ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και Υλοποίηση κοινωνικού δικτύου που επιτρέπει στους χρήστες να μοιράζονται την τοποθεσία τους με άλλους χρήστες σε περιπτώσεις κινδύνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Της

Ελπίδας Ν. Ρουκά

Επιβλέπων: Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και Υλοποίηση κοινωνικού δικτύου που επιτρέπει στους χρήστες να μοιράζονται την τοποθεσία τους με άλλους χρήστες σε περιπτώσεις κινδύνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Της

Ελπίδας Ν. Ρουκά

Επιβλέπων: Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Ιουλίου 2017.

.....	.....	.....
Ι. Βενιέρης	Δ.Θ. Κακλαμάνη	Γ. Ματσόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2017

.....

Ελπίδα Ν. Ρουκά

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι έξυπνες φορητές συσκευές αποτελούν πλέον αναπόσπαστο κομμάτι της καθημερινότητας μας. Σε αυτό έχει συμβάλλει η πληθώρα των εφαρμογών που έχουν αναπτυχθεί και αφορούν κάθε τομέα της ανθρώπινης δραστηριότητας.

Στόχος αυτής της διπλωματικής είναι η ανάπτυξη μιας εφαρμογής που λειτουργεί ως κοινωνικό δίκτυο, βοηθώντας το χρήστη να αποφύγει ή να αντιμετωπίσει περιπτώσεις κινδύνου. Η εφαρμογή προορίζεται για συσκευές με λειτουργικό σύστημα iOS, δηλαδή μπορεί να λειτουργήσει σε iPhone, iPod και iPad.

Η εφαρμογή δίνει τη δυνατότητα στο χρήστη να δημιουργεί σήματα κινδύνου μόλις βρεθεί σε επείγουσες καταστάσεις, ενημερώνοντας έτσι τους φίλους του για το που βρίσκεται και για το τι ακριβώς του συνέβη. Τα δεδομένα αυτά συλλέγονται και εξάγουν πληροφορίες σχετικά με την επικινδυνότητα κάθε περιοχής, στις οποίες οι χρήστες έχουν πρόσβαση μόλις βρεθούν στην εκάστοτε περιοχή. Όλα αυτά συμβαίνουν στα πλαίσια ενός κοινωνικού δικτύου που παρέχει λειτουργίες αιτημάτων φιλίας και δημιουργίας/επεξεργασίας δημοσιεύσεων, οι οποίες ουσιαστικά αντιπροσωπεύουν τα σήματα κινδύνου.

Επιπλέον δίνεται στο χρήστη η δυνατότητα να προσδιορίζει κάποιο χρονικό διάστημα για το οποίο κάποιος φίλος του θα μπορεί να παρακολουθεί ζωντανά τη θέση του. Το χρονικό αυτό διάστημα προσδιορίζεται είτε άμεσα από το χρήστη είτε έμμεσα με την επιλογή κάποιου προορισμού. Η συγκεκριμένη λειτουργία είναι καίρια, ειδικά αν ληφθούν υπ' όψιν οι κίνδυνοι τους οποίους αντιμετωπίζουν οι άνθρωποι σήμερα στις σύγχρονες μεγαλουπόλεις.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε ένας υπολογιστής MacBook Pro της Apple, με λειτουργικό σύστημα macOS Sierra (Version 10.12.4), και συγκεκριμένα το εργαλείο Xcode 8. Για τη δοκιμή της εφαρμογής χρησιμοποιήθηκε ένα κινητό τηλέφωνο iPhone 5 και ο προσομοιωτής του Xcode σε iPhone 7. Για τη βάση δεδομένων και το server χρησιμοποιήθηκε η πλατφόρμα Google Firebase.

Λέξεις κλειδιά: iOS, iPhone, εφαρμογή, κοινωνικό δίκτυο, κίνδυνος, NoSQL βάση, υπηρεσίες τοποθεσίας, τοπικές ειδοποιήσεις, Firebase.

Abstract

Nowadays, smart and portable devices are an integral part of our everyday lives. What has contributed to this, is the great number of applications that have been developed and are related to every sector of human activity.

The purpose of this diploma thesis is to develop an application which functions as a social network and helps the user avoid or deal with dangerous situations. The application is designed for devices that run the iOS operating system, which are iPhone, iPod and iPad.

The application enables the user to create emergency signals and share them with his friends, informing them about his location and declaring what happened to him. These data are collected and produce information about how dangerous each area is. This information becomes available to the user when he enters each specific area. All this happens inside a social network, which provides the basic functions of friend requests, as well as the creation and editing of posts that represent the emergency signals.

Furthermore, the user can specify a time interval, during which a friend can observe his location in real-time. This time interval can be specified directly from the user or indirectly by specifying his destination. This function is of great significance, considering the dangers people confront in big cities nowadays.

The application was developed on an Apple MacBook Pro, running the operating system macOS Sierra (Version 10.12.4). The tool Xcode 8 was used. For testing puproses, I used an iPhone 5 device as well as the iPhone 7 Xcode Simulator. The Google Firebase platform was used for the database and the server.

Keywords: iOS, iPhone, application, social network, danger, NoSQL Database, location based services, local notifications, Firebase.

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής, την ανάπτυξη και σχεδίαση της εφαρμογής και γενικά για την ολοκλήρωση του κύκλου σπουδών μου νιώθω την ανάγκη να ευχαριστήσω κάποια άτομα των οποίων η βοήθεια υπήρξε καθοριστική.

Αρχικά θέλω να ευχαριστήσω θερμά τον επιβέποντα της διπλωματικής, καθηγητή Ιάκωβο Βενιέρη, για την πολύτιμες γνώσεις και συμβουλές που μου έδωσε. Κυρίως όμως θέλω να τον ευχαριστήσω για την ευκαιρία που μου προσέφερε να ασχοληθώ με ένα τελείως καινούριο αντικείμενο το οποίο με ενέπνευσε και ελπίζω να το ακολουθήσω και στη μετέπειτα σταδιοδρομία μου.

Επίσης, θέλω να ευχαριστήσω τα μέλη της επιτροπής, την καθηγήτρια Δήμητρα Θεοδώρα Κακλαμάνη και τον αναπληρωτή καθηγητή Γεώργιο Ματσόπουλο για την υποστήριξή τους κατά τη διάρκεια εκπόνησης της εργασίας και τη σημαντική συμβολή τους στην περαίωσή της.

Ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω στους υποψήφιους διδάκτορες Πέτρο-Φλώριο Μπάκαλο και Εμμανουήλ Καραμανή, χωρίς τους οποίους θα ήταν αδύνατη η ολοκλήρωση της διπλωματικής. Η συμβολή τους στην εξέλιξη και περάτωσή της υπήρξε καθοριστική. Παράλληλα θα ήθελα να ευχαριστήσω όλο το προσωπικό του Εργαστηρίου Ευφυών Επικοινωνιών και Δικτύων Ευρείας Ζώνης που μου προσέφερε τη βοήθειά του όποτε αυτή ήταν απαραίτητη.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου, που με στήριξε όλους αυτούς του μήνες και πίστεψε στις δυνατότητές μου, αλλά και την Άντα, την Ελένη, τη Δήμητρα και τον Αντρέα χωρίς τους οποίους οι σπουδές μου δεν θα είχαν τελειώσει τόσο ομαλά, ούτε τόσο σύντομα.

Πίνακας περιεχομένων

Εισαγωγή

1.1 Έξυπνες συσκευές - Internet of Things

Η ραγδαία εξέλιξη των κινητών τηλεφώνων ξεκίνησε όταν ο Martin Cooper, μηχανικός της Motorola και πρωτοπόρος των ασύρματων επικοινωνιών, εφηύρε το 1983 το πρώτο κινητό τηλέφωνο, το Motorola DynaTAC 8000X. Δεκαεπτά χρόνια αργότερα, το 2000, κυκλοφόρησε στην αγορά το Ericsson R380, η πρώτη συσκευή που συνδύαζε δυνατότητες κινητού τηλεφώνου και PDA και χαρακτηρίστηκε ως Smartphone. Η εξάπλωση των iPhone ξεκινά στις 9 Ιανουαρίου του 2007, όταν ο σχεδιαστής και εφευρέτης της Apple Inc. Steve Jobs, ανακοίνωσε την επικείμενη κυκλοφορία του πρώτου iPhone. Από τότε, η δραματική διάδοση των έξυπνων τηλεφώνων έχει οδηγήσει σε 2.32 δισεκατομμύρια χρήστες παγκοσμίως το 2017, με μία πρόβλεψη 6.1 δισεκατομμυρίων χρηστών για το 2020, αριθμός που αντιστοιχεί στο 70% του παγκόσμιου πληθυσμού.

Τα έξυπνα τηλέφωνα έχουν ορισμένα βασικά χαρακτηριστικά, όπως δυνατότητα λήψης φωτογραφιών, αποστολή γραπτών και ηχητικών μηνυμάτων, ημερολόγιο, Media Player, GPS και προσωπικούς ψηφιακούς βοηθούς (π.χ. Siri, Google Assistant). Αυτό όμως που τα ξεχωρίζει είναι η δυνατότητα εγκατάστασης από το χρήστη λογισμικού (Εφαρμογών) μέσω του Google Play Store και του Apple App Store, καθώς και οι δυνατότητες ασύρματης επικοινωνίας μέσω πρωτοκόλλων όπως Bluetooth, 3G-4G δίκτυα και Wi-Fi. Το αξιοσημείωτο είναι ότι τα τελευταία χρόνια έχουν κυκλοφορήσει, εκτός από κινητά τηλέφωνα, και άλλα είδη έξυπνων συσκευών όπως έξυπνα ρολόγια, tablets, έξυπνα αυτοκίνητα και εν τέλει έξυπνες οικιακές συσκευές που συνθέτουν τα έξυπνα σπίτια (smarthomes).

Άρρηκτα συνδεδεμένη με την εξέλιξη των έξυπνων συσκευών είναι η έννοια του Internet of Things, της διαδικτύωσης δηλαδή φυσικών συσκευών, τεχνολογία που άρχισε να γίνεται δημοφιλής το 1999 μέσω της ερευνητικής ομάδας Auto-ID Labs του MIT. Η βασική ιδέα του Internet of Things είναι η ασύρματη επικοινωνία συσκευών και ενσωματωμένων συστημάτων μέσω λογισμικού, αισθητήρων (sensors) και ενεργοποιητών (actuators) που καθιστούν αυτές τις συσκευές ικανές να συλλέγουν, να επεξεργάζονται και να ανταλλάσσουν δεδομένα αυτόματα. Η τεχνολογία αυτή επιτρέπει επίσης στους χρήστες να χειρίζονται τις έξυπνες συσκευές εξ' αποστάσεως, οδηγώντας σε αποτελεσματικότητα, ακρίβεια, μειωμένη ανάγκη για ανθρώπινη παρέμβαση και τελικά βελτιωμένη εμπειρία χρήστη. Είναι αδιαμφισβήτητο το γεγονός ότι ο σημερινός ρυθμός εξέλιξης του Internet of Things θα οδηγήσει σε έναν τελείως διαφορετικό, πλήρως αυτοματοποιημένο τρόπο ζωής τα προσεχή χρόνια.

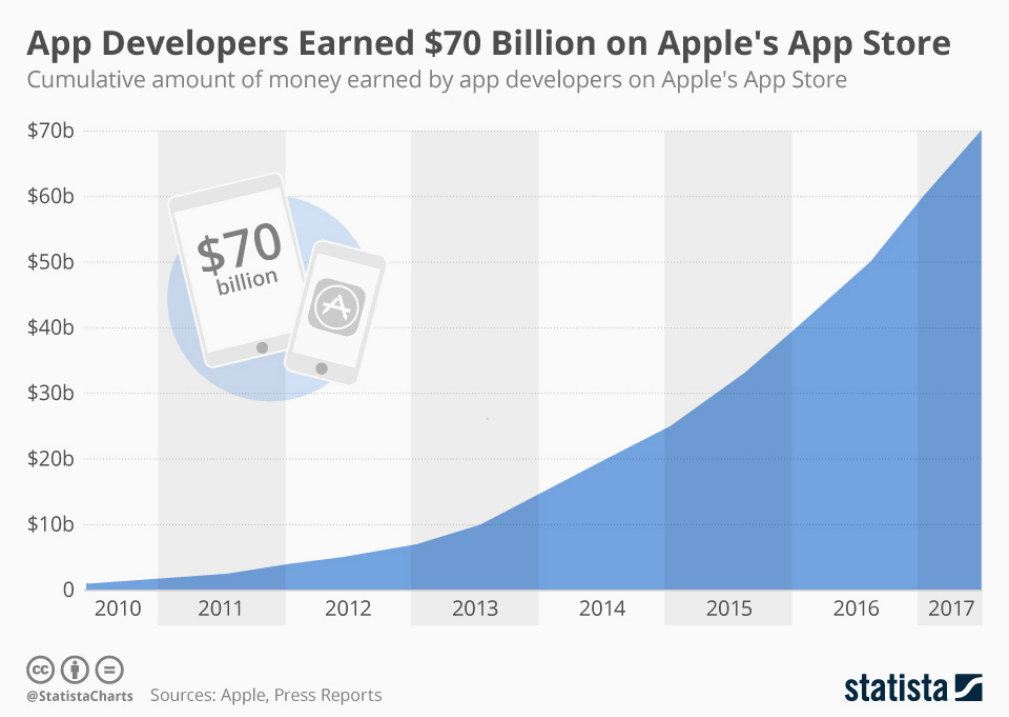
1.2 Λειτουργικά Συστήματα κι Εφαρμογές

Τα τρία βασικά λειτουργικά συστήματα των σύγχρονων έξυπνων τηλεφώνων είναι το iOS της Apple, το Android της Google και το Windows Phone της Microsoft. Μεταξύ αυτών, το πιο

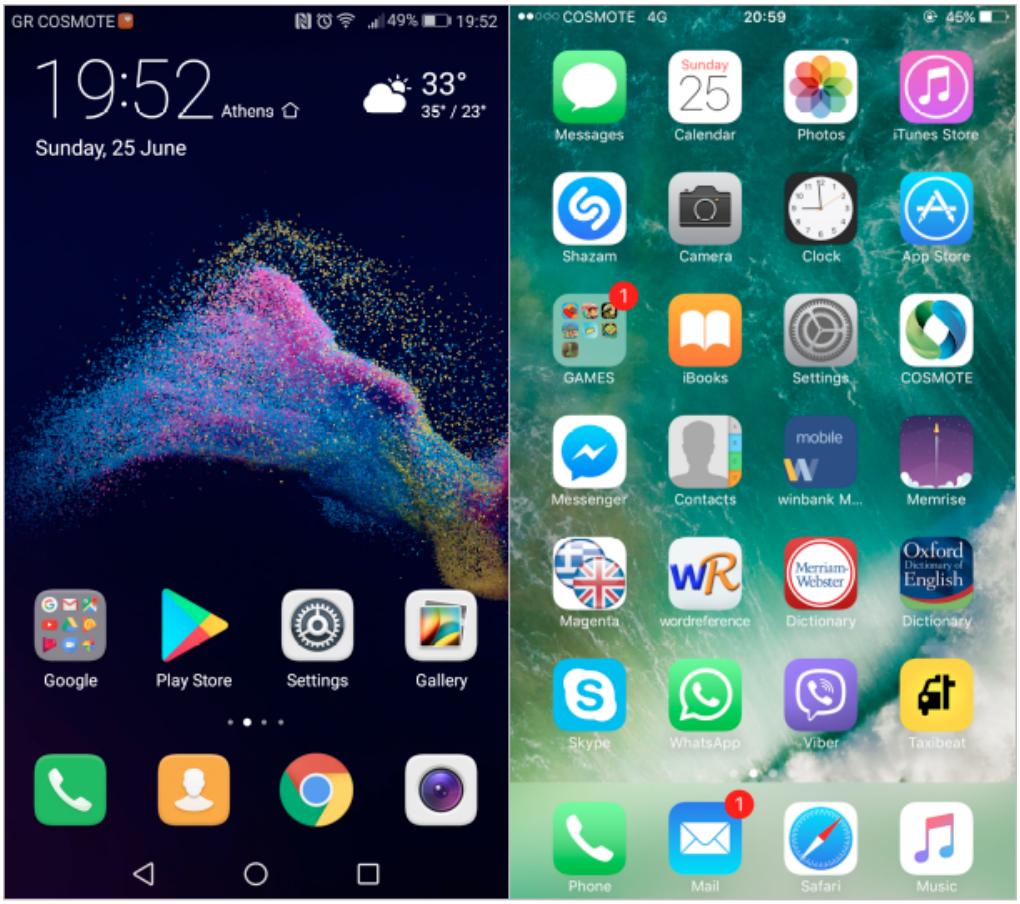
διαδεδομένο με μεγάλη διαφορά είναι το Android, του οποίου οι πωλήσεις φτάνουν τα 380 εκατομμύρια τηλέφωνα για το 2017, καταλαμβάνοντας το 87% της παγκόσμιας αγοράς Smartphones. Ακολουθεί το iOS, με 52 εκατομμύρια πωλήσεις αυτή τη χρονιά και το Windows Phone, λιγότερο δημοφιλές, με λιγότερο από 1 εκατομμύριο πωλήσεις παγκοσμίως φέτος. Σύμφωνα με την εταιρεία ερευνών Statista, παρόμοια συμπεριφορά είχαν οι πωλήσεις αυτών των τριών λειτουργικών από το 2011 μέχρι σήμερα.

Εικόνα 1: Ποσοστά πωλήσεων λειτουργικών συστημάτων Smartphone την περίοδο 2009-2017.

Οι πιο ευρέως χρησιμοποιούμενες εφαρμογές των Smartphones σήμερα περιλαμβάνουν εφαρμογές κλήσης, αποστολής μηνυμάτων, ηλεκτρονικού ταχυδρομείου, πρόβλεψης καιρού, περιήγησης ιστού, λήψης φωτογραφιών, παιχνιδιών και αναπαραγωγής/ επεξεργασίας πολυμέσων. Ιδιαίτερα δημοφιλείς πλέον είναι και οι εφαρμογές instant messaging και κοινωνικής δικτύωσης. Οι περισσότερες από αυτές προσφέρονται δωρεάν στους χρήστες, όμως πολλές πιο εξελιγμένες απαιτούν ένα αντίτιμο που μπορεί να κυμαίνεται από λίγα cents μέχρι εκατοντάδες ευρώ, ανάλογα με την πολυπλοκότητα της εφαρμογής. Μάλιστα, από την σύλληψη του Apple App Store το 2008 μέχρι σήμερα, οι πωλήσεις εφαρμογών iOS ανέρχονται αθροιστικά στα 70 δισεκατομμύρια δολλάρια.



Εικόνα 2: Αθροιστικά κέρδη προγραμματιστών στο Apple App Store την περίοδο 2010-2017.



Εικόνα 3: Οι αρχικές οθόνες των λειτουργικών συστημάτων Android και iOS. Αριστερά το Android 7.0 στο Huawei P9 lite. Δεξιά το iOS 10 στο iPhone 7.

Είναι γεγονός ότι οι διαρκώς αυξανόμενες ανάγκες και απαιτήσεις των χρηστών τροφοδοτούν τις παραπάνω τάσεις και καθιστούν το επάγγελμα του μηχανικού λογισμικού, και ειδικά εφαρμογών, όλο και περισσότερο προσοδοφόρο. Οι εφαρμογές που έχουν κυκλοφορήσει στην αγορά σχετίζονται πλέον με κάθε πτυχή της ζωής μας, από την καθημερινή ζωή, την οδήγηση, τη μόρφωση και τη συλλογή πληροφοριών μέχρι την άμεση επικοινωνία, τη διασκέδαση καθώς και κάθε είδους δραστηριότητα. Λαμβάνοντας υπ' όψιν ότι οι διαθέσιμες εφαρμογές ανέρχονταν σε δύο εκατομμύρια διακόσιες χιλιάδες (2200000) στο Apple App Store και σε δύο εκατομμύρια οχτακόσιες χιλιάδες (2800000) στο Google Play Store στις αρχές του 2017, είναι εμφανής ο ανταγωνισμός και η εξέλιξη που υπάρχουν σήμερα στον τομέα ανάπτυξης mobile εφαρμογών.

1.3 Υπηρεσίες βάσει τοποθεσίας

Οι υπηρεσίες βάσει τοποθεσίας (Location Based Services, LBS) είναι ένα σύνολο υπηρεσιών λογισμικού που χρησιμοποιούν την τοποθεσία του χρήστη για να συλλέξουν και να επεξεργαστούν ορισμένα δεδομένα. Τέτοιου είδους υπηρεσίες χρησιμοποιούνται εκτεταμένα σήμερα στα κοινωνικά δίκτυα, σε εφαρμογές διασκέδασης, ασφάλειας, καθοδήγησης, προσδιορισμού της κίνησης στους δρόμους αλλά και ξενάγησης σε εσωτερικούς χώρους. Επιπλέον, οι υπηρεσίες αυτές είναι ιδιαίτερα χρήσιμες σε επιχειρήσεις και κυβερνητικούς οργανισμούς που επιθυμούν να εξάγουν πληροφορίες και μοτίβα για τις δραστηριότητες που λαμβάνουν μέρος σε συγκεκριμένες περιοχές και να κατανοήσουν τις σχέσεις μεταξύ τους. Αποτελούν επομένως μια πτυχή της εξόρυξης δεδομένων (Data Mining).

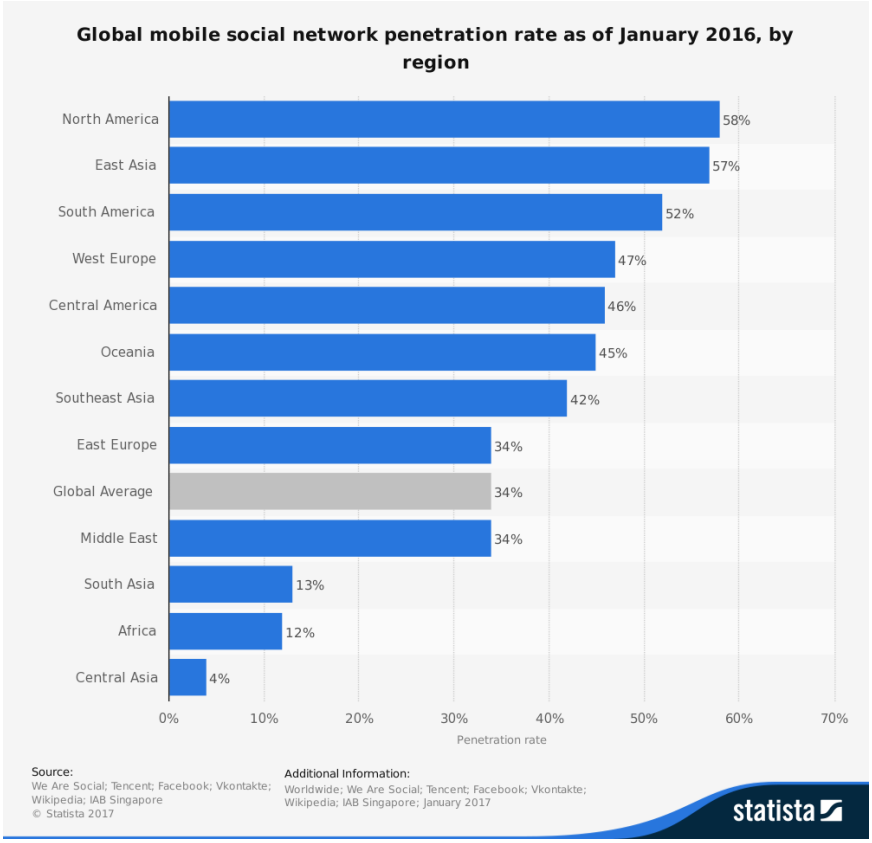
Για να προσδιοριστεί η τοποθεσία του χρήστη μπορούν να χρησιμοποιηθούν διάφορες τεχνικές. Η πιο ακριβής από αυτές είναι το GPS (Global Positioning System, Παγκόσμιο Σύστημα Θεσιθεσίας). Το GPS βασίζεται σε ένα "πλέγμα" εικοσιτεσσάρων δορυφόρων της Γης, εφοδιασμένων με ειδικές συσκευές εντοπισμού, οι οποίες ονομάζονται "πομποδέκτες GPS". Οι πομποδέκτες αυτοί παρέχουν ακριβείς πληροφορίες για τη θέση ενός σημείου, το υψόμετρό του, την ταχύτητα και την κατεύθυνση της κίνησης του. Το σύστημα ξεκίνησε από το Υπουργείο Άμυνας των ΗΠΑ και ονομάστηκε NAVSTAR GPS (Navigation Signal Timing and Ranging Global Positioning System). Το δορυφορικό αυτό σύστημα ρυθμίζεται καθημερινά από τη Βάση Πολεμικής Αεροπορίας Schriever με κόστος 400 εκατομμύρια δολάρια το χρόνο. Η χρήση αυτής της μεθόδου είναι ιδιαίτερα διαδεδομένη σήμερα, που τα περισσότερα smartphones έχουν ενσωματωμένους δέκτες GPS.

Για τα κινητά τηλέφωνα που δεν υποστηρίζουν υπηρεσίες GPS, υπάρχει μέθοδος προσδιορισμού της θέσης που βασίζεται στα δίκτυα τηλεφωνίας. Συγκεκριμένα χρησιμοποιείται το Cell ID, ένας μοναδικός αριθμός που προσδιορίζει το Σταθμό Πομποδεκτών Βάσης (Base Transciever Station, BTS) στον οποίο είναι συνδεδεμένο το κινητό εκείνη τη χρονική στιγμή. Από τη στιγμή που αυτό αποφασιστεί, τίθεται ως τοποθεσία του χρήστη η τοποθεσία του BST.

Αξιοσημείωτο είναι το γεγονός ότι, πλέον, οι πλατμόρμες ανάπτυξης λογισμικού και ειδικά αυτές που υποστηρίζουν ανάπτυξη εφαρμογών για smartphones, διαθέτουν APIs (Application Programming Interfaces, Διεπαφές Προγραμματισμού Εφαρμογών) που υποστηρίζουν Location Based Services.

1.4 Κοινωνικά δίκτυα

Τα κοινωνικά δίκτυα (Social Networks) είναι εικονικά δίκτυα που επιτρέπουν σε χρήστες (άτομα ή οργανισμούς) να αναπτύσσουν δεσμούς, να αλληλεπιδρούν και να επικοινωνούν μεταξύ τους. Τα δημοφιλέστερα κοινωνικά δίκτυα σήμερα είναι το Facebook, το Instagram, το Twitter και το Linkedin, με τους χρήστες των κοινωνικών δικτύων να ανέρχονται σε 2.5 δισεκατομμύρια παγκοσμίως το 2017. Το Facebook συγκεκριμένα, το οποίο πλέον χρησιμοποιείται από άτομα όλων των ηλικιών έχει αυτή τη σιγμή 1.7 δισεκατομμύρια ενεργούς χρήστες.



Εικόνα 4: Ποσοστά χρήσης κοινωνικών δικτύων παγκοσμίως τον Ιανουάριο του 2016.

Μορφές επικοινωνίας στα κοινωνικά δίκτυα αποτελούν αποτελούν τα μηνύματα, τα σχόλια, οι δημοσιεύσεις (posts), η μεταφόρτωση φωτογραφιών, και οι δεσμοί που δημιουργούνται μεταξύ των χρηστών συνήθως ονομάζονται “friend” ή “follow”. Ακριβώς επειδή ο αριθμός των χρηστών είναι τόσο μεγάλος και ταυτόχρονα η πληροφορία που δημοσιοποιείται αφορά προσωπικά στοιχεία των χρηστών, πρέπει να δίνεται μεγάλη σημασία σε θέματα ασφάλειας (security) και ιδιωτικότητας (privacy) στα κοινωνικά δίκτυα.

Η ευρεία χρήση των κοινωνικών δικτύων και οι συνεχείς αλληλεπιδράσεις που συμβαίνουν στο πλαίσιο αυτών τα καθιστά αντικείμενο έρευνας των κοινωνικών και ψυχολογικών επιστημών. Οι δεσμοί που αναπτύσσουν τα άτομα μέσω των κοινωνικών δικτύων μπορούν να αποκαλύψουν τάσεις της κοινωνιολογίας, της οικονομίας, του εμπορίου, της υγείας, της κουλτούρας, ακόμη και της εγκληματικότητας. Εξάλλου, τα κοινωνικά δίκτυα είναι ένας από τους κύριους τρόπους διαφήμισης προϊόντων και συνεπώς επηρρεάζουν άμεσα τις επιλογές των καταναλωτών.

1.5 Αντικείμενο διπλωματικής εργασίας

Το αντικείμενο της διπλωματικής εργασίας είναι η σχεδίαση, η ανάπτυξη και η υλοποίηση εφαρμογής που λειτουργεί ως ένα κοινωνικό δίκτυο προσανατολισμένο σε περιπτώσεις κινδύνου. Οι χρήστες έχουν τη δυνατότητα να δημιουργήσουν φιλίες με πολλαπλούς άλλους χρήστες, και να εκπέμψουν “σήμα κινδύνου” στους φίλους τους αν αυτό είναι απαραίτητο, που περιλαμβάνει την ακριβή τοποθεσία τους εκείνη τη στιγμή. Στη συνέχεια, μπορούν να παρέχουν πληροφορίες σχετικά με το περιστατικό που αντιμετώπισαν και να δηλώσουν αν διευθετήθηκε από την αστυνομία. Ταυτόχρονα, καθώς μετακινούνται λαμβάνουν αυτόματα στατιστικά σχετικά με το πόσο ασφαλής είναι η περιοχή στην οποία βρίσκονται. Τέλος, δίνεται η δυνατότητα να μοιράζεται ο χρήστης την τοποθεσία και τον προορισμό του με κάποιο φίλο του για ένα προεπιλεγμένο χρονικό διάστημα.

Η εφαρμογή υλοποιήθηκε στην πλατφόρμα iOS της Apple, με το εργαλείο ανάπτυξης εφαρμογών Xcode. Χρησιμοποιήθηκε το iOS SDK, που περιλαμβάνει όλα τα απαραίτητα πλαίσια (frameworks) για την ανάπτυξη οποιασδήποτε εφαρμογής. Για τον έλεγχο της εφαρμογής χρησιμοποιήθηκε συσκευή iPhone 5 και το εργαλείο iOS Simulator για τον έλεγχο σε πιο σύγχρονα μοντέλα (iPhone 6, 6s, 7). Για τον Server και τη Βάση Δεδομένων χρησιμοποιήθηκε η πλατφόρμα Firebase της Google. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε στο Client-side μέρος της εφαρμογής είναι η αντικειμενοστραφής γλώσσα Swift 3.0 της Apple.

1.6 Διάρθρωση της εργασίας

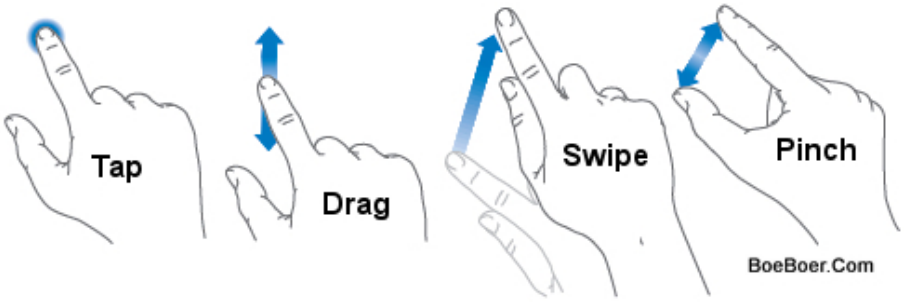
Έγινε προσπάθεια να ακολουθηθούν τα προβλεπόμενα βήματα της τεχνολογίας λογισμικού που αναφέρονται στα επόμενα κεφάλαια (Σχεδίαση, Ανάπτυξη, Υλοποίηση). Στο δεύτερο κεφάλαιο περιγράφονται οι τεχνολογίες που χρησιμοποιήθηκαν. Στο τρίτο κεφάλαιο αναλύονται οι λειτουργικές απαιτήσεις της εφαρμογής και παρουσιάζονται ορισμένα σενάρια χρήσης σε γλώσσα UML. Στο τέταρτο κεφάλαιο παρουσιάζεται η σχεδίαση της εφαρμογής και περιγράφονται οι οθόνες, οι κλάσεις και οι ιδιότητες τους, καθώς και η μορφή της βάσης στο Firebase. Στο πέμπτο κεφάλαιο καταγράφονται τα κυριότερα σημεία του κώδικα και περιγράφεται αναλυτικά ένα σενάριο χρήσης. Τέλος, στο έκτο κεφάλαιο συνοψίζονται οι βασικές λειτουργίες της εφαρμογής και προτείνονται μελλοντικές επεκτάσεις.

Τεχνολογίες

2.1 Apple iOS

Το iOS (προηγουμένως γνωστό ως iPhone OS) είναι ένα λογισμικό για κινητά τηλέφωνα το οποίο αναπτύχθηκε και διανέμεται από την Apple Inc. Αρχικά παρουσίαστηκε το 2007 για το iPhone, ενώ υποστηρίζει και άλλες συσκευές της Apple όπως το iPod touch (Σεπτέμβριος 2007), το iPad (Ιανουάριος 2010) και το Apple TV (δεύτερης γενιάς) (Σεπτέμβριος 2010). Σε αντίθεση με το Windows Phone και το Android, το iOS μπορεί να εγκατασταθεί μόνο σε συσκευές Apple.

Η διεπαφή χρήστη (User Interface) του iOS βασίζεται στον άμεσο χειρισμό μιας οθόνης επαφής, μέσω πολλαπλών χειρονομιών επαφής (multi-touch gestures). Κάποιες από αυτές είναι οι swipe, drag, tap, pinch. Τα αποτελέσματα τέτοιου είδους χειρονομιών ελέγχονται από τον προγραμματιστή μέσω ειδικών συναρτήσεων στο Xcode. Επιπλέον, εσωτερικά επιταχυνσιόμετρα (accelerometers) χρησιμοποιούνται από διάφορες εφαρμογές για να γίνει αντιληπτό το κούνημα ή η περιστροφή της συσκευής στις τρεις διαστάσεις.

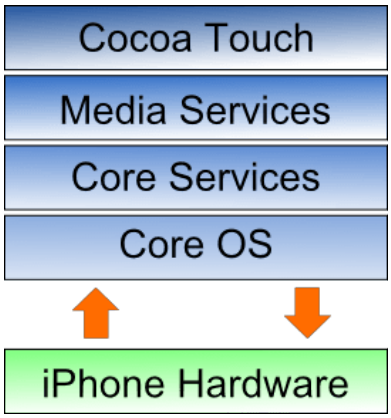


Εικόνα 5: Βασικές χειρονομίες στην οθόνη επαφής του iOS.

Τον Ιούνιο του 2016 ανακοινώθηκε το νέο σύστημα αρχείων της Apple Inc., για macOS, iOS, tvOS and watchOS. Το Apple File System (APFS) είναι βελτιστοποιημένο για Flash Disks και Solid-State Drives και επικεντρώνεται ιδιαίτερα στην κρυπτογράφηση. Χρησιμοποεί αριθμούς inode 64-bit που σημαίνει ότι μπορεί να αποθηκεύσει μέχρι και διαφορετικά αρχεία στον ίδιο δίσκο. Για την ακεραιότητα των δεδομένων (Data Integrity) παράγονται αθροίσματα ελέγχου (checksums) βασισμένα στα μεταδεδομένα των αρχείων. Γενικά, στο APFS έχουν αυξηθεί οι ταχύτητες των λειτουργιών ανάγνωσης-εγγραφής λόγω του τρόπου με τον οποίο υπολογίζονται τα διαθέσιμα δεδομένα. Τέλος, το APFS παρέχει προστασία από κατάρρευση συστήματος (Crash Protection) ως εξής: Αντί να κάνει απευθείας overwrite τα δεδομένα, δημιουργεί μια εξ' ολοκλήρου καινούρια εγγραφή, έναν δείκτη σε αυτήν, και μετά διαγράφει το παλιό αρχείο. Γενικά, μπορούμε να πούμε ότι αποτελεί μια δραματική αλλαγή από τον προκάτοχο του, το HFS+, η επίδοση της οποίας αναμένεται να κριθεί από τους καταναλωτές.

2.1.1 Η διαστρωματωμένη αρχιτεκτονική του iOS

Η αρχιτεκτονική του iOS χωρίζεται σε πέντε στρώματα, καθένα από τα οποία παρέχει προγραμματιστικά πλαίσια (frameworks) για την ανάπτυξη εφαρμογών. Μία εφαρμογή μπορεί απευθείας να επικοινωνήσει με frameworks σε οποιοδήποτε στρώμα. Ωστόσο, επειδή κάθε στρώμα παρέχει ένα αυξανόμενο επίπεδο αφαίρεσης από κάτω προς τα πάνω, συστήνεται στους προγραμματιστές να χρησιμοποιούν κυρίως frameworks του Cocoa Touch Layer ώστε να δουλεύουν μακριά από το hardware αποφεύγοντας τα πολλά bugs.



Εικόνα 6: Τα στρώματα της αρχιτεκτνικής του iOS.

Τα frameworks του **Cocoa Touch Layer** είναι αυτά που χρησιμοποιούνται κυρίως στις εφαρμογές. Γραμμένο κυρίως σε Objective-C, το στρώμα αυτό βασίζεται στο Mac OS X Cocoa API και έχει επεκταθεί και τροποποιηθεί ώστε να αντοποκρίνεται στις απαιτήσεις του iPhone. Περιλαμβάνει τα εξής frameworks: UIKit, MapKit, Push Notification Service, MessageUI, Address Book UI και GameKit framework.

Το **Media Services Layer** παρέχει στο iPhone δυνατότητες που σχετίζονται με ήχο, βίντεο, animation και γραφικά. Τα παρεχόμενα frameworks είναι: Core Graphics, Quartz Core, OpenGLES, iPhone Audio Support, AVFoundation, Core Audio Frameworks, Open Audio Library και Media Player framework.

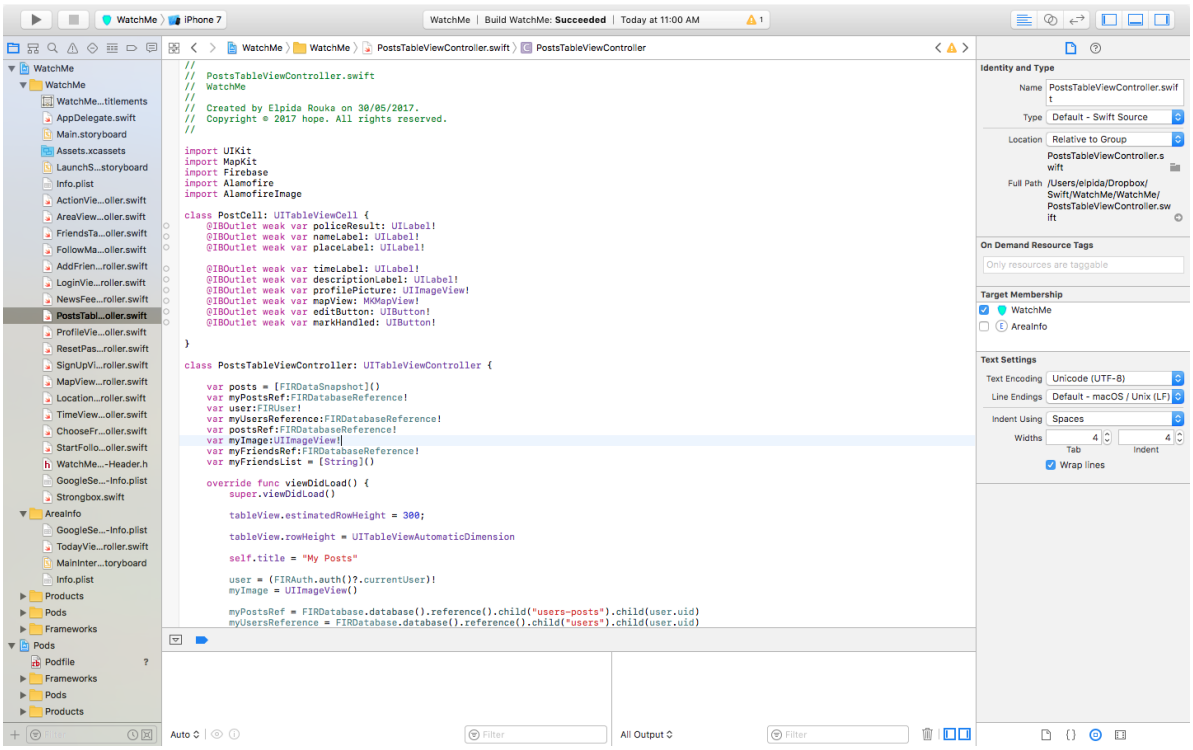
Το **Core Services Layer** παρέχει τα θεμέλια πάνω στα οποία έχουν χτιστεί τα frame- works των προηγούμενων στρωμάτων. Αποτελείται από τα ακόλουθα χαμηλού επιπέδου frameworks: Address Book, Core Data, Core Foundation, Foundation, Core Location, Store Kit και SQLite library.

Τέλος, το κοντινότερο στο hardware **Core OS Layer** παρέχει ορισμένες υπηρεσίες χαμηλού επιπέδου, όπως υπηρεσίες δικτύωσης στη στοίβα TCP/IP, επικοινωνία με εξαρτήματα του κινητού καθώς και θεμελιώδεις υπηρεσίες του λειτουργικού συστήματος όπως διαχείρηση μνήμης, αλληλεπίδραση με το σύστημα αρχείων και νήματα (threads). Τα frameworks που περιλαμβάνονται στο Core OS Layer είναι: CFNetwork, External Accessory, Security Framework και LibSystem. Η πρόσβαση σε αυτό το επίπεδο είναι περιορισμένη για λόγους ευστάθειας και ασφάλειας.

2.1.2 Εργαλεία ανάπτυξης εφαρμογών

Για την ανάπτυξη εφαρμογής iOS, ο προγραμματιστής πρέπει να διαθέτει έναν Intel-Based υπολογιστή Apple με λειτουργικό σύστημα Macintosh και τα εργαλεία του Xcode. Το Xcode είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment, IDE) για τη δημιουργία εφαρμογών για Mac, iPhone, iPad, Apple Watch, και Apple TV. Περιλαμβάνει εργαλεία για σχεδίαση και ανάπτυξη κώδικα, έλεγχο, αποσφαλμάτωση και υποβολή στο App Store, καθώς και δυνατότητα Source/Version Control.

Το βασικό παράθυρο του Xcode αποτελείται από τον συντάκτη κειμένου στο κέντρο, έναν Project Navigator στα αριστερά όπου βρίσκονται όλα τα αρχεία του Project, και έναν File Inspector στα δεξιά που παρέχει πληροφορίες για το επιλεγμένο αρχείο. Στην περιοχή Debugging στο κάτω σημείο εμφανίζονται τα μηνύματα κονσόλας όταν η εφαρμογή τρέχει σε μια συνδεδεμένη συσκευή. Στο ανώτερο σημείο του παραθύρου βρίσκεται το Toolbar, μέσω του οποίου επιλέγεται σε ποια συσκευή θα τρέξει η εφαρμογή.



Εικόνα 7: Το User Interface του Xcode.

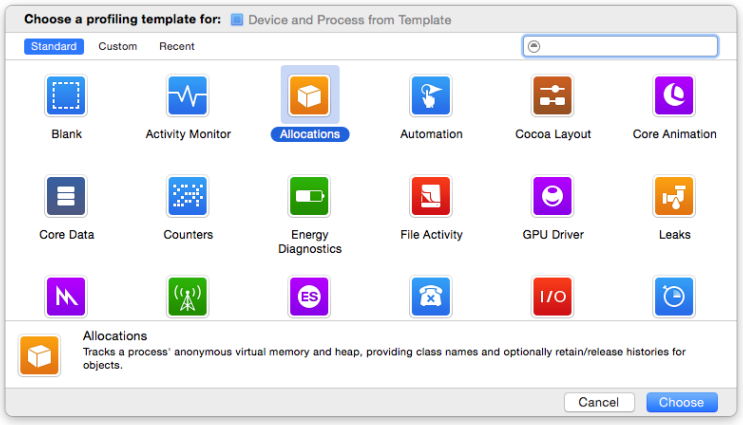
Όταν ένα λειτουργικό μέρος μιας εφαρμογής έχει γραφτεί και μεταγλωττιστεί στο Xcode, υπάρχουν δύο δυνατοί τρόποι να τρέξει: είτε σε μια συσκευή iPhone είτε στον iOS Simulator του Xcode. Επειδή όμως ο Simulator δεν υποστηρίζει αρκετές βασικές λειτουργίες, ο πιο αξιόπιστος έλεγχος γίνεται πάντα σε μια κανονική συσκευή.

Εικόνα 8: iOS Simulator, iPhone 7.

Ένα επιπλέον εργαλείο που ανήκει στην σουίτα του Xcode είναι το Instruments. Πρόκειται για ένα ευέλικτο εργαλείο ανάλυσης επίδοσης και δοκιμών (testing) που βοηθά τον προγραμματιστή να κατανοήσει και να βελτιστοποιήσει την επίδοση της εφαρμογής του μέσω ορισμένων μετρήσεων. Με αυτό τον τρόπο βοηθά να κατανοηθούν και να επιλυθούν τάσεις και συμπεριφορές του προγράμματος. Μερικές από τις λειτουργίες του είναι:

- Εξέταση συμπεριφοράς της εφαρμογής
- Εξέταση χαρακτηριστικών της συσκευής όπως Wi-Fi, Bluetooth
- Profiling (μέτρηση χρονικής και χωρικής πολυπλοκότητας, χρησιμοποίηση εντολών και διάρκεια κλήσεων συναρτήσεων) σε μία ή περισσότερες συσκευές
- Εύρεση προβλημάτων στον κώδικα
- Ανάλυση επίδοσης
- Εύρεση προβλημάτων μνήμης
- Επίτευξη αποδοτικότητας ισχύος

Παρόλο που είναι ενσωματωμένο στο Xcode, το Instruments είναι μια ξεχωριστή εφαρμογή που μπορεί να χρησιμοποιηθεί ανεξάρτητα και είναι ιδιαιτέρα απαραίτητη όταν ένα προϊόν πρόκειται να κυκλοφορήσει στο App Store.



Εικόνα 9: Instruments Templates

2.1.3 Today Extensions

Κάνοντας swipe προς τα δεξιά στην αρχική οθόνη του iOS εμφανίζεται μια οθόνη που ονομάζεται “Today View”. Σε αυτή την οθόνη βρίσκονται επεκτάσεις των εφαρμογών που ονομάζονται Today Extensions (ή αλλιώς Widgets) και παρέχουν γρήγορη πρόσβαση σε πληροφορίες που είναι σημαντικές εκείνη τη χρονική στιγμή. Παραδείγματα τέτοιων πληροφοριών είναι οι καιρικές συνθήκες, οι τιμές των μετοχών ή κάποια προγραμματισμένη συνάντηση. Επίσης, είναι δυνατό με το πάτημα ενός κουμπιού να επιτελείται κάποια απλή λειτουργία. Επειδή οι Today Extensions δεν υποστηρίζουν είσοδο από πληκτρολόγιο, ο χρήστης συνήθως μπορεί να τις διαμορφώσει μέσω της αντίστοιχης εφαρμογής. Επειδή ο διαθέσιμος χώρος σε ένα Widget είναι περιορισμένος, οι πληροφορίες που παρέχει πρέπει να είναι σύντομες και επικεντρωμένες στο λόγο για τον οποίο δημιουργήθηκαν.

Οι Today Extensions παρέχουν ένα API μέσω του οποίου ο προγραμματιστής χειρίζεται τις ενημερώσεις στο περιεχόμενό τους. Για να φαίνεται μόνιμα ενημερωμένο το Widget, το iOS περιοδικά λαμβάνει στιγμιότυπά του. Όταν το Widget ξαναγίνει ορατό, εμφανίζεται το πιο πρόσφατο στιγμιότυπο μέχρι να αναεωθούν τα δεδομένα του.

Για να ενημερωθεί ένα Widget πριν ληφθεί ένα στιγμιότυπο, πρέπει να υλοποιεί το πρωτόκολλο “NCWidgetProviding”. Η συνάρτηση “widgetPerformUpdate (completionHandler:)” του συγκεκριμένου πρωτοκόλλου καλείται πριν ληφθεί ένα στιγμιότυπο. Μέσα σε αυτήν ο προγραμματιστής ανανεώνει το περιεχόμενο του Widget καλώντας τον Completion Handler με ένα από τα παρακάτω ορίσματα:

- updateResult.newData: Το Widget ανανεώνεται σύμφωνα με τις νέες τιμές που έχουν δοθεί στις μεταβλητές και τα νέα κείμενα στα labels και στα buttons.
- updateResult.noData: Το widget δεν χρειάζεται ανανέωση.
- updateResult.failed: Ένα σφάλμα προέκυψε κατά την ανανέωση.

Εάν το Widget χρειάζεται να είναι ορατό στην Today View μόνο κάτω από ορισμένες προϋποθέσεις (π.χ. όταν διαθέτει καινούριες ή αξιοσημείωτες πληροφορίες) χρησιμοποιείται η μέθοδος setHasContent(forWidget: , withBundleIdentifier:) της κλάσης NCWidgetController. Μέσω της μεθόδου αυτής δηλώνεται η κατάσταση του Widget που με τη σειρά της προτρέπει το σύστημα να το αποκρύψει ή να το εμφανίσει. Η μέθοδος αυτή καλείται από την αντίστοιχη εφαρμογή -και όχι από το ίδιο το Widget- με τις τιμές YES (για εμφάνιση) ή NO (για απόκρυψη).

Ένας άλλος τρόπος επικοινωνίας του Today Extension με την περικλείουσα εφαρμογή είναι τα UserDefaults, ένα αντικείμενο που έχει πρόσβαση σε μια βιβλιοθήκη όπου αποθηκεύονται προτιμήσεις του χρήστη σχετικά με κάθε εφαρμογή. Η πληροφορία που πρέπει να διαμοιραστεί μεταξύ της εφαρμογής και της επέκτασης αποθηκεύεται στα UserDefaults από την κύρια εφαρμογή. Έπειτα, την διαβάζει από εκείνο το σημείο η επέκταση ώστε να ανανεωθεί.

2.1.4 Keychain

Το Keychain είναι ένα σύστημα διαχείρησης κωδικών στα macOS και iOS που αναπτύχθηκε από την Apple. Εκτός από κωδικούς, χρησιμοποιείται για να αποθηκεύει κλειδιά, στοιχεία λογαριασμών, πιστοποιητικά και ασφαλείς σημειώσεις. Το Keychain αποθηκεύει τα αρχεία κρυπτογραφημένα, με αλγορίθμους AES 256-bit, ασύμμετρη κρυπτογραφία ελλειπτικών καμπυλών και αναδίπλωση κλειδιών. Ένας χρήστης αποκτά πρόσβαση στο keychain της συσκευής του με έναν κωδικό, ενώ κάθε εφαρμογή που χρησιμοποιεί το Keychain έχει πρόσβαση μόνο σε κωδικούς που έχει τοποθετήσει η ίδια εκεί, μέσω του Keychain Services API.

Μία συνήθης χρήση του Keychain είναι το αυτοματοποιημένο login σε εφαρμογές. Η διαδικασία που ακολουθείται είναι η εξής:

1. Ο χρήστης εκκινεί την εφαρμογή
2. Η εφαρμογή ελέγχει στο Keychain αν υπάρχουν γι' αυτήν αποθηκευμένα στοιχεία εισόδου(email-password). Αν υπάρχουν, πηγαίνουμε στο βήμα 7
3. Ο χρήστης καλείται να εισάγει email και κωδικό
4. Τα στοιχεία αυτά στέλνονται στον Server της εφαρμογής για έλεγχο. Αν δεν είναι έγκυρα, πηγαίνουμε στο βήμα 3.
5. Ο χρήστης ρωτάται αν θέλει να αποθηκευτούν αυτά τα στοιχεία στο Keychain. Αν όχι, πηγαίνουμε στο βήμα 8
6. Αποθήκευση των στοιχείων στο Keychain. Πηγαίνουμε στο βήμα 8
7. Τα στοιχεία αυτά στέλνονται στον Server της εφαρμογής για έλεγχο. Αν δεν είναι έγκυρα, πηγαίνουμε στο βήμα 3.
8. Τέλος

Στην εφαρμογή υλοποιήσαμε αυτήν ακριβώς τη διαδικασία χρησιμοποιώντας έναν wrapper του Keychain Services API, το Strongbox, που αποτελεί μια υψηλού επιπέδου αφαίρεση των λειτουργιών του Keychain.

2.2 JSON

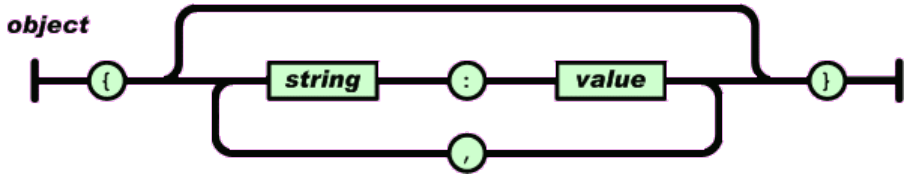
Το JSON (JavaScript Object Notation) είναι μια μορφή αποθήκευσης και ανταλλαγής δεδομένων, που καθιστά εύκολη την ανάγνωση και εγγραφή δεδομένων. Βασίζεται σε ένα υποσύνολο της γλώσσας προγραμματισμού JavaScript, όμως επειδή τα αρχεία JSON είναι σε μορφή κειμένου είναι δυνατό να διαβαστούν και να γραφούν και από πολλές άλλες γλώσσες προγραμματισμού.

Το JSON βασίζεται σε δύο δομές:

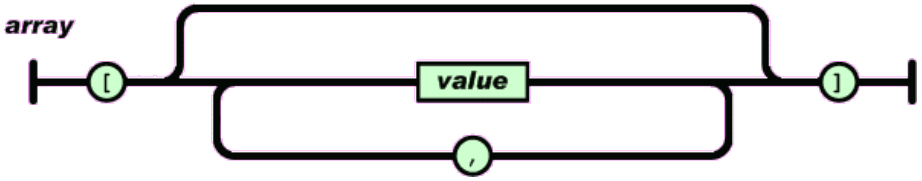
- Μία συλλογή από ζεύγη κλειδιού-τιμής, που μπορούν να χαρακτηριστούν ως "Αντικείμενα", δομές, εγγραφές, λεξικά ή πίνακες κατακερματισμού

- Μία ταξινομημένη λίστα τιμών, που στις περισσότερες γλώσσες υλοποιείται ως πίνακας ή λίστα.

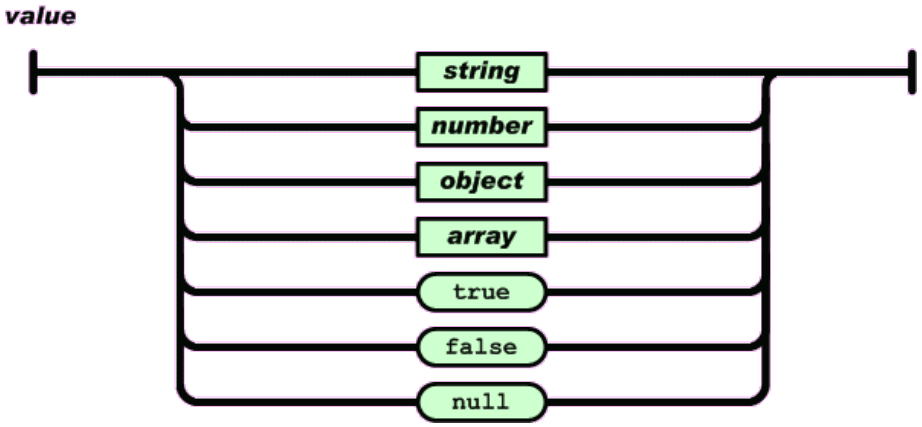
Είναι λοιπόν εμφανής η συμβατότητα του JSON με τις περισσότερες γλώσσες προγραμματισμού. Στις ακόλουθες εικόνες παρουσιάζεται ο τρόπος δημιουργίας JSON Objects.



Εικόνα 10: Αλγόριθμος δημιουργίας αντικειμένου σε αρχείο JSON.



Εικόνα 11: Αλγόριθμος δημιουργίας πίνακα σε αρχείο JSON.



Εικόνα 12: Τύποι τιμών σε αρχείο JSON.

2.3 Google Firebase

Το Firebase είναι μια πλατφόρμα ανάπτυξης mobile και web εφαρμογών. Αποτελείται από συμπληρωματικά χαρακτηριστικά που οι προγραμματιστές μπορούν να αναμείξουν για να ικανοποιήσουν τις ανάγκες της εφαρμογής τους. Η εταιρεία Firebase ιδρύθηκε το 2011 στην Καλιφόρνια από τους Andrew Lee και James Tamplin. Το αρχικό προϊόν ήταν μια πραγματικού χρόνου βάση δεδομένων, που παρείχε ένα API το οποίο επιτρέπει στους προγραμματιστές να συγχρονίζουν δεδομένα μεταξύ πολλαπλών πελατών/συσκευών. Η εταιρεία αγοράστηκε από την Google τον Οκτώβριο του 2014 και σήμερα έχει επεκταθεί σε μία πλήρη σουίτα ανάπτυξης εφαρμογών.

Οι παροχές του Firebase χωρίζονται σε 4 κατηγορίες: Analytics, Develop, Grow και Earn. Περιληπτικά, οι λειτουργίες κάθε κατηγορίας είναι οι εξής:

Analytics

Firebase Analytics: Παρέχει μετρήσεις και στατιστικά σχετικά με τη χρησιμοποίηση της εφαρμογής και τους μηνιαία ενεργούς χρήστες ανάλογα με το φύλο, την ηλικία και την περιοχή.

Develop

Authentication: Καθιστά δυνατή την αυθεντικοποίηση των χρηστών μόνο με client-side κώδικα, έχοντας το back-end ήδη υλοποιημένο. Επιτρέπει το login, εκτός από e-mail και κωδικό, και με εξωτερικούς παρόχους login (Facebook, Github, Twitter, Google).

Realtime Database: Είναι μία noSQL βάση δεδομένων αποθηκευμένη στο Cloud σε μορφή JSON. Αντί για κλασσικά HTTP Requests χρησιμοποιεί συγχρονισμό δεδομένων, επιτρέποντας την ενημέρωση οποιασδήποτε συσκευής που τρέχει την εκάστοτε εφαρμογή μέσα σε milliseconds. Η πρόσβαση στη βάση μπορεί να περιοριστεί από τον προγραμματιστή μέσω κανόνων ασφάλειας (Security Rules).

Storage: Προσφέρει ασφαλή μεταφόρτωση και λήψη αρχείων για τις εφαρμογές του Firebase, ανεξάρτητα της ποιότητας δικτύου. Χρησιμοποιείται για αποθήκευση αρχείων εικόνας, ήχου και βίντεο των χρηστών.

Hosting: Χρησιμοποείται σε Web εφαρμογές, για την ανάπτυξη και κυκλοφορία στατικού περιεχομένου (π.χ. αρχεία CSS, HTML, JavaScript) σε ένα δίκτυο διανομής περιεχομένου (Content-Delivery Network, CDN) μέσω HTTPS/SSL.

Cloud Functions: Δίνει τη δυνατότητα στον προγραμματιστή να γράφει back-end κώδικα που ενεργοποιείται αυτόματα όταν συμβαίνουν συγκεκριμένες αλλαγές ή αιτήσεις στη βάση. Ο κώδικας αποθηκεύεται στο Google Cloud.

Test Lab: Παρέχει μία υποδομή δοκιμών (testing) για εφαρμογές Android. Τα αποτελέσματα των δοκιμών (screenshots, logs, videos) γίνονται διαθέσιμα στον προγραμματιστή στην κονσόλα του Firebase. Βοηθά ιδιάιτερα στην ανεύρεση προβλημάτων που οφείλονται σε κάποιο συγκεκριμένο συνδυασμό συσκευής – λογισμικού.

Crash Reporting: Δημιουργεί λεπτομερείς αναφορές (reports) για σφάλματα που συμβαίνουν στο πρόγραμμα.

Performance Monitoring: Βοηθά τον προγραμματιστή να διεισδύσει στα προβλήματα επίδοσης της εφαρμογής του. Παρέχει ένα SDK συλλογής δεδομένων επίδοσης τα οποία έπειτα παρουσιάζονται και αναλύονται στην κονσόλα.

Firebase Cloud Messaging (FCM): Εργαλείο για αποστολή μηνυμάτων μεταξύ των χρηστών, που διέρχονται από τον server της εφαρμογής, καθώς και απομακτυσμένων ειδοποιήσεων (remote notifications) ως απόκριση σε ορισμένα γεγονότα.

Grow

Notifications: Δωρεάν υπηρεσία αποστολής μηνυμάτων και ειδοποιήσεων από τον προγραμματιστή σε συγκεκριμένες συσκευές με σκοπό το testing.

Remote Configuration: Υπηρεσία που καθιστά ικανούς τους προγραμματιστές να αλλάξουν τη συμπεριφορά και τα γραφικά της εφαρμογής χωρίς να απαιτηθεί από το χρήστη να εκτελέσει κάποιο update.

Dynamic Links: Σύνδεσμοι προς το περιβάλλον της εφαρμογής που συμπεριφέρονται ανάλογα με τη συσκευή στην οποία βρίσκεται ο χρήστης, για παράδειγμα Mobile View ή Desktop View της εφαρμογής. Ακόμη, σε περίπτωση που η εφαρμογή δεν είναι εγκατεστημένη, μπορούν να παραπέμπουν το χρήστη στο App Store.

Earn

AdMob: Πλατφόρμα τοποθέτησης διαφημίσεων εντός της εφαρμογής που καθιστά τον προγραμματιστή ικανό να αποκτήσει εισόδημα μέσω αυτής.

Από όλες τις παραπάνω λειτουργίες, τέσσερις χρησιμοποιήθηκαν πιο εντατικά για τη δημιουργία της παρούσας εφαρμογής. Αυτές αναλύονται στη συνέχεια.

2.3.1 Authentication

Οι περισσότερες εφαρμογές χρειάζεται να εξακριβώνουν την ταυτότητα των χρηστών τους και να περιορίζουν την πρόσβαση σε ευαίσθητα προσωπικά δεδομένα. Το Firebase Authentication παρέχει τις κατάλληλες back-end υπηρεσίες, εύχρηστα SDKs και βιβλιοθήκες που επιτρέπουν τη σύνδεση χρηστών μέσω email, κωδικών αλλά και εξωτερικών παρόχων login.

Ένα αντικείμενο χρήστη Firebase (Firebase User Object) αντιπροσωπεύει έναν λογαριασμό χρήστη που έχει εγγραφεί στην εφαρμογή και περιλαμβάνει μία διεύθυνση e-mail, ένα μοναδικό αναγνωριστικό χρήστη (UserID), ένα όνομα (Display name) και ένα URL φωτογραφίας προφίλ. Οι εφαρμογές συνήθως έχουν πολλούς εγγεγραμμένους χρήστες, αποθηκευμένους στο User Database του Firebase project. Απ' την άλλη πλευρά, ένα αντικείμενο Firebase Auth περιέχει πληροφορίες σχετικά με την αυθεντικοποίηση. Όταν ένας χρήστης συνδέεται στην εφαρμογή το User Object που του αντιστοιχεί τίθεται στο πεδίο "currentUser" του αντικειμένου Auth που τρέχει στην συγκεκριμένη συσκευή. Όταν ο χρήστης αποσυνδεθεί (Sign out) το αντικείμενο Auth παύει να υπάρχει στη μνήμη.

Ο προτεινόμενος τρόπος παρατήρησης του Firebase Auth Object είναι μέσω παρατηρητών (observers) που τρέχουν στον Client και ειδοποιούνται όταν συμβαίνει κάποια αλλαγή στο αντικείμενο Auth. Πέρα από τις περιπτώσεις Sign in και Sign out, ένας τέτοιος παρατηρητής

ειδοποιείται αν ο χρήστης αλλάξει τον κωδικό του ή αν ανανεωθεί το Authentication Token. Το Authentication Token είναι μια συμβολοσειρά που παράγεται ξεχωριστά για κάθε χρήστη με βάση τα διαπιστευτήριά του, είναι απαραίτητο για την αυθεντικοποίηση και ανανεώνεται κάθε 60 λεπτά για λόγους ασφαλείας. Η αποστολή ενός έγκυρου Authentication Token στον Server ταυτοποιεί τον χρήστη με την ίδια αποτελεσματικότητα που τον ταυτοποιεί το ζεύγος email-password.

Δύο ξεχωριστές δυνατότητες που δίνει επίσης το Firebase Authentication είναι η λειτουργίες επαλήθευσης email και επαναπροσδιορισμού κωδικού. Και οι δύο υλοποιούνται με αποστολή email στο χρήστη. Η επαλήθευση email χρήστη ελέγχεται από την Boolean μεταβλητή "isEmailVerified" και τη συνάρτηση "sendEmailVerification" του αντικειμένου User. Αντίστοιχα, ο επαναπροσδιορισμός κωδικού πραγματοποιείται με τη συνάρτηση "sendPasswordReset" του αντικειμένου User, που στέλνει ένα email στη διεύθυνση του συγκεκριμένου χρήστη προτρέποντάς τον να εισάγει νέο κωδικό.

2.3.2 Real Time Database

Η πραγματικού χρόνου βάση δεδομένων του Firebase αποτελείται από δύο μέρη: Το αρχείο δεδομένων σε JSON format και τους κανόνες ασφαλείας. Θα εξετάσουμε αυτές τις δύο συνιστώσες ξεχωριστά.

2.3.2.1 Data

Η Firebase Realtime Database επιτρέπει ασφαλή πρόσβαση στα δεδομένα απευθείας από Client-side κώδικα. Τα δεδομένα αποθηκεύονται και τοπικά στη συσκευή ώστε ακόμα και εκτός σύνδεσης να προσφέρουν ρεαλιστική εμπειρία χρήστη. Αφού επανέλθει η σύνδεση, συγχρονίζονται με την απομακρυσμένη βάση και συγχωνεύονται οι αλλαγές. Πρέπει να τονίσουμε ότι επειδή πρόκειται για NoSQL βάση χρειάζονται διαφορετικού είδους βελτιστοποιήσεις και λειτουργικότητες από μία σχεσιακή βάση. Το API της Realtime Database είναι σχεδιασμένο ώστε να επιτρέπει μόνο λειτουργίες που μπορούν να εκτελεστούν γρήγορα. Επομένως, είναι ιδιαίτερα σημαντικό να κατανοήσει ο προγραμματιστής τον τρόπο με τον οποίο οι χρήστες θα πρέπει να επικοινωνούν με τη βάση και να σχεδιάσει τη δομή της με βάση αυτές τις απαιτήσεις.

Μία πρωταρχική χρήση της Realtime Database είναι η αποθήκευση δεδομένων για κάθε χρήστη, πέρα από αυτά που αποθηκεύονται στον πίνακα των χρηστών στο Firebase Authentication. Για παράδειγμα μια εφαρμογή μπορεί να αποθηκεύει την ηλικία, το φύλο του κάθε χρήστη, τις αγαπημένες του ταινίες ή οποιαδήποτε άλλη πληροφορία. Τονίζουμε ότι στοιχεία που δεν μπορούν να παρουσιαστούν σε μορφή κειμένου (όπως εικόνες, αρχεία ήχου και βίντεο) αποθηκεύονται αντί της βάσης στο Firebase Storage.

Η ανάγνωση και εγγραφή δεδομένων στη βάση γίνεται δημιουργώντας μία αναφορά σε αυτήν και συγκεκριμένα στον κόμβο του JSON δέντρου τον οποίο θέλουμε να επεξεργαστούμε. Για την ανάγνωση, επισυνάπτεται σε αυτή την αναφορά ένας παρατηρητής (observer), ο κώδικας του οποίου γίνεται trigger κάθε φορά που ο κόμβος αυτός ή κάποιο παιδί του μεταβληθεί και

επιστρέφεται ένα snapshot του ανανεωμένου κόμβου.Για λειτουργίες εγγραφής καλείται πάνω στην αναφορά κάποια συνάρτηση set, update ή delete που παίρνει ως όρισμα τις νέες τιμές (η delete ισοδυναμεί με όρισμα null) και επιτελεί την αντίστοιχη λειτουργία.

Οι observers δεν σταματούν αυτόματα να συγχρονίζουν δεδομένα μόλις κλείσει μια οθόνη. Συνεπώς, όταν ένας observer σταματά να είναι απαραίτητος ή όταν μια οθόνη ετοιμάζεται να κλείσει, ο προγραμματιστής πρέπει να προνοεί αποσυνδέοντας τον observer από την αντίστοιχη αναφορά.

2.3.2.2 Security Rules

Οι κανόνες ασφάλειας της βάσης δεδομένων του Firebase καθορίζουν ποιος έχει πρόσβαση εγγραφής και ανάγνωσης στη βάση, πώς είναι δομημένο το JSON δέντρο και τι ευρετήρια υπάρχουν. Οι κανόνες βρίσκονται στον Server του Firebase και επιβάλλονται αυτόματα ανά πάσα στιγμή. Κάθε αίτηση ανάγνωσης και εγγραφής λαμβάνει χώρα μόνο αν οι κανόνες το επιτρέπουν.

Υπάρχουν τέσσερα είδη κανόνων:

- *.read* : Καθορίζει αν και πότε τα δεδομένα επιτρέπεται να διαβαστούν
- *.write* : Καθορίζει αν και πότε τα δεδομένα επιτρέπεται να γραφτούν
- *.validate* : Καθορίζει πως θα φαίνεται μια σωστά φορμαρισμένη τιμή, τον τύπο δεδομένων της και το ποια παιδιά θα έχει
- *.indexOn*: Καθορίζει ποιος κόμβος-παιδί θα υποστηρίζει ταξιμομήσεις και ερωτήματα

Επίσης, το Firebase διαθέτει ορισμένες προκαθορισμένες, βοηθητικές μεταβλητές που μπορούν να προσπελαστούν μέσα στον ορισμό των κανόνων ασφαλείας. Αυτές είναι οι εξής:

- *now*: Ο τρέχων χρόνος σε milliseconds μετρημένος με αφετηρία το Unix Epoch (00:00:00 UTC στις 1 Ιανουαρίου 1970). Χρησιμοποιείται κυρίως για τον έλεγχο χρονοσφραγίδων (timestamps).
- *root*: Αντιπροσωπεύει τον κόμβο-ρίζα του JSON δέντρου, όπως εμφανίζεται πριν την λειτουργία προς υλοποίηση.
- *newData*: Το key-value pair όπως θα εμφανιζόταν μετά την λειτουργία προς υλοποίηση.
- *data:* Το key-value pair όπως εμφανίζεται πριν την λειτουργία προς υλοποίηση.
- *auth:* Αντιπροσωπεύει το ωφέλιμο φορτίο του token ενός αυθεντικοποιημένου χρήστη.

Ταυτόχρονα, ο ίδιος ο χρήστης μπορεί να ορίσει δικές του μεταβλητές που έχουν τη μορφή συμβολοσειρών, ξεκινούν με το σύμβολο “$” και αντιπροσωπεύουν IDs και κλειδιά κόμβων-παιδιών. Αυτές οι μεταβλητές ονομάζονται wildcards.

Αυθεντικοποίηση

Ένα πρώτο βήμα για την ασφάλεια μιας εφαρμογής είναι η ταυτοποίηση (ή αυθεντικοποίηση, authentication) των χρηστών, που όπως είδαμε προηγουμένως υλοποιείται στο Firebase Authentication. Οι προεπιλεγμένοι κανόνες στο firebase, αυτοί δηλαδή που

δημιουργούνται αυτόματα στην αρχή μαζί με το project, ορίζουν ότι ένας χρήστης έχει πρόσβαση σε όλο το JSON δέντρο αν και μόνο αν είναι αυθεντικοποιημένος, ή με απλά λόγια αν έχει ολοκληρώσει επιτυχώς το Sign in. Αυτό εκφράζεται μέσω της μεταβλητής “auth” ως εξής:

```
{
  "rules": {
  ".read": "auth != null",
  ".write": "auth != null"
  }
}
```

Ως τιμή των παραπάνω key-value pairs μπορούν να οριστούν και οι μεταβλητές true και false που καθορίζουν αντίστοιχα πρόσβαση ή αποκλεισμό οποιουδήποτε στον εν λόγω κόμβο, άσχετα με το αν έχει αυθεντικοποιηθεί ή όχι.

Εξουσιοδότηση

Ακόμη κι αν ένας χρήστης έχει ταυτοποιηθεί, υπάρχουν δεδομένα τα οποία απαγορεύεται να αλλάξει (για παράδειγμα προσωπικά δεδομένα άλλων χρηστών) αλλά και δεδομένα που δεν επιτρέπεται ούτε να διαβάσει χωρίς να πληρεί ορισμένες προϋποθέσεις. Για παράδειγμα, αν επιθυμούμε να επιτρέψουμε σε όλους τους αυθεντικοποιημένους χρήστες να διαβάζουν τον κόμβο users αλλά ο καθένας να γράφει μόνο στα δικά του προσωπικά στοιχεία, χρησιμοποιούμε τους εξής κανόνες:

```
{
  "rules": {
    "users": {
      ".read": "auth != null",
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Αυτό βέβαια προϋποθέτει ότι έχουμε αποθηκεύσει κάθε χρήστη στη βάση με το αναγνωριστικό που του δόθηκε όταν γράφτηκε πρώτη φορά στην εφαρμογή (Sign up). Ένας ακόμη ενδιαφέρων κανόνας που ουσιαστικά επιτρέπει μόνο τις λειτουργίες write και delete, απαγορεύοντας τη λειτουργία update είναι ο εξής:

```
".write": "!data.exists() || !newData.exists()"
```

Έλεγχος Εγκυρότητας

Καθώς στα αρχεία JSON δεν δηλώνεται κάπου ο τύπος των μεταβλητών ούτε ο αριθμός των παδιών κάθε κόμβου, είναι σημαντικό με κάποιο τρόπο τα δεδομένα να παραμένουν συνεπή και να έχουν την αναμενόμενη μορφή. Αυτό επιτυγχάνεται με τον κανόνα “.validate”. Για παράδειγμα, αν θέλουμε ως τιμή του /foo/ να γράφονται μόνο συμβολοσειρές με λιγότερους από 100 χαρακτήρες ορίζουμε τον εξής κανόνα:

```
{
  "rules": {
```

```
    "foo": {
      ".validate": "newData.isString() && newData.val().length < 100"
    }
  }
}
```

Ορισμός Ευρετηρίων

Όσο οι ποσότητες των δεδομένων διατηρούνται μικρές, τα ευρετήρια δεν είναι απαραίτητα για την γρήγορη εκτέλεση των ερωτημάτων. Ωστόσο, πριν μια εφαρμογή κυκλοφορήσει στην αγορά και χρειαστεί να υποστηρίξει ίσως και εκατομμύρια χρήστες, είναι απαραίτητο να προστεθούν ευρετήρια που θα εξασφαλίζουν τη γρήγορη απόκρισή της. Για παράγειγμα, αν στην εφαρμογή χρειάζεται συχνά η ταξινόμηση δεινοσαύρων ανάλογα με ύψος και το βάρος τους, πρέπει να προστεθεί ο ακόλουθος .indexOn κανόνας:

```
{
  "rules": {
    "dinosaurs": {
      ".indexOn": ["height", "length"]
    }
  }
}
```

Τέλος, είναι σημαντικό να κατανοηθεί ο τρόπος με τον οποίο αποτιμώνται οι κανόνες. Ειδικά όσον αφορά στην εγγραφή και την ανάγνωση η αποτίμηση γίνεται ως εξής: Έστω ότι η λειτουργία αφορά κάποιον κόμβο-παιδί, τότε το Firebase ξεκινά να ελέγχει τις άδειες από τον κόμβο-ρίζα. Όσο συναντά κανόνες απαγόρευσης προχωρά στο αμέσως επόμενο επίπεδο μέχρι να βρει –πριν ή στον ζητούμενο κόμβο- κάποιον κανόνα που να του επιτρέπει να εκτελέσει τη λειτουργία. Μόλις φτάσει σε τέτοιο κόμβο η λειτουργία εκτελείται άσχετα με τους κανόνες που υπάρχουν πιο βαθιά στο δέντρο. Αν φτάσει στον ζητούμενο κόμβο χωρίς να έχει βρει κάποιον επιτρεπτικό κανόνα η λειτουργία μπλοκάρεται. Τονίζουμε ότι η απουσία κανόνα .read/.write σε κάποιο κόμβο ταυτίζεται με την ύπαρξη αντίστοιχου κανόνα με τιμή false.

Μετά τον καθορισμό των κανόνων ο προγραμματιστής μπορεί να ελέγξει τη λειτουργικότητά τους χρησιμοποιώντας τον Security Rules Simulator που βρίσκεται στην κονσόλα του Firebase. Εκεί ο προγραμματιστής ορίζει τι λειτουργία επιθυμεί να εκτελέσει και σε ποιο κόμβο του JSON δέντρου, αν είναι αυθεντικοποιημένος ή όχι, ποιο είναι το Authentication Token του καθώς και τη μορφή των παλιών και νέων δεδομένων (για περιπτώσεις εγγραφής) και πατά Run. Αμέσως εμφανίζεται ένα αποτέλεσμα “Allowed” ή “Denied”.

Η κατανόηση του ιδιαίτερου τρόπου γραφής και αποτίμησης των κανόνων ασφάλειας είναι καίρια. Αυτό που θα καταστήσει την ασφάλεια μιας εφαρμογής επιτυχημένη δεν είναι ένας μεμονωμένος κανόνας, αλλά ένα σύνολο κανόνων που αλληλεπιδρούν σωστά μεταξύ τους και αλληλοσυμπληρώνονται.

2.3.3 Storage

Στις περισσότερες εφαρμογές ο χρήστης έχει τη δυνατότητα να ανεβάζει (upload) ή να κατεβάζει (download) αρχεία εικόνας, ήχου ή βίντεο που δεν μπορούν να παρασταθούν σε μορφή κειμένου. Τέτοιου είδους αρχεία αποθηκεύονται στο Firebase Storage μέσω κατάλληλων SDKs που παρέχει το Firebase. Η ευκολία και ταχύτητα χρήσης του Firebase Storage έγκειται στο γεγονός ότι, αν κατά τη διάρκεια ενός upload ή download χαθεί η σύνδεση δικτύου, ο πελάτης μπορεί να συνεχίσει τη λειτουργία από το σημείο που σταμάτησε μόλις η σύνδεση επανέλθει. Το Storage διαθέτει, όπως και η βάση, Security Rules που ελέγχουν την πρόσβαση των χρηστών στα αρχεία.

Για να ανέβει κάποιο αρχείο στο Storage προσδιορίζεται αρχικά το path στο οποίο θα γραφτεί και το όνομα του αρχείου. Στη συνέχεια δημιουργείται μια αναφορά (reference) στο Storage και εκτελείται μια λειτουργία “put” πάνω σε αυτήν την αναφορά, με όρισμα το συγκεκριμένο path.

Με το που αποθηκευτεί κάποιο αρχείο στο Storage δημιουργείται ένα URL προς αυτό, το οποίο ο προγραμματιστής μπορεί να αποθηκεύσει πλέον στη βάση. Όταν χρειαστεί να ανακτηθεί (download) το συγκεκριμένο αρχείο, εκτελείται μία ασύγχρονη λειτουργία λήψης στον κώδικα με όρισμα το συγκεκριμένο URL. Η λειτουργία αυτή θα αποθηκεύσει το αρχείο σε κάποια μεταβλητή της εφαρμογής καθιστώντας έτσι δυνατή την προσπέλασή του.

Ανάλυση

3.1 Απαιτήσεις κοινωνικού δικτύου για περιπτώσεις κινδύνου

Η αύξηση του εγκλήματος στις σύγχρονες μεγαλουπόλεις καθώς και η επικινδυνότητα συγκεκριμένων, κυρίως φτωχών, περιοχών οδηγεί στην ανάγκη για μια αίσθηση ασφάλειας των πολιτών, η οποία πολλές φορές είναι δύσκολο να επιτευχθεί. Σε αυτήν την κατεύθυνση μπορούν να βοηθήσουν τα smartphones, τα οποία πλέον διαθέτουν τόσες δυνατότητες ώστε να μπορούν να καλύψουν και τις πιο απαιτητικές ανάγκες των χρηστών.

Οι χρήστες, συγκεκριμένα, νιώθουν ασφάλεια αν μοιράζονται με κοντινούς τους ανθρώπους την τοποθεσία τους και αν μπορούν να ενημερώσουν αυτόματα την οικογένειά τους αν κάτι κακό τους συμβεί. Η εφαρμογή βοηθά τους χρήστες να το επιτύχουν αυτό μόνο με λίγες κινήσεις. Για να είναι λοιπόν χρήσιμο ένα τέτοιου είδους εργαλείο, πρέπει να ικανοποιεί κάποιες βασικές προδιαγραφές. Μερικές από αυτές είναι:

- Άμεση πληροφόρηση ενός χρήστη για την επικινδυνότητα της περιοχής στην οποία βρίσκεται.
- Ενημέρωσή του για την δραστηριότητα της αστυνομίας στη συγκεκριμένη περιοχή.
- Άμεση ενημέρωση του χρήστη όταν συμβεί ένα επείγον περιστατικό σε κάποιον φίλο του.
- Δυνατότητα να μοιράζεται με κάποιον φίλο την τοποθεσία του για όσο μετακινείται μέσα στην πόλη.
- Όλες αυτές οι ενέργειες να συμβαίνουν γρήγορα, αποτελεσματικά και σε περιβάλλον φιλικό προς το χρήστη.

3.2 Το κοινωνικό δίκτυο WatchMe

Η εφαρμογή που αναπτύσσω στην παρούσα διπλωματική εργασία ονομάζεται WatchMe. Παρέχει όλες τις βασικές δυνατότητες ενός κοινωνικού δικτύου (αιτήματα φιλίας, αποδοχή ή διαγραφή φίλων, δημοσιεύσεις) καθώς και λειτουργίες, όπως αυτές που αναφέρθηκαν παραπάνω που βοηθούν το χρήστη να αντιμετωπίζει περιπτώσεις κινδύνου. Η εφαρμογή προσφέρει ένα φιλικό περιβάλλον χρήστη, καθιστώντας δυνατή την καθημερινή της χρήση.

Αρχικά κάθε χρήστης εγγράφεται στην εφαρμογή με τη χρήση ενός email και ενός password. Για να αποφευχθούν ανενεργοί χρήστες και μη λειτουργικά email, έχει προστεθεί στην εφαρμογή λειτουργία επαλήθευσης email, πριν ο χρήστης αποθηκευτεί στη βάση δεδομένων. Αφού ένας χρήστης έχει εγγραφεί και επιβεβαιώσει το email του επιτυχώς, μπορεί να συνδεθεί (login) στην εφαρμογή. Επιπλέον, υπάρχει σύστημα επαναφοράς κωδικού, σε περίπτωση που ο χρήστης ξεχάσει τον κωδικό του, που επιτυγχάνεται με αποστολή email στο χρήστη.

Κάθε χρήστης μπορεί να αναπτύσσει δεσμούς φιλίας με άλλους χρήστες. Η διαδικασία είναι η εξής: ο χρήστης Α αναζητά το χρήστη Β με βάση το όνομά του, του στέλνει αίτημα φιλίας, ο χρήστης Β βλέπει το αίτημα και είτε το αποδέχεται είτε το απορρίπτει. Όλοι οι δεσμοί φιλίας, ολοκληρωμένοι ή σε φάση αίτησης, είναι αποθηκευμένοι στη βάση δεδομένων. Κάθε χρήστης

μπορεί να δει τους φίλους του μέσω του προφίλ του. Στο προφίλ επίσης μπορεί να αλλάξει τη φωτογραφία προφίλ του, να δει και να επεξεργαστεί τις δημοσιεύσεις του (η λειτουργία αυτή θα αναλυθεί στη συνέχεια) και να αποσυνδεθεί από την εφαρμογή (logout).

Οποιαδήποτε στιγμή ο χρήστης βρεθεί σε κίνδυνο, μπορεί να στείλει ένα σήμα κινδύνου σε όλους τους φίλους του, το οποίο καταγράφεται αυτόματα και στην βάση. Το σήμα αυτό -το οποίο στη συνέχεια θα αναφέρεται ως δημοσίευση- περιλαμβάνει το μοναδικό αναγνωριστικό του χρήστη που το δημιούργησε, την τοποθεσία του χρήστη εκείνη τη στιγμή σε γεωγραφικό πλάτος και μήκος, μία χρονοσφραγίδα, μία περιγραφή, καθώς και πληροφορία για το αν η αστυνομία χειρίστηκε ή όχι το περιστατικό. Τα δύο τελευταία στοιχεία μπορεί να τα επεξεργαστεί ο χρήστης εκ των υστέρων μέσω του προφίλ του. Κάθε χρήστης θα μπορεί να δει τις δημοσιεύσεις όλων των φίλων του στην αρχική σελίδα της εφαρμογής του.

Ταυτόχρονα, θα είναι δυνατό για το χρήστη να ελέγξει ανά πάσα στιγμή την ασφάλεια της περιοχής στην οποία βρίσκεται. Αυτό θα υπολογίζεται ανάλογα με το πόσα περιστατικά (δημοσιεύσεις) έχουν λάβει χώρα σε ακτίνα ενός χιλιομέτρου τον τελευταίο χρόνο, και θα παρουσιάζεται η διαφορά από τον προηγούμενο χρόνο. Ο υπολογισμός θα λαμβάνει υπ’ όψιν δημοσιεύσεις όλων των χρηστών της εφαρμογής και όχι μόνο των φίλων του ώστε το αποτέλεσμα να είναι όσο το δυνατόν πιο ρεαλιστικό.

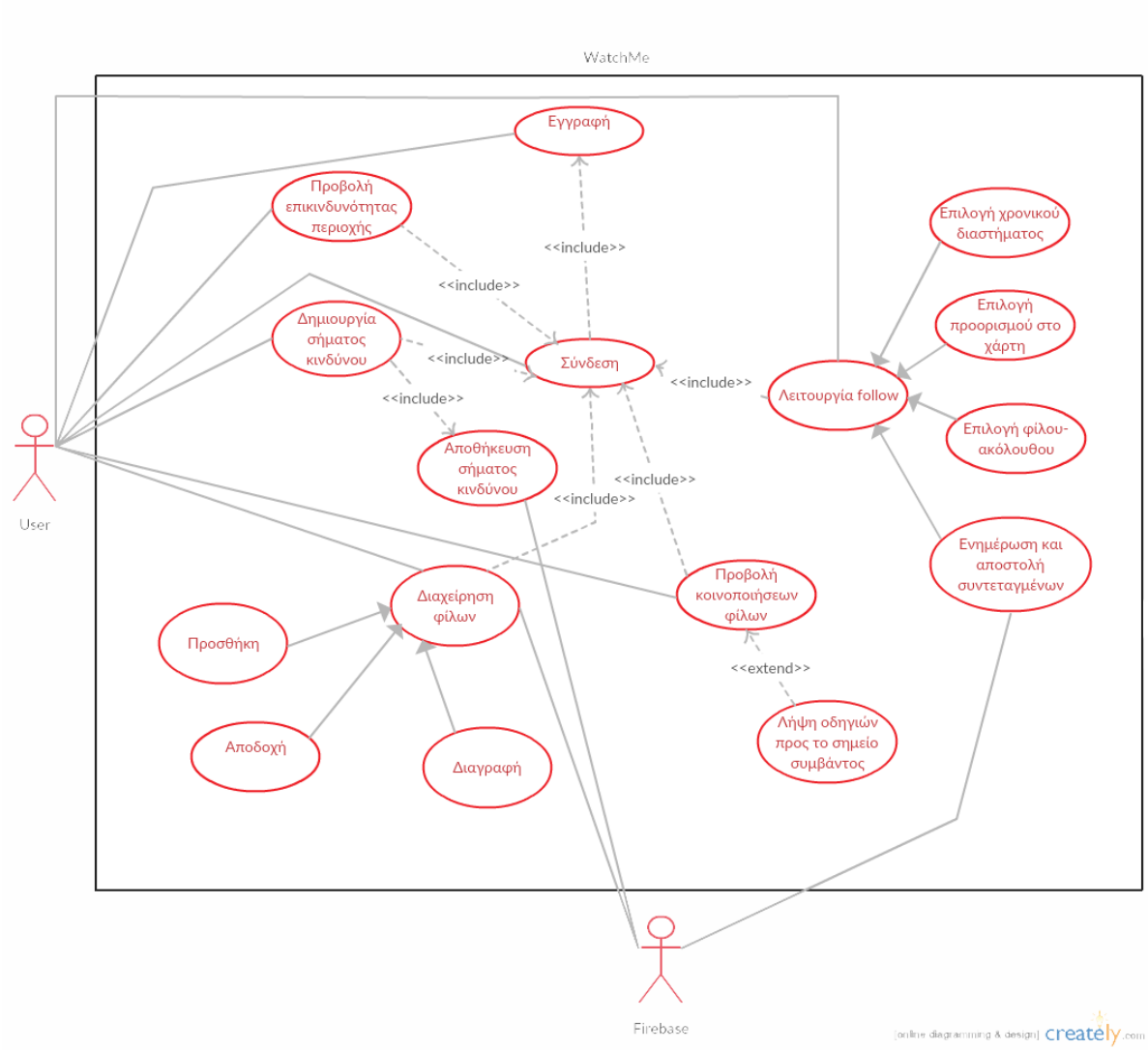
Τέλος, θα δίνεται η δυνατότητα στους χρήστες να μοιράζονται την τοποθεσία τους με κάποιο φίλο τους για συγκεκριμένο χρονικό διάστημα, το οποίο θα προσδιορίζεται είτε απευθείας από το χρήστη είτε έμμεσα από την εφαρμογή, αφού ο χρήστης έχει επιλέξει τον προορισμό του. Ιδιαίτερα αυτή η τελευταία λειτουργία μπορεί να προσφέρει μεγάλη αίσθηση ασφάλειας, εφόσον ένα έμπιστο άτομο ξέρει ανά πάσα στιγμή που βρίσκεσαι και μπορεί να καταφθάσει γρήγορα σε περίπτωση που συμβεί κάτι.

Συνοπτικά λοιπόν, οι λειτουργίες που πρέπει να περιλαμβάνει η εφαρμογή είναι:

- Εγγραφή χρήστη και αποθήκευσή του στη βάση
- Σύνδεση χρήστη
- Προσθήκη, αποδοχή, απόρριψη και διαγραφή φίλων
- Δημιουργία σήματος κινδύνου και αποθήκευσή του στη βάση
- Έλεγχος επικινδυνότητας περιοχής
- Προβολή δημοσιεύσεων των φίλων κάθε χρήστη
- Μετάβαση στο σημείο όπου συνέβη επείγον περιστατικό
- Επεξεργασία δημοσιεύσεων
- Καθορισμός χρονικού διαστήματος ή προορισμού και πρόσκληση σε φίλο να παρακολουθήσει το χρήστη για αυτό το διάστημα

3.3 Σενάρια χρήσης

Για την εξαγωγή των απαιτήσεων της εφαρμογής παρουσιάζεται το παρακάτω διάγραμμα σεναρίων χρήσης.



Εικόνα 13: Διάγραμμα σεναρίων χρήσης.

3.3.1 Εγγραφή χρήστη

Παρουσιάζονται τα βασικά βήματα εγγραφής νέου χρήστη στην εφαρμογή, μαζί με τα σφάλματα που μπορεί να προκύψουν.

3.3.1.1 Βασική ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Εκκίνηση εφαρμογής 2. Επιλογή συνδέσμου “Sign up”		
	3. Μετάβαση στη φόρμα εγγραφής	
4. Συμπλήρωση πεδίων: email, κωδικός, επιβεβαίωση κωδικού, όνομα		

	5. Έλεγχος συμπληρωθέντων στοιχείων 6. Αποστολή στοιχείων στο Firebase	
		7. Έλεγχος μοναδικότητας email χρήστη 8. Δημιουργία εγγραφής στο Authentication table 9. Δημιουργία εγγραφής στην Realtime Database 10. Αποστολή Verification email
	11. Εμφάνιση ειδοποίησης αποστολής verification email 12. Επιστροφή στην οθόνη sign up με κενά πεδία	

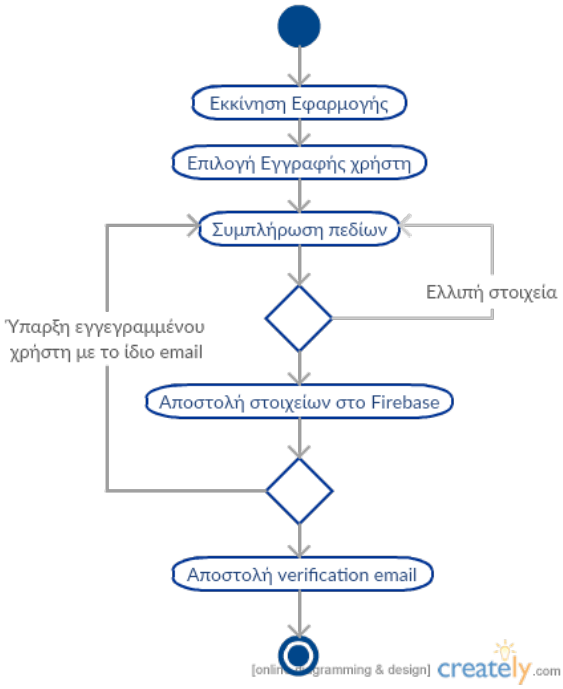
3.3.1.2 Ροή διαχείρησης σφάλματος – Ελλιπής συμπλήρωση φόρμας / λανθασμένη μορφή email

Χρήστης	Εφαρμογή	Firebase
	6. Εμφάνιση μηνύματος προσδιορισμού του λάθους 7. Επιστροφή στην οθόνη Sign up	

3.3.1.3 Ροή διαχείρησης σφάλματος – Μη μοναδικό email

Χρήστης	Εφαρμογή	Firebase
		8. Επιστροφή error
	9. Εμφάνιση μηνύματος μη μοναδικού email 10. Επιστροφή στην οθόνη Sign up	

3.3.1.4 Διάγραμμα δραστηριοτήτων



Εικόνα 14: Εγγραφή νέου χρήστη.

3.3.2 Σύνδεση χρήστη

Παρουσιάζονται τα βήματα σύνδεσης ενός χρήστη μαζί με τα σφάλματα που μπορεί να προκύψουν.

3.3.2.1 Βασική ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Εκκίνηση εφαρμογής 2. Συμπλήρωση πεδίων: email, κωδικός πρόσβασης		
	3. Έλεγχος συμπληρωθέντων στοιχείων 4. Αποστολή στοιχείων στο Firebase	
		5. Έλεγχος στοιχείων 6. Επιστροφή επιτυχούς αυθεντικοποίησης
	7. Εμφάνιση κεντρικής οθόνης	

3.3.2.2 Ροή διαχείρησης σφάλματος – Ελλιπής συμπλήρωση φόρμας

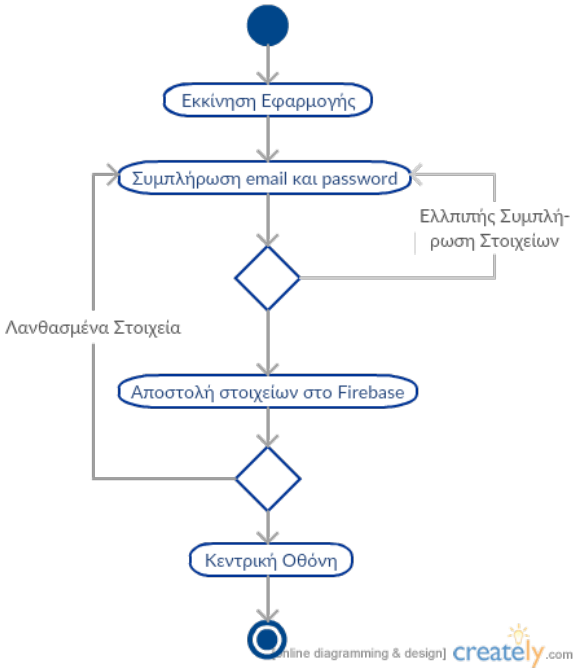
Χρήστης	Εφαρμογή	Firebase

	4. Εμφάνιση μηνύματος προσδιορισμού του λάθους 5. Επιστροφή στην οθόνη Login	

3.3.2.3 Ροή διαχείρησης σφάλματος – Λανθασμένο ζεύγος email, password

Χρήστης	Εφαρμογή	Firebase
		6. Επιστροφή σφάλματος αυθεντικοποίησης
	7. Εμφάνιση μηνύματος λανθασμένων στοιχείων 8. Επιστροφή στην οθόνη Login	

3.3.2.4 Διάγραμμα δραστηριοτήτων



Εικόνα 15: Σύνδεση χρήστη.

3.3.3 Διαχείρηση φίλων

Παρουσιάζονται οι βασικές λειτουργίες που σχετίζονται με την “φιλία” στο κοινωνικό δίκτυο. Θεωρούμε πως ο χρήστης είναι ήδη συνδεδεμένος.

3.3.3.1 Αποδοχή ή Απόρριψη αιτήματος φιλίας

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Profile”, κελιού “My Friends”		

	2. Αίτηση στο Firebase για τη λίστα φίλων και αιτημάτων φιλίας	
		3. Επιστροφή λιστών
	4. Εμφάνιση λιστών	
5. Στο section “PENDING REQUESTS”, επιλογή αποδοχής ή απόρριψης αιτήματος		
	6. Αποστολή πληροφορίας στο Firebase	
		7. Ανανέωση βάσης
	8. Ανανέωση λιστών	

3.3.3.2 Διαγραφή φίλου

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Profile”, κελιού “My Friends”		
	2. Αίτηση στο Firebase για τη λίστα φίλων και αιτημάτων φιλίας	
		3. Επιστροφή λιστών
	4. Εμφάνιση λιστών	
5. Στο section “MY FRIENDS”, επιλογή διαγραφής φίλου		
	6. Αποστολή πληροφορίας στο Firebase	
		7. Ανανέωση βάσης
	8. Ανανέωση λιστών	

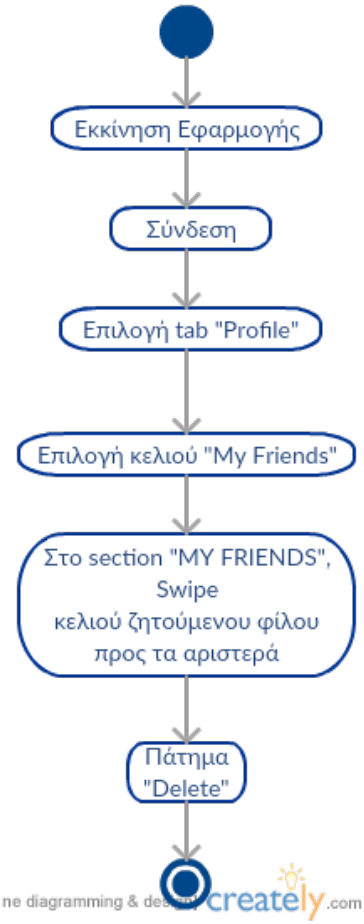
3.3.3.3 Προσθήκη νέου φίλου

Χρήστης	Εφαρμογή	Firebase

1. Επιλογή tab “Profile”, κελιού “My Friends”		
	2. Αίτηση στο Firebase για τη λίστα φίλων και αιτημάτων φιλίας	
		3. Επιστροφή λιστών
	4. Εμφάνιση λιστών	
5. Επιλογή κουμπιού “+” 6. Πληκτρολόγηση ονόματος φίλου		
	7. Ερώτημα στο Firebase για ονόματα χρηστών που ξεκινούν με τη συγκεκριμένη συμβολοσειρά	
		8. Επιστροφή λίστας
	9. Προβολή λίστας	
10. Επιλογή κελιού που αντιστοιχεί στον επιθυμητό χρήστη		
	11. Αποστολή πληροφορίας στο Firebase	
		12. Ανανέωση βάσης

3.3.3.4 Διαγράμματα δραστηριοτήτων

Εικόνα 16: Διαχειριση αιτήματος φιλίας.



Εικόνα 17: Διαγραφή φίλου.

Εικόνα 18: Προσθήκη νέου φίλου.

3.3.4 Δημιουργία σήματος κινδύνου

Παρουσιάζεται η διαδικασία δημιουργίας σήματος κινδύνου. Θεωρείται ότι ο χρήστης είναι ήδη συνδεδεμένος.

3.3.4.1 Ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Action” 2. Πάτημα κουμπιού “Danger”		
	3. Λήψη τοποθεσίας, ώρας 4. Αποστολή σήματος στο Firebase	
		5. Αποθήκευση σήματος (ως δημοσίευση) στη βάση

3.3.4.2 Διάγραμμα δραστηριοτήτων

Εικόνα 19: Δημιουργία σήματος κινδύνου.

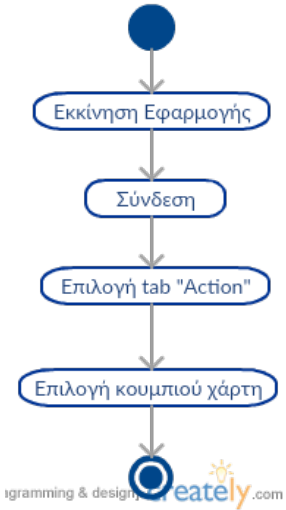
3.3.5 Προβολή επικινδυνότητας περιοχής

Παρουσιάζεται η διαδικασία προσδιορισμού επικινδυνότητας περιοχής. Θεωρείται ότι ο χρήστης είναι ήδη συνδεδεμένος.

3.3.5.1 Ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Action” 2. Πάτημα κουμπιού χάρτη		
	3. Αίτηση προς το firebase λίστας δημοσιεύσεων	
		4. Επιστροφή λίστας δημοσιεύσεων
	5. Φιλτράρισμα όσων δημοσιεύσεων συνέβησαν εντός ενός χιλιομέτρου 6. Εύρεση όσων συνέβησαν την τρέχουσα και την περασμένη χρονιά και υπολογισμός διαφοράς 7. Παρουσίαση των σημείων στο χάρτη και των αποτελεσμάτων/στατιστικών	

3.3.5.2 Διάγραμμα δραστηριοτήτων



Εικόνα 20: Έλεγχος επικινδυνότητας περιοχής.

3.3.6 Λειτουργία Follow

Παρουσιάζεται η διαδικασία ενεργοποίησης διαφορετικών μορφών της λειτουργίας follow. Θεωρείται ότι ο χρήστης είναι ήδη συνδεδεμένος.

3.3.6.1 Ενεργοποίηση follow για συγκεκριμένο χρονικό διάστημα

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Follow me” 2. Επιλογή “Choose time” 3. Επιλογή χρονικού διαστήματος (ώρες, λεπτά) 4. Πάτημα “Next” 5. Επιλογή φίλου 6. Πάτημα “Go”		
	7. Αποστολή των επιλογών στο firebase	
		8. Δημιουργία εγγραφής στη βάση
	9. Λήψη και αποστολή τοποθεσίας ανά τακτά χρονικά διαστήματα	
		10. Ανανέωση των δεδομένων στην εγγραφή
	11. Εμφάνιση τοπικής ειδοποίησης (local notification) μετά το πέρας του χρονικού διαστήματος	

3.3.6.2 Ενεργοποίηση follow μέχρις ότου να φτάσει ο χρήστης σε συγκεκριμένο προορισμό

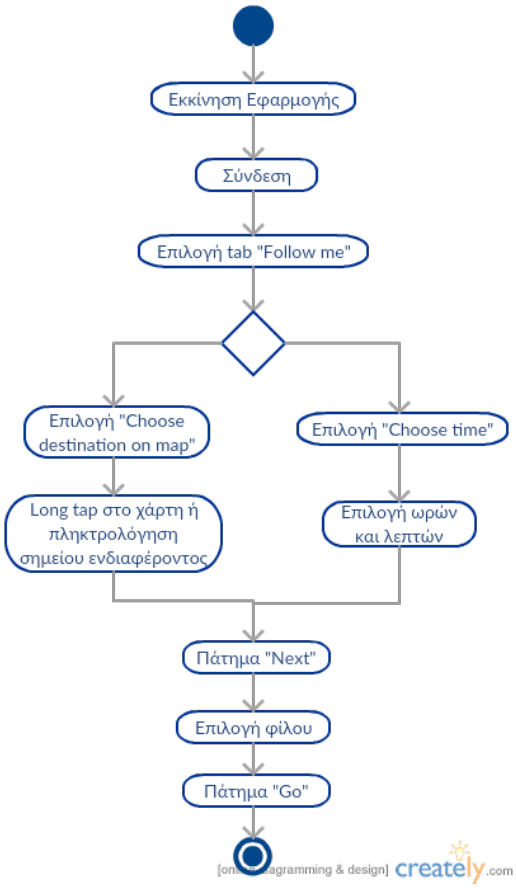
Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Follow me” 2. Επιλογή “Choose destination on Map” 3. Επιλογή προορισμού στο χάρτη με long tap ή αναζήτηση σημείου ενδιαφέροντος 4. Πάτημα “Next” 5. Επιλογή φίλου 6. Πάτημα “Go”		
	7. Υπολογισμός χρονικού διαστήματος ανάλογα με τον προορισμό 8. Αποστολή των επιλογών στο firebase	
		9. Δημιουργία εγγραφής στη βάση
	10. Λήψη και αποστολή τοποθεσίας ανά τακτά χρονικά διαστήματα	
		11. Ανανέωση των δεδομένων στην εγγραφή
	12. Εμφάνιση τοπικής ειδοποίησης (local notification) μετά το πέρας του αναμενόμενου χρονικού διαστήματος	

3.3.6.3 Αποδοχή πρόσκλησης follow από κάποιο φίλο

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Profile”, κελιού “My Friends”		

	2. Αίτηση στο Firebase για τη λίστα φίλων και αιτημάτων φιλίας	
		3. Επιστροφή λιστών
	4. Εμφάνιση λιστών	
5. Κλικ στο κελί του φίλου το οποίο είναι μαρκαρισμένο με την ένδειξη “follow”		
	6. Μετάβαση στην οθόνη χάρτη “Follow friend” 7. Λήψη από το Firebase της τοποθεσίας του φίλου όποτε αυτή ενημερώνεται	
		8. Αποστολή τοποθεσίας
	9. Ανανέωση τοποθεσίας φίλου στο χάρτη	

3.3.6.4 Διάγραμμα δραστηριοτήτων



Εικόνα 21: Ενεργοποίηση λειτουργίας follow.

3.3.7 Προβολή δημοσιεύσεων φίλων / Λήψη οδηγιών προς τοποθεσία συμβάντος

Παρουσιάζεται η διαδικασία προβολής συμβάντων από τους φίλους του χρήστη. Προαιρετικά, είναι δυνατή η εκκίνηση της εφαρμογής Maps με ταυτόχρονη λήψη οδηγιών προς την τοποθεσία κάποιου σημβάντος. Θεωρείται ότι ο χρήστης είναι ήδη συνδεδεμένος στην εφαρμογή.

3.3.7.1 Ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “Friends’ Posts”		
	2. Αίτηση στο Firebase για λήψη δημοσιεύσεων που δημιουργήθηκαν από φίλους	
		3. Αποστολή σχετικών δημοσιεύσεων
	4. Εμφάνιση των αποτελεσμάτων από το πιο πρόσφατο	
5. (προαιρετικά) Επιλογή κουμπιού “Go” στο κελί κάποιας συγκεκριμένης δημοσίευσης		
	6. Άνοιγμα εφαρμογής Maps με pin στην τοποθεσία του συγκεκριμένου συμβάντος	

3.3.7.2 Διάγραμμα δραστηριοτήτων

Εικόνα 22: Παρουσίαση δημοσιεύσεων φίλων και λήψη οδηγιών προς την τοποθεσία κάποιας από αυτές.

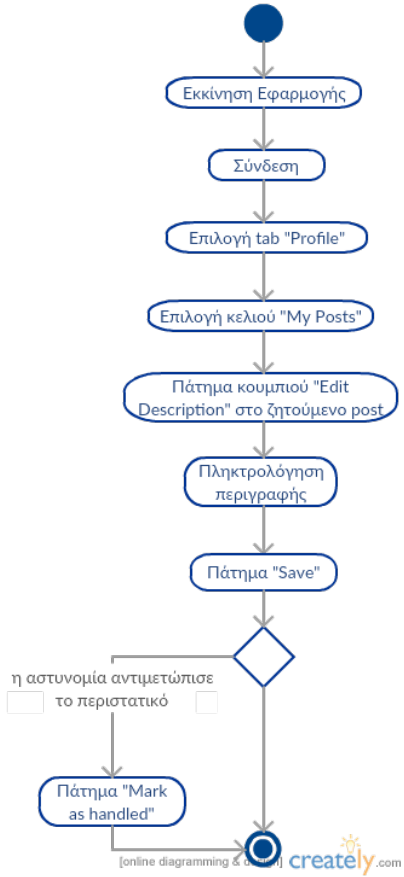
3.3.8 Επεξεργασία δημοσίευσης

Παρουσιάζεται η διαδικασία μέσω της οποίας ο χρήστης επεξεργάζεται δημοσιεύσεις του. Ο χρήστης θεωρείται ήδη συνδεδεμένος.

3.3.8.1 Ροή γεγονότων

Χρήστης	Εφαρμογή	Firebase
1. Επιλογή tab “My Profile”, κελιού “My Posts” 2. Εύρεση δημοσίευσης προς επεξεργασία 3. Πάτημα κουμπιού “Edit Description” 4. Πληκτρολόγηση περιγραφής 5. Πάτημα “Save”		
	6. Αποστολή τροποιημένης περιγραφής στο Firebase	
		7. Ενημέρωση εγγραφής στη βάση
8. (προαιρετικά) Πάτημα κουμπιού “Mark as handled” αν το περιστατικό αντιμετωπί-στηκε από την αστυνομία		
	9. Αποστολή δεδομένου στο Firebase	
		10. Ενημέρωση εγγραφής στη βάση

3.3.8.2 Διάγραμμα δραστηριοτήτων



Εικόνα 23: Επεξεργασία συμβάντος

Σχεδίαση

4.1 Η εφαρμογή WatchMe στο iOS

4.1.1 Οι οθόνες της εφαρμογής

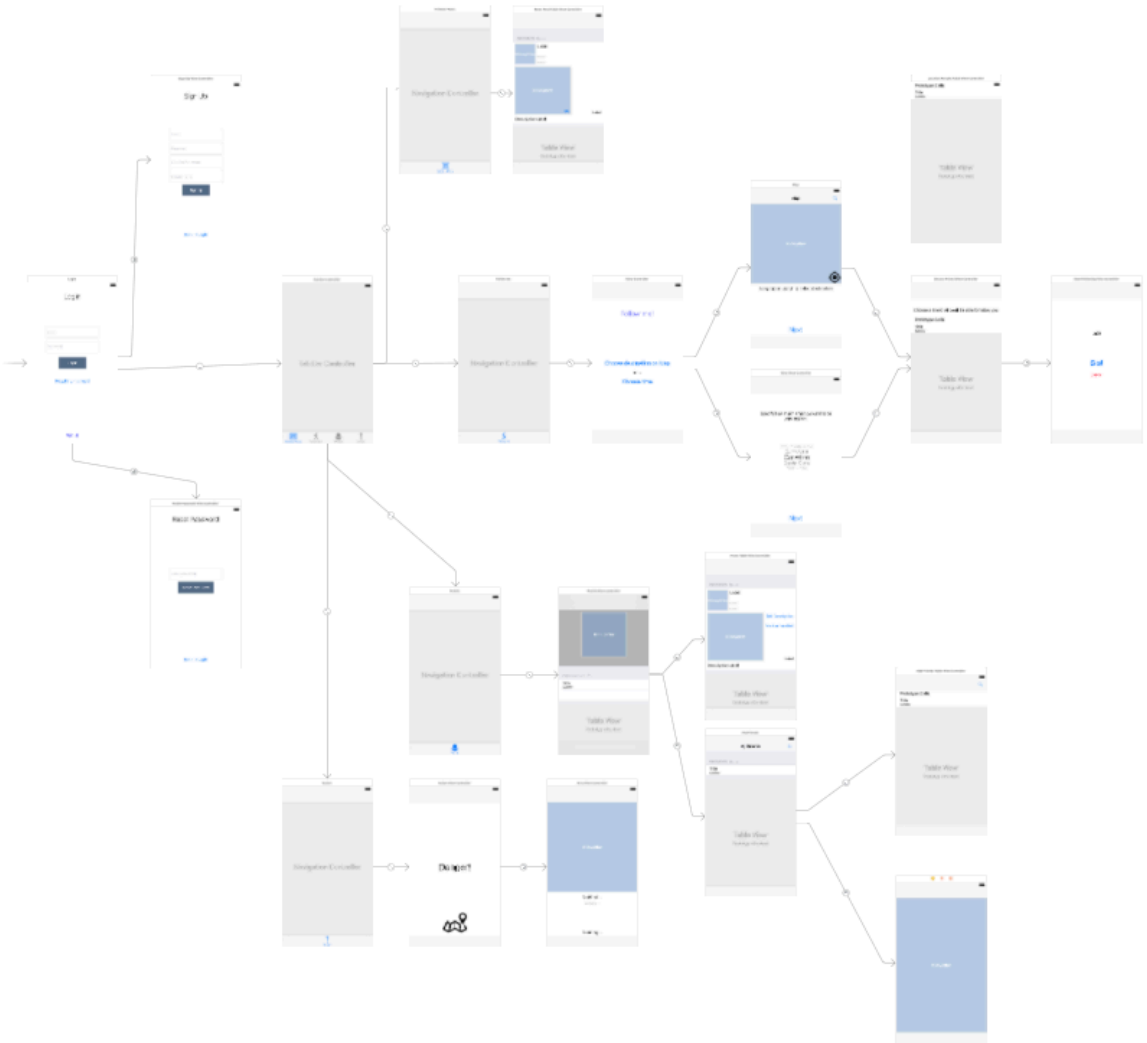
4.1.1.1 Περιγραφή οθονών

Προκειμένου να επιτευχθούν οι διάφορες λειτουργίες της εφαρμογής, έχουν δημιουργηθεί ορισμένες οθόνες. Παρακάτω αναφέρονται οι ιδιότητες της καθεμιάς.

- **Εναρκτήρια οθόνη Login**: Η πρώτη οθόνη που εμφανίζεται στο χρήστη όταν ανοίγει την εφαρμογή. Ο χρήστης μπορεί να συνδεθεί στο λογαριασμό του εισάγοντας τα στοιχεία του. Διαθέτει δύο ακόμη κουμπιά που οδηγούν στις οθόνες επαναφοράς κωδικού και εγγραφής.
- **Οθόνη επαναφοράς κωδικού**: Ο χρήστης είναι δυνατό, εισάγοντας το email του, να λάβει από το Firebase email επαναφοράς κωδικού σε περίπτωση που διαθέτει λογαριασμό και έχει ξεχάσει τον κωδικό εισόδου του.
- **Οθόνη εγγραφής**: Ένας νέος χρήστης μπορεί να παραθέσει τα στοιχεία του και να εγγραφεί στην εφαρμογή.
- **Οθόνη προβολής δημοσιεύσεων φίλων**: Παρατίθενται, η μία κάτω απ' την άλλη και ξεκινώντας από την πιο πρόσφατη, οι δημοσιεύσεις των φίλων του συνδεδεμένου χρήστη. Κάθε δημοσίευση περιλαμβάνει το όνομα και την φωτογραφία προφίλ του χρήστη, την ώρα, ημερομηνία και περιοχή όπου συνέβη το περιστατικό, έναν χάρτη με ένα pin στο ακριβές σημείο όπου συνέβη, ένα κουμπί λήψης οδηγιών, μία περιγραφή και μία ένδειξη για το αν η αστυνομία χειρίστηκε το περιστατικό ή όχι.
- **Οθόνη προφίλ**: Περιλαμβάνει τη φωτογραφία προφίλ του χρήστη, κουμπί αποσύνδεσης (logout) και συνδέσμους προς τις οθόνες φίλων και δημοσιεύσεων χρήστη.
- **Οθόνη δημοσιεύσεων χρήστη**: Περιλαμβάνει τις δημοσιεύσεις του συνδεδεμένου χρήστη (σήματα κινδύνου που εκείνος έχει δημιουργήσει) και επιτρέπει την επεξεργασία τους.
- **Οθόνη φίλων**: Περιλαμβάνει λίστα των φίλων του συνδεδεμένου χρήστη (με δυνατότητα διαγραφής φίλου) και τη λίστα των αιτημάτων φιλίας που εκκρεμούν (με δυνατότητα αποδοχής ή απόρριψης αιτήματος). Διαθέτει επίσης κουμπί που οδηγεί στην οθόνη προσθήκης νέου φίλου.
- **Οθόνη προσθήκης νέου φίλου**: Ο χρήστης μπορεί να πληκτρολογήσει το όνομα κάποιου χρήστη της εφαρμογής και να του στείλει αίτημα φιλίας.
- **Οθόνη παρακολούθησης φίλου**: Δυνατότητα ζωντανής παρακολούθησης της τοποθεσίας ενός φίλου όταν έχει ληφθεί πρόσκληση follow από τον συγκεκριμένο φίλο.
- **Οθόνη δράσης**: Δυνατότητα άμεσης αποστολής σήματος κινδύνου μέσω ενός κουμπιού. Διαθέτει σύνδεσμο προς την οθόνη προβολής επικινδυνότητας περιοχής.
- **Οθόνη προβολής επικινδυνότητας περιοχής**: Περιέχει χάρτη στον οποίο φαίνονται τα περιστατικά που συνέβησαν τον τελευταίο χρόνο εντός ενός χιλιομέτρου. Παρέχει ορισμένα στατιστικά.

- **Αρχική οθόνη follow**: Δίνει τη δυνατότητα στο χρήστη να προχωρήσει στην ενεργοποίηση της λειτουργίας follow είτε για συγκεκριμένο χρονικό διάστημα είτε επιλέγοντας προορισμό.
- **Οθόνη επιλογής προορισμού στο χάρτη** (εντός λειτουργίας follow): Ο χρήστης επιλέγει τον προορισμό του είτε με long tap στο χάρτη είτε με εύρεση σημείου ενδιαφέροντος.
- **Οθόνη αποτελεσμάτων αναζήτησης χάρτη** (εντός λειτουργίας follow): Εμφανίζει τα αποτελέσματα της αναζήτησης της προηγούμενης οθόνης, αν ο χρήστης πληκτρολογήσει σημείο ενδιαφέροντος ή διεύθυνση (αντί για long tap).
- **Οθόνη επιλογής χρονικού διαστήματος** (εντός λειτουργίας follow): Ο χρήστης επιλέγει το χρονικό διάστημα για το οποίο θα παρακολουθείται σε ώρες και λεπτά.
- **Οθόνη επιλογή φίλου-ακολούθου** (εντός λειτουργίας follow): Ο χρήστης επιλέγει το φίλο στον οποίο θα στείλει πρόσκληση για να τον παρακολουθεί.
- **Οθόνη ολοκλήρωσης λειτουργίας follow**: Περιέχει μια περίληψη των επιλογών του χρήστη και ένα κουμπί ενεργοποίησης την λειτουργίας follow.

4.1.1.2 Αλληλεπιδράσεις μεταξύ οθονών



Εικόνα 24: Οι αλληλεπιδράσεις μεταξύ όλων των οθονών της εφαρμογής

4.1.2 Περιγραφή των κλάσεων

4.1.2.1 Η κλάση AppDelegate

Η κλάση αυτή είναι ένας observer του αντικειμένου UIApplication. Αυτό σημαίνει ότι χειρίζεται ειδικές καταστάσεις στις οποίες μπορεί να φτάσει η εφαρμογή, φροντίζοντας για την ομαλή αλληλεπίδρασή της με το σύστημα καθώς και με άλλες εφαρμογές. Παραδείγματα γεγονότων τα οποία χειρίζεται αυτή η κλάση είναι η μετάβαση από foreground σε background mode και η λήψη ειδοποιήσεων (απομακρυσμένων ή τοπικών).

Όνομα κλάσης	**AppDelegate**	UIResponder
Μεταβλητές	*window*	UIWindow
Μέθοδοι	• ***application*(_:didFinishLaunchingWithOptions:)** Αρχικοποίηση και διαμόρφωση συνιστωσών της εφαρμογής. • ***applicationWillResignActive(_:)*** Καλείται πριν η εφαρμογή μεταβεί από ενεργή σε ανενεργή κατάσταση. Αυτό μπορεί να συμβεί λόγω διακοπών της, όπως σε περίπτωση εισερχόμενης κλήσης ή SMS. • ***applicationDidEnterBackground(_:)*** Καλείται μόλις η εφαρμογή περάσει στο background. • ***applicationWillEnterForeground(_:)*** Καλείται πριν η εφαρμογή επανέλθει στο foreground. • ***applicationDidBecomeActive(_:)*** Καλείται μόλις η εφαρμογή ξαναγίνει ενεργή, πιθανώς επανεκκινώντας εργασίες ή ανανεώνοντας το user interface. • ***applicationWillTerminate(_:)*** Καλείται αμέσως πριν τερματιστεί η εφαρμογή, πιθανώς σώζοντας απαραίτητα δεδομένα.	

4.1.2.2 Η κλάση Strongbox

Η κλάση αυτή λειτουργεί ως ένας wrapper του Keychain που διευκολύνει τις λειτουργίες του. Αποτελεί πρότζεκτ ανοιχτού κώδικα. Παρακάτω παρουσιάζονται μόνο οι υψηλού επιπέδου συναρτήσεις του.

Όνομα κλάσης	**Strongbox**	
Μεταβλητές	*keyPrefix*	String
	lastStatus	OSStatus
Μέθοδοι	• ***archive*** Εισάγει στο keychain ένα ζεύγος κλειδιού-τιμής. Η τιμή nil ισοδυναμεί με διαγραφή του συγκεκριμένου κλειδιού από το keychain. • ***unarchive*** Ανακτά από το keychain την τιμή ενός κλειδιού	

4.1.2.3 Κλάσεις οθονών

Κάθε οθόνη της εφαρμογής στο iOS περιγράφεται από μια συγκεκριμένη κλάση. Η μόνη οθόνη που δεν χρειάστηκε κώδικα για την υλοποίηση της είναι η αρχική οθόνη follow, λόγω του ότι

υλοποιεί εξαιρετικά απλές λειτουργίες που προσδιορίζονται μέσω του αρχείου “Main.storyboard”. Παρακάτω αναλύονται οι κλάσεις των προαναφερθέντων οθονών.

Όνομα κλάσης	**LoginViewController**	UIViewController
Μεταβλητές	*email_input*	UITextField
	password_input	UITextField
	login_button	UIButton
	sb	Strongbox
Μέθοδοι	• ***check_server*** Ελέγχει ασύγχρονα αν δοσμένο ζεύγος email και password αντιστοιχεί σε χρήστη. Επιστρέφει διαφορετικές τιμές για κάθε πιθανό error ή επιτυχία μέσω ενός handler. • ***check_keychain*** Ελέγχει αν υπάρχει στο keychain αποθηκευμένο ζεύγος email και password για την εφαρμογή. Αν υπάρχει, καλεί την check_server για αυτό το ζεύγος. • ***DoLogin*** Καλείται με το πάτημα του κουμπιού login_button και καλεί την check_server. Ανάλογα με το αποτέλεσμα αυτής εισάγει ή όχι τον χρήστη στο λογαριασμό του.	

Όνομα κλάσης	**ResetPasswordViewController**	UIViewController
Μεταβλητές	*resetButton*	UIButton
	emailField	UITextField
Μέθοδοι	• ***resetAction*** Καλείται με το πάτημα του κουμπιού resetButton. Αν το email έχει οριστεί σωστά στέλνει email επαναφοράς κωδικού, αλλιώς εμφανίζει μήνυμα λάθους.	

Όνομα κλάσης	**SignUpViewController**	UIViewController
Μεταβλητές	*emailTextField*	UITextField
	passwordTextField	UITextField
	confirmTextField	UITextField
	usernameTextField	UITextField
	signUpButton	UIButton
Μέθοδοι	• ***createAccountAction*** Καλείται μόλις πατηθεί το κουμπί signUpButton. Εμφανίζει πιθανά errors λόγω ελλιπούς συμπλήρωσης στοιχείων. Αν τέτοια errors δεν υπάρχουν, καλεί την create_account. • ***create_account*** Ελέγχει αν υπάρχει εγγεγραμμένος χρήστης με το ίδιο email. Αν όχι, εισάγει τον χρήστη στο Authentication table του firebase, στέλνει verification email και καλεί την create_record. • ***create_record***	

	Δημιουργεί εγγραφή για τον χρήστη στη βάση δεδομένων.

Όνομα κλάσης	**NewsFeedTableViewController**	UITableViewController
Μεταβλητές	*timeline*	[FIRDataSnapshot]
	myTimelineRef	FIRDatabaseReference
	user	FIRUser
	usersReference	FIRDatabaseReference
	locationManager	CLLocationManager
Μέθοδοι	• ***initializeObservers*** Αρχικοποιεί τους observers στα απαραίτητα references του firebase. • ***tableView(_:cellForRowAt:)*** Διαμορφώνει τα κελιά του πίνακα με βάση τις μεταβλητές της κλάσης NewsFeedCell.	
Περιεχόμενη κλάση	**NewsFeedCell**	UITableViewCell
Μεταβλητές NewsFeedCell	*goButton*	UIButton
	descriptionLabel	UILabel
	nameLabel	UILabel
	mapView	MKMapView
	placeLabel	UILabel
	profilePicture	UIImageView
	timeLabel	UILabel
	policeResult	UILabel

Όνομα κλάσης	**ProfileViewController**	UIViewController
Μεταβλητές	*profilePicture*	UIImageView
	profileTable	UITableView
	user	FIRUser

Μέθοδοι	• ***getProfilePicture*** Λαμβάνει –μέσω ενός observer- το snapshot που αντιστοιχεί στην εγγραφή του συνδεδεμένου χρήστη στη βάση, και μετατρέπει το URL της φωτογραφίας προφίλ σε εικόνα μέσω του framework Alamofire. • ***handleSelectProfileImageView*** Καλείται αν ο χρήστης κάνει κλικ πάνω στη φωτογραφία προφίλ. Εμφανίζει το Gallery του iPhone ώστε ο χρήστης να επιλέξει νέα φωτογραφία προφίλ. • ***imagePickerController(_:didFinishPickingMediaWithInfo:)*** Καλείται μόλις ο χρήστης επιλέξει φωτογραφία από το Gallery. Ανεβάζει τη νέα φωτογραφία στο Storage, λαμβάνει το νέο URL και το αποθηκεύει στην εγγραφή του χρήστη στη βάση. • ***logout*** Αποσυνδέει το χρήστη, διαγράφοντας τα στοιχεία του, αν υπάρχουν, από το keychain. Εκκινεί την αρχική οθόνη login.

Όνομα κλάσης	**PostsTableViewController**	UITableViewController
Μεταβλητές	*posts*	[FIRDataSnapshot]
	myPostsRef	FIRDatabaseReference
	user	FIRUser
	myUsersReference	FIRDatabaseReference
	postsRef	FIRDatabaseReference
	myImage	UIImageView
	myFriendsRef	FIRDatabaseReference
	myFriendsList	[String]
Μέθοδοι	• ***initialize_observers*** Αρχικοποιεί τους observers στα απαραίτητα references του firebase. • ***editPostPressed*** Καλείται μόλις πατηθεί το editButton κάποιου κελιού. Παρουσιάζει alert controller με δυνατότητα εισαγωγής κειμένου, το οποίο έπειτα ανανεώνει στις αντίστοιχες εγγραφές στο Firebase. • ***markPressed*** Καλείται μόλις πατηθεί το markHandled κουμπί κάποιου κελιού.Ανανεώνει τις αντίστοιχες εγγραφές στο firebase με την πληροφορία ότι το περιστατικό αντιμετωπίστηκε.	
Περιεχόμενη κλάση	**PostCell**	UITableViewCell
Μεταβλητές PostCell	*policeResult*	UILabel
	descriptionLabel	UILabel
	nameLabel	UILabel
	mapView	MKMapView
	placeLabel	UILabel

	profilePicture	UIImageView
	timeLabel	UILabel
	policeResult	UILabel
	editButton	UIButton
	markHandled	UIButton

Όνομα κλάσης	**FriendsTableViewController**	UITableViewController
Μεταβλητές	*friendsList*	[String:Bool]
	pendingList	[String:Bool]
	wantedList	[String:Bool]
	user	FIRUser
	myFriendsReference	FIRDatabaseReference
	usersReference	FIRDatabaseReference
	timelineRef	FIRDatabaseReference
	followRef	FIRDatabaseReference
Μέθοδοι	• ***initialize_observers*** Αρχικοποιεί τους observers στα απαραίτητα references του firebase. • ***tableView(_:cellForRowAt:)*** Αρχικοποιεί τα κελιά, ταξινομώντας τους φίλους σε αυτούς που έχουν προστεθεί και σε αυτούς που έχουν στείλει αίτημα. Από αυτούς που έχουν προστεθεί, ελέγχει ποιοι έχουν στείλει στο συνδεδεμένο χρήστη πρόσκληση follow. • ***tableView(_:didSelectRowAt:)*** Για τα κελιά των φίλων που έχουν στείλει στο συνδεδεμένο χρήστη πρόσκληση follow, δημιουργεί σύνδεσμό τους με την οθόνη παρακολούθησης φίλου. • ***prepare(for:sender:)*** Προετοιμάζει την οθόνη παρακολούθησης φίλου στέλνοντάς της τις απαραίτητες πληροφορίες.	

Όνομα κλάσης	**AddFriendsTableViewController**	UITableViewController
Μεταβλητές	*searchBarButton*	UIBarButtonItem
	searchController	UISearchController
	filtered	[FIRDataSnapshot]
	user	FIRUser
	usersReference	FIRDatabaseReference
	myFriendsReference	FIRDatabaseReference
Μέθοδοι	• ***initialize_search*** Κάνει τις απαραίτητες αρχικοποιήσεις για να χρησιμοποιηθεί ο searchController • ***searchBar(_:textDidChange:)*** Μόλις αλλάζει το κείμενο στην αναζήτηση, ανανεώνεται ένα query στη βάση ώστε να ληφθούν όλοι οι χρήστες με όνομα που αρχίζει με τη συμβολοσειρά της αναζήτησης.	

	• ***tableView(_:didSelectRowAt:)*** Κάνει τις απαραίτητες ενέργειες που περιλαμβάνει ένα αίτημα φιλίας, μόλις ο χρήστης κάνει κλικ στο κελί κάποιου χρήστη που δεν είναι φίλος του.

Όνομα κλάσης	**FollowMapViewController**	UIViewController
Μεταβλητές	*mapView*	MKMapView
	pointAnnotation	MKPointAnnotation
	friendID	String
	wantedRef	FIRDatabaseReference
	currentRef	FIRDatabaseReference
	destRef	FIRDatabaseReference
	friendRef	FIRDatabaseReference
	user	FIRUser
	lat	CLLocationDegrees
	lon	CLLocationDegrees
Μέθοδοι	• ***initialize_observers*** Αρχικοποιεί τους observers στα απαραίτητα references του firebase. • ***updateAnnotation*** Καλείται κάθε φορά που ανανεώνεται η τοποθεσία του φίλου τον οποίο παρακολουθεί ο χρήστης. Ανανεώνει τον χάρτη ώστε να φαίνεται η νέα τοποθεσία.	

Όνομα κλάσης	**ActionViewController**	UIViewController
Μεταβλητές	*showAreaButton*	UIButton
	dangerButton	UIButton
	user	FIRUser
	ref	FIRDatabaseReference
	postsRef	FIRDatabaseReference
	timelineRef	FIRDatabaseReference
	myFriendsRef	FIRDatabaseReference
	locManager	CLLocationManager
	myFriendsList	[String]
Μέθοδοι	• ***inDanger*** Καλείται με το πάτημα του dangerButton. Λαμβάνει την ώρα, ημερομηνία και τοποθεσία και δημιουργεί εγγραφή του περιστατικού στη βάση.	

Όνομα κλάσης	**AreaViewController**	UIViewController
Μεταβλητές	*numberLabel*	UILabel
	comparisonLabel	UILabel
	myAreaMap	MKMapView
	resultLabel	UILabel
	circle	MKCircle
	center	CLLocationCoordinate2D

	rad	CLLocationDistance
	locationManager	CLLocationManager
	postsRef	FIRDatabaseReference
	nearestEvents	[FIRDataSnapshot]
	num	Int
Μέθοδοι	• ***locationManager(_:didUpdateLocations:)*** Καλείται κάθε φορά που συμβαίνει μια σημαντική αλλαγή στην τοποθεσία του χρήστη. Καλεί τις updateNearestPosts, calculateLastYear, updateAnnotations • ***updateNearestPosts*** Λαμβάνει από τη βάση και αποθηκεύει στον πίνακα nearestEvents τα περιστατικά που συνέβησαν εντός ενός χιλιομέτρου από την παρούσα τοποθεσία του χρήστη. • ***calculateLastYear*** Υπολογίζει τα περιστατικά που συνέβησαν τον τελευταίο χρόνο και κρατά μόνο αυτά στον πίνακα nearestEvents. Υπολογίζει τα περιστατικά που συνέβησαν τον προτελευταίο χρόνο. Υπολογίζει τη διαφορά τους και ανανεώνει τα labels. • ***updateAnnotations*** Ανανεώνει τον χάρτη τοποθετώντας pins στα σημεία όπου συνέβησαν τα εναπομείναντα περιστατικά του πίνακα nearestEvents.	

Όνομα κλάσης	**MapViewController**	UIViewController
Μεταβλητές	*nextButton*	UIButton
	mapView	MKMapView
	zoomToMe	UIButton
	searchController	UISearchController
	pointAnnotation	MKPointAnnotation
	locationManager	CLLocationManager
	userLocation	CLLocation
	quickestRouteForSegment	MKRoute
	selectedPin	MKPlacemark
	minutes	Int
Μέθοδοι	• ***determineMyCurrentLocation*** Αρχικοποίηση του location manager και zoom στην τοποθεσία του χρήστη. • ***addAnnotationOnLongPress*** Δημιουργεί pin στο σημείο του χάρτη όπου ο χρήστης θα κάνει long tap. Καλεί την createRoute • ***createRoute*** Δημιουργεί γραμμή πάνω στο χάρτη η οποία δείχνει τον τρόπο με τον οποίο ο χρήστης θα φτάσει εκεί από τον παρούσα τοποθεσία του. • ***showSearchBar***	

	Καλείται μόλις πατηθεί το κουμπί αναζήτησης. Αρχικοποιεί τον search controller και ορίζει σαν οθόνη αποτελεσμάτων την LocationResultsTableViewController • ***dropPin*** Ανήκει στην επέκταση HandleMapSearch του MapViewController. Λαμβάνει ως όρισμα ένα placemark στο σημείο του οποίου τοποθετεί ένα pin.

Όνομα κλάσης	**LocationResultsTableViewController**	UITableViewController
Μεταβλητές	*matchingItems*	[MKMapItem]
	mapView	MKMapView
	handleMapSearchDelegate	HandleMapSearch
Μέθοδοι	• ***parseAddress*** Από μία δηεύθυνση ή σημείο ενδιαφέροντος που έχει πληκτρολογήσει ο χρήστης, δημιουργεί ευανάγνωστη συμβολοσειρά ώστε να τοποθετηθεί στο αντίστοιχο κελί του πίνακα αποτελεσμάτων. • ***tableView(_:didSelectRowAt:)*** Καλεί την dropPin για το χάρτη της προηγούμενης οθόνης, με όρισμα το σημείο ενδιαφέροντος που επέλεξε ο χρήστης στον πίνακα αναζητησης. • ***updateSearchResults*** Ανήκει στην επέκταση UISearchResultsUpdating του LocationResultsTableViewController. Καλείται κάθε φορά που αλλάζει το κείμενο αναζήτησης, ενημερώνοντας τα αποτελέσματα στον πίνακα.	

Όνομα κλάσης	**TimeViewController**	UIViewController
Μεταβλητές	*timePicker*	UIPickerView
Μέθοδοι	• ***prepare(for:sender:)*** Προετοιμάζει την επόμενη οθόνη στέλνοντας της το χρόνο σε λεπτά που επέλεξε ο χρήστης.	

Όνομα κλάσης	**ChooseFriendViewController**	UIViewController
Μεταβλητές	*minutes*	Int
	destlat	CLLocationDegrees
	destlon	CLLocationDegrees
	friendsList	[String]
	user	FIRUser
	myFriendsReference	FIRDatabaseReference
	usersReference	FIRDatabaseReference
	myTable	UITableView
Μέθοδοι	• ***prepare(for:sender:)*** Προετοιμάζει την επόμενη οθόνη (Οθόνη ολοκλήρωσης λειτουργίας follow) στέλνοντας της το χρόνο σε λεπτά που	

	επέλεξε ο χρήστης, τον προορισμό του, και το όνομα και user ID του φίλου στον οποίο θα στείλει πρόσκληση follow.

Όνομα κλάσης	**StartFollowingViewController**	UIViewController
Μεταβλητές	*minutes*	Int
	destlat	CLLocationDegrees
	destlon	CLLocationDegrees
	followerID	String
	followerName	String
	locManager	CLLocationManager
	ref	FIRDatabaseReference
	postsRef	FIRDatabaseReference
	timelineRef	FIRDatabaseReference
	myFriendsRef	FIRDatabaseReference
	user	FIRUser
	myFriendsList	[String]
	followCurRef	FIRDatabaseReference
	followDestRef	FIRDatabaseReference
	currentLocation	CLLocation
	goButton	UIButton
	messageLabel	UILabel
	cancelButton	UIButton
Μέθοδοι	• ***goTapped*** Καλείται μόλις πατηθεί το goButton. Λαμβάνει την τρέχουσα τοποθεσία του χρήστη και καλεί την setupNotificationSettings και την enable_following. • ***setupNotificationSettings*** Προγραμματισμός τοπικής ειδοποίησης μετά από τόσα λεπτά όσα δηλώνει η μεταβλητή minutes, εμπλουτισμένης με συγκεκριμένες ενέργειες • ***enable_following*** Ανεβάζει στο firebase, σε ειδικό σημείο του JSON δέντρου, την τρέχουσα τοποθεσία και τον πιθανό προορισμό του χρήστη, ώστε να τα διαβάσει από εκεί ο φίλος-ακόλουθος.	

4.2 Η εφαρμογή WatchMe στο Firebase

4.2.1 Η τεχνική Fan-out

Το fan-out είναι μια τεχνική που εφαρμόζεται στις βάσεις δεδομένων. Περιλαμβάνει την παραγωγή διπλότυπων αντιγράφων (duplicates) από τα δεδομένα, αποφεύγοντας έτσι τις χρονοβόρες ενώσεις (joins) και επιταχύνοντας τις λειτουργίες ανάγνωσης. Φυσικά, η τεχνική αυτή αυξάνει τη χωρική πολυπλοκότητα, απαιτώντας έτσι από τον προγραμματιστή καλή σχεδίαση της βάσης και προσδιορισμό των αντιγράφων ανάλογα με τις ανάγκες της εφαρμογής.

Το δύσκολο κομμάτι της τεχνικής fan-out είναι ότι τα διπλότυπα πρέπει να παραμένουν ενημερωμένα. Η τελευταία έκδοση των Firebase SDKs προνοεί για αυτό, παρέχοντας τη δυνατότητα ενημέρωσης αντιγράφων σε πολλαπλά μονοπάτια με ατομικές λειτουργίες. Με αυτό τον τρόπο ενημερώνονται είτε όλα τα αντίγραφα ή κανένα (σε περίπτωση βλάβης του συστήματος), διατηρώντας τη βάση συνεπή και ταυτόχρονα αποδοτική. Ο συνδυασμός της λογικής του Firebase και της τεχνικής fan-out ονομάζεται client-side fan-out technique.

Η τεχνική αυτή χρησιμοποιείται πλέον από τους γίγαντες των μέσων κοινωνικής δικτύωσης. Όταν μια εφαρμογή αποκτά όλο και περισσότερους χρήστες, βασική μέριμνα είναι η ικανοποίηση των χρηστών σε βάρος της εξοικονόμησης χώρου. Για παράδειγμα, όταν ένας χρήστης κάνει μια δημοσίευση, η δημοσίευση αυτή πρέπει να εμφανιστεί στις αρχικές σελίδες όλων των φίλων του. Δεν είναι λοιπόν δυνατό, κάθε φορά που ο χρήστης φορτώνει την αρχική του σελίδα να ψάχνει τις δημοσιεύσεις όλων των φίλων του και μάλιστα αυτές να ταξινομούνται και χρονολογικά. Αυτό που γίνεται είναι ότι για κάθε χρήστη υπάρχει ένας κόμβος timeline στο JSON δέντρο, και όταν κάποιος χρήστης κάνει μια δημοσίευση αυτή γράφεται απευθείας στο timeline κάθε φίλου του. Το ίδιο ακριβώς συμβαίνει και με την ενημέρωση μιας δημοσίευσης.

Το fan-out υλοποιείται μέσω ενός αντικειμένου (συνήθως πίνακα) το οποίο διατηρεί όλα τα μονοπάτια-προορισμούς για τα δεδομένα που πρέπει να γραφούν στη βάση. Έπειτα, εκτελείται λειτουργία insert ή update στο αντικείμενο αυτό, αντί σε κάθε μονοπάτι ξεχωριστά. Έτσι η λειτουργία καθίσταται ατομική. Μπορούμε να πούμε ότι το εν λόγω αντικείμενο κατά κάποιο τρόπο «διαχέει» τα δεδομένα. Στη συνέχεια θα παρουσιάσω αναλυτικά πως αξιοποιήθηκε η τεχνική αυτή σε διάφορα σημεία της εφαρμογής.

4.2.2 Το JSON δέντρο

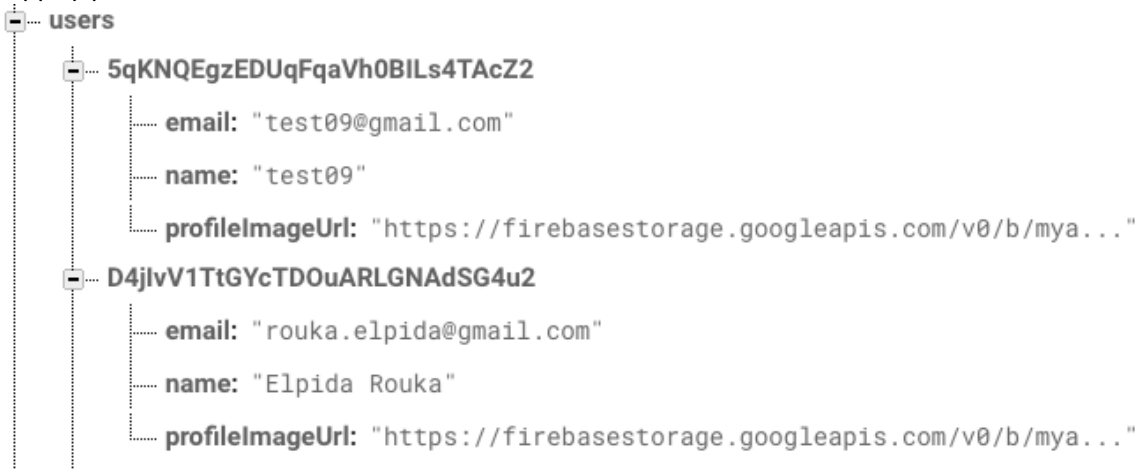
Κάτω από τη ρίζα του δέντρου υπάρχουν 6 αρχικοί κόμβοι. Αυτοί είναι οι κόμβοι “posts”, “timeline”, “users”, “users-follow”, “users-friends” και “users-posts”.

Εικόνα 25: Οι αρχικοί κόμβοι στο JSON δέντρο της εφαρμογής.

4.2.2.1 Κόμβος users

Εδώ αποθηκεύονται τα στοιχεία κάθε χρήστη και συγκεκριμένα το email του, το όνομά του και το URL της φωτογραφίας προφίλ του. Κάθε παιδί του κόμβου αυτού είναι το id του

συγκεκριμένου χρήστη, που είναι μοναδικό και δημιουργείται κατά την εγγραφή του στην εφαρμογή.

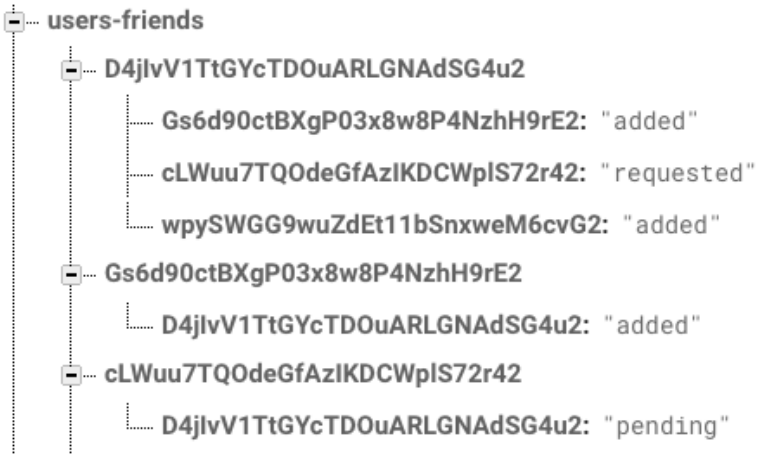


Εικόνα 26: Τμήμα κόμβου "users".

4.2.2.2 Κόμβος users-friends

Ο κόμβος αυτός διατηρεί δεδομένα για τα στάδια φιλίας μεταξύ των χρηστών. Κάθε παιδί του είναι το id κάποιου χρήστη, για τον οποίο εμφανίζονται:

- Οι χρήστες με τους οποίους είναι ήδη φίλος (added)
- Οι χρήστες στους οποίους έχει στείλει αίτημα φιλίας (requested)
- Οι χρήστες οι οποίοι του έχουν στείλει αίτημα φιλίας (pending)



Εικόνα 27: Τμήμα κόμβου "users-friends".

Για παράδειγμα, στο παραπάνω μέρος του δέντρου, φαίνεται ότι ο χρήστης με UID D4jlvV1TtGYcTDOuARLGNAdSG4u2 έχει στείλει αίτημα φιλίας στο χρήστη με UID cLWuu7TQOdeGfAzIKDCWplS72r42, εκείνος όμως δεν έχει απαντήσει ακόμα. Αν ο τελευταίος απαντήσει θετικά, και οι δύο εγγραφές θα αποκτήσουν την τιμή “added”, αν απαντήσει αρνητικά και οι δύο εγγραφές θα διαγραφούν. Αξίζει να παρατηρήσουμε εδώ ότι οι πληροφορίες για την φιλία αποθηκεύονται ξεχωριστά από τον κόμβο “users”, ενώ θα μπορούσαν να αποθηκευτούν σε αυτόν. Ο λόγος που συμβαίνει αυτό είναι ότι, κατά το αίτημα φιλίας, ο client του αιτούντα δημιουργεί μια εγγραφή στον κόμβο του αιτούμενου με το UID του και την τιμή “pending”. Παρ’ όλα αυτά, όπως θα φανεί αργότερα στην ανάλυση των κανόνων ασφαλείας, δεν θέλουμε άλλοι χρήστες να έχουν δικαιώματα εγγραφής στα προσωπικά στοιχεία κάθε χρήστη στον κόμβο “users”. Το πρόβλημα αυτό παρακάμπτεται με την αποθήκευση των αιτημάτων στον ξεχωριστό κόμβο “users-friends”.

4.2.2.3 Κόμβος posts

Αυτό είναι ένα από τα σημεία όπου γράφεται κάθε post που εκτελεί κάθε χρήστης, περικλείοντας έτσι όλες τις δημοσιεύσεις τις εφαρμογής. Κατά την αρχική δημιουργία κάθε post παράγεται ένα μοναδικό αναγνωριστικό που αποτελεί τον "τίτλο" του post σε όλα τα υπόλοιπα σημεία της βάσης όπου αυτό θα γραφεί. Ο κόμβος αυτός είναι ιδιαίτερα χρήσιμος στην οθόνη προβολής επικινδυνότητας περιοχής όπου λαμβάνονται υπ' όψη οι δημοσιεύσεις όλων των χρηστών, ασχέτως αν είναι φίλοι ή όχι του συνδεδεμένου χρήστη. Ομοίως, αυτός ο κόμβος είναι ο μόνος τον οποίο χρησιμοποιεί το Widget της εφαρμογής, για να εκτελέσει μια παρόμοια, απλοποιημένη λειτουργία της προαναφερθείσας οθόνης.

Κάθε post περιλαμβάνει 6 πεδία: την περιγραφή του (description), την τοποθεσία όπου συνέβη σε γεωμετρικό πλάτος (latitude) και μήκος (longitude), μία Boolean τιμή που δηλώνει αν αντιμετωπίστηκε από την αστυνομία (policeArrived), μία χρονοσφραγίδα (timestamp) και το UID του χρήστη ο οποίος το προκάλεσε (user). Ένα μέρος του κόμβου posts φαίνεται παρακάτω.

Εικόνα 28: Τμήμα κόμβου "posts".

4.2.2.4 Κόμβος users-posts

Ένα από τα σημεία στα οποία γίνεται fan-out το κάθε post. Κάθε παιδί του κόμβου αυτού έχει ως τίτλο κάποιο UID, κάτω από το οποίο αποθηκεύονται όλα τα posts του συγκεκριμένου χρήστη. Ο κόμβος αυτός είναι ιδιαίτερα χρήσιμος στην οθόνη δημοσιεύσεων χρήστη, ώστε να μπορούν να διαβάζονται όλα τα posts του κάθε χρήστη ταχύτατα, σε μία λειτουργία read. Παρακάτω φαίνεται ένα μέρος του συγκεκριμένου κόμβου.

Εικόνα 29: Τμήμα κόμβου "users-posts".

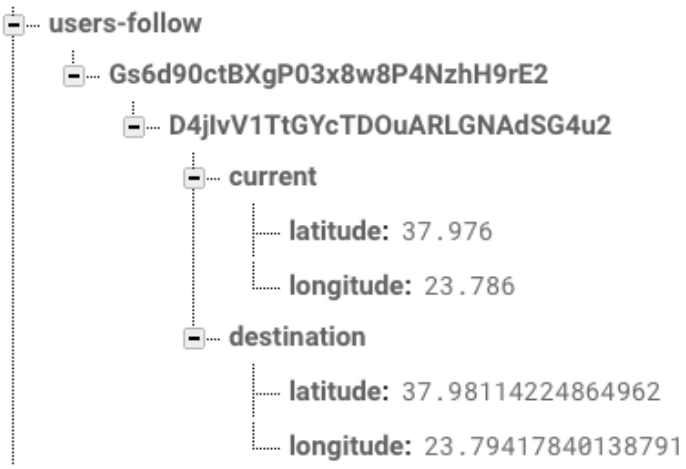
4.2.2.5 Κόμβος timeline

Σε αυτόν τον κόμβο υλοποιείται η ουσιαστική λειτουργία της τεχικής fanout που περιγράψαμε προηγουμένως. Κάθε παιδί του έχει τίτλο το UID κάποιου χρήστη και περιλαμβάνει όλα τα posts των φίλων του συγκεκριμένου χρήστη. Προφανώς, ο κόμβος αυτός αξιοποιείται στην οθόνη προβολής δημοσιεύσεων φίλων.

Εικόνα 30: Τμήμα κόμβου "timeline".

4.2.2.6 Κόμβος users-follow

Ο κόμβος αυτός χρησιμοποιείται κατά τη λειτουργία follow. Τα παιδιά του έχουν τίτλους UIDs χρηστών οι οποίοι έχουν δεχτεί πρόσκληση να παρακολουθήσουν κάποιον/κάποιους φίλους τους. Οι χρήστες οι έχουν στείλει την πρόσκληση φαίνονται στο αμέσως επόμενο επίπεδο ιεραρχίας. Τέλος, σε κάθε χρήστης που έχει στείλει πρόσκληση σε κάποιον φίλο του, αντιστοιχούν δύο κόμβοι “current” και “destination” στους οποίους καταγράφεται η τωρινή τοποθεσία του χρήστη και ο προορισμός του αντίστοιχα. Αν ο χρήστης έχει επιλέξει να παρακολουθείται για συγκεκριμένο χρονικό διάστημα χωρίς να προσδιορίσει προορισμό, το πεδίο του προορισμού είναι κενό.



Εικόνα 31: Τμήμα κόμβου "users-follow".

Το παραπάνω τμήμα δηλώνει ότι ο χρήστης με UID Gs6d90ctBXgP03x8w8P4NzhH9rE2 έχει προσκληθεί να παρακολουθήσει το χρήστη με UID D4jlvV1TtGYcTDOuARLGNAdSG4u2 του οποίου η τωρινή τοποθεσία βρίσκεται στον κόμβο current (ο οποίος συνεχώς ανανεώνεται) και ο προορισμός του βρίσκεται στον κόμβο destination (ο οποίος παραμένει αμετάβλητος).

4.2.3 Οι κανόνες ασφαλείας

Παρακάτω παρατίθενται όλοι οι κανόνες ασφαλείας που χρειάστηκαν για την εφαρμογή WatchMe.

```
{
  "rules": {
    "posts": {
      ".read": true,
      "$postid": {
        ".write": "auth != null",
        ".validate": "newData.hasChildren(['user', 'policeArrived', 'timestamp',
'latitude', 'longitude'])",

       "user": {
         ".validate": "newData.val() == auth.uid"
       },
       "policeArrived": {
         ".validate": "newData.isBoolean()"
       },
       "timestamp": {
         ".validate": "newData.isNumber()"
       },
       "latitude": {
         ".validate": "newData.isNumber()"
       },
       "longitude": {
          ".validate": "newData.isNumber()"
       }
     }
    },
    "users": {
       ".read": "auth != null",

       "$uid": {
         ".write": "auth.uid == $uid"
       }
    },
    "timeline": {
      "$userid": {
        ".read": "auth.uid == $userid",
```

```
        ".write": "root.child('users-friends').child(auth.uid).child($userid).
      val() == 'added'",

       "$postid": {
            ".read": "root.child('timeline').child($userid).child($postid).
            child('user').val() == auth.uid",

            ".validate": "newData.hasChildren(['user', 'policeArrived',
      'timestamp', 'latitude', 'longitude'])",

            "user": {
            ".validate": "newData.val() == auth.uid"
            },
            "policeArrived": {
              ".validate": "newData.isBoolean()"
            },
            "timestamp": {
              ".validate": "newData.isNumber()"
            },
            "latitude": {
              ".validate": "newData.isNumber()"
            },
            "longitude": {
             ".validate": "newData.isNumber()"
            }
         }
       }
    },
    "users-posts": {
      "$userid": {
        ".read": "auth.uid == $userid",
        ".write": "auth.uid == $userid",
        "$postid": {
            ".validate": "newData.hasChildren(['user', 'policeArrived',
'timestamp', 'latitude', 'longitude'])",

            "user": {
            ".validate": "newData.val() == auth.uid"
            },
            "policeArrived": {
              ".validate": "newData.isBoolean()"
            },
            "timestamp": {
              ".validate": "newData.isNumber()"
            },
            "latitude": {
              ".validate": "newData.isNumber()"
            },
            "longitude": {
             ".validate": "newData.isNumber()"
            }
        }
      }
    },
    "users-follow":{
      "$follower": {
        ".read": "$follower == auth.uid",

        "$walker": {
          ".write": "$walker == auth.uid && root.child('users-
friends').child($follower).child(auth.uid).val() == 'added'"
        }
      }
    },
    "users-friends": {
```

```
      "$u1": {
        ".read": "$u1 == auth.uid",

        "$u2": {
          ".write": "$u2 == auth.uid && !data.exists() && newData.val()=='pending'
|| $u1 == auth.uid && !data.exists() && newData.val()=='requested' || $u1 ==
auth.uid && data.val()=='pending' && newData.val()=='added' || $u2 == auth.uid &&
data.val()=='requested' && newData.val()=='added' || !newData.exists()"
        }
      }
    }
  }
}
```

Αρχικά, στον κόμβο posts ορίζουμε ότι όλοι οι χρήστες, αυθεντικοποιημένοι ή όχι, έχουν δικαίωμα ανάγνωσης. Αυτό συμβαίνει για τον εξής λόγο: Επειδή το widget της εφαρμογής χρησιμοποιεί τον κόμβο posts για να εξάγει στατιστικά, θέλουμε η διαδικασία αυτή να πραγματοποιείται άμεσα χωρίς ο χρήστης να χρειάζεται πρώτα να συνδεθεί, ώστε το Widget να είναι εύχηστο. Όσον αφορά στην εγγραφή, ορίζουμε ότι ο χρήστης πρέπει να είναι αυθεντικοποιημένος (ώστε να ληφθεί το UID του) και κάθε post να περιλαμβάνει τα παιδιά user, policeArrived, timestamp, latitude, longitude ώστε να είναι πλήρες. Για το παιδί user ορίζουμε ότι πρέπει να έχει ως τιμή το UID του αυθεντικοποιημένου χρήστη (ώστε ένας κακόβουλος χρήστης να μην μπορεί να παραστήσει κάποιον άλλον) και για τα υπόλοιπα παιδιά ορίζουμε τους τύπους των τιμών.

Για τον κόμβο users ορίζουμε ότι μπορεί να διαβαστεί από αυθεντικοποιημένους μόνο χρήστες και κάθε κόμβος-παιδί να γραφεί μόνο από τον χρήστη στον οποίο αντιστοιχεί.

Για τον κόμβο timeline ορίζουμε ότι κάθε κόμβος-παιδί (timeline κάποιου χρήστη) μπορεί να διαβαστεί μόνο από το συγκεκριμένο χρήστη καθώς και ότι στον κόμβο αυτό έχουν δικαιώματα εγγραφής μόνο οι φίλοι του. Για κάθε post μέσα στο timeline κάθε χρήστη ισχύουν οι ίδιοι κανόνες με αυτούς που ίσχυαν στον κόμβο posts. Επίσης, κάθε posts μπορεί επίσης να διαβαστεί από αυτόν που το δημιούργησε.

Στον κόμβο users-posts, στον κόμβο-παιδί κάθε χρήστη έχει δικαιώματα εγγραφής και ανάγνωσης μόνο ο συγκεκριμένος χρήστης. Για κάθε post κάθε χρήστη ισχύουν οι ίδιοι κανόνες με αυτούς που ίσχυαν στον κόμβο posts.

Στον κόμβο users-follow, δηλώνουμε ότι ο κάθε follower έχει πρόσβαση μόνο στον δικό του κόμβο-παιδί. Αυτό πρακτικά σημαίνει ότι κάποιος χρήστης δεν μπορεί να διαβάσει την τοποθεσία κάποιου άλλου χρήστη χωρίς να έχει δεχθεί πρόσκληση από αυτόν. Επίσης, στον κόμβο κάθε walker έχει δικαίωμα εγγραφής μόνο ο συγκεκριμένος walker (ώστε κάποιος κακόβουλος χρήστης να μην μπορεί να μεταβάλλει την τοποθεσία κάποιου άλλου) και επίσης κάθε walker πρέπει να είναι φίλος του αντίστοιχου follower.

Τέλος, στον κόμβο users-friends ορίζουμε αρχικά ότι ο κάθε χρήστης έχει δικαίωμα ανάγνωσης μόνο στα αιτήματα στα οποία εμπλέκεται εκείνος. Με τον τελευταίο μεγάλο κανόνα ορίζουμε τα εξής: Ο εσωτερικός χρήστης έχει δικαίωμα να γράψει μόνο στον δικό του υποκόμβο (και αν δεν υπάρχει ήδη εγγραφή μπορεί να οριστεί μόνο ως “pending”) ο εξωτερικός χρήστης μπορεί να ορίσει κάποιον εσωτερικό ως requested και ο εσωτερικός χρήστης μπορεί να μετατρέψει το “pending” σε “added” (αποδοχή αιτήματος). Τέλος, δίνεται η δυνατότητα σε κάθε χρήστη να διαγράψει οποιοδήποτε φίλο άσχετα με την προηγούμενη κατάσταση του.

Σε περίπτωση που ένας προγραμματιστής-χρήστης επιδιώξει μέσω κώδικα να παραβιάσει τους παραπάνω κανόνες η βάση δεν θα ανταποκριθεί. Για παράδειγμα, ένας πολύ απλός τρόπος

ελέγχου του κανόνα ".read": "auth != null" στον κόμβο users είναι να επιδιώξουμε να αποκτήσουμε πρόσβαση σε αυτόν μέσω της εντολής curl των Linux, χωρίς να έχουμε αυθεντικοποιηθεί. Η πρόσβαση προφανώς δεν είναι εφικτή. Μία αντίστοιχη προσπάθεια πρόσβασης στον κόμβο posts όπου ισχύει ο κανόνας ".read": true, είναι επιτυχής. Τα αποτελέσματα φαίνονται παρακάτω:

```
➜  ~ curl 'https://myapp-4c309.firebaseio.com/users.json'
{
  "error" : "Permission denied"
}
➜  ~ curl 'https://myapp-4c309.firebaseio.com/posts.json'
{"-KltV7ukoBEDle5R9dgM":{"description":"wkEIJF LAWHFAWKJFNWHFK BJBAJ HBAJKFJ AHjaebf arbf bfjhabrfjh","la
titude":37.974,"longitude":23.783,"policeArrived":false,"timestamp":1496687087,"user":"wpySWGG9wuZdEt11bS
nxweM6cvG2"},"-KltVItloWU_TuuwJ62b":{"description":"","latitude":37.975,"longitude":23.782,"policeArrived
":true,"timestamp":1496687132,"user":"wpySWGG9wuZdEt11bSnxweM6cvG2"},"-KltVVtlSUOb2qcN07GF":{"description
":"works","latitude":37.975,"longitude":23.782,"policeArrived":true,"timestamp":1496687185,"user":"Gs6d90
ctBXgP03x8w8P4NzhH9rE2"},"-KltVl-lzemyUSw3tobu":{"description":"ntua","latitude":37.975,"longitude":23.78
7,"policeArrived":false,"timestamp":1496687251,"user":"D4jIvV1TtGYcTDOuARLGNAdSG4u2"},"-KltVstKWPS9_sBQYX
➜  ~
```

Υλοποίηση

5.1 Υλοποίηση της εφαρμογής

Για την υλοποιήση της εφαρμογής WatchMe σε γλώσσα Swift χρησιμοποιήθηκαν πολλά SDKs, καθένα από τα οποία εξυπηρέτησε κι έναν καθοριστικό σκοπό. Για της υπηρεσίες τοποθεσίας χρησιμοποιήθηκαν τα CoreLocation και MapKit, για την λήψη εικόνων από URLs το AlamorifeImage, για την επικοινωνία με το Keychain το Security, για τη δημιουργία τοπικών ειδοποιήσεων (local notifications) το UserNotifications και για την επικοινωνία με το Firebase χρησιμοποιήθηκε το Firebase SDK. Στη συνέχεια θα παρουσιαστούν κάποια τμήματα κώδικα που ήταν κρίσιμα για τις βασικές λειτουργίες της εφαρμογής.

Εγγραφή στην εφαρμογή

Μέθοδοι	create_account, create_record
Κλάση	SignUpViewController

```
func create_account() {
        FIRAuth.auth()?.createUser(withEmail: emailTextField.text!, password:
passwordTextField.text!) { (user, error) in

            if error == nil {

                let user = FIRAuth.auth()?.currentUser
                if let user = user {
                    let changeRequest = user.profileChangeRequest()

                    changeRequest.displayName = self.usernameTextField.text
                    changeRequest.photoURL = URL(string:
"https://firebasestorage.googleapis.com/v0/b/myapp-4c309.appspot.com/o/profile_images%2Ficon-
user-default-300x300.png?alt=media&token=71734fdd-98da-4f9f-a09c-93964cef461c")
                    changeRequest.commitChanges { error in
                        if error != nil {
                            // An error happened.
                        } else {
                            // Profile updated.
                        }
                    }
                }

                FIRAuth.auth()?.currentUser!.sendEmailVerification(completion: { (error) in
                })


                let alert = UIAlertController(title: "Account Created", message: "Please verify
your email by confirming the sent link.", preferredStyle: UIAlertControllerStyle.alert)
                alert.addAction(UIAlertAction(title: "Ok", style: UIAlertActionStyle.default,
handler: nil))
                self.present(alert, animated: true, completion: nil)

                // add user into database

                if let thisuser = user{
                    self.create_record(user: thisuser)
                }

            } else {
                let alertController = UIAlertController(title: "Error", message:
error?.localizedDescription, preferredStyle: .alert)

                let defaultAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
                alertController.addAction(defaultAction)

                self.present(alertController, animated: true, completion: nil)
            }
        }

    }
```

```
    func create_record(user: FIRUser) {
        let ref = FIRDatabase.database().reference()
        let userReference = ref.child("users").child((user.uid))


        userReference.updateChildValues(["name": self.usernameTextField.text!, "email":
self.emailTextField.text!, "profileImageUrl":
"https://firebasestorage.googleapis.com/v0/b/myapp-4c309.appspot.com/o/profile_images%2Ficon-
user-default-300x300.png?alt=media&token=71734fdd-98da-4f9f-a09c-93964cef461c"],
withCompletionBlock: { (err, ref) in

            if let err = err {
                print(err)
                return
            }

            self.dismiss(animated: true, completion: nil)
        })
        reinitialize_form()

    }
```

Η συνάρτηση create_account() καλείται αφού ο χρήστης έχει εισάγει όλα τα απαραίτητα στοιχεία. Δημιουργεί εγγραφή στο Authentication table του Firebase μέσω της συνάρτησης createUser του αντικειμένου auth, αποθηκεύει το URL της default φωτογραφίας προφίλ και στέλνει στο χρήστη email ενεργοποίησης του λογαριασμού. Σε περίπτωση που συμβεί κάποιο σφάλμα κατά την εγγραφή, το εμφανίζει. Η συνάρτηση create_record δημιουργεί μια αναφορά στη βάση και μέσω αυτής μια εγγραφή με τα εν λόγω στοιχεία στον κόμβο users.

Σύνδεση στην εφαρμογή

Μέθοδοι	check_keychain, check_server, DoLogin
Κλάση	LoginViewController

```
let sb = Strongbox()
func check_keychain(){
        let user = sb.unarchive(objectForKey: "wmUsername") as? String
        let pass = sb.unarchive(objectForKey: "wmPassword") as? String
        if let user = user, let pass = pass {
            print("found!", user, pass)
            check_server(username: user, password: pass) { (result) -> Void in
                //Go to the HomeViewController if the login is sucessful
                if (result == 0) {
                    let vc = self.storyboard?.instantiateViewController(withIdentifier: "Home")
                    self.present(vc!, animated: true, completion: nil)

                }
            }
        }
        else {
            print("not found…")
        }

}
```

```
// 0: logged in, 1: wrong email/pass, 2: not verified email, 3: other error
func check_server(username:String, password:String, handler:@escaping(_ result:Int)->Void){
        FIRAuth.auth()?.signIn(withEmail: username, password: password) { (user, error) in

            if error == nil {

                if let user = FIRAuth.auth()?.currentUser {
                    if !user.isEmailVerified{
                        let alertVC = UIAlertController(title: "Error", message: "Sorry. Your
email address has not yet been verified. Do you want us to send another verification email to
\(self.email_input.text!)?", preferredStyle: .alert)
                        let alertActionOkay = UIAlertAction(title: "Okay", style: .default) {
                            (_) in
                            user.sendEmailVerification(completion: nil)
                        }
                        let alertActionCancel = UIAlertAction(title: "Cancel", style: .default,
handler: nil)
                        alertVC.addAction(alertActionOkay)
                        alertVC.addAction(alertActionCancel)
```

```
                        self.present(alertVC, animated: true, completion: nil)
                        handler(2)
                    }
                    else {
                        print("You have successfully logged in")
                        handler(0)
                    }
                }
                else {

            //Tells the user that there is an error and then gets firebase to tell them the error
                    let alertController = UIAlertController(title: "Error", message:
error?.localizedDescription, preferredStyle: .alert)

                    let defaultAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
                    alertController.addAction(defaultAction)

                    self.present(alertController, animated: true, completion: nil)
                    handler(3)
                }
            }
            else {
                // no user with this info
                handler(1)
            }
        }
}
```

```
@IBAction func DoLogin(_ sender: AnyObject) {
        if self.email_input.text == "" || self.password_input.text == "" {

            //Alert to tell the user that there was an error because they didn't fill anything
            let alertController = UIAlertController(title: "Error", message: "Please enter an
email and password.", preferredStyle: .alert)

            let defaultAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
            alertController.addAction(defaultAction)

            self.present(alertController, animated: true, completion: nil)

        } else {
            check_server(username: self.email_input.text!, password: self.password_input.text!)
{ (result) -> Void in
                if result == 0 {
                    let saveAlert = UIAlertController(title: "Save in Keychain", message: "Do
you want to save your email and password in the keychain for easier login?", preferredStyle:
.alert)
                    let alertActionYes = UIAlertAction(title: "Yes", style: .default) {(_) in
                        if self.sb.archive(self.email_input.text!, key: "wmUsername"){
                            print("username store done")
                        }
                        else {
                            print("username store error")
                        }
                        if self.sb.archive(self.password_input.text, key: "wmPassword"){
                            print("password store done")
                        }
                        else {
                            print("password store error")
                        }
                        let vc=self.storyboard?.instantiateViewController(withIdentifier:"Home")
                        self.present(vc!, animated: true, completion: nil)
                    }
                    let alertActionNo = UIAlertAction(title: "No", style: .default) {(_) in
                        let vc=self.storyboard?.instantiateViewController(withIdentifier:"Home")
                        self.present(vc!, animated: true, completion: nil)
                    }
                    saveAlert.addAction(alertActionYes)
                    saveAlert.addAction(alertActionNo)
                    self.present(saveAlert, animated: true, completion: nil)
                }
                else if result == 1 {
                    let alertVC = UIAlertController(title: "Error", message: "No user with this
email and password", preferredStyle: .alert)
                    let alertActionOk = UIAlertAction(title: "OK", style: .default, handler:nil)
                    alertVC.addAction(alertActionOk)
                    self.present(alertVC, animated: true, completion: nil)
                }
            }
        }
}
```

Εδώ ακολουθείται ακριβώς η διαδικασία που περιγράφηκε στο κεφάλαιο 2.1.4. Η συνάρτηση check_keychain, μέσω της συνάρτησης unarchive του Strongbox ελέγχει αν η εφαρμογή έχει αποθηκεύσει κάποιο ζεύγος email-password στο keychain. Αν υπάρχει τέτοιο ζεύγος, στέλνεται για έλεγχο στον server. Η συνάρτηση check_server πραγματοποιεί μια ασύγχρονη αίτηση στο server για να γίνει αυτός ο έλεγχος, μέσω της συνάρτησης signIn του αντικειμένου auth. Το αποτέλεσμα λαμβάνεται μέσω ενός handler. Σε περίπτωση που ο έλεγχος αποτύχει ή δεν υπάρχει εξ' αρχής αποθηκευμένο ζεύγος στο keychain, ο χρήστης καλείται να εισάγει τα στοιχεία του και να πατήσει το κουμπί Login. Τότε καλείται η συνάρτηση DoLogin που πραγματοποιεί τον έλεγχο γι αυτά τα στοιχεία και, αν είναι σωστά, δίνει τη δυνατότητα στο χρήστη να τα εισάγει στο keychain μέσω της συνάρτησης archive του Strongbox.

Προβολή δημοσιεύσεων φίλων

Μέθοδοι	initializeObservers, tableView(_:cellForRowAt:)
Κλάση	NewsFeedTableViewController

```
func initializeObservers(){

        self.user = (FIRAuth.auth()?.currentUser)!
        myTimelineRef = FIRDatabase.database().reference().child("timeline").child(user.uid)
        usersReference = FIRDatabase.database().reference().child("users")

        myTimelineRef.observe(.childAdded, with: { (snapshot) in
            self.timeline.insert(snapshot, at: 0)
            self.tableView.reloadData()
        })
        myTimelineRef.observe(.childChanged, with: { (snapshot) in
            let index = self.timeline.index(where: { (item) -> Bool in
                item.key == snapshot.key // test if this is the item you're looking for
            })
            self.timeline.remove(at: index!)
            self.timeline.insert(snapshot, at: index!)
            self.tableView.reloadData()
        })
        myTimelineRef.observe(.childRemoved, with: { (snapshot) in
            self.timeline = self.timeline.filter{$0.key != snapshot.key}
            self.tableView.reloadData()
        })
}
```

```
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "newsfeedCell", for: indexPath)
as! NewsFeedCell
        cell.selectionStyle = .none
        cell.goButton.tag = indexPath.section
        let post = timeline[indexPath.section].value as! [String:Any]
        let friend_id = post["user"]

        // get name and image
        usersReference.child(friend_id as! String).observeSingleEvent(of: .value, with: {
(snaphot) in

            cell.nameLabel.text = (snaphot.childSnapshot(forPath: "name").value as! String)
            let urlstr = (snaphot.childSnapshot(forPath: "profileImageUrl").value as! String)
            let url = URL(string: urlstr)

            cell.profilePicture.af_setImage(withURL: url!, placeholderImage: UIImage(named:
"icon-user-default.png"), completion: { (downloadedImage) in
                //print("done")
            })
        })

        // get date and time
        let date = Date(timeIntervalSince1970: post["timestamp"] as! TimeInterval)
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "MMM dd, yyyy HH:mm" //Specify your format that you want
        cell.timeLabel.text = dateFormatter.string(from: date)

        // get description
        cell.descriptionLabel.text = post["description"] as? String
        cell.descriptionLabel.sizeToFit()
```

```
        // get police status
        if post["policeArrived"] as! Bool == true {
            cell.policeResult.text = "Handled by police"
            cell.policeResult.textColor = UIColor.green
        }
        else{
            cell.policeResult.text = "Not handled by police"
            cell.policeResult.textColor = UIColor.red
        }

        // get location
        let lat = post["latitude"]
        let lon = post["longitude"]
        let pointAnnotation = MKPointAnnotation()
        pointAnnotation.coordinate = CLLocationCoordinate2D(latitude: lat as! CLLocationDegrees,
longitude: lon as! CLLocationDegrees)
        let pinAnnotationView =  MKPinAnnotationView(annotation: pointAnnotation,
reuseIdentifier: nil)
        cell.mapView.centerCoordinate = pointAnnotation.coordinate
        cell.mapView.removeAnnotations(cell.mapView.annotations)
        cell.mapView.addAnnotation(pinAnnotationView.annotation!)
        cell.mapView.showAnnotations([pointAnnotation], animated: true)

        // get location text
        let geoCoder = CLGeocoder()
        let location = CLLocation(latitude: lat as! CLLocationDegrees, longitude: lon as!
CLLocationDegrees)

        geoCoder.reverseGeocodeLocation(location) { (placemarks, error) in
            // Place details
            var placeMark: CLPlacemark!
            placeMark = placemarks?[0]
            let region = placeMark.subLocality ?? ""
            let country = placeMark.country ?? ""
            let adm = placeMark.administrativeArea
            let subadm = placeMark.subAdministrativeArea
            var city:String
            if subadm != nil {
                city = subadm ?? ""
            }
            else if adm != nil{
                city = adm ?? ""
            }
            else {
                city = ""
            }

            cell.placeLabel.text = region + " " + city + " " + country
        }
        return cell
    }
```

Στη συνάρτηση initializeObservers δημιουργούμε τρεις observers στο timeline του συνδεδεμένου χρήστη. Ο κώδικας του πρώτου (.childAdded) θα καλείται κάθε φορά που ένας κόμβος-παιδί προστίθεται στο συγκεκριμένο timeline, ο δεύτερος(.childChanged) όταν ένας κόμβος-παιδί μεταβάλλεται και ο τρίτος (.childRemoved) όταν ένας κόμβος-παιδί διαγράφεται. Σε καθεμία από αυτές τις περιπτώσεις ανανεώνεται ο πίνακας της οθόνης, δηλαδή καλείται η συνάρτηση tableView(_:cellForRowAt:) για κάθε κελί. Σε αυτήν καλούμε έναν observer για το κάθε post ώστε να ληφθούν οι απαραίτητες πληροφορίες. Ο observer αυτός, για λόγους αποδοτικότητας, καλείται μία μόνο φορά και μετά κλείνει αυτόματα (observeSingleEvent).

Προβολή και επεξεργασία δημοσιεύσεων συνδεδεμένου χρήστη

Μέθοδοι	editPostPressed, markPressed
Κλάση	PostsTableViewController

```
@IBAction func editPostPressed(_ sender: Any) {

        let buttonSection = (sender as AnyObject).tag
        let post_id = posts[buttonSection!].key
        let alertController = UIAlertController(title: "Add a description for the incident",
message: "", preferredStyle: UIAlertControllerStyle.alert)
```

```
        let saveAction = UIAlertAction(title: "Save", style: UIAlertActionStyle.default,
handler: {
            alert -> Void in

            let descriptiontext = alertController.textFields![0].text
            let myGroup = DispatchGroup()

            // create fanout object
            var fanoutObject = ["/posts/\(post_id)/description": descriptiontext]
            fanoutObject["/users-posts/\(self.user.uid)/\(post_id)/description"] =
descriptiontext
            for friend_id in self.myFriendsList{
                // job enters the group
                myGroup.enter()

                // only if the post exists, add the new value
FIRDatabase.database().reference().child("timeline").child(friend_id).child(post_id).observeSing
leEvent(of: FIRDataEventType.value, with: { (snapshot) in
                    if snapshot.exists(){
                        fanoutObject["/timeline/\(friend_id)/\(post_id)/description"] =
descriptiontext
                    }
                    myGroup.leave()
                })
            }
            // when all friends checked, fanout the post atomically
            myGroup.notify(queue: .main) {
               FIRDatabase.database().reference().updateChildValues(fanoutObject as!
[String:String])
            }
        })
        let cancelAction = UIAlertAction(title: "Cancel", style: UIAlertActionStyle.default,
handler: {
            (action : UIAlertAction!) -> Void in
        })
        alertController.addTextField { (textField : UITextField!) -> Void in

self.postsRef.child(self.posts[buttonSection!].key).child("description").observeSingleEvent(of:
FIRDataEventType.value, with: { (snapshot) in
                textField.text = snapshot.value as? String
            })
        }
        alertController.addAction(saveAction)
        alertController.addAction(cancelAction)

        self.present(alertController, animated: true, completion: nil)
    }
```

```
@IBAction func markPressed(_ sender: Any) {

        let buttonSection = (sender as AnyObject).tag

        let post_id = posts[buttonSection!].key

        // let updatedPost = ["policeArrived": true]
        print(myFriendsList)

        let myGroup = DispatchGroup()

        // create fanout object
        var fanoutObject = ["/posts/\(post_id)/policeArrived": true]
        fanoutObject["/users-posts/\(self.user.uid)/\(post_id)/policeArrived"] = true

        for friend_id in self.myFriendsList{

            // job enters the group
            myGroup.enter()
            // only if the post exists, add the new value
FIRDatabase.database().reference().child("timeline").child(friend_id).child(post_id).observeSing
leEvent(of: FIRDataEventType.value, with: { (snapshot) in
                if snapshot.exists(){

                    fanoutObject["/timeline/\(friend_id)/\(post_id)/policeArrived"] = true
                }
                myGroup.leave()
            })
        }
        // when all friends checked, fanout the post atomically
        myGroup.notify(queue: .main) {
            FIRDatabase.database().reference().updateChildValues(fanoutObject)
```

```
        }
    }
```

Η κλάση αυτή είναι παρόμοια με την NewsFeedTableViewController, με τη διαφορά ότι παρέχει τις συναρτήσεις editPostPressed και markPressed για την επεξεργασία κάθε post. Και οι δύο αυτές συναρτήσεις υλοποιούν την τεχνική fan-out, γράφοντας το ανανεωμένο post στις εξής τοποθεσίες: στον κόμβο posts, στον κόμβο users-posts και στο timeline κάθε φίλου του συνδεδεμένου χρήστη. Παρατηρούμε ότι σε φίλους που προστέθηκαν μετά από την αρχική δημιουργία του post, αυτό δεν εμφανίζεται. Χρησιμοποιείται η τεχνική των Dispatch Groups ώστε πριν γίνει το fan-out να έχουν ληφθεί και προστεθεί στο fanout Object όλοι οι φίλοι του συνδεδεμένου χρήστη από τον αντίστοιχο observer.

Εύρεση και προσθήκη φίλου

Μέθοδοι	initialize_search, searchBar(_:textDidChange:), tableView(_:didSelectRowAt:)
Κλάση	AddFriendsTableViewController

```
    func initialize_search(){

        searchController = UISearchController(searchResultsController: nil)
        searchController.hidesNavigationBarDuringPresentation = false
        self.searchController.searchBar.delegate = self
        present(searchController, animated: true, completion: nil)
        searchController.dimsBackgroundDuringPresentation = false
        searchController.searchBar.setValue("Done", forKey:"_cancelButtonText")
    }
```

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {

        let usersRef = FIRDatabase.database().reference().child("users")
        if (searchText != ""){
            usersRef.queryOrdered(byChild: "name").queryStarting(atValue:
searchText).queryEnding(atValue: searchText+"\u{f8ff}").observe(.value, with: { (snapshot) in
                self.filtered = snapshot.children.allObjects as! [FIRDataSnapshot]
                self.tableView.reloadData()
            }, withCancel: nil)
        }
        else {
            self.filtered = []
            self.tableView.reloadData()
        }
}
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

        let userid = filtered[indexPath.row].key

        if self.user.uid == userid{
            return
        }
        let label = tableView.cellForRow(at: indexPath)?.accessoryView as! UILabel
        if label.text == "" {
            // Implement friend request

            // set the user as requested in my friends
            self.myFriendsReference.updateChildValues([userid : "requested"])

            // set me as pending in his friends
            FIRDatabase.database().reference().child("users-
friends").child(userid).updateChildValues([user.uid : "pending"])
            tableView.reloadData()
        }
}
```

Στη συνάρτηση initialize_search αρχικοποιείται ο UISearchController, ένα αντικείμενο που παρέχει μια μπάρα αναζήτησης και θα χρησιμοποιηθεί στην εύρεση φίλων. Η συνάρτηση searchBar(_:textDidChange:) καλείται όποτε αλλάζει το κείμενο στην μπάρα αναζήτησης κι εκτελεί ένα query στο Firebase το οποίο εμφανίζει όλους τους χρήστες των οποίων το όνομα αρχίζει από αυτή τη συμβολοσειρά. Η συνάρτηση searchBar(_:textDidChange:) καλείται όταν ο χρήστης πατήσει στο κελί κάποιου από αυτούς του εμφανιζόμενους φίλους ο οποίος δεν είναι ούτε φίλος του, ούτε του έχει ήδη στείλει αίτημα φιλίας ούτε έχει λάβει από αυτόν αίτημα φιλίας. Η συνάρτηση υλοποιεί

τις απαραίτητες ενέργειας και εγγραφές στη βάση που συνδέονται με ένα αίτημα φιλίας, όπως αυτές περιγράφονται στην παράγραφο 4.2.2.2.

Παρακολούθηση φίλου στο χάρτη

Μέθοδοι	initialize_observers, updateAnnotation
Κλάση	FollowMapViewController

```
func initialize_observers(){
        user = FIRAuth.auth()?.currentUser
        friendRef =
FIRDatabase.database().reference().child("users").child(friendID).child("name")

        friendRef.observeSingleEvent(of: .value, with: { (snapshot) in
            self.title = snapshot.value as? String
        })

        wantedRef = FIRDatabase.database().reference().child("users-
follow").child(user.uid).child(friendID)
        wantedRef.observeSingleEvent(of: .value, with: { (snapshot) in
            if snapshot.hasChild("destination") {
                let destAnnotation = MKPointAnnotation()
                destAnnotation.coordinate.latitude = snapshot.childSnapshot(forPath:
"destination").childSnapshot(forPath: "latitude").value as! CLLocationDegrees
                destAnnotation.coordinate.longitude = snapshot.childSnapshot(forPath:
"destination").childSnapshot(forPath: "longitude").value as! CLLocationDegrees
                destAnnotation.title = "Destination"
                self.mapView.addAnnotation(destAnnotation)

            }
        })
        currentRef = FIRDatabase.database().reference().child("users-
follow").child(user.uid).child(friendID).child("current")
        destRef = FIRDatabase.database().reference().child("users-
follow").child(user.uid).child(friendID).child("destination")

        currentRef.observe(.value, with: { (snapshot) in
            if snapshot.hasChild("latitude") && snapshot.hasChild("longitude"){
              self.lat = snapshot.childSnapshot(forPath:"latitude").value as! CLLocationDegrees
              self.lon = snapshot.childSnapshot(forPath:"longitude").value as! CLLocationDegrees
              self.updateAnnotation()
            }
            else{
                _ = self.navigationController?.popViewController(animated: true)

            }
        })
    }
```

```
func updateAnnotation() {
        mapView.removeAnnotation(pointAnnotation)
        pointAnnotation.coordinate = CLLocationCoordinate2D(latitude: lat, longitude: lon)
        self.mapView.addAnnotation(pointAnnotation)
        mapView.showAnnotations(mapView.annotations, animated: true)
    }
```

Στη συνάρτηση initialize_observers αυτής της κλάσης, ελέγχεται αρχικά αν υπάρχει καταγεγραμμένος προορισμός στη βάση και αν υπάρχει εμφανίζεται με μια πινέζα στο χάρτη της οθόνης. Έπειτα, αρχικοποιείται ένας observer στον κόμβο current (όπως εξηγήθηκε στην παράγραφο 4.2.2.6) ο οποίος λαμβάνει συνεχώς ενημερώσεις για την τοποθεσία και τις αποθηκεύει στις μεταβλητές lat και lon. Με κάθε ανανέωση καλείται η συνάρτηση updateAnnotation η οποία ενημερώνει στο χάρτη το pin που αντιστοιχεί στην τοποθεσία του παρακολουθούμενου χρήστη.

Δημιουργία σήματος κινδύνου

Μέθοδοι	inDanger
Κλάση	ActionViewController

```
@IBAction func inDanger(_ sender: UIButton) {
        var currentLocation = CLLocation()

        if( CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedWhenInUse
|| CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedAlways){

            currentLocation = locManager.location!
```

```
        }
        timelineRef = ref.child("timeline")
        postsRef = ref.child("posts")
        let mypostRef = postsRef.childByAutoId()

        let uid = user.uid
        let timestamp = Int(Date().timeIntervalSince1970)
        let lat = currentLocation.coordinate.latitude
        let lon = currentLocation.coordinate.longitude

        let mypost = ["user": uid, "timestamp": timestamp, "latitude": lat, "longitude": lon,
"description": "", "policeArrived": false] as [String : Any]

        // create fanout object
        var fanoutObject = ["/posts/\(mypostRef.key)": mypost]
        for friend_id in myFriendsList{
            fanoutObject["/timeline/\(friend_id)/\(mypostRef.key)"] = mypost
        }
        fanoutObject["/users-posts/\(self.user.uid)/\(mypostRef.key)"] = mypost

        // fanout the post atomically
        ref.updateChildValues(fanoutObject)
}
```

Η συνάρτηση καλείται μόλις πατηθεί το κουμπί DANGER και λαμβάνει όλες εκείνες τις πληροφορίες που πρέπει να περιέχει ένα post (τοποθεσία, χρόνο, χρήστη, περιγραφή και ένδειξη αστυνομίας αρχικοποιημένη στο false). Στη συνέχεια δημιουργείται ένα αντικείμενο fan-out που γράφει το post στα απαραίτητα μονοπάτια της βάσης μέσω της συνάρτησης updateChildValues του Firebase.

Προβολή επικινδυνότητας περιοχής

Μέθοδοι	locationManager(_:didUpdateLocations:), updateNearestPosts, calculateLastYear, updateAnnotations
Κλάση	AreaViewController

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

        postsRef.observeSingleEvent(of: .value, with: { (snapshot) in

            self.updateNearestPosts(snapshot: snapshot)
            self.calculateLastYear()
            self.updateAnnotations()
        })
}
```

```
func updateNearestPosts(snapshot: FIRDataSnapshot) {
        nearestEvents.removeAll()
        for rest in snapshot.children.allObjects {
            let long = (rest as! FIRDataSnapshot).childSnapshot(forPath: "longitude").value
            let lat = (rest as! FIRDataSnapshot).childSnapshot(forPath: "latitude").value
            let curloc = self.locationManager.location
            let postloc = CLLocation(latitude: lat as! CLLocationDegrees, longitude: long as!
CLLocationDegrees)
            let distanceInMeters = curloc?.distance(from: postloc) // result is in meters
            if (Int(distanceInMeters!) <= 1000){
                self.nearestEvents.append(rest as! FIRDataSnapshot)
            }
        }
}
```

```
func calculateLastYear() {

        var events_thisyear = 0
        var events_previousyear = 0
        var indexes_to_remove:[Int] = []

        for event in nearestEvents {
            let timestamp = event.childSnapshot(forPath: "timestamp").value
            let date = Date(timeIntervalSince1970: timestamp as! TimeInterval)
            let years=Calendar.current.dateComponents([.year], from: date, to: Date()).year ?? 0
            if years < 1{
                events_thisyear += 1
            }
            else if years < 2{
                events_previousyear += 1
```

```
                indexes_to_remove.append(nearestEvents.index(of: event)!)
            }
            else{
                indexes_to_remove.append(nearestEvents.index(of: event)!)
            }
        }
        for index in indexes_to_remove{
            nearestEvents.remove(at: index)
        }
        self.numberLabel.text = "\(events_thisyear) event(s) during last year within one mile"
        if events_thisyear < 5 {
            self.resultLabel.text = "Safe Area"
            self.resultLabel.textColor = UIColor.green
        }
        else if events_thisyear < 10 {
            self.resultLabel.text = "Medium Safety Area"
            self.resultLabel.textColor = UIColor.orange
        }
        else {
            self.resultLabel.text = "Dangerous Area"
            self.resultLabel.textColor = UIColor.red
        }
        if (events_thisyear >= events_previousyear){
            self.comparisonLabel.text = "\(events_thisyear - events_previousyear) event(s) more
than previous year"
        }
        else {
            self.comparisonLabel.text = "\(events_previousyear - events_thisyear) event(s) less
than previous year"
        }
    }
```

```
func updateAnnotations() {

        myAreaMap.removeAnnotations(myAreaMap.annotations)
        let myGroup = DispatchGroup()
        for event in nearestEvents{
            myGroup.enter()
            let lat = event.childSnapshot(forPath: "latitude").value
            let lon = event.childSnapshot(forPath: "longitude").value
            let pointAnnotation = myAnnotation()
            pointAnnotation.handled = (event.childSnapshot(forPath: "policeArrived").value as!
Bool)
            pointAnnotation.coordinate = CLLocationCoordinate2D(latitude: lat as!
CLLocationDegrees, longitude: lon as! CLLocationDegrees)
            let date = Date(timeIntervalSince1970: event.childSnapshot(forPath:
"timestamp").value as! TimeInterval)
            let dateFormatter = DateFormatter()
            dateFormatter.dateFormat = "dd-MMM-yyyy" //Specify your format that you want
            pointAnnotation.subtitle = dateFormatter.string(from: date)
            let user = event.childSnapshot(forPath: "user").value
            FIRDatabase.database().reference().child("users").child(user as!
String).observeSingleEvent(of: .value, with: { (snapshot) in
              pointAnnotation.title = (snapshot.childSnapshot(forPath: "name").value as! String)
              pointAnnotation.imageUrl = snapshot.childSnapshot(forPath:
"profileImageUrl").value as! String
              self.myAreaMap.addAnnotation(pointAnnotation)
              myGroup.leave()
            })
        }
       // fit zoom to show all annotations
       myGroup.notify(queue: .main) {
            self.myAreaMap.showAnnotations(self.myAreaMap.annotations, animated: true)
       }
}
```

Η συνάρτηση locationManager(_:didUpdateLocations:) καλείται κάθε φορά που ανανεώνεται η τοποθεσία του συνδεδεμένου χρήστη και καλεί τις άλλες 3 συναρτήσεις. Η updateNearestPosts αποθηκεύει σε ένα πίνακα όσα posts υπάρχουν στη βάση τη δεδομένη στιγμή, τα οποία έχουν συμβεί εντός ενός χιλιομέτρου από την τρέχουσα τοποθεσία του χρήστη. Η calculateLastYear φιλτράρει περαιτέρω αυτά τα posts, υπολογίζοντας πόσα έχουν συμβεί τον τελευταίο χρόνο και πόσα τον προηγούμενο (μέσω του πεδίου timestamp) και παρουσιάζει τα αποτελέσματα μαρκάροντας ταυτόχρονα την περιοχή ως πολύ, μέτρια ή καθόλου ασφαλή. Τέλος, η συνάρτηση updateAnnotations παρουσιάζει τις τοποθεσίες των φιλτραρισμένων δημοσιεύσεων στο χάρτη της οθόνης, χρησιμοποιώντας τα πεδία latitude και longitude.

Εκκίνηση λειτουργίας follow

Μέθοδοι	goTapped, setupNotificationSettings, enable_following
Κλάση	StartFollowingViewController

```
@IBAction func goTapped(_ sender: Any) {

        locManager.startUpdatingLocation()
        goButton.isEnabled = false
        navigationItem.hidesBackButton = true
        self.cancelButton.isEnabled = true
        setupNotificationSettings()
        enable_following()
}
```

```
func setupNotificationSettings() {

        // Specify the notification actions. (arrived, danger, reschedule)
        let arrivedAction = UNNotificationAction(identifier: "arrived", title: "I have
arrived!", options: [])

        let dangerAction = UNNotificationAction(identifier: "danger", title: "I am in danger",
options: [.destructive])

        let rescheduleAction = UNNotificationAction(identifier: "reschedule", title: "Remind me
in 10 minutes", options: [])

        // Specify the category related to the above actions.
        let checkInCategory = UNNotificationCategory(identifier: "checkInCategory", actions:
[arrivedAction, dangerAction, rescheduleAction], intentIdentifiers: [], options: [])
        UNUserNotificationCenter.current().setNotificationCategories([checkInCategory])

        let seconds = self.minutes * 60
        let content = UNMutableNotificationContent()
        content.title = "Check in reminder"
        content.body = "Hey, have you arrived safely?"
        content.categoryIdentifier = "checkInCategory"

        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: TimeInterval(seconds),
repeats: false)

        let requestIdentifier = "checkInNotif"

        let request = UNNotificationRequest(identifier: requestIdentifier, content: content,
trigger: trigger)
        UNUserNotificationCenter.current().add(request, withCompletionHandler: { (error) in
            print(error ?? " ")
        })
}
```

```
func enable_following() {
        if( CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedWhenInUse
||
            CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedAlways){

            currentLocation = locManager.location!
        }
        if (destlat != nil && destlon != nil){
            followDestRef = ref.child("users-
follow").child(followerID).child(user.uid).child("destination")
            followDestRef.updateChildValues(["latitude": destlat!, "longitude": destlon!])
        }
        let lat = currentLocation.coordinate.latitude
        let lon = currentLocation.coordinate.longitude

        followCurRef = ref.child("users-
follow").child(followerID).child(user.uid).child("current")
        followCurRef.updateChildValues(["latitude": lat, "longitude": lon])
}
```

Η συνάρτηση goTapped καλείται μόλις ο χρήστης πατήσει το κουμπί εκκίνησης της λειτουργίας follow, αφού έχει επιλέξει κάποιον φίλο να τον ακολουθεί καθώς και προορισμό/χρόνο. Αρχικά καλεί τη συνάρτηση setupNotificationSettings μέσω της οποίας προγραμματίζεται μια local ειδοποίηση έπειτα από τον επιλεγμένο χρόνο η οποία θα προτρέπει το χρήστη είτε να δηλώσει ότι έφτασε με ασφάλεια, είτε ότι βρίσκεται σε κίνδυνο είτε επιτρέπει τον επαναπρογραμματισμό της. Η συνάρτηση enable_following -η οποία καλείται όποτε ανανεώνεται η τοποθεσία του

συνδεδεμένου χρήστη- ενημερώνει με τη νέα τοποθεσία τον κόμβο current που του αντιστοιχεί, όπως εξηγήθηκε στην παράγραφο 4.2.2.6.

5.2 Υλοποίηση του Today Extension

Το Today Extension (Widget) της εφαρμογής, υλοποιεί όπως αναφέρθηκε και παραπάνω την λειτουργία της προβολής επικινδυνότητας περιοχής, ελαφρώς όμως απλοποιημένη. Παρακάτω καταγράφεται ο κώδικας του όπου έγινε χρήση των τεχνικών και μεθόδων που αναφέρθηκαν στο κεφάλαιο 2.1.3.

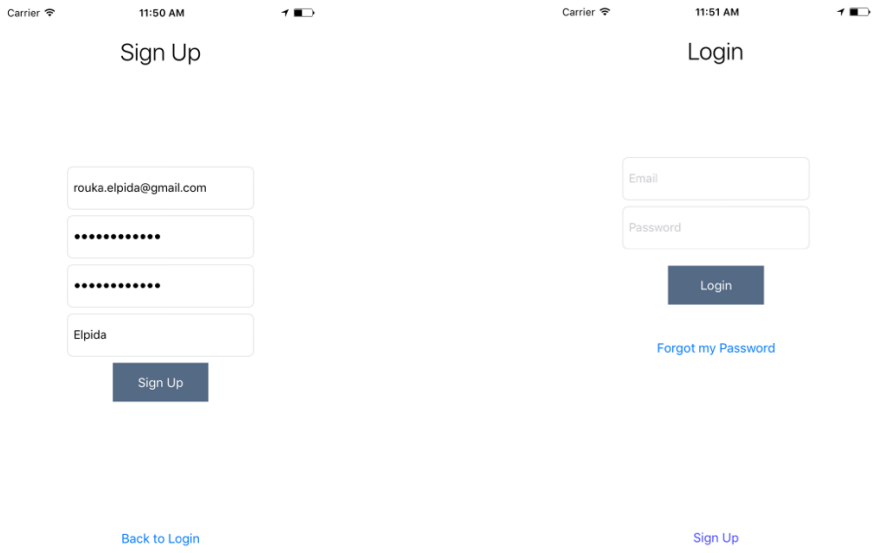
Κλάση	TodayViewController

```
class TodayViewController: UIViewController, NCWidgetProviding, CLLocationManagerDelegate {

    @IBOutlet weak var numberLabel: UILabel!
    @IBOutlet weak var resultLabel: UILabel!
    var postsRef:FIRDatabaseReference!
    var prevPosts:Int? = nil
    static var isAlreadyLaunchedOnce = false // Used to avoid 2 FIRApp configure
    var locationManager: CLLocationManager = CLLocationManager()
    var currentLocation: CLLocation?
    private var updateResult = NCUpdateResult.noData

    override func viewDidLoad() {
        super.viewDidLoad()
        setLabels()
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        if !TodayViewController.isAlreadyLaunchedOnce {
            FIRApp.configure()
            TodayViewController.isAlreadyLaunchedOnce = true
            numberLabel.text = "Loading..."
            resultLabel.text = "Loading..."
        }
        postsRef = FIRDatabase.database().reference().child("posts")
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(true)
        locationManager.requestLocation()
    }

    func locationManager(_ manager:CLLocationManager,didUpdateLocations locations:[CLLocation]){

        if let currentLocation = locations.first {
            postsRef.observeSingleEvent(of: .value, with: { (snapshot) in

                let posts = self.calcNearestPosts(snapshot: snapshot, curloc: currentLocation)
                if posts != self.prevPosts {
                    self.updateResult = .newData
                    self.prevPosts = posts
                    self.setLabels()
                }
                else {
                    self.updateResult = .noData
                }
            })
        }
    }

    func calcNearestPosts(snapshot: FIRDataSnapshot, curloc: CLLocation) -> Int{
        var events = 0
        for rest in snapshot.children.allObjects {
            let long = (rest as! FIRDataSnapshot).childSnapshot(forPath: "longitude").value
            let lat = (rest as! FIRDataSnapshot).childSnapshot(forPath: "latitude").value
            let curloc = self.locationManager.location
            let postloc = CLLocation(latitude: lat as! CLLocationDegrees, longitude: long as!
CLLocationDegrees)
            let distanceInMeters = curloc?.distance(from: postloc) // result is in meters

            let timestamp = (rest as! FIRDataSnapshot).childSnapshot(forPath: "timestamp").value
            let date = Date(timeIntervalSince1970: timestamp as! TimeInterval)
            let years = Calendar.current.dateComponents([.year],from:date, to: Date()).year ?? 0

            if (Int(distanceInMeters!) <= 1000 && years <= 1){
                events += 1
            }
```

```
        }
        return events
    }

    func setLabels(){

        if let posts = prevPosts {
        self.numberLabel.text = "\(posts) event(s) during the last year within one mile"
            if posts < 5 {
                self.resultLabel.text = "Safe Area"
                self.resultLabel.textColor = UIColor.green
            }
            else if posts < 10 {
                self.resultLabel.text = "Medium Safety Area"
                self.resultLabel.textColor = UIColor.orange
            }
            else {
                self.resultLabel.text = "Dangerous Area"
                self.resultLabel.textColor = UIColor.red
            }
        }
    }

    func locationManager(_ manager: CLLocationManager, didFailWithError error: Error) {
        self.updateResult = .failed
        self.numberLabel.text = "Open location to retrieve data"
        self.resultLabel.text = ""
        print(error.localizedDescription)
    }

    func widgetPerformUpdate(completionHandler: (@escaping (NCUpdateResult) -> Void)) {
        setLabels()
        locationManager.requestLocation()
        completionHandler(updateResult)
    }
}
```

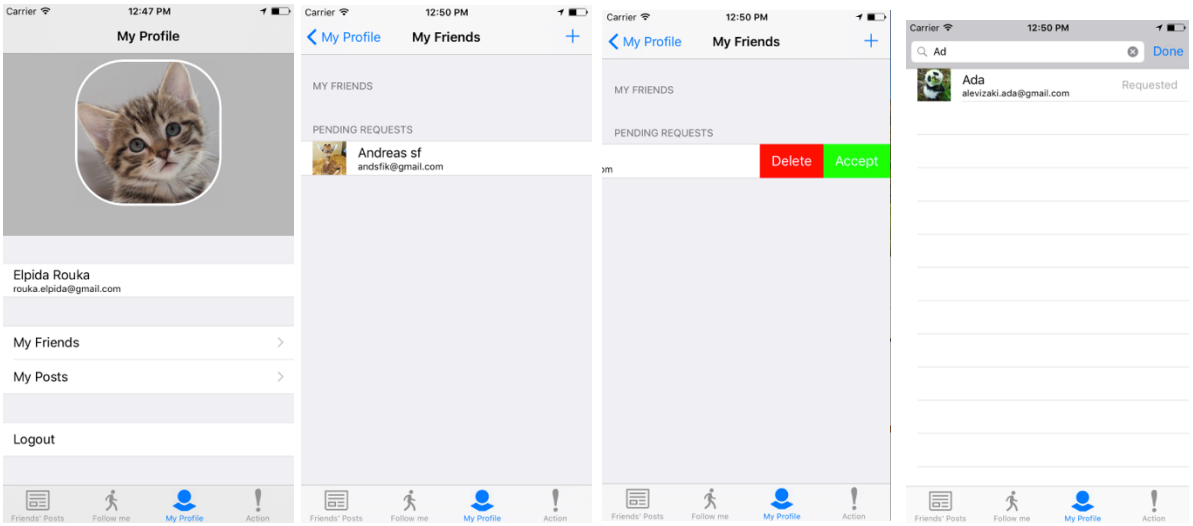
5.3 Παράδειγμα χρήσης

Σε αυτή την ενότητα θα παρουσιαστεί ένα σενάριο χρήσης της εφαρμογής μέσω του οποίου θα κατανοηθούν πρακτικά όλες οι λειτουργίες της. Στο σενάριο θα συμμετέχουν 4 χρήστες, η Άντα η Ελένη και η Ελπίδα και ο Αντρέας. Η Ελπίδα αρχικά δεν έχει λογαριασμό στην εφαρμογή. Γι αυτό, μόλις την εγκαταστήσει, πατά στην αρχική οθόνη το κουμπί “Sign Up” και συμπληρώνει τα στοιχεία της. Οι υπόλοιποι χρήστες έχουν ήδη λογαριασμό οπότε συμπηρώνουν κατευθείαν τα στοιχεία τους στην αρχική οθόνη και εισέρχονται στην εφαρμογή.



Εικόνα 32: Οθόνες εγγραφής και σύνδεσης.

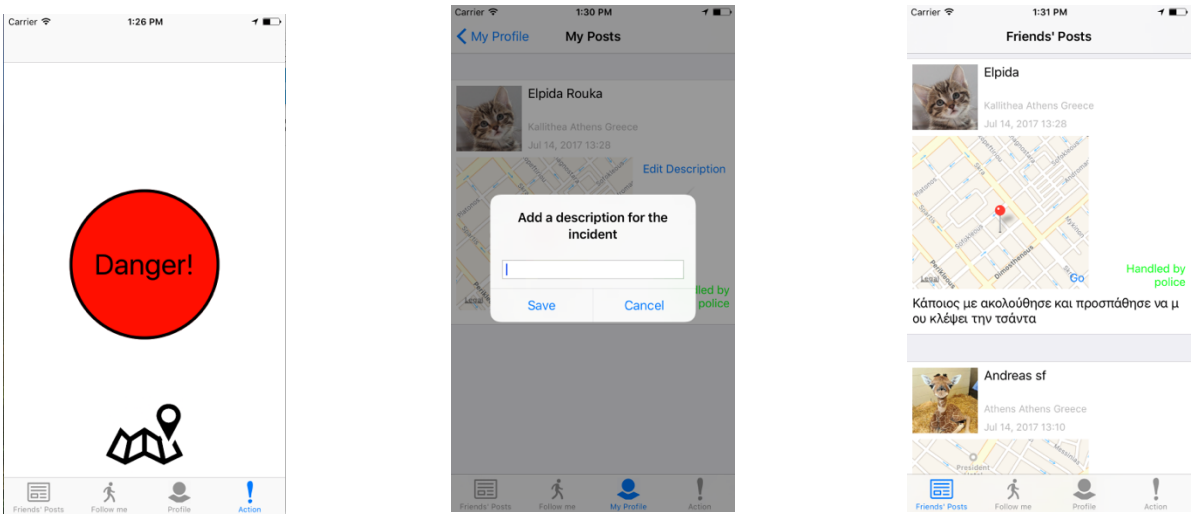
Η Ελένη και η Άντα είναι ήδη φίλες μεταξύ τους. Η Ελπίδα θέλει να προσθέσει την Άντα. Αφού κατευθύνεται στο προφίλ της και αλλάζει φωτογραφία προφίλ, πατά στο κελί “My Friends” και βλέπει ότι έχει αίτημα φιλίας από τον Αντρέα. Το αποδέχεται, πάει στην οθόνη προσθήκης φίλων, βρίσκει την Άντα και την προσθέτει (Εικόνα 33). Η Άντα αποδέχεται την Ελπίδα.



Εικόνα 33: Αποδοχή και δημιουργία αιτήματος φιλίας.

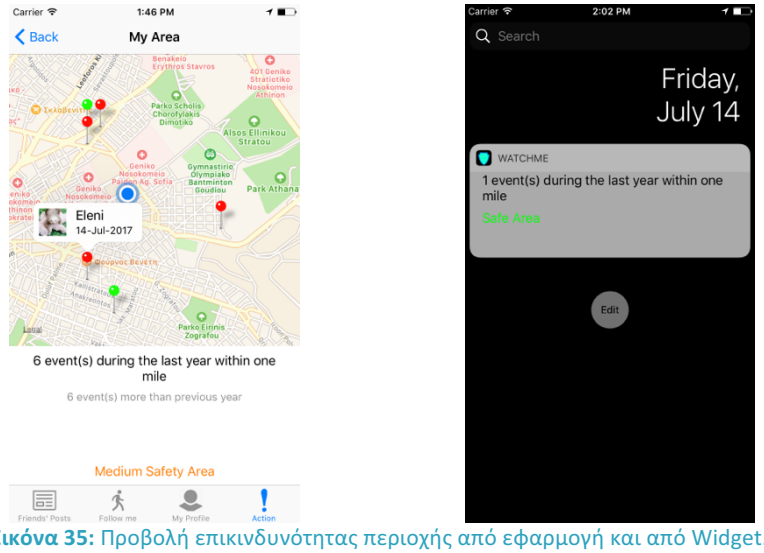
Η Ελπίδα, κάνοντας βόλτα στην Καλλιθέα παρατηρεί ότι κάποιος την ακολουθεί και αμέσως στέλνει σήμα κινδύνου από το κινητό της. Ευτυχώς στην περιοχή υπήρχε αστυνομία που συνέλαβε τον κλέφτη. Η Ελπίδα, γυρνώντας ασφαλής σπίτι της επεξεργάζεται τη δημοσίευση που μόλις

δημιούργησε, περιγράφοντας πως έγινε το περιστατικό και δηλώνοντας ότι η αστυνομία το χειρίστηκε. Η Άντα, που το επόμενο πρωί ανοίγει την εφαρμογή, βλέπει στην αρχική της οθόνη τι έχει συμβεί (Εικόνα 34).



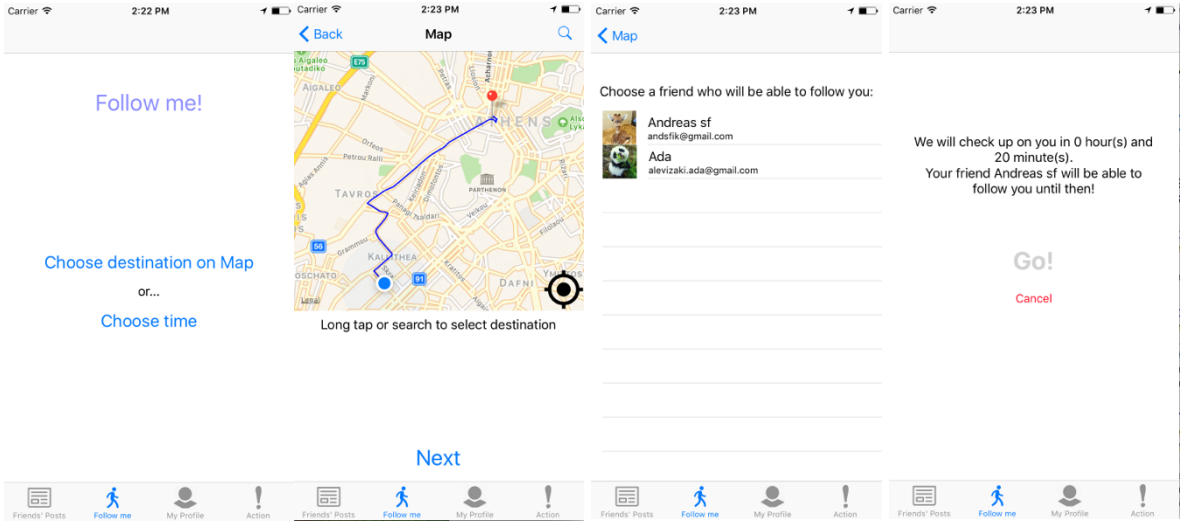
Εικόνα 34: Δημιουργία, επεξεργασία σήματος κινδύνου και εμφάνιση στον timeline φίλου.

Θεωρούμε ότι έχει περάσει κάποιος καιρός οπότε έχουν συμβεί αρκετά περιστατικά από διάφορους χρήστες της εφαρμογής. Μία μέρα που η Ελένη κυκλοφορεί στου Ζωγράφου, θυμάται ότι κάποιος είχε προσπαθήσει να της επιτεθεί πριν λίγο καιρό εκεί κοντά. Ανοίγοντας της εφαρμογή στην οθόνη προβολής επικινδυνότητας περιοχής βλέπει, μαζί με το δικό της, άλλα 5 περιστατικά που έχουν συμβεί τον τελευταίο χρόνο εντός ενός χιλιομέτρου καθιστώντας την περιοχή μειωμένης ασφάλειας. Μάλιστα παρατηρεί ότι μόνο τα 2 από τα 5 περιστατικά αντιμετωπίστηκαν από την αστυνομία (πράσινα pins). Την ίδια στιγμή, η Άντα που περπατά στην Καλλιθέα πραγματοποιεί τον ίδιο έλεγχο απευθείας από το Widget της εφαρμογής, διαπιστώνοντας ότι η περιοχή στην οποία βρίσκεται εκείνη είναι ασφαλής (Εικόνα 35).



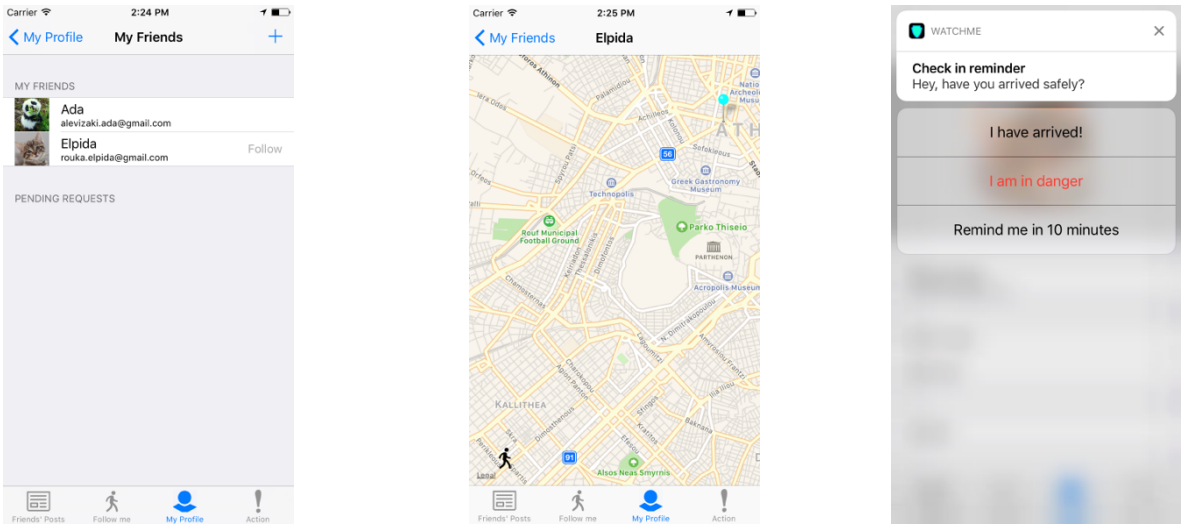
Εικόνα 35: Προβολή επικινδυνότητας περιοχής από εφαρμογή και από Widget.

Λίγες μέρες μετά, η Ελπίδα έχει μια υποχρέωση σε μια όχι τόσο ασφαλή περιοχή και προτιμά να την παρακολουθεί κάποιος μέχρι να φτάσει στον προορισμό της. Γι αυτό το σκοπό κατευθύνεται στην οθόνη follow, επιλέγει τον προορισμό της στο χάρτη, μετά επιλέγει να την ακολουθήσει ο Αντρέας και εκκινεί τη λειτουργία.



Εικόνα 36: Ρύθμιση και εκκίνηση λειτουργίας Follow.

Ο Αντρέας τώρα, πηγαίνοντας στην οθόνη των φίλων του, βλέπει πως έχει δεχθεί πρόσκληση να ακολουθήσει την Ελπίδα αφού εμφανίζεται η ένδειξη “follow” δίπλα στο όνομά της. Πατώντας, στο κελί που της αντιστοιχεί, μεταφέρεται σε έναν χάρτη όπου με γαλάζιο pin δηλώνεται ο προορισμός της Ελπίδας και με ένα ανθρωπάκι η θέση της. Έτσι, είναι δυνατό να μαθαίνει αμέσως τις αλλαγές της τοποθεσίας της, καθώς και αν κατευθύνεται όντως προς τον προορισμό της. Μετά από αυτά τα αναμενόμενα 20 λεπτά που έχουν υπολογιστεί από την εφαρμογή, και ενώ η Ελπίδα έχει φτάσει ασφαλής στον προορισμό της, της εμφανίζεται μια τοπική ειδοποίηση. Σε αυτήν η Ελπίδα δηλώνει πως έφτασε με ασφάλεια οπότε και ο Αντρέας σταματά να μπορεί να την παρακολουθεί. Σε διαφορετική περίπτωση, θα μπορούσε να έχει δημιουργήσει ένα σήμα κινδύνου δηλώνοντας στην ειδοποίηση ότι είναι σε κίνδυνο.



Εικόνα 37: Παρακολούθηση φίλου και λήψη τοπικής ειδοποίησης από τον ακολουθούμενο.

Επίλογος

6.1 Συμπεράσματα

Κατά τη σχεδίαση και υλοποίηση αυτής της εφαρμογής προέκυψαν πολλά εμπόδια και πολλές προκλήσεις. Η ανάπτυξη εφαρμογών για κινητά τηλέφωνα και ειδικά για το λετουργικό σύστημα iOS έχει εξελιχθεί ταχύτατα τα τελευταία χρόνια, προσφέροντας στους προγραμματιστές εξαιρετικές δυνατότητες που παλαιότερα οι χρήστες κινητών τηλεφώνων δεν θα μπορούσαν ούτε να τις φανταστούν. Ωστόσο, απαιτείται μεγάλη τριβή με τις συγκεκριμένες τεχνολογίες για να έχει ο προγραμματιστής μια πλήρη ιδέα των δυνατοτήτων που του παρέχονται και να μπορεί να τις συνδυάσει και να τις αξιοποιήσει επιτυχώς.

Ιδιαίτερα οι υπηρεσίες τοποθεσίας που χρησιμοποιήθηκαν στο μεγαλύτερο κομμάτι της εφαρμογής, τείνουν να γίνουν άκρως απαραίτητες στη σύγχρονη εποχή. Η ανάγκη του ανθρώπου για σταθερά υπολογιστικά συστήματα έχει μειωθεί σημαντικά, ενώ διαρκώς αυξάνεται η εξοικείωση και “εξάρτηση” του από φορητές συσκευές. Οι υπηρεσίες τοποθεσίας είναι άμεσα συνυφασμένες με αυτή την ιδέα και έχουν εισχωρήσει σε πολλές πλευρές της ανθρώπινης δραστηριότητας τα τελευταία χρόνια.

Αξίζει επίσης να αναφέρω την τρομερή επανάσταση που επιφέρει το Firebase στη διαχείρηση βάσεων δεδομένων. Αν και, λόγω της πρόσφατης ανάπτυξης του, ελάχιστο ποσοστό των σημερινών εφαρμογών το χρησιμοποιούν, θεωρώ πως μπορεί να προσφέρει μια τελείως διαφορετική, εκμοντερνισμένη προγραμματιστική εμπειρία, καθιστώντας λειτουργίες που απαιτούσαν παλιά εκατοντάδες γραμμές κώδικα εφικτές μόνο σε μερικές γραμμές. Η λογική του -αποκλειστικά- client-side programming μπορεί να επιτρέψει σε αρχάριους προγραμματιστές να πειραματιστούν εύκολα και σε πεπειραμένους να ξεπεράσουν τον εαυτό τους προσανατολίζοντας όλες τους τις δυνατότητες σε μια εξαιρετική εμπειρία χρήστη.

Γενικά, ο ανταγωνισμός που επικρατεί σήμερα στο χώρο της ανάπτυξης εφαρμογών είναι πολύ υψηλός λόγω των δισεκατομμύρια εφαρμογών που υπάρχουν. Πέρα λοιπόν από την εμπειρία και την ικανότητα, ένας προγραμματιστής οφείλει να είναι δημιουργικός, ανοιχτόμυαλος και ικανός να σκεφτεί τι θα περίμενε ο κάθε χρήστης από την εκάστοτε εφαρμογή. Μόνο έτσι, και φυσικά με τη συνεργασία ταλαντούχων γραφιστών μπορεί μια εφαρμογή να πετύχει και τελικά να επιφέρει κέρδος στους δημιουργούς της.

6.2 Μελλοντικές επεκτάσεις

Η εφαρμογή WachMe σίγουρα βρίσκεται ακόμη σε ένα αρκετά πειραματικό στάδιο. Πολλές βελτιώσεις είναι εφικτές, κυρίως στα γραφικά, στην απόδοση και στην ταχύτητα της επικοινωνίας μεταξύ εφαρμογής και Firebase. Επιπλέον, οι κανόνες ασφαλείας του Firebase είναι δυνατό να επεκταθούν περαιτέρω ώστε να ανταποκρίνονται πλήρως στις ανάγκες ιδιωτικότητας μιας μελλοντικής εφαρμογής. Επίσης, εφόσον τα μέσα κοινωνικής δικτύωσης χρησιμοποιούνται ευρύτατα σήμερα, θα ήταν εύστοχο να προστεθεί η δυνατότητα σύνδεσης του χρήστη μέσω Facebook ή άλλων κοινωνικών δικτύων –μία δυνατότητα που το Firebase προσφέρει εύκολα. Ταυτόχρονα, μπορούν να υπάρξουν επεκτάσεις ώστε η εφαρμογή να λειτουργεί και σε Android συσκευές, και συνεπώς να αποκτήσει περισσότερους χρήστες.

Ωστόσο, η σημαντικότερη κατά τη γνώμη μου επέκταση είναι να προστεθούν στην εφαρμογή απομακρυσμένες ειδοποιήσεις (remote notifications). Με αυτόν τον τρόπο η εφαρμογή θα ανταποκρίνεται πιο ρεαλιστικά στις ανάγκες των χρηστών της. Για παράδειγμα, απομακρυσμένες ειδοποιήσεις μπορεί να λαμβάνει ο χρήστης σε περιπτώσεις αιτημάτων φιλίας, όταν κάποιος φίλος του εκπέμπει σήμα κινδύνου ή όταν κάποιος φίλος προσκαλεί τον χρήστη να τον ακολουθήσει. Σε αυτό το κλίμα, μπορεί να προστεθεί η λειτουργία αυτόματης αποστολής γραπτού μηνύματος όταν ένας φίλος βρίσκεται σε κίνδυνο, ώστε να μην χρειάζεται ο παραλήπτης να είναι συνδεδεμένος στο διαδίκτυο.

Τέλος, στο πλαίσιο της επικείμενης κυκλοφορίας του iOS 11 το φθινόπωρο του 2017, θα ήταν εύλογο να εντάξουμε στην εφαρμογή τη σχετικά νέα τεχνολογία της Apple που ονομάζεται Siri. Στις εκδόσεις iOS που πρόκειται να έρθουν, είναι πιθανό η χρήση του να είναι πιο ευέλικτη, επιτρέποντας σε πολλά είδη εφαρμογών να το χρησιμοποιήσουν. Είναι φανερό ότι σε μια εφαρμογή όπως το WatchMe που προσανατολίζεται σε επείγοντες περιπτώσεις κινδύνου, η ενεργοποίηση λειτουργιών μόνο με τη φωνή μπορεί να εκτοξεύσει την ικανοποίηση των χρηστών.

Αυτές είναι μόνο μερικές από τις πιθανές επεκτάσεις. Η διαφορετικότητα στις ιδέες κάθε προγραμματιστή και οι ταχύτατα εξελισσόμενες τεχνολογίες θα μπορούσαν να εξελίξουν μία τέτοιου είδους πειραματική εφαρμογή σε ένα επιτυχημένο τελικό προϊόν μέσα στα επόμενα χρόνια.

Βιβλιογραφία

[1] Apple, *The Swift Programming Language Swift 3.1 Edition*, 2014

[2] Chris Adamson, Janie Clayton, *iOS 10 SDK Development: Creating iPhone and iPad Apps with Swift*, 2017

[3] Matt Neuburg, *iOS 10 Programming Fundamentals with Swift*, 2016

[4] Nutting Jack, Clark Peter, *Learn Cocoa on the Mac*, 2013

[5] Apple Developer Portal, https://developer.apple.com/

[6] Firebase Documentation, https://firebase.google.com/docs/

[7] Statista, https://statista.com

[8] CocoaPods, https://cocoapods.org

[9] Wikipedia, http://en.wikipedia.org

[10] Techopedia, https://www.techopedia.com/