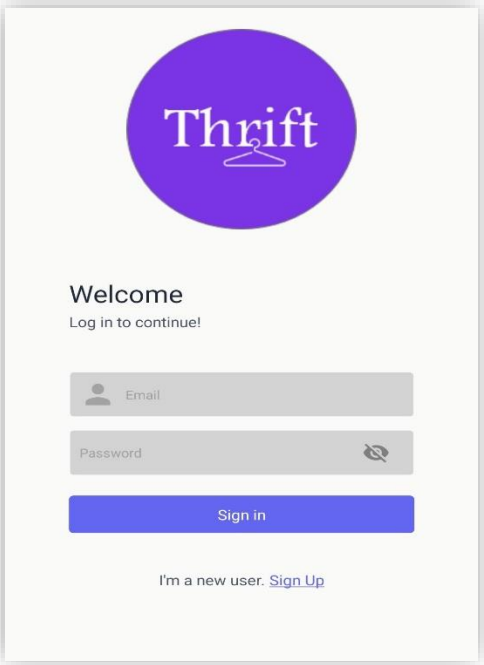
```
<Stack.Screen name="PostScreen" component={PostScreen}
  options={{
    headerShown: false
  }}
/>
<Stack.Screen name="PostScreen2" component={PostScreen2}
  options={{
    headerShown: false
  }}
/>
<Stack.Screen name="PostScreen3" component={PostScreen3}
  options={{
    headerShown: false
  }}
/>
```

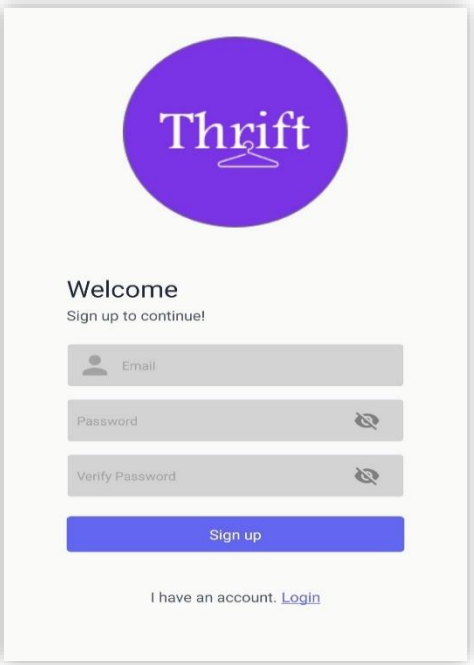
```
<Stack.Screen name="PostScreen4" component={PostScreen4}
  options={{
    headerShown: false
  }}
/>
<Stack.Screen name="PostScreen5" component={PostScreen5}
  options={{
    headerShown: false
  }}
/>
<Stack.Screen name="PostScreen6" component={PostScreen6}
  options={{
    headerShown: false
  }}
/>
```

d. ProfileStack.js
Το αρχείο ProfileStack.js περιέχει μέσα την οθόνη με το προφίλ του χρήστη και κάποιες ρυθμίσεις (ProfileScreen.js) , την οθόνη με την λίστα από τις αναρτήσεις που έχει κάνει ο χρήστης σαν πωλητής (MyPostedListScreen.js) και την οθόνη με την οποία μπορεί να επεξεργαστεί κάθε ανάρτηση (EditPostScreen.js).

```
<Stack.Navigator >
  <Stack.Screen name="ProfileScreen" component={ProfileScreen}
    options={{
      headerShown: false
    }}
  />
  <Stack.Screen name="MyPostedListScreen" component={MyPostedListScreen}
    options={{
      headerShown: false
    }}
  />
  <Stack.Screen name="EditPostScreen" component={EditPostScreen}
    options={{
      headerShown: false
    }}
  />
</Stack.Navigator >
```

4.3.2 SignIn & SignUp Screen





Η οθόνη SignUpScreen.js είναι η πρώτη οθόνη που εμφανίζεται όταν ο χρήστης ανοίγει την εφαρμογή για πρώτη φορά. Αν έχει ήδη λογαριασμό χρησιμοποιώντας το Link στο τέλος της οθόνης ο χρήστης μπορεί να πάει από την μια οθόνη στην άλλη και ανάποδα.

Ακόμη η SignUpScreen όταν εμφανίζεται στην οθόνη η πρώτη συνάρτηση που καλείται είναι η useEffect η οποία είναι hook της ReactNative. Αυτή η συνάρτηση ελέγχει χρησιμοποιώντας την συνάρτηση της firebase Authentication , onAuthStateChange η οποία αναγνωρίζει αν έχει αλλάξει η κατάσταση του προηγούμενος συνδεδεμένου χρήστη (Persistent authentication). Αν ο χρήστης είναι ήδη συνδεδεμένος στο σύστημα τότε η ροή αυθεντικοποίησης παραβλέπετε και το πρόγραμμα συνεχίζει στην βασική ροή και καλείτε η BottomTabNavigation που για πρώτη οθόνη έχει την HomeScreen.js

```
useEffect(() => {
  firebase.auth().onAuthStateChanged(authenticate => {
    if (authenticate) {
      myContext.setUserId(authenticate.uid)
      myContext.setUserEmail(authenticate.email)
      navigation.replace('BottomTabNavigation');
    } else {
      setLoading(false)
    }
  });
}, [])
```

Για τις φόρμες χρησιμοποιήθηκαν τα FormControl και Input (τύπου Email , Password αντίστοιχα). Το εικονίδιο στα δεξιά κάθε πεδίου τύπου Password χρησιμοποιείται για να κάνει εμφανίζει η όχι τους χαρακτήρες του πεδίου. Στην οθόνη εγγραφής η διαδικασία έχει ένα βήμα παραπάνω πριν πάει η διαδικασία στην Firebase ελέγχει αν οι δυο κωδικοί ταιριάζουν μεταξύ τους αν δεν ταιριάζουν τότε το FormControl κοκκινίζει το περίγραμμά του δευτέρου πεδίου του κωδικού και εμφανίζει προειδοποιητικό μήνυμα. Αν πατηθεί το κουμπί σύνδεσης η εγγραφής τότε εκτελείται μια συνάρτηση η οποία καλεί την Firebase και προσπαθεί να δημιουργήσει έναν νέο χρήστη η να συνδέσει έναν ήδη υπάρχον ανάλογα την περίπτωση. Αν κάτι πάει στραβά από οποιαδήποτε άποψης η Firebase ενημερώνει τον χρήστη για το τι πήγε στραβά. Μετά την αυθεντικοποίηση από οποιαδήποτε οθόνη η εφαρμογή συνεχίζει στο BottomTabNavigation.

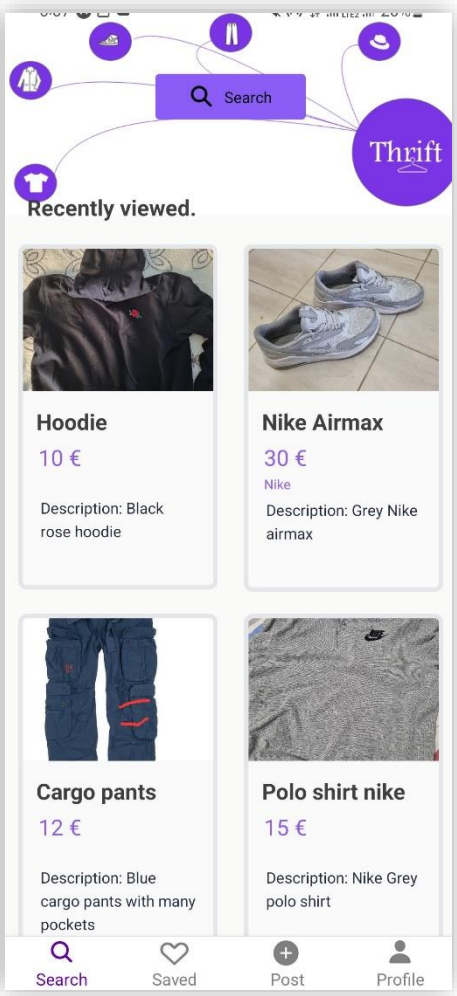
Μέθοδος που εκτελεί την εγγραφή του χρήστη στην Firebsase.

```
const signUpUser = () => {
  if (VerifyPassword == Password) {
    setVerify(false);
    setLoading(true)
    firebase
      .auth()
      .createUserWithEmailAndPassword(Email, Password)
      .then((userCredential) => {
        navigation.replace('BottomTabNavigation');
      })
      .catch((error) => {
        setLoading(false)
        const errorCode = error.code;
        const errorMessage = error.message;
        Alert.alert(errorMessage);
      });
  } else {
    setVerify(true);
  }
}
```

Η μέθοδος που κάνει την σύνδεση στην εφαρμογή.

```
const logIn = () => {
  setLoading(true)
  firebase
    .auth()
    .signInWithEmailAndPassword(Email, Password)
    .then(() => { })
    .catch(err => {
      setLoading(false)
      alert(err.message)
    })
}
```

4.3.3 Home Screen



Η οθόνη Home Screen είναι η πρώτη οθόνη που συναντάει ο χρήστης μετά από την σύνδεση του στην εφαρμογή. Βρίσκεται στην καρτέλα με τίτλο ‘Αναζήτηση’ γιατί από αυτή την οθόνη μπορεί να προηγηθεί στην οθόνη αναζήτησης μέσω του κουμπιού στο πάνω μέρος. Στο υπόλοιπο κομμάτι της οθόνης υπάρχει μια λίστα με πρόσφατες αναρτήσεις που παρατήρησε ο χρήστης. Σε αυτή τη λίστα υπάρχει μια σύντομη περιγραφή του ρούχου που πωλείται Το κάθε χαρακτηριστικό του προϊόντος εμφανίζεται μόνο αν έχει καταχωρηθεί από τον πωλητή (Conditional Rendering).

Η λίστα που εμφανίζεται βρίσκεται αποθηκευμένη κάτω από τον κωδικό του χρήστη στη βάση (Realtime Database) και ενημερώνεται κάθε φορά που αυτός πλοηγείτε στις λεπτομέρειες κάποιας ανάρτησης.
Για να ανακτηθούν τα δεδομένα της κάθε ανάρτησης και να προβληθούν γίνεται η κλήση στην βάση με link.

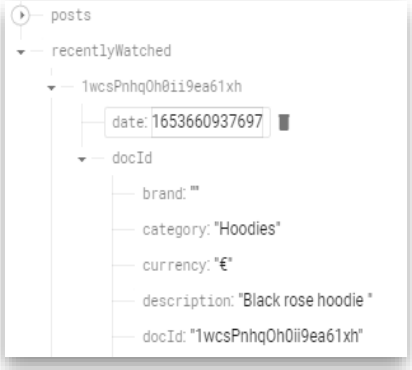
```
firebase.database().ref('users/' + myContext.UserIdValue + "/recentlyWatched/");
```

Δηλαδή για να ανακτηθεί η πληροφορία, πρώτα χρησιμοποιείται η βάση του δέντρου δηλαδή οι χρήστες (users), μετά ανακτάτε ο κωδικός του χρήστη μέσω context (useContext ReactNative Hook) δηλαδή μιας global μεταβλητής που καθορίζεται κατά την σύνδεση και στη συνέχεια η λίστα που είναι αντικείμενο ενδιαφέροντος για αυτήν την οθόνη. Στη συνέχεια ανακτώνται οι φωτογραφίες από την βάση (Storage) και αντιστοιχούνται στις αγγελίες οι φωτογραφίες αναρτιούνται με τον ίδιο τρόπο μέσω link όπου για να προσπεραστούν οι φωτογραφίες της κάθε ανάρτησης χρησιμοποιείτε η βάση του δέντρου photos στη συνέχεια το Id της ανάρτησης και στο τέλος χρησιμοποιείται ο αριθμός μηδέν για να παρθεί η πρώτη φωτογραφία από την συλλογή της ανάρτησης.

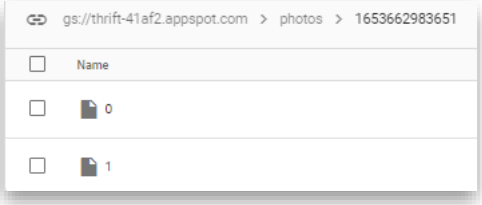
```
firebase.storage().ref('photos/' + x.docId.id + '/0')
```

Η μορφή των δεδομένων στην firebase realtime database και στην firebase storage

Realtime

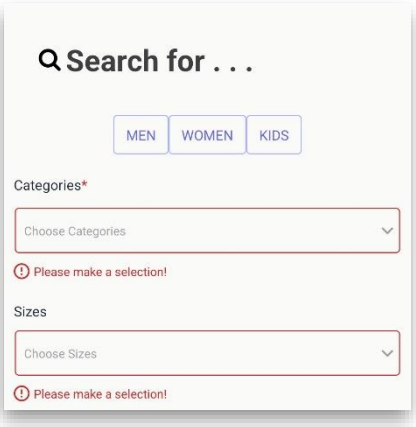


Storage



4.3.4 Search Screen

Η SearchScreen.js είναι μια από τις πιο σημαντικές οθόνες της εφαρμογής. Αυτή η οθόνη περιέχει τα φίλτρα μπορούν να εφαρμοστούν ώστε ο χρήστης να μπορεί να βλέπει μόνο τα αποτελέσματα που τον ενδιαφέρουν. Αποτελείται από δυο υποχρεωτικά παιδιά 'Κατηγορία ρούχου' , 'Νούμερο' τα υπόλοιπα πεδία δεν είναι υποχρεωτικό να συμπληρωθούν , αυτά είναι η το εύρος τιμής η μάρκα και το εύρος της περιοχής που αναζητά καθώς ξεκινάει με προεπιλογή στο ένα χιλιόμετρο από την τωρινή του τοποθεσία.

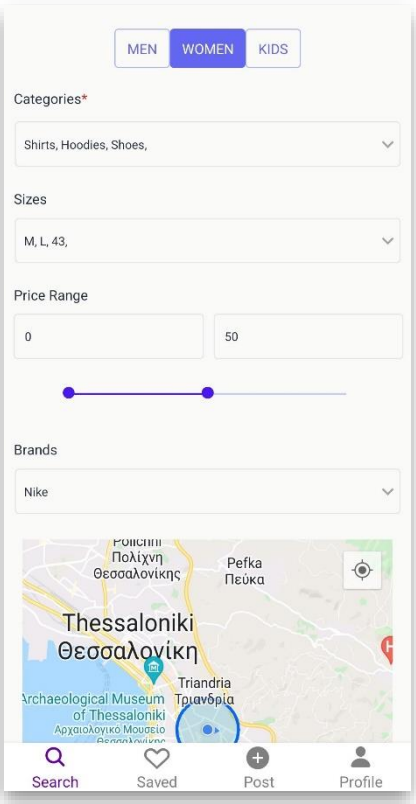


Ξεκινώντας η οθόνη ζητάει άδεια από τον χρήστη για να χρησιμοποιήσει την τοποθεσία του για να εκκινήσει τους χάρτες και να θέσει τον κύκλο που θα αποτελεί την ακτίνα του χρήστη αυτό γίνεται εφικτό με την βιβλιοθήκη expo-location και react-native-maps.

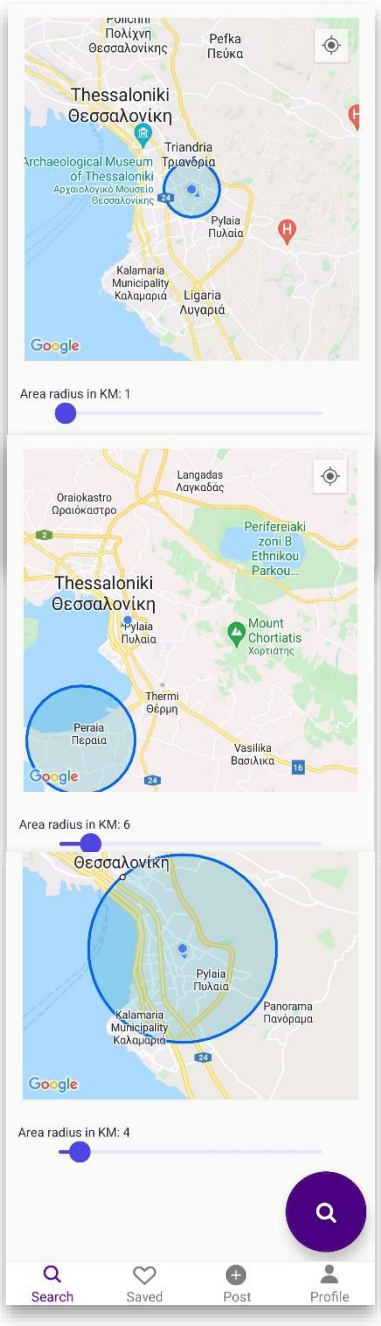
```
let { status } = await Location.requestForegroundPermissionsAsync();
```

Ακόμη και αν δεν δώσει την άδεια του , η εφαρμογή του επιτρέπει αργότερα να καθορίζει μια τοποθεσία αναζήτησης χειροκίνητα.

Στη συνέχεια ο χρήστης μπορεί να ρυθμίσει τα φίλτρα του από τις επιλογές που του δίνονται για την κατηγορία, το νούμερο, την μάρκα που επιθυμεί χρησιμοποιώντας το στοιχείο Select. Ωστόσο το στοιχείο Select της native-base δεν υποστήριζε πολλαπλή επιλογή παρά μόνο μια επιλογή. Δεν θα ήταν αποτελεσματική η αναζήτηση όμως αν ο χρήστης επιτρεπόταν να ψάξει μόνο μια κατηγορία ή μέγεθος την φορά γιατρό και δημιουργήθηκε μέθοδος η οποία αποθηκεύει όλες τις επιλογές του χρήστη μέσω λίστας και τις παρουσιάζει σαν μια συμβολοσειρά με όριο τις τρεις επιλογές λόγο των κανόνων της βάσης δεδομένων. Για την επιλογή του γένους του οποίου απευθύνεται η δημοσίευση χρησιμοποιήθηκαν γραφικά που συμπεριφέρονται ως διακόπτες. Όσο αφορά την τιμή ο τρόπος εισαγωγής της θα έπρεπε να γύναι ξεκάθαρος και εύκολος στην χρήση για αυτό χρησιμοποιήθηκε το στοιχείο MultSlider από την βιβλιοθήκη '@ptomasroos/react-native-multi-slider'. Με το οποίο μπορεί ο χρήστης να ρυθμίσει το εύρος της τιμής σέρνοντας δυο δείκτες έναν για την χαμηλότερη τιμή και έναν για την υψηλότερη που επιθυμεί. Επίσης για να γίνει ακόμα πιο εύκολο στην χρήση του, χρησιμοποιήθηκαν και δυο αριθμητικά πεδία από πάνω για τον ίδιο σκοπό.



Για τον κανονισμό της τοποθεσίας στην οποία θα ήθελε να ψάξει ο χρήστης χρησιμοποιήθηκε ένας χάρτης σε συνδυασμό με έναν απλό slider. Ο χρήστης μέσω στου slider μπορεί να ρυθμίσει την χιλιομετρική απόσταση στην οποία είναι διατεθειμένος να παραλάβει τα προϊόντα. Όσο ο χρήστης ανεβάζει και κατεβάζει τον slider

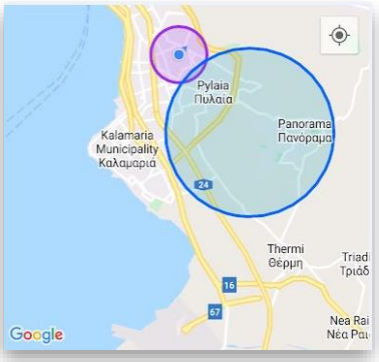



```
<MapView
  style={styles.map}
  showsUserLocation={true}
  region={Loc}
  loadingEnabled={true}
  onRegionChangeComplete={(region) => setLoc(region)}
  onPress={onLocationSelect}
>
  <Circle
    strokeWidth={3}
    strokeColor={'#0166FE'}
    fillColor={'rgba(10,157,180,0.2)'}
    radius={KM * 1000}
    center={pin}
    scrollEnabled={false}
  />
</MapView>
<Text style={{ marginTop: 30 }}> Area radius in KM: {KM}</Text>
<Center>
  <Slider w="3/4" maxW="300" onChange={(v) => { setKM(v) }}
    defaultValue={1} minValue={0} maxValue={50} step={1}
    colorScheme={colorMode == 'light' ? "indigo" : "cyan"}
  >
```

```
const onLocationSelect = (event: MapEvent) => {
  setPin(event.nativeEvent.coordinate)
}
```

ανανεώνει μια μεταβλητή τύπου state (useState ReactNative hook) την 'KM' η οποία ρυθμίζει την ακτίνα του κύκλου που υπάρχει Μέσα στον χάρτη ανάλογα. Κάθε φορά που ο χρήστης αλλάζει τον χάρτη καλείται η συνάρτηση onRegionChangeComplete η οποία ανανεώνει τις συντεταγμένες του χάρτη. Σε περίπτωση που ο χρήστης δεν έχει δώσει άδεια για την χρήση της τοποθεσίας του υπάρχει η συνάρτηση onLocationSelect η οποία καλείται από όταν χρήστης κάνει πάτημα σε κάποιο σημείο του χάρτη δέχεται σαν όρισμα ένα MapEvent και το μόνο που κάνει είναι να ανανεώνει τις συντεταγμένες του κύκλου.

Τέλος μόνο όταν τα υποχρεωτικά πεδία συμπληρωθούν δηλαδή οι κατηγορίες και τα μεγέθη του ρούχου που αναζητάει ο χρήστης τότε εμφανίζεται κάτω δεξιά το κουμπί που του επιτρέπει να κάνει την αναζήτηση βάση των φίλτρων που είχε καταχωρήσει μέχρι εκείνη την στιγμή.




```
query = postRef
  .where("price", ">", PriceRange[0])
  .where("price", "<", PriceRange[1])
  .where("category", "in", CategorySelected)
  .orderBy("price")
  .limit(100)
```

Όταν όλα είναι έτοιμα για να γίνει η αναζήτηση ο χρήστης πατάει το κουμπί και καλείται η ασύγχρονη συνάρτηση 'Search' η οποία είναι υπεύθυνη για να κάνει την κλήση στην Firebase firestore με τα κατάλληλα φίλτρα. Καθώς η firebase φιλτράρει τα δεδομένα της εφαρμογής στον δικό της server έχει κανόνες έτσι ώστε να μην εξαντλούνται άσκοπα οι πόροι της, για αυτό τον λόγο το μισό φιλτράρισμα γίνεται στους server της firebase και το άλλο μισό γίνεται πάνω στα δεδομένα που επέστρεψε η βάση στον client. Ακόμη στην μεριά του client γίνεται και ο υπολογισμός της απόστασης από την περιοχή του χρήστη στην περιοχή των αγγελιών. Ο αλγόριθμος υπολογισμού απόστασης επιστρέφει μόνο τις αγγελίες οι οποίες έχουν απόσταση μικρότερη από το άθροισμα της ακτίνας του πωλητή και της ακτίνας του αγοραστή. Και τέλος αφαιρεί τις αγγελίες που έχουν δημοσιευτεί από τον συνδεδεμένο χρήστη.

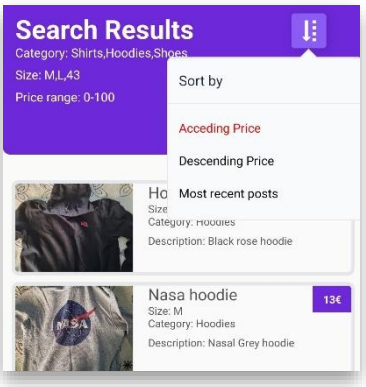
Μέθοδος υπολογισμού απόστασης δυο συντεταγμένων.

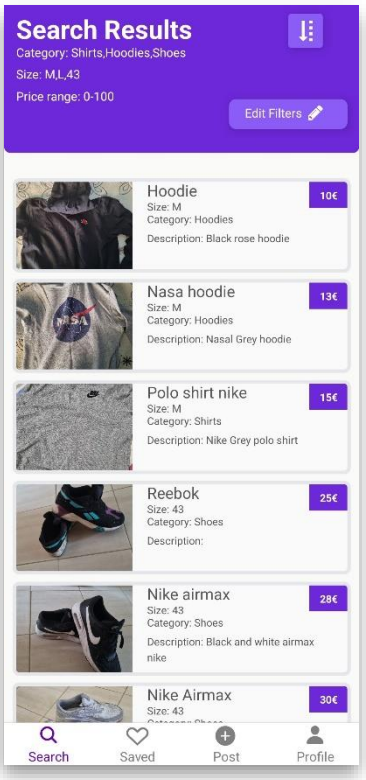
```
function getDistanceFromLatLonInKm(lat1, lon1, lat2, lon2) {
  var R = 6371;
  var dLat = deg2rad(lat2 - lat1);
  var dLon = deg2rad(lon2 - lon1);
  var a =
    Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
    Math.sin(dLon / 2) * Math.sin(dLon / 2)
    ;
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  var d = R * c;
  return d;
}

function deg2rad(deg) {
  return deg * (Math.PI / 180)
}
}
```

Φιλτράρισμα στην μεριά του client

```
const moreFiltering = (result) => {
  let data = result.filter((item) => {
    if (Gender != null) {
      if (Brand) {
        return (SizeSelected.includes(item.size)) && (Brand == item.brand) &&
          (item.gender == Gender) &&
          (getDistanceFromLatLonInKm(item.longitude, item.latitude, CurrentLoc.coords.longitude, CurrentLoc.coords.latitude) <= item.km + KM)
      } else {
        return (SizeSelected.includes(item.size)) &&
          (item.gender == Gender) &&
          (getDistanceFromLatLonInKm(item.longitude, item.latitude, CurrentLoc.coords.longitude, CurrentLoc.coords.latitude) <= item.km + KM)
      }
    } else {
      if (Brand) {
        return (SizeSelected.includes(item.size)) && (Brand == item.brand) &&
          (getDistanceFromLatLonInKm(item.longitude, item.latitude, CurrentLoc.coords.longitude, CurrentLoc.coords.latitude) <= item.km + KM)
      } else {
        return (SizeSelected.includes(item.size)) &&
          (getDistanceFromLatLonInKm(item.longitude, item.latitude, CurrentLoc.coords.longitude, CurrentLoc.coords.latitude) <= item.km + KM)
      }
    }
  })
  data = data.filter(item => item.user !== myContext.UserIdValue)
  navigation.navigate('Results',
    {
      data: data, CategorySelected: CategorySelected, SizeSelected: SizeSelected,
      Brand: Brand, PriceRange: PriceRange, Gender: Gender, searchPin: pin, searchRadius: KM
    })
  setLoading(false)
}
```

4.3.5 Results Screen





Η οθόνη Results Screen.js είναι αυτή που προβάλει τα αποτελέσματα της αναζήτησης του χρήστη από την οθόνη αναζήτησης , στέλνοντας την λίστα με τις αγγελίες που επέστρεψε η βάση μέσω Navigation Route. Σε αυτό το σημείο ο χρήστης μπορεί να κάνει μια γρήγορη ανασκόπηση των αγγελιών χωρίς απαραίτητα να πλοηγηθεί στην οθόνη με τις λεπτομέρειες (PostDetailScreen.js) καθώς δίνονται σημαντικά χαρακτηριστικά και μια φωτογραφία. Στην επικεφαλίδα της οθόνης βρίσκονται τα φίλτρα που χρησιμοποιήθηκαν για τα αποτελέσματα καθώς και δυο κουμπιά. Με το κουμπί 'Edit Filters' ο χρήστης πλοηγείτε γρηγορά στην οθόνη αναζήτησης για να επεξεργαστεί τα φίλτρα. Με το δεύτερο κουμπί μπορεί να αλλάξει σε ζωντανό χρόνο την διάταξη των αποτελεσμάτων οι επιλογές που δίνονται είναι με αύξουσα τιμή, με φθίνουσα μέσω του πεδίου 'price' και με σειρά ημερομηνίας με την πιο πρόσφατη να είναι πρώτη για αυτό τον σκοπό χρησιμοποιείτε το πεδίο 'id' το οποίο στην ουσία είναι ένα timestamp (κωδικός ημερομηνίας) της ημερομηνίας που δημιουργήθηκε η αγγελία. Για να δει ο χρήστης περισσότερες λεπτομέρειες για κάποιο προϊόν απλά θα πρέπει να πατήσει πάνω σε ένα από τα αποτελέσματα τότε η οθόνη θα τον οδηγήσει στην οθόνη με τις λεπτομέρειες στέλνοντας τα δεδομένα του προϊόντος που πατήθηκε

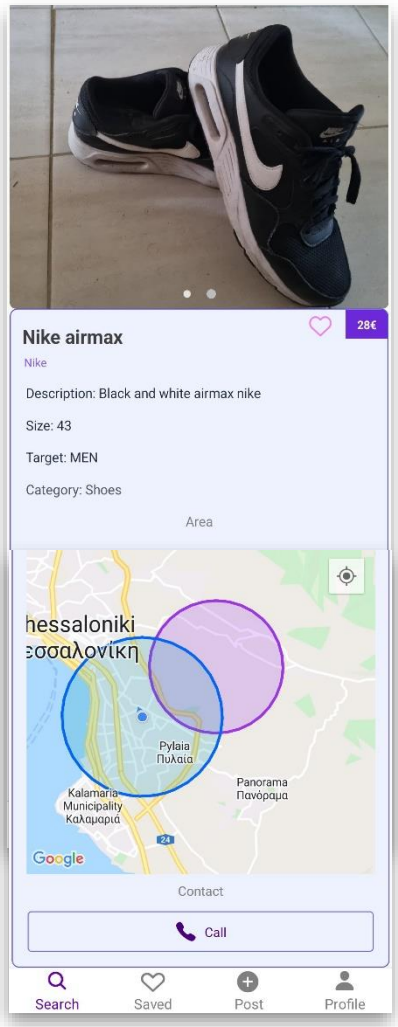
```
<TouchableOpacity onPress={() => { navigation.navigate('PostDetailScreen', { item, searchPin, searchRadius, mode: "Search" }) }}>
```

Ακόμη στέλνεται ένα όρισμα με το όνομα 'mode' και τιμή 'search' για να ρυθμίσει την επόμενη έτσι ώστε να παρουσιάσει το προϊόν σαν αποτέλεσμα αναζητήσεις. Αυτό συμβαίνει γιατί η επόμενη οθόνη χτίστηκε με τρόπο έτσι ώστε να ξαναχρησιμοποιείτε σε διαφορετικές περιπτώσεις όπως για παράδειγμα την παρουσίαση των λεπτομερειών από την αρχική οθόνη ή την λίστα αγαπημένων. Σε κάθε περίπτωση η επόμενη οθόνη εμφανίζεται διαφορετικά έτσι ώστε να κάνει εμφανές στον χρήστη αν έχει παραχθεί από αναζήτηση η από δεδομένα του χρήστη

4.3.6 Post Detail Screen

Η οθόνη PostDetailScreen.js έχει όλες τις λεπτομέρειες που χρειάζεται να γνωρίζει ο χρήστης για το προϊόν. Αρχικά όταν ολοκληρωθεί η φόρτωση της οθόνης το πρώτο πράμα που κάνει μέσω της useEffect είναι να τοποθέτηση την αγγελία στην λίστα με τις πρόσφατες αγγελίες του χρήστη που προβλήθηκαν.

```
const addToRecentlyWatched = () => {
  firebase.database().ref('users/' + myContext.UserIdValue + "/recentlyWatched/" + item.docId)
    .set({ docId: item, date: Date.now()})
}
```



Η δομή της οθόνης είναι τέτοια ώστε να παρουσιάζει όλες τις πληροφορίες στον χρήστη με κατανοητό τρόπο. Για την επίτευξη αυτού χρησιμοποιήθηκε στην κορυφή της οθόνης ένα carousel με εικόνες του προϊόντος από την βιβλιοθήκη 'react-native-image-slider-box'. Οι φωτογραφίες ανακτώνται από την firebase storage με την ίδια κλήση όπως στην λίστα αλλά αυτή την φορά χωρίς να ζητείτε μόνο η πρώτη εικόνα. Πιο κάτω υπάρχουν λεπτομέρειες για τα χαρακτηριστικά όπως η κατηγορία, η τιμή το μέγεθος, η μάρκα και μια περιγραφή.

Για να γίνει πιο κατανοητό το σενάριο της απόστασης που υπάρχει μεταξύ του αγοραστή και του πωλητή δημιουργήθηκε άλλος ένα χάρτης , αυτή τη φορά όμως με δυο κύκλους ο ένας (μπλε) δηλώνει το ευρέως της αναζήτησης και ο άλλος (μωβ) το εύρος που εμφανίζεται η αγγελία. Έτσι είναι ξεκάθαρο στον αγοραστή ποιο είναι το σημείο τομής των δυο πλευρών.

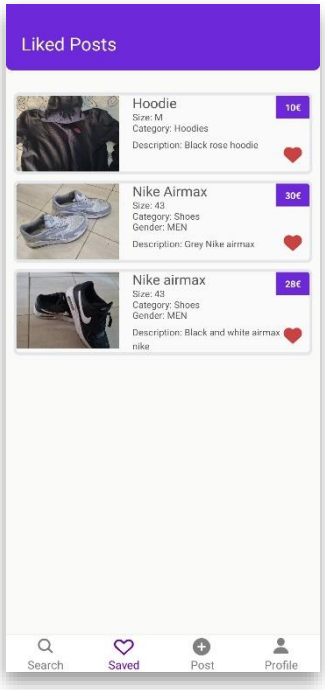
Αυτή η οθόνη εκτός από πληροφορίες προσφέρει και κάποιες γρήγορες ενέργειες. Η πρώτη βρίσκετε δίπλα στην τιμή και είναι ένα κουμπί με μια καρδιά για σύμβολο που λειτουργεί σαν διακόπτης αν πατηθεί τοποθετεί την αγγελία στη λίστα με τις αγαπημένες του χρήστη , αν ξαναπατηθεί την αφαιρεί χρησιμοποιώντας πάντα την ίδια μέθοδο με link.

```
if (Like == 'heart') {
  setLike("hearto")
  firebase.database().ref('users/' + myContext.UserIdValue + "/likedPosts/" + item.docId).remove()
} else {
  setLike("heart")
  firebase.database().ref('users/' + myContext.UserIdValue + "/likedPosts/" + item.docId)
    .set({ docId: item })
```

Τέλος η τελευταία ενέργεια που μπορεί να κάνει ο χρήστης είναι να πάρει τηλέφωνο στον πωλητή μέσω του κουμπιού στο τέλος το οποίο καλεί κατευθείαν τον πωλητή με το τηλέφωνο του χρήστη αρκεί να υπάρχει συνδεδεμένη κάρτα SIM. Αυτό γίνεται εφικτό με την λειτουργία Linking της React Native.

```
<Button variant="outline" _dark={{ color: 'cyan', _text: 'cyan' }} _light={{ borderColor: 'indigo.500' }}
  leftIcon={<Ionicons name="md-call" size={24} color={colorMode == 'light' ? "indigo" : "cyan"} />}
  onPress={() => { Linking.openURL(`tel:${item.phone}`) }}
>
  <Text style={{ color: colorMode == 'light' ? "indigo" : "cyan" }}>Call</Text>
</Button>
```

4.3.7 Saved Screen

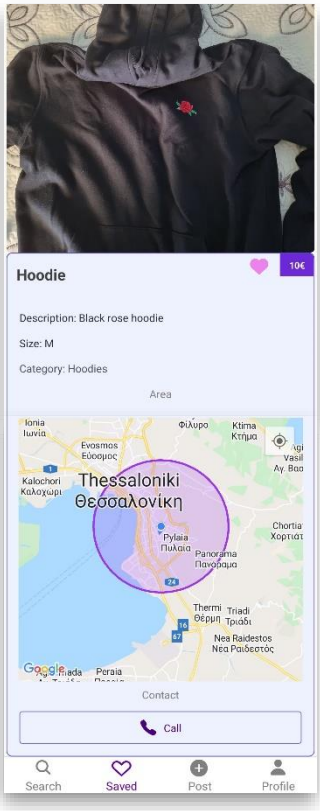


Η οθόνη SavedScreen.js περιέχει μέσα μια λίστα με τις δημοσιεύσεις που άρεσαν στον χρήστη και τις αποθήκευσε έτσι ώστε να τις ξαναδεί αργότερα. Αυτή η οθόνη δημιουργήθηκε χρησιμοποιώντας το αρχείο Helper.js το οποίο δέχεται κάποια ορίσματα και παρουσιάζει μια λίστα.

Έγινε η χρήση αυτού του αρχείου καθώς υπήρχαν δυο οθόνες οι οποίες ήταν ακριβώς ίδιες εμφανισιακά και στη λογική και το μόνο που άλλαζε ήταν η πληροφορία. Έτσι για να μην γραφεί σχεδόν ο ίδιος κώδικας 2 φορές φτιάχτηκε η Helper.js στην οποία έχει υλοποιηθεί μια φορά η λογική στην μέθοδο List και απλώς καλείται με διαφορετικά ορίσματα από διαφορετικά σημεία (SavedScreen.js , MyPostedListScreen.js). Με αυτόν τον τρόπο καταφέρθηκε να μειωθεί δραματικά ο κώδικας που χρειάστηκε για να υλοποιηθεί η οθονη.

Στην List έχει υλοποιηθεί η λογική της βάσης και το μόνο που χρειάζεται να ξέρει είναι τα ορίσματα που στέλνονται δηλαδή το navigation έτσι ώστε να μπορέσει να δρομολογήσει τον χρήστη στην οθονη με τις λεπτομέρειες όταν πατήσει σε κάποιο προϊόν , το label το οποίο στην ουσία είναι η επικεφαλίδα της οθόνης και το όρισμα list το οποίο είναι ο κόμβος που πρέπει να ψάξει η βάση μέσω του link που θα χρησιμοποιήσει για να βρει την λίστα.

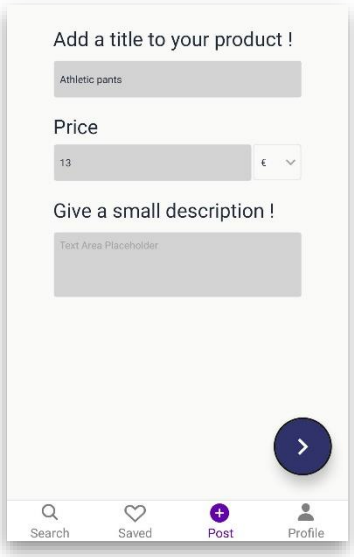
```
import React from "react"
import { List } from "../components/Helper";
export default SavedScreen = ({ navigation }) => {
  return (
    <List navigation={navigation} label={'Liked Posts'} list={'likedPosts'} />
  )
}
```



Όταν ο χρήστης προηγηθεί στην οθονη με τις λεπτομέρειες της κάθε αγγελίας (PostDetailScreen.js), αυτή την φορά θα δει το κουμπί σε σχήμα καρδίας ήδη επιλεγμένο και έχει την δυνατότητα να αφαίρεση την αγγελία πατώντας το και τον χάρτη να έχει μόνο έναν κύκλο μέσα (μωβ) αυτόν που δηλώνει την περιοχή του πωλητή. Ενώ η οθονη είναι η ίδια που χρησιμοποιείται από την λίστα αποτελεσμάτων αναζήτησης , λίστα προσφάτων αναρτήσεων και λίστα αποθηκευμένων, κάθε φορά μέσω τον ορισμάτων αναγνωρίζει τι πρέπει να δείξει στον χρήστη (Conditional Rendering). Έτσι σε αυτήν την περίπτωση αναγνωρίζει ότι δεν υπήρξε αναζήτηση προηγούμενος οπότε δεν πρέπει να δείξει τον δεύτερο κύκλο (μπλε) που δηλώνει το εύρος της αναζήτησης.

```
navigation.navigate('PostDetailScreen', { item, mode: "noSearch" })
```

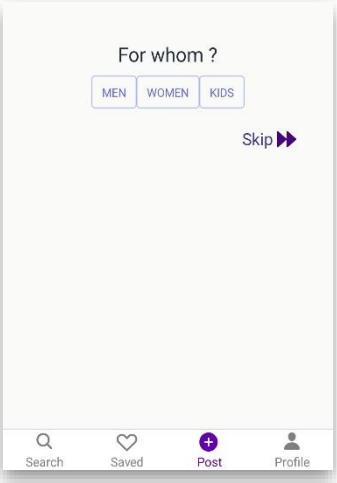
4.3.8 Post Screen



Η PostScreen.js είναι η πρώτη οθονη που συναντάει ο χρήστης όταν μπαίνει στην διαδικασία να κάνει μια ανάρτηση κάποιου προϊόντος. Για να ολοκληρωθεί η ανάρτηση ο χρήστης πρέπει να διασχίσει μια σειρά από οθόνες (PostScreen.js – PostScreen6.js). Η διαδικασία ανάρτησης είναι στημένη έτσι ώστε να καθοδηγεί τον χρήστη σε κάθε βήμα ώστε να μην κάνει λάθος και να μην μπορεί να προχωρήσει παραπέρα αν δεν έχει παραχωρήσει τα κατάλληλα δεδομένα. Σε κάθε μεταφορά του χρήστη από μια οθονη στην επόμενη μεταφέρουμε ένα αντικείμενο που κρατάει τις πληροφορίες που βάζει ο χρήστης το οποίο ενημερώνεται ανάλογα στην κάθε οθονη.

Αυτή η οθονη αποτελείται από τέσσερα πεδία, τον τίτλο του προϊόντος, την τιμή (numeric) , το νομιμά (selection) και ένα μεγαλύτερο πεδίο που μπορεί ο χρήστης να βάλει μια περιγραφή. Από αυτά τα πεδία τα υποχρεωτικά είναι αυτά του τίτλου και της τιμής γιατί χωρίς αυτά ο χρήστης δε πρέπει να μπορεί να προχωρήσει, το πεδίο του νομίσματος είναι υποχρεωτικό αλλά υπάρχει τιμή από την αρχή και δεν μπορεί να είναι κενό. Για αυτό και μόλις συμπληρωθούν αυτά εμφανίζεται κάτω δεξιά ένα κουμπί ‘Συνέχεια’ που με αυτό μπορεί ο χρήστης να συνεχίσει την ανάρτηση του.

4.3.9 Post Screen 2

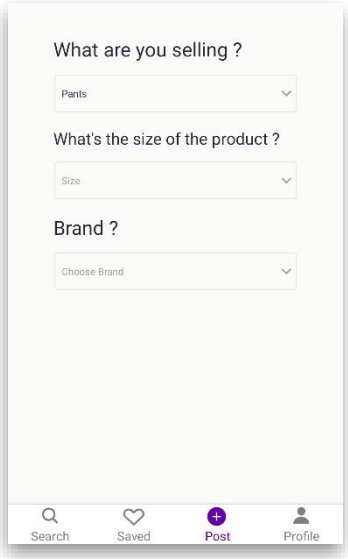


Στην PostScreen.js μπορεί να επιλεχθεί το target group της αγγελίας ανάμεσα σε τρεις επιλογές για άντρες , για γυναίκες και για παιδιά. Οι επιλογές είναι τύπου διακόπτης δηλαδή όταν επιλεγεί μια, θα σβήσει την προ υπάρχουσα επιλογή αν υπάρχει ακόμη αν επιλεχθεί η ήδη υπάρχουσα επιλογή επιλογή σβήνει και το στοιχείο επιστρέφει στην αρχική του κατάσταση.

Η πληροφορία όμως δεν είναι υποχρεωτική για να συνεχίσει ο χρήστης. Έτσι υπάρχει ένα κουμπί ‘Skip’ με το οποίο ο χρήστης μπορεί να προχωρήσει στην επόμενη οθονη χωρίς να επιλέξει κάτι. Αν επιλέξει τελικά, τότε το κουμπί ‘Skip’ εξαφανίζεται και εμφανίζεται κάτω δεξιά το κουμπί ‘Συνέχεια’ που μπορεί να μεταφέρει τον χρήστη στην επόμενη οθονη.

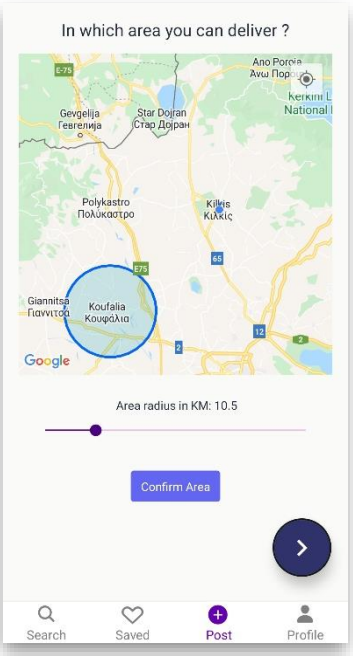
4.3.10 Post Screen 3

Το επόμενο βήμα για την ανάρτηση ενός ρούχου είναι η οθονη PostScreen3.js στην οποίο ο χρήστης μπορεί να προσθέσει κάποια παραπάνω χαρακτηριστικά όπως είναι η κατηγορία, το μέγεθος και η μάρκα όλα εκ των οποίον είναι τύπου επιλογής. Από αυτά τα πεδία τα υποχρεωτικά είναι η κατηγορία και το μέγεθος, η κατηγορία είναι η υποχρεωτική για να διευκολύνει τους αγοραστές να βρίσκουν πιο εύκολα αυτό που ψάχνουν και το μέγεθος καθώς δεν μπορεί να υπάρξει ρούχο χωρίς μέγεθος. Μόνο όταν συμπληρωθούν τα υποχρεωτικά πεδία ο χρήστης θα δει του κουμπί για να προχωρήσει στην επόμενη οθονη.

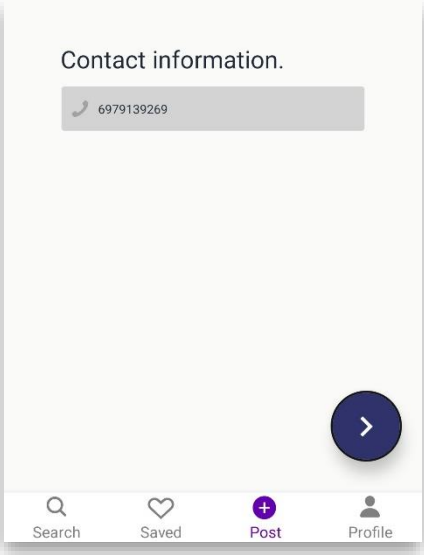


4.3.11 Post Screen 4

Στην οθονη PostScreen4.js ο χρήστης μπορεί να επιλέξει την περιοχή την οποία θα εμφανίζεται η ανάρτηση. Χρησιμοποιήθηκε ξανά ένας χάρτης με έναν slider που ρυθμίζει την ακτίνα του κύκλου που θα μπορεί να εμφανιστεί η ανάρτηση. Ακόμη υπάρχει δυνατότητα να επιλεχθεί διαφορετική τοποθεσία από αυτή που βρίσκετε ο χρήστης κάνοντας πάτημα σε οποιοδήποτε σημείο του χάρτη κάνοντας έτσι το νέο σημείο κέντρο του κύκλου. Αφού ο χρήστης πατήσει το κουμπί ‘επιβεβαίωση περιοχής’ ο χάρτης και ο slider κλειδώνουν και δεν μπορούν να αλλάξουν και εμφανίζεται ξανά το κουμπί ‘Συνέχεια’. Αν ο χρήστης δεν αλλάξει τίποτα και πατήσει επιβεβαίωση και συνέχεια τότε τα δεδομένα που στέλνονται είναι η τωρινή του τοποθεσία με ακτίνα ενός χιλιομέτρου που είναι η προεπιλογή του slider.

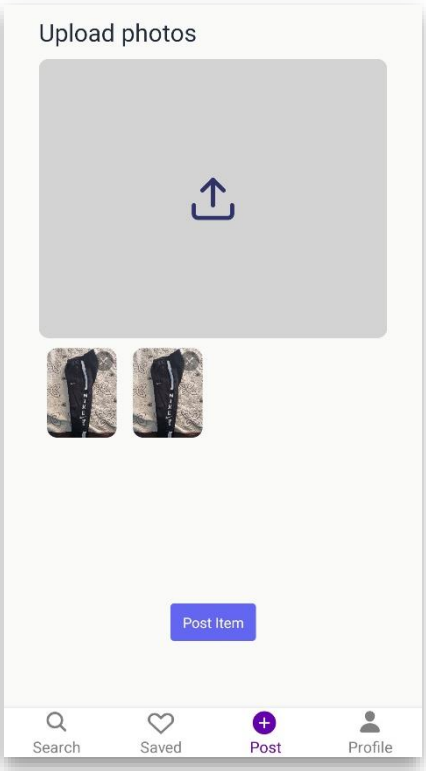


4.3.12 Post Screen 5



Δε θα μπορούσε να είναι υπάρξει ανάρτηση χωρίς να υπάρχει μέσω επικοινωνίας για αυτό και την PostScreen5.js ο χρήστης πρέπει να καταχωρήσει ένα αριθμό στον οποίο θα μπορούν να καλούν οι ενδιαφερόμενοι αγοραστές. Το πεδίο που χρησιμοποιείτε είναι αριθμητικό δηλαδή επιτρέπει μόνο αριθμούς και για να μπορέσει να συνεχίσει την ανάρτηση ο χρήστης θα πρέπει να καταχωρήσει ακριβώς δέκα ψηφιά μόνο τότε εμφανίζεται το κουμπί ‘Συνέχεια’.

4.3.13 Post Screen 6



Η τελευταία οθονη στην διαδικασία ανάρτησης είναι η PostScreen6.js στην οποία ο χρήστης μπορεί να ανεβάσει τις φωτογραφίες του προϊόντος καθώς για την πώληση ρούχων είναι απαραίτητο να υπάρχουν φωτογραφίες.

Για το ανέβασμα των φωτογραφιών χρησιμοποιήθηκε η βιβλιοθήκη ‘expo-image-picker’ η οποία δίνει στον χρήστη την δυνατότητα να ανεβάσει φωτογραφίες που έχει αποθηκευμένες στο smartphone του. Μπορεί να ανεβάσει μια την φορά καθώς δεν υπάρχει μαζική επιλογή και το όριο είναι μέχρι και οχτώ φωτογραφίες για να αποφευχθεί η κατάχρηση της βάσης δεδομένων.

```
let result = await ImagePicker.launchImageLibraryAsync({
  mediaTypes: ImagePicker.MediaTypeOptions.All,
  quality: 1.0,
  cancelled: false
});
```

Στην οθονη υπάρχει ένα κουμπί για το ανέβασμα και μια λίστα στην οποία φαίνονται οι φωτογραφίες που έχουν επιλεχθεί. Ακόμη έχει υλοποιηθεί η λειτουργία αφαίρεσης εικόνων με ένα διακριτικό κουμπί στην πάνω δεξιά γωνία της εικόνας.

Όταν δοθεί έστω και μια εικόνα εμφανίζεται το κουμπί ‘Δημοσίευση Προϊόντος’. Όταν πατηθεί

ξεκινάει η δημοσιοποίηση του προϊόντος με τα στοιχεία που καταχωρήθηκαν. Για την αποθήκευση της πληροφορίας έπρεπε να χρησιμοποιηθεί κάθε κομμάτι της firebase.

Τα χαρακτηρίστηκα της ανάρτησης που συγκεντρώθηκαν από όλες της οθόνες της ροής ανάρτησης αποθηκεύονται στην firebase storage, όταν ολοκληρωθεί η αποθήκευση η ανάρτηση αποθηκεύεται στην λίστα με τις αναρτήσεις του χρήστη μέσω της Firebase realtime database και όταν ολοκληρωθεί και αυτή η διαδικασία ανεβάζει τις φωτογραφίες στην firebase storage. Όταν τελείωση η διαδικασία εμφανίζεται ένα παράθυρο επιβεβαίωσης (Alert) και ο χρήστης μεταφέρεται στην αρχική οθονη (HomeScreen.js).

```
firebase.firestore().collection('post')
  .add({
```

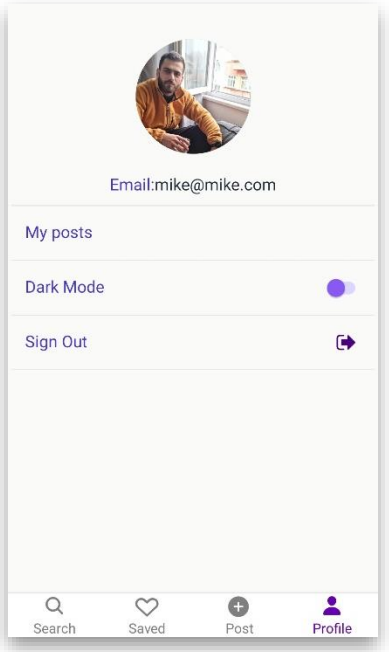
• • •

```
  })
  .then((docRef) => {
    firebase.database().ref('users/' + myContext.UserIdValue + "/posts/" + docRef.id)
      .set({
```

• • •

```
        }
      }).then(async() => {
        for (let i = 0; i < Gallery.length; i++) {
          console.log(Gallery[i])
          const response = await fetch(Gallery[i]);
          const blob = await response.blob();
          var ref = firebase.storage().ref().child('photos/' + timestamp + '/' + i);
          ref.put(blob);
        }
      })
  }).finally(() => {
  Alert.alert('Your item is now online!')
  navigation.popToTop()
  navigation.replace('BottomTabNavigation', { screen: 'HomeHomeScreen' })
})
```

4.3.14 Profile Screen



Η ProfileScreen.js είναι η οθονη με τις λεπτομερείς του χρήστη δηλαδή το email του , την φωτογραφία , την λίστα με τις αναρτήσεις του χρήστη και τις επιλογές σκοτεινή λειτουργία (Dark Mode) και αποσύνδεση.

Το email του χρήστη ανακτάται μέσω ενός hook της ReactNative που λέγεται useContext και θέτει όσες μεταβλητές οριστούν ως global σε όλη την εφαρμογή.

Για την φωτογραφεία του χρήστη χρησιμοποιήθηκε η ίδια βιβλιοθήκη με την οθονη ανάρτησης φωτογραφίας (‘expo-image-picker’) αλλά αυτή την φορά δεν αποθηκεύονται στον φάκελο με τον κωδικό ανάρτησης αλλά στον φάκελο με τις φωτογραφίες των χρηστών πιο συγκεκριμένα στον φάκελο με τον κωδικό του συνδεδεμένου χρήστη. Μετά το ανέβασμα της εικόνας κάθε φορά που ο χρήστης μεταβαίνει στην οθονη η πρώτη μέθοδος που καλείται ελέγχει αν υπάρχει ήδη φωτογραφία για να εμφανίσει αλλιώς εμφανίζει μια προεπιλεγμένη.

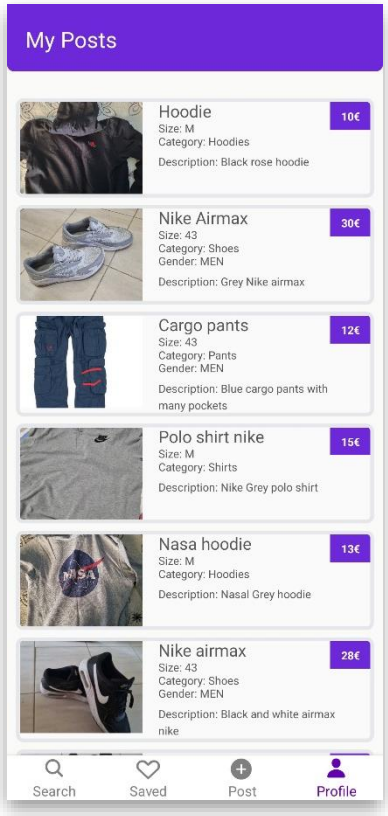
Μέσω του κουμπιού My posts ο χρήστης πλοηγείτε στην οθονη MyPostsListsScreen.js στην οποία υπάρχουν όλες οι αναρτήσεις που έγιναν από τον συνδεδεμένο χρήστη. Και από εκεί μπορεί να γίνει επεξεργασία των αναρτήσεων.

Η επιλογή Dark Mode αλλάζει όλη την εμφάνιση της εφαρμογής στη σκούρη έκδοση. Για αυτή τη λειτουργία χρησιμοποιήθηκε η λειτουργία της βιβλιοθήκης Native Base η useColorMode. Για να λειτουργήσει το Dark Mode σε κάθε γραφικό σε όλη την εφαρμογή χρησιμοποιήθηκαν δυο τρόποι εμφάνισης οι οποίοι ορίζονται κατά την φόρτωση ανάλογα με την τιμή της global μεταβλητής ColorMode.

Τέλος υπάρχει και το κουμπί αποσύνδεσης το οποίο αποσυνδέει τον χρήστη και η εφαρμογή μεταβαίνει στην αρχική της κατάσταση δηλαδή στην οθονη εγγραφής. Για την αποσύνδεση χρησιμοποιείται η Firebase Authentication και η μέθοδος sigOut().

```
const signOutUser = () => {
  firebase
    .auth()
    .signOut()
    .then(() => { navigation.replace('SignUp') })
    .catch(err => { alert(err) })
}
```

4.3.15 My Post List Screen



Η MyPostListScreen.js περιέχει μέσα το reusable component List της Helper.js όπως και η SavedScreen.js μόνο που σε αυτή τη περίπτωση στέλνονται διαφορετικοί παράμετροι.

```
<List navigation={navigation} label={'My Posts'} list={'posts'} />
```

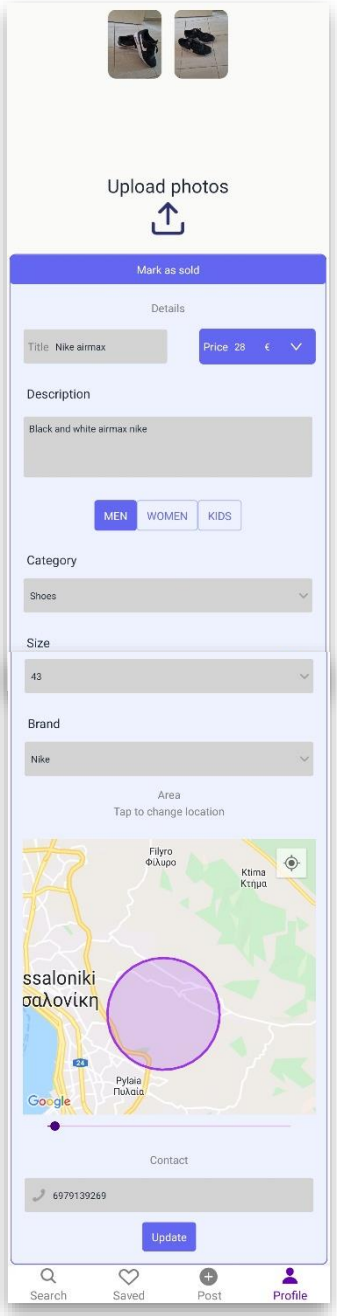
Σε αυτήν την περίπτωση στέλνεται σαν label η συμβολοσειρά ‘My Posts’ η οποία είναι η επικεφαλίδα της οθόνης και σαν list στέλνεται η λέξη post ώστε να γνωρίζει η List πια λίστα θα ζητήσει από τα δεδομένα του χρήστη.

Σε αυτή την ο οθονη ο χρήστης μπορεί να δει όλες τις αναρτήσεις που έχει κάνει και πατώντας πάνω σε μια ανάρτηση μπορεί να την επεξεργαστεί στην οθονη EditPostScreen.js.

Τα δεδομένα ανακτώνται από την Firebase Realtime database μέσω λινκ, βρίσκονται κάτω από τον κωδικό του χρήστη στην λίστα με τις αναρτήσεις.

```
firebase.database().ref('users/' + myContext.UserIdValue + "/" + list + "/");
```

4.3.16 Edit Post Screen



Για να είναι ευκολότερο για τον χρήστη να διαχειριστεί τις δημοσιεύσεις του πέρα από την λίστα δημιουργήθηκε και μια οθονη που δείχνει της λεπτομερείς για κάποια από τις αγγελίες του με την δυνατότητα επεξεργασίας.

Η EditPostScreen.js παίρνει όλα τα δεδομένα από την προηγουμένη οθονη MyPostListScreen.js και τα ξαναπαρουσιάζει όλα στην μορφή που φαίνονται και σε άλλες οθόνες με λεπτομερείς (PostDetailScreen.js) έτσι ώστε να είναι πιο εύκολο στον χρήστη να καταλάβει τι επεξεργάζεται.

Στην ουσία αυτή η οθονη επεξεργάζεται όλες τις πληροφορίες που καταχωθήκαν από τον χρήστη στη ροή ανάρτησης (PostScreen.js – PostScreen6.js) με πιο γρήγορο τρόπο καθώς πρόκειται για επεξεργασία και όχι δημιουργία. Πιο αναλυτικά υπάρχει η δυνατότης διαγράφης των φωτογραφιών και ανάρτησης νέων, δυνατότητα επεξεργασίας όλων των πεδίων και του χάρτη με όλες τις δυνατότητες του δηλαδή αλλαγή τοποθεσίας και αλλαγή ακτίνας. Ακόμη σε όλα τα πεδία υπάρχουν ελέγχει που δεν αφήνουν να επεξεργαστούν λάθος, δηλαδή να μην υπάρχουν κενά σε υποχρεωτικά πεδία , ο αριθμός των ψηφίων στο πεδίο του τηλεφώνου και να υπάρχει τουλάχιστον μια φωτογραφία. Με το κουμπί ‘Update’ η ανάρτηση ενημερώνεται με τις νέες πληροφορίες και ενημερώνει την ανάρτηση οπού αλλού υπάρχει τέλος μεταφέρει τον χρήστη στην οθονη ProfileScreen.js

Επιπρόσθετη λειτουργία αυτής της οθόνης πέρα από την επεξεργασία της ανάρτησης είναι η δήλωση ότι το προϊόν πουλήθηκε. Για να γίνει αυτό υπάρχει ένα κουμπί πάνω από τον τίτλο το οποίο όταν πατηθεί προβάλλεται πρώτα μια ειδοποίηση η οποία ζητάει από τον χρήστη να επιβεβαίωση την ενέργεια του, αυτό γιατί αν συνεχίσει η αγγελία θα διαγραφεί από την γενική βάση που υπάρχουν όλες οι αναρτήσεις Firestore , μετρά από την λίστα του χρήστη που την έκανε και τέλος από οποιαδήποτε άλλη λίστα αλλού χρήστη είτε αυτή είναι λίστα αγαπημένων είτε είναι λίστα με πρόσφατες αναρτήσεις.

```
const markAsSold = () => {
  firebase.firestore().collection('post').doc(item.docId).delete().then(() => {
  const ref = firebase.database().ref('users');
  ref.once('value').then(snapshot => {
    const users = snapshot.val();
    Object.entries(users).forEach ((user) => {
      firebase.database().ref('/users/' + user[0] + "/posts/" + item.docId + '/docId').remove()
      firebase.database().ref('/users/' + user[0] + "/likedPosts/" + item.docId + '/docId').remove()
      firebase.database().ref('/users/' + user[0] + "/recentlyWatched/" + item.docId).remove()
    })
  })
  }).catch((error) => {
    console.error("Error removing document: ", error);
  });
  navigation.goBack();
}
```

4.3.17 Loadings

Το αχρείο Loadings.js περιέχει μέσα πέντε μεθόδους οι οποίοι αντιστοιχούνται σε πέντε τύπους φόρτωσης μιας οθόνης. Η λογική των οθονών φόρτισης είναι να φαίνονται στον χρήστη μέχρι να φορτωθούν τα δεδομένα της οθόνης που πρόκειται να δει καθώς η ανάκτηση ή η προώθηση πληροφοριών στην βάση δεδομένων παίρνει κάποιες φορές ανάλογα την διαδικασία μέχρι και κάποια δευτερόλεπτα. Για αυτόν τον χρόνο λοιπόν προκειμένου να μην υπάρχει κενό χρησιμοποιήθηκαν οθόνες που δίνουν στον χρήστη να καταλάβει ότι πρέπει να περιμένει μέχρι να ολοκληρωθεί κάποια διαδικασία.

Για την κατασκευή των οθονών φόρτισης χρησιμοποιήθηκε το στοιχείο Skeleton της βιβλιοθήκης Native-Base. Αυτό το στοιχείο δίνει την δυνατότητα σχεδίασης ενός κινουμένου UI χωρίς λεπτομερείς, οι οθόνες δημιουργήθηκαν έτσι ώστε το UI που δείχνουν στην οθονη φόρτωσης να είναι παραπλήσιο με αυτό που θα δείξει η τελική οθονη. Αυτό δίνει μια ενδιαφέρουσα αίσθηση στην εφαρμογή που φαίνεται σαν να φορτώνει τα γραφικά στοιχεία σιγά σιγά ενώ στην πραγματικότητα είναι δυο διαφορετικά πράματα που αλλάζουν.

Για να χρησιμοποιηθούν αυτές οι οθόνες θα πρέπει να εισαχθούν στο βασικό αρχείο οθόνης και να χρησιμοποιηθούνε αντί της κανονικής εμφάνισης μέχρι να γίνει η ανάκτηση η προώθηση των πληροφοριών, αυτό πετυχαίνετε χρησιμοποιώντας μια μεταβλητή State (useState ReactNative hook) η οποία συνηθώς ονομάζεται ‘Loading’ και ρόλος της είναι να γίνει ψευδείς όταν θα έχει ολοκληρωθεί η λογική που χρειάζεται για να γίνει προβολή της οθόνης έτσι τελειώνει να προβάλετε η οθονη φόρτωσης και προβάλλεται η πραγματική οθονη.

Στο αρχείο Loadings,js υπάρχουν πέντε οι μέθοδοί DetailLoadinScreen.js η οποία χρησιμοποιείτε για την οθονη λεπτομερειών μιας ανάρτησης , GenericLoading χρησιμοποιείτε γενικά οπού δεν υπάρχει μεγάλη αναμονή και αποτελείται απλά από έναν spinner, HomeLoading για την αρχική οθονη, η SavedScreenLoading που χρησιμοποιείτε για τις οθόνες που έχουν λίστες και η LoadingResultsScreen Που χρησιμοποιείται στην οθονη αποτελεσμάτων. Θα μπορούσε ο κώδικας που υπάρχει σε κάθε μέθοδο της Loading.js να χρησιμοποιηθεί απευθείας στα αρχεία της κάθε οθόνης άλλα αυτό θα είχε ως αποτέλεσμα δυο αρνητικά: 1) αρχεία με εκτεταμένο κώδικα
2) πανομοιότυπος κώδικας που θα χρησιμοποιούταν σε πολλά σημεία.

ResultsScreen.js

```
{Loading
  ?
  <VStack _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }} style={{ flex: 1 }}>
    <LoadingResultScreen />
  </VStack>
  :
  <VStack _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }} style={{ flex: 1 }}>
```

LoadingResultsScreen

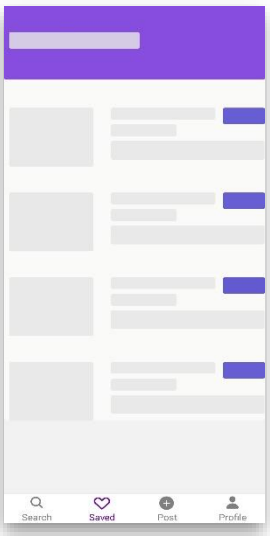
Loadings.js Μέθοδος: LoadingResultsScreen

```
return (
  <Center w="100%" >
    <VStack w="100%" space={1} overflow="hidden" rounded="md"
      _dark={{ bg: '#03203C' }}
      _light={{ bg: 'warmGray.50' }}
    >
      <Skeleton h={windowHeight / 3.5} startColor={colorMode === 'light' ? "violet.700" : '#00E5FF'} />
      <VStack style={{ position: 'absolute' }} space={2} marginLeft={2}>
        <HStack space={20} marginTop={10}>
          <Skeleton size="7" w={windowWidth / 2} rounded="full" />
          <Skeleton size="7" w={windowWidth / 7.9} rounded="full" startColor={colorMode === 'light' ? "indigo.700" : "cyan.700"} />
        </HStack>
        <Skeleton size="4" w={windowWidth / 3} rounded="full" />
        <Skeleton size="4" w={windowWidth / 3.9} rounded="full" />
        <HStack space={150}>
          <Skeleton size="4" w={windowWidth / 3} rounded="full" />
          <Skeleton size="7" w={windowWidth / 4} rounded="full" startColor={colorMode === 'light' ? "indigo.700" : "cyan.700"} />
        </HStack>
      </VStack>
      <LoadingListItem />
      <LoadingListItem />
      <LoadingListItem />
      <LoadingListItem />
    </VStack>
  </Center>
```

SavedListScreem.js MyPostedListScreen.js >Helper.js

```
{Loading
  ?
  <SavedScreenLoading />
  :
  <VStack _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }} style={{ flex: 1 }}>
```

SavedScreenLoading



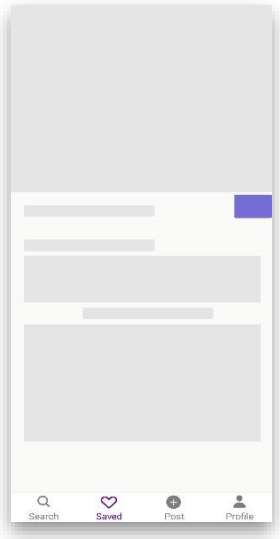
Loadings.js Μέθοδος: SavedScreenLoading

```
return (
  <Center>
    <VStack w="100%" space={1} overflow="hidden" rounded="md"
      _dark={{ bg: '#03203C' }}
      _light={{ bg: 'warmGray.50' }}
    >
      <Skeleton h={windowHeight / 5.5} startColor={colorMode === 'light' ? "violet.700" : '#00E5FF'} />
      <VStack style={{ position: 'absolute' }} space={2} marginLeft={2}>
        <HStack marginTop={20} space={20}>
          <Skeleton size="7" w={windowWidth / 2} rounded="full" />
        </HStack>
      </VStack>
      <LoadingListItem />
      <LoadingListItem />
      <LoadingListItem />
      <LoadingListItem />
    </VStack>
  </Center>
)
```

PostDetailScreen.js

```
{Loading
  ?
  <VStack _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }} style={{ flex: 1 }}>
    <DetailLoadingScreen item={item} />
  </VStack >
  :
```

DetailLoadingScreen



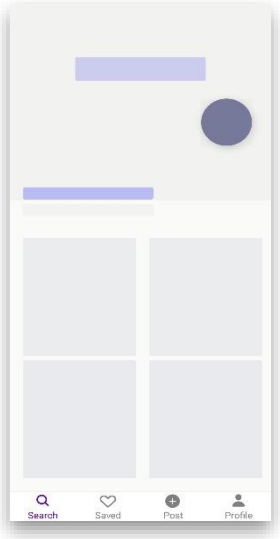
Loadings.js Μέθοδος: DetailLoadingScreen

```
export function DetailLoadingScreen({ item }) {
  const { colorMode, toggleColorMode } = useColorMode();
  return (
    <Center w="100%" >
      <VStack w="100%" space={1} overflow="hidden" rounded="md"
        _dark={{ bg: '#03203C' }}
        _light={{ bg: 'warmGray.50' }}
      >
        <Skeleton h={windowHeight / 2.5} />
        <Stack space={2}>
          <Skeleton size="10" startColor={colorMode === 'light' ? "indigo.700" : "cyan.700"} rounded="full" w={60} style={{ alignSelf: 'flex-end' }} />
          <Skeleton size="5" w={windowWidth / 2} rounded="full" top={-30} marginLeft={5} />
          {item.brand
            ?
            <Skeleton size="5" w={windowWidth / 3} rounded="full" marginLeft={5} top={-30} startColor={colorMode === 'light' ? "indigo.700" : "cyan.700"} />
            :
            null
          }
          <Skeleton size="5" w={windowWidth / 2} rounded="full" marginLeft={5} />
          <Skeleton size="5" w={windowWidth / 1.1} h={20} rounded="full" marginLeft={5} />
          <Center>
            <Skeleton size="5" w={windowWidth / 2} rounded="full" marginLeft={5} />
          </Center>
          <Skeleton size="5" w={windowWidth / 1.1} h={200} rounded="full" marginLeft={5} />
        </Stack>
      </VStack>
    </Center>
  )
};
```

HomseScreen.js

```
{Loading
  ?
  <VStack _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }} style={{ flex: 1 }}>
    <HomeLoading />
  </VStack>
  :
  <Box flex={1} _dark={{ bg: '#03203C' }} _light={{ bg: 'warmGray.50' }}>
```

HomeLoading



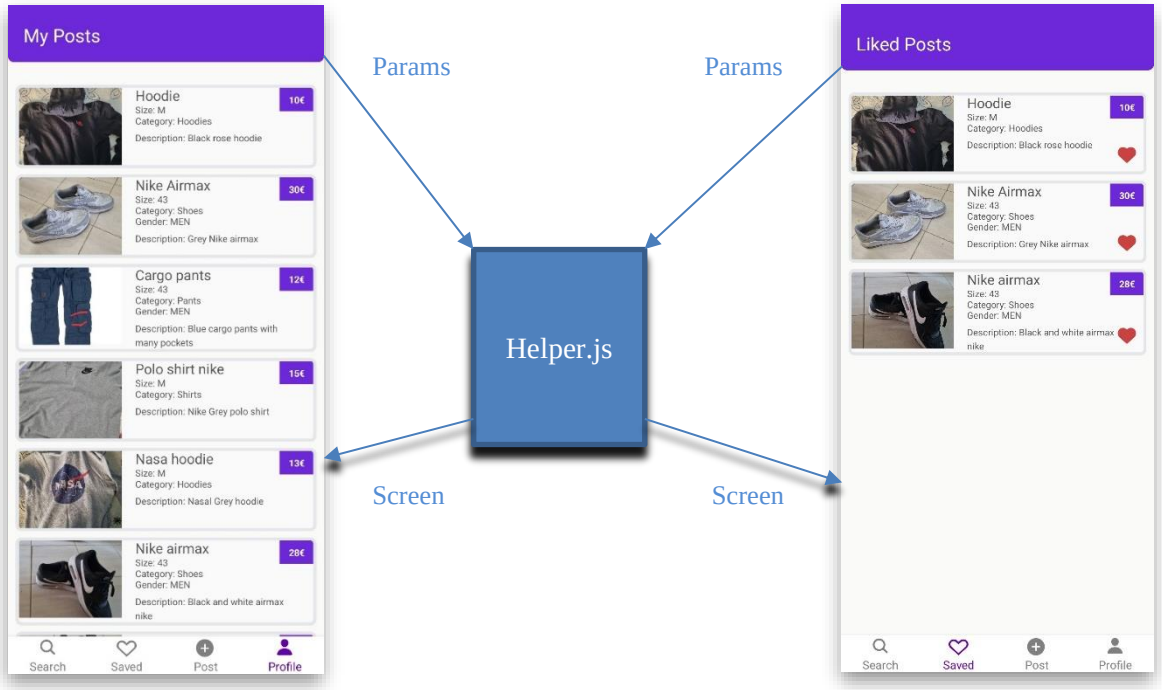
Loadings.js Μέθοδος: HomeLoading

```
export function HomeLoading() {
  const { colorMode, toggleColorMode } = useColorMode();
  return (
    <Box w="100%" flex={1}>
      <VStack w="100%" rounded="md"
        space={2}
        _dark={{ bg: '#03203C' }}
        _light={{ bg: 'warmGray.50' }}
      >
        <Center>
          <Skeleton style={{ position: 'absolute', top: windowHeight / 7 }} startColor={colorMode === 'light' ? "indigo.500" : "cyan.500"} size="5" w={windowWidth / 2} rounded
        </Center>
        <Skeleton size="10" style={colorMode === 'light' ? styles.plusButtonLight : styles.plusButtonDark} rounded="full" w={80} />
        <Skeleton h={windowHeight / 2.5} />
        <Stack space={2}>
          <Skeleton startColor={colorMode === 'light' ? "indigo.500" : "cyan.500"} size="5" w={windowWidth / 2} rounded="full" top={-30} marginLeft={5} />
          <Skeleton size="5" w={windowWidth / 2} rounded="full" top={-30} marginLeft={5} />
          <HStack>
            <Skeleton startColor={colorMode === 'light' ? "coolGray.300" : "indigo.500"} size="5" w={windowWidth / 2.3} h={200} rounded="full" marginLeft={5} />
            <Skeleton startColor={colorMode === 'light' ? "coolGray.300" : "indigo.500"} size="5" w={windowWidth / 2.3} h={200} rounded="full" marginLeft={5} />
          </HStack>
          <HStack>
            <Skeleton startColor={colorMode === 'light' ? "coolGray.300" : "indigo.500"} size="5" w={windowWidth / 2.3} h={200} rounded="full" marginLeft={5} />
            <Skeleton startColor={colorMode === 'light' ? "coolGray.300" : "indigo.500"} size="5" w={windowWidth / 2.3} h={200} rounded="full" marginLeft={5} />
          </HStack>
        </Stack>
      </VStack>
    </Box>
  )
}
```

4.3.18 Helper

Η Helper.js λειτουργεί σαν ένα επαναχρησιμοποιήσιμο στοιχείο που δέχεται τρία ορίσματα navigation , label και list. Την εκμεταλλεύονται δυο σχεδόν ιδίες οθόνες η SavedListScreen.js και η MyPostsListScreen.js οπού η κάθε μια στέλνει διαφορετικά ορίσματα. Η Helper.js στην ουσία υλοποιεί μια λογική και εμφανίζει ένα UI που χωρίς αυτήν την συνάρτηση θα έπρεπε να γραφτούν δυο φορές με αποτέλεσμα να υπάρχει επαναλαμβανόμενος κώδικας.

Στην ουσία αυτό που κάνει αυτό το στοιχείο είναι να καλεί την Firebase Realtime και να παίρνει μια λίστα την οποία και παρουσιάζει. Οι παράμετροι βοηθούν στην διαδικασία δίνοντας τις πληροφορίες που χρειάζεται για να ξεχωρίσει τις δυο περιπτώσεις. Δηλαδή με την παράμετρο label να εμφανίσει διαφορετική επικεφαλίδα, με την παράμετρο list να καλεί διαφορετική λίστα από την βάση τέλος η παράμετρος navigation χρειάζεται για να γίνει η πλοήγηση του χρήστη σε άλλη οθονη.



SavedList καλεί την List της Helper.js

```
import React from "react"
import { List } from "../components/Helper";
export default SavedScreen = ({ navigation }) => {
  return (
    <List navigation={navigation} label={'Liked Posts'} list={'likedPosts'} />
  )
}
```

MyPostedList καλεί την List της Helper.js

```
import React from "react"
import { List } from "../components/Helper";
export default MyPostedListScreen = ({ navigation }) => {
  return (
    <List navigation={navigation} label={'My Posts'} list={'posts'} />
  )
}
```

Helper.js

```
export function List({ navigation, label, list }) {
```

Χρήση της παραμέτρου list στην getData της List για την ανάκτηση των δεδομένων.

```
const getData = () => {
  let records = []
  var starCountRef = firebase.database().ref('users/' + myContext.UserIdValue + "/" + list + "/");
  starCountRef.on('value', (snapshot) => {
    if (snapshot.val()) {
      records = []
      const data = snapshot.val();
      (Object.values(data)).forEach(x => {
        firebase.storage().ref('photos/' + x.docId.id + '/0')
          .getDownloadURL().then((url) => {
            x.image = url
          })
        records = [...records, x.docId]
        setSavedData(records)
      })
    } else {
      setSavedData([])
    }
  })
}
```

Η παράμετρος label που καθορίζει την επικεφαλίδα της οθόνης.

```
<Text
  fontWeight="500" ml="-0.5" mt="5" fontSize="2xl" marginLeft={5}
  _light={{ color: "white", }}
  _dark={{ color: "white", }}
>
  {label}
</Text>
```

Χρήση της παραμέτρου navigation και της παραμέτρου list με τις οποίες αποφασίζεται σε πια οθονη θα προηγηθεί στη συνέχεια ο χρήστης.

```
const navigationCondition = (item) => {
  {
    list === "likedPosts"
    ?
    navigation.navigate('PostDetailScreen', { item, mode: "noSearch" })
    :
    navigation.navigate('EditPostScreen', { item })
  }
}
```

4.3.19 App Context

Το αρχείο AppContext.js το μόνο που κάνει είναι να αρχικοποίησή και να δημιουργεί συνδέσμους για κάποιες μεταβλητές προς όλα τα άλλα στοιχεία. Δηλαδή χρησιμοποιώντας το Hook της React Native useContext ο προγραμματιστής έχει την δυνατότητα να καλεί κάποιες μεταβλητές από οποιαδήποτε οθονη της εφαρμογής.

4.3.20 App

Η App.js είναι η αρχή της εφαρμογής είναι η πρώτη οθονη που καλείται δεν έχει γραφικά, αυτό που κάνει είναι αρχικοποίησή την βάση με τα στοιχεία του χρήστη να δηλώνει κάποιες global μεταβλητές που θα χρησιμοποιηθούν από την AppContext.js και να καλεί τον πρώτο navigator χρησιμοποιώντας ένα επιπλέον στοιχείο την NtaivePaseProvider που βοηθάει στην εμφάνιση της εφαρμογής.

4.3.21 App.json & Google API

Το αρχείο App.json δεν είναι κάποια οθονη ούτε κατέχει κάποια λογική που εφαρμόζετε σε κάποια διαδικασία της εφαρμογής ο ρόλος του είναι να ρυθμίζει κάποια κομμάτια της εφαρμογής πριν γίνει η εκκίνηση της όπως για παράδειγμα την οθονη φόρτισης και την συνδεσιμότητα με κάποιο εξωτερικό API.

Για κάθε χάρτη που χρησιμοποιήθηκε σε όλο το εύρος της εφαρμογής παροχή της υπηρεσίας ήταν η βιβλιοθήκη react-native-maps και η google. Για την λειτουργία των χαρτών θα έπρεπε να συνδεθεί η εφαρμογή με τις υπηρεσίες της google (google cloud platform) για να γίνει αυτό χρειάστηκε να φτιαχτεί μια εφαρμογή στην πλατφόρμα της google στη οποία καταχωρήθηκαν στοιχεία της εφαρμογής όπως το package name και το SH1 certificate fingerprint της εφαρμογής. Ακόμη έπρεπε να ενεργοποιηθούν τα API google maps SDK for android και google maps SDK for IOS. Αυτή η διαδικασίας είχε ως αποτέλεσμα την δημιουργία του API key, το οποίο χρησιμοποιήθηκε στην συνέχεια από την μεριά της εφαρμογής στο αρχείο App.json ώστε να γίνει η σύνδεση και να έχει άδεια η εφαρμογή να χρησιμοποιεί την υπηρεσία της google.

5 Αποτελέσματα

5.1 Συμπεράσματα

Καθώς ολοκληρώνεται η παρούσα πτυχιακή εργασία έχει δημιουργηθεί μία εφαρμογή για τις δύο δημοφιλέστερες πλατφόρμες φορητών συσκευών. Η συγκεκριμένη πτυχιακή συμβάλλει στην ευκολότερη πώληση και αγορά ενδυμάτων σε ατομικό επίπεδο με την χρήση ενός κινητού τηλεφώνου που στις μέρες μας όλοι έχουν.

Η συγκεκριμένη εφαρμογή προσφέρει την δυνατότητα σε όλους όσους έχουν την διαθέσει να εξοικονομήσουν χρήματα για την αγορά ρούχων ή να εκμεταλλευτούν με οικονομικό αντάλλαγμα τα ρούχα που δεν χρειάζονται πια.

Η γλώσσα που χρησιμοποιήθηκε για την δημιουργία της πτυχιακής είναι React Native καθώς δίνει τη δυνατότητα με έναν μόνο κώδικα να δημιουργηθεί μία εφαρμογή και για τις δύο πλατφόρμες. Η βάση δεδομένων που χρησιμοποιήθηκε είναι η Firebase με στόχο την ευκολία διαχείρισης των δεδομένων από τρίτους. Τέλος χρησιμοποιήθηκε η Native Base για να δώσει στην εφαρμογή ένα μοντέρνο και ταυτόχρονα κατανοητό design.

5.2 Μελλοντική Εργασία και Επεκτάσεις

Καμία εφαρμογή που βρίσκεται αυτή τη στιγμή εγκατεστημένη στα κινητά τηλεφωνά όλων μας δεν φτιάχτηκε από την αρχή έτσι όπως την βλέπουμε σήμερα, όλες πέρασαν στάδια και σε κάθε στάδιο υπήρχαν ιδέες για την εξελίσσει ή την βελτιστοποίηση των παροχών. Έτσι και για αυτή την εφαρμογή υπάρχουν σχέδια που μπορούν να την ακόμα καλύτερη και πι ο φιλική στην προς στον χρήστη.

1) Περισσότερα στοιχεία χρήστη

   Για την καλύτερη επικοινωνία των χρηστών θα μπορούσαν να υπάρχουν περισσότερες πληροφορίες για τους χρήστες στις οθόνες των αναρτήσεων όπως για παράδειγμα ονοματεπώνυμο, email και φωτογραφία προφίλ.

2) Ενσωματωμένο σύστημα chater

   Στα πλαίσια της βελτιστοποίησης της επικοινωνίας μεταξύ αγοραστή και πωλητή θα μπορούσε να δημιουργηθεί ένας σύστημα ανταλλαγής μνημάτων μέσα στην εφαρμογή έτσι ώστε ο αγοραστής να μην είναι υποχρεούμενος να τηλεφωνήσει για να μάθει λεπτομερείς.

3) Facebook Login

   Για την γρηγορότερη σύνδεση του χρήστη θα μπορούσε να χρησιμοποιηθεί το API του Facebook (Facebook Dev) σε συνδυασμό με την Firebase που θα επέτρεπε στον χρήστη να συνδέεται στην εφαρμογή χρησιμοποιώντας τον λογαριασμό του στο Facebook. Ακόμη θα έδειχνε την δυνατότητα να γίνει η επικοινωνία μέσω του chating του Facebook Messenger που θα άνοιγε κατευθείαν από την εφαρμογή. Τέλος με αυτήν την αναβάθμιση θα μπορούσε να υπάρχει μια λίστα στο προφίλ του χρήστη που θα του έδειχνε τους φίλους του στο Facebook που χρησιμοποιούν και αυτοί την εφαρμογής.

4) Σύστημα δημοπρασίας

   Πολλές φορές οι πωλητές βρίσκονται στην δύσκολη θέση να μην ξέρουν σε ποιον θα πρέπει να δώσουν το προϊόν τους αφού οι ενδιαφερόμενοι ενδιαφέρθηκαν την ιδιά χρονική στίμη. Για αυτό το πρόβλημα θα μπορούσε να δημιουργηθεί ένα σύστημα δημοπρασίας το οποίο θα ενεργοποιούσε πωλητής για κάποιο προϊόν και θα έβαζε τους αγοραστές σε μια διαδικασία ώστε να κερδίσουν το προϊόν δίνοντας παραπάνω χρήματα.

5) Σύστημα ειδοποιήσεων

   Για να γίνει η εφαρμογή πιο φιλική προς τον χρήστη θα μπορούσε να ενταχθεί στις λειτουργίες της η βιβλιοθήκη expo-notifications με την οποία θα μπορούσε η εφαρμογής να ενημερώνει τους χρήστες για μηνύματα, αγορές ή και ακόμα για προϊόντα που έχουν βάλει στις λίστες τους

6) Σύστημα απομακρυσμένης πληρωμής

   Σε αυτή την εκδώσει της εφαρμογής για να γίνει ολοκλήρωση μιας αγοράς θα πρέπει οι δυο πλευρές να συναντηθούν και να κάνουν την συναλλαγή. Μια ακόμα χρήσιμη εξέλιξη του συστήματος θα ήταν να δημιουργηθεί ροή που μεταφέρει τον αγοραστή σε κάποιο τραπεζικό σύστημα με τα στοιχεία του πωλητή έτσι ώστε να μπορέσει να του στείλει τα χρήματα.

Βιβλιογραφία

https://reactnativeexample.com/a-customizable-react-native-component-that-implements-an-image-slider-ui/

https://www.codegrepper.com/code-examples/javascript/find+the+distance+between+two+coordinates+javascript

https://github.com/react-native-maps

https://github.com/ptomasroos/react-native-multi-slider

https://docs.nativebase.io/?utm_source=HomePage&utm_medium=header&utm_campaign=NativeBase_3

https://icons.expo.fyi/

https://reactnative.dev/docs/components-and-apis

https://docs.expo.dev/

https://www.wikiwand.com/en/Android_(operating_system)

https://www.wikiwand.com/en/IOS

https://www.wikiwand.com/en/Firebase

https://www.wikiwand.com/en/React_Native

https://www.wikiwand.com/en/JavaScript

https://www.wikiwand.com/en/Android_Studio

https://www.wikiwand.com/en/Xcode

https://www.wikiwand.com/en/Cloud_database