ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Εφαρμογή παιχνιδιών σε iOS»

Του φοιτητή
Βέτσου Νικόλαου
Αρ. Μητρώου: 510014

Επιβλέπων
Παναγιώτης Τζέκης
Καθηγητής

Πρόλογος

Την περίοδο του covid και των lockdown μας έλλειψε η επαφή με τους δικούς μας ανθρώπους και είχαμε ανάγκη να ξεφεύγουμε από την καθημερινότητα. Τα παιχνίδια ρόλων με παρέα είναι αυτό που προσωπικά μας βοήθησε να περάσουμε ευχάριστα και δημιουργικά εκείνες τις μέρες. Η εφαρμογή που επέλεξα να δημιουργήσω συνδυάζει τα ενδιαφέροντα μου και έχει σκοπό να βοηθήσει χωρίς πείρα παίκτες να συμμετέχουν με ευκολία σε αυτά.

Περίληψη

Η διπλωματική εργασία είναι μία εφαρμογή σε λειτουργικό σύστημα iOS τριών παιχνιδιών που μπορούν να παίξουν παρέες από κοντά. Έχει σκοπό την ολοκληρωμένη τεκμηρίωση των στοιχείων που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής, καθώς και η ανάλυση των κλάσεων του κώδικα που χρησιμοποιήθηκαν και γράφτηκαν. Στο εν λόγο κείμενο μπορεί ο αναγνώστης να έχει μία πλήρη εικόνα για τον προγραμματισμό συσκευών που έχουν το λειτουργικό σύστημα iOS και μπορεί να το χρησιμοποιήσει ως ένα εγχειρίδιο για τον προγραμματισμό δικών του εφαρμογών.

«Gaming app in iOS»

«Nikolaos Vetsos»

Abstract

The thesis is an iOS app of three games that can be played in close groups. It aims at the comprehensive documentation of the elements used to create the application, as well as the analysis of the code classes used and written. With this text and documentation, the reader can have a complete picture of how to program apps for devices that have the iOS operating system and can use it as a manual for programming their own applications.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: iOS development

1.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφονται οι βασικές γνώσεις που χρειάζεται να κατέχει κάποιος προγραμματιστής για την δημιουργία εφαρμογών σε συσκευές με λειτουργικό σύστημα iOS. Για αρχή παρουσιάζεται το λειτουργικό σύστημα iOS, η γλώσσα swift στην οποία γράφονται οι εφαρμογές και έπειτα τα εργαλεία που χρησιμοποιούνται για την γραφή του κώδικα. Έπειτα αναλύεται το βασικότερο framework όλων των εφαρμογών το UIKit και τέλος οι σημαντικότερες κλάσεις του.

1.2 iOS

Το iOS είναι ένα λειτουργικό σύστημα που δημιουργήθηκε από την Apple αποκλειστικά για τις συσκευές που κατασκευάζει η ίδια η εταιρεία. Είναι το λειτουργικό σύστημα που χρησιμοποιείται σε όλες τις κινητές συσκευές της, συμπεριλαμβανομένων των iPhone, iPad και iPod Touch. Είναι το δεύτερο πιο δημοφιλές κινητό λειτουργικό σύστημα παγκοσμίως μετά το Android.

Πρωτοπαρουσιάστηκε το 2007, για το iPhone. Τώρα το iOS έχει επεκταθεί για να υποστηρίξει και άλλες συσκευές της Apple: όπως το iPod Touch (Σεπτέμβριος 2007) και το iPad (Ιανουάριος 2010). Τον Νοέμβριο του 2022 το App Store της Apple περιέχει περισσότερα από 1,8 εκατομμύρια iOS εφαρμογές, για συσκευές iPad και iPhone.

Η διεπαφή χρήστη iOS βασίζεται σε άμεσο χειρισμό, χρησιμοποιώντας χειρονομίες πολλαπλής αφής. Τα στοιχεία ελέγχου της διασύνδεσης αποτελούνται από διακόπτες και κουμπιά. Για την αλληλεπίδραση με το λειτουργικό σύστημα χρησιμοποιούνται ειδικές χειρονομίες: ελαφρύ χτύπημα, μετακινούμενο άγγιγμα με ένα δάχτυλο, μετακινούμενο άγγιγμα με δύο δάχτυλα και συνθέσεις αυτών. Τα παραπάνω διαθέτουν ειδικούς ορισμούς στα πλαίσια του λειτουργικού συστήματος iOS και των διεπαφών πολλαπλής αφής. Επίσης τα εσωτερικά επιταχυνσιόμετρα των συσκευών χρησιμοποιούνται σε αρκετές εφαρμογές.

Σημαντικές εκδόσεις του iOS κυκλοφορούν ετησίως. Το iOS 16 είναι η πιο πρόσφατη έκδοση του λειτουργικού συστήματος iOS της Apple, και είναι διάδοχος του iOS 15. Ανακοινώθηκε κατά τη διάρκεια του συνεδρίου προγραμματιστών, WWDC22, της Apple, στις 6 Ιουνίου 2022 και κυκλοφόρησε στο ευρύ κοινό στις 12 Σεπτεμβρίου 2022. Αισθητικά μοιάζει με το iPadOS 16. Είναι το πρώτο λογισμικό που δουλεύει αποκλειστικά σε iPhones, καθώς τα tablet της εταιρίας έχουν πλέον το δικό τους λογισμικό. Περιλαμβάνει αρκετές ενημερώσεις και νέα χαρακτηριστικά. Βασική προσθήκη στο iOS 16 είναι η βελτίωση του multitasking.[1]

1.3 Swift

Η Swift είναι μια γλώσσα προγραμματισμού γενικής χρήσης που κατασκευάζεται χρησιμοποιώντας μια σύγχρονη προσέγγιση για την ασφάλεια, την απόδοση και τα σχέδια σχεδιασμού λογισμικού.

Ο στόχος της γλώσσας Swift είναι να δημιουργήσει την καλύτερη διαθέσιμη γλώσσα για χρήσεις που κυμαίνονται από τον προγραμματισμό συστημάτων έως εφαρμογές για κινητά και υπολογιστές, κλιμακούμενες μέχρι υπηρεσίες cloud. Το πιο σημαντικό είναι ότι η Swift έχει σχεδιαστεί για να διευκολύνει το γράψιμο και τη συντήρηση των προγραμμάτων για τον προγραμματιστή.

Η Swift είναι:

•Ασφαλής. Τα λάθη του προγραμματιστή πρέπει να πιάνονται πριν από την παραγωγή του λογισμικού. Η Swift έχει μηχανισμούς οι οποίοι προλαμβάνουν τα λάθη αυτά και είναι προαιρετικοί. Η επιλογή για ασφάλεια σημαίνει μερικές φορές ότι η Swift είναι αυστηρή γλώσσα, αλλά η σαφήνεια που έχει σαν αποτέλεσμα εξοικονομεί χρόνο μακροπρόθεσμα.

•Γρήγορη. Η Swift προορίζεται να αντικαταστήσει τις γλώσσες που βασίζονται σε C (C, C ++ και Objective-C). Ως εκ τούτου, η Swift είναι συγκρίσιμη με αυτές τις γλώσσες στην απόδοση για τις περισσότερες εργασίες. Η απόδοση είναι επίσης προβλέψιμη και συνεπής, όχι μόνο γρήγορη σε κάποιες περιπτώσεις.

•Εκφραστική. Η Swift επωφελείται από δεκαετίες προόδου στην επιστήμη των υπολογιστών για να προσφέρει σύνταξη που είναι ευχάριστη στη χρήση, με σύγχρονα χαρακτηριστικά που περιμένουν οι προγραμματιστές. Αλλά η Swift δεν έχει τελειώσει. Η γλώσσα συνεχώς εξελίσσεται για να γίνεται ακόμη καλύτερη.

Τα εργαλεία αποτελούν κρίσιμο μέρος της Swift. Εντάσσονται καλά στο σύνολο εργαλείων ενός προγραμματιστή, δημιουργούν κώδικα γρήγορα, παρουσιάζουν διαγνώσεις για τον κώδικα και δίνουν τη δυνατότητα διαδραστικών εμπειριών ανάπτυξης. Τα εργαλεία μπορούν να κάνουν τον προγραμματισμό πιο ισχυρό. Ένα παράδειγμα εργαλείου είναι το Swift-based playgrounds που υπάρχει στο Xcode. [2]

1.3.1 Χαρακτηριστικά / Features

Η Swift περιλαμβάνει λειτουργίες που διευκολύνουν την ανάγνωση και γραφή του κώδικα, δίνοντας στον προγραμματιστή τον απαραίτητο έλεγχο σε μια πραγματική γλώσσα προγραμματισμού συστημάτων. Η Swift υποστηρίζει υποτιθέμενους τύπους για να καταστήσει τον κώδικα καθαρότερο και λιγότερο επιρρεπή σε λάθη, και οι ενότητες εξαλείφουν τις επικεφαλίδες και παρέχουν χώρους ονομάτων. Η μνήμη διαχειρίζεται αυτόματα και δεν χρειάζεται καν η πληκτρολόγηση κάποιου συμβόλου στο τέλος κάθε γραμμής κώδικα για την ένδειξη ολοκλήρωσης γραμμής. Η Swift δανείζεται επίσης από άλλες γλώσσες. Για παράδειγμα οι ονομαζόμενες παράμετροι που προωθούνται από την γλώσσα Objective-C εκφράζονται σε μια καθαρή σύνταξη που κάνει τα API στη Swift εύκολα να διαβαστούν και να διατηρηθούν. [3]

Τα χαρακτηριστικά της Swift έχουν σχεδιαστεί για να συνεργάζονται και να δημιουργήσουν μια γλώσσα που είναι ισχυρή, αλλά διασκεδαστική στη χρήση. Ορισμένες πρόσθετες λειτουργίες του Swift περιλαμβάνουν:

•Closures ενοποιημένα με δείκτες λειτουργίας

•Tuples και πολλαπλές επιστρεφόμενες τιμές

•Generics

•Γρήγορη και συνοπτική επανάληψη σε ένα range ή ένα collection

•Structs που υποστηρίζουν μεθόδους, επεκτάσεις και πρωτόκολλα

•Functional programming patterns, π.χ. map και filter

•Ισχυρό ενσωματωμένο χειρισμό σφαλμάτων

•Προχωρημένη ροή ελέγχου με την εκτέλεση do, guard, defer, και repeat λέξεων-κλειδιών

1.3.2 Ασφάλεια

Η Swift σχεδιάστηκε εξ αρχής για να είναι ασφαλέστερη από τις γλώσσες που βασίζονται σε C και εξαλείφει ολόκληρες κατηγορίες μη ασφαλούς κώδικα. Οι μεταβλητές αρχικοποιούνται πάντα πριν από τη χρήση, οι πίνακες και οι ακέραιοι αριθμοί ελέγχονται για υπερχείλιση και η μνήμη διαχειρίζεται αυτόματα. Η σύνταξη ρυθμίζεται με ευκολία - για παράδειγμα, οι απλές λέξεις-κλειδιά τριών χαρακτήρων ορίζουν μια μεταβλητή (var) ή μια σταθερά (let).

Ένα άλλο χαρακτηριστικό ασφαλείας είναι ότι από προεπιλογή τα αντικείμενα Swift δεν μπορούν ποτέ να είναι μηδενικά και η προσπάθεια να δημιουργηθεί ή να χρησιμοποιηθεί ένα μηδενικό αντικείμενο θα έχει ως αποτέλεσμα ένα compile-time error. Αυτό καθιστά τον κώδικα γραφής πολύ καθαρότερο και ασφαλέστερο και αποτρέπει μια κοινή αιτία των συνθηκών εκτέλεσης. Ωστόσο, υπάρχουν περιπτώσεις όπου μία τιμή είναι κατάλληλη για αυτές τις καταστάσεις η Swift διαθέτει ένα καινοτόμο χαρακτηριστικό γνωστό ως optional. Ένα optional μπορεί να περιέχει τιμή αλλά μπορεί και όχι, η σύνταξη της Swift αναγκάζει τη διαχείριση αυτού με ασφάλεια χρησιμοποιώντας κάποιες τεχνικές.

Η πρώτη τεχνική είναι χρησιμοποιώντας ένα «?» για να υποδείξει στον compiler την κατανόηση της συμπεριφοράς και τον χειρισμό με ασφάλεια. Όταν θα φτάσει η στιγμή της εκτέλεσης του προγράμματος και θα χρειαστεί να χρησιμοποιηθούν τα δεδομένα της μεταβλητής η οποία έχει ερωτηματικό, αν η μεταβλητή δεν έχει τιμή τότε η εφαρμογή θα συνεχίσει χωρίς την εκτέλεση της γραμμής αυτής.

Στην δεύτερη τεχνική χρησιμοποιείται ένα «!» που ορίζει στον compiler πως αν χρειαστεί η ανάγνωση της τιμής αυτής της μεταβλητής αλλά έχει μηδενική τιμή τότε υποχρεωτικά πρέπει να τερματιστεί η εφαρμογή. Η τεχνική αυτή χρησιμοποιείται σε συγκεκριμένες περιπτώσεις και είναι αρκετά χρήσιμη για της εύρεση λαθών κατά την ανάπτυξη κώδικα, πριν την δημοσίευση της τελικής εφαρμογής.

Η τρίτη τεχνική είναι η ανάθεση προκαθορισμένης τιμής και χρησιμοποιείται με τα σύμβολα «??». Μετά τα σύμβολα ακολουθεί η προκαθορισμένη τιμή. Όταν κατά την εκτέλεση του προγράμματος χρειαστεί η τιμή της μεταβλητής αυτής, τότε αν δεν υπάρχει θα χρησιμοποιηθεί η προκαθορισμένη τιμή. [4]

1.3.3 Υποστήριξη

Μια από τις πιο συναρπαστικές πτυχές της ανάπτυξης με τη γλώσσα Swift είναι ότι γίνεται να μεταφερθεί σε ένα ευρύ φάσμα από πλατφόρμες, συσκευές και περιπτώσεις χρήσεως.

Στόχος είναι να παρέχεται συμβατότητα πηγής για τη Swift σε όλες τις πλατφόρμες, παρόλο που οι πραγματικοί μηχανισμοί εφαρμογής ενδέχεται να διαφέρουν από τη μία πλατφόρμα στην άλλη. Το κύριο παράδειγμα είναι ότι οι πλατφόρμες της Apple περιλαμβάνουν Objective-C runtime, το οποίο απαιτείται για την πρόσβαση σε πλαίσια της πλατφόρμας της Apple όπως UIKit και AppKit. Στις άλλες πλατφόρμες, όπως το Linux, δεν υπάρχει Objective-C runtime, επειδή δεν είναι απαραίτητο.

Το έργο Swift Core Libraries στοχεύει στην επέκταση των δυνατοτήτων cross-platform της Swift παρέχοντας φορητές υλοποιήσεις βασικών πλαισίων της Apple (όπως το Foundation) χωρίς εξαρτήσεις για το χρόνο εκτέλεσης της Objective-C. Αν και οι βασικές βιβλιοθήκες βρίσκονται σε πρώιμο στάδιο ανάπτυξης, παρέχουν τελικά βελτιωμένη συμβατότητα πηγής για τον κώδικα Swift σε όλες τις πλατφόρμες.

1.3.4 Πλατφόρμες της Apple

Η Swift ανοιχτού κώδικα μπορεί να χρησιμοποιηθεί σε Mac για να στοχεύσει όλες τις πλατφόρμες της Apple: iOS, macOS, watchOS και tvOS. Επιπλέον, οι δυαδικές κατασκευές Swift ανοιχτού κώδικα ενσωματώνονται με τα εργαλεία ανάπτυξης προγραμματιστών Xcode, συμπεριλαμβανομένης της πλήρους υποστήριξης για το σύστημα δημιουργίας Xcode, την ολοκλήρωση του κώδικα στον επεξεργαστή και την ολοκληρωμένη αποσφαλμάτωση, επιτρέποντας σε οποιονδήποτε να πειραματιστεί με τις τελευταίες εξελίξεις της Swift σε ένα γνωστό Cocoa και Cocoa Touch περιβάλλον ανάπτυξης. Παρέχουν τελικά βελτιωμένη συμβατότητα πηγής για τον κώδικα Swift σε όλες τις πλατφόρμες.

1.4 Xcode

Το Xcode είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για macOS που περιέχει μια σειρά από εργαλεία ανάπτυξης λογισμικού που δημιουργήθηκαν από την Apple για macOS, iOS, watchOS και tvOS. Πρώτα κυκλοφόρησε το 2003. Η πιο πρόσφατη σταθερή έκδοση είναι η έκδοση 14.1 και διατίθεται δωρεάν μέσω του Mac App Store για χρήστες MacOS High Sierra και MacOS Mojave. Οι εγγεγραμμένοι προγραμματιστές μπορούν να πραγματοποιήσουν λήψη ενημερωμένων εκδόσεων και προγενέστερων εκδόσεων μέσω της ιστοσελίδας της Apple Developer.

Το Xcode υποστηρίζει τον πηγαίο κώδικα για τις γλώσσες προγραμματισμού C, C ++, Objective-C, Objective-C ++, Java, AppleScript, Python, Ruby, ResEdit και Swift με διάφορα μοντέλα προγραμματισμού, Cocoa, Carbon και Java. Με επιπρόσθετα εργαλεία υπάρχει υποστήριξη για το GNU Pascal, Free Pascal, Ada, C #, Perl, D και Fortran.

Το Xcode μπορεί να δημιουργήσει fat binary αρχεία που περιέχουν κώδικα για πολλαπλές αρχιτεκτονικές με το εκτελέσιμο αρχείο Mach-O. Αυτά ονομάζονται καθολικά δυαδικά αρχεία, τα οποία επιτρέπουν το λογισμικό να λειτουργεί τόσο σε πλατφόρμες PowerPC όσο και Intel (x86) και μπορεί να περιλαμβάνει κώδικα 32 bit και 64 bit και για τις δύο αρχιτεκτονικές. Χρησιμοποιώντας το iOS SDK, το Xcode μπορεί επίσης να χρησιμοποιηθεί για να μεταγλωττίσει και να εντοπίσει σφάλματα σε εφαρμογές για iOS που εκτελούνται σε επεξεργαστές αρχιτεκτονικής ARM.

Το Xcode περιλαμβάνει το εργαλείο GUI Instruments, το οποίο τρέχει πάνω από ένα δυναμικό πλαίσιο ανίχνευσης, το DTrace, το οποίο δημιουργήθηκε από την Sun Microsystems και κυκλοφόρησε ως μέρος του OpenSolaris.

Η κύρια εφαρμογή της σουίτας είναι το ολοκληρωμένο περιβάλλον ανάπτυξης (IDE), που ονομάζεται επίσης Xcode. Το Xcode περιλαμβάνει τα περισσότερα έγγραφα τεκμηρίωσης της Apple και ενσωματωμένο Interface Builder, μια εφαρμογή που χρησιμοποιείται για την κατασκευή γραφικών διεπαφών χρήστη.

Έως το Xcode 4.1, η σουίτα Xcode περιλαμβάνει μια τροποποιημένη έκδοση της συλλογής μεταγλωττιστή GNU. Στο Xcode 3.1 μέχρι το Xcode 4.6.3 περιλάμβανε τον μεταγλωττιστή LLVM-GCC, με front ends από τη συλλογή μεταγλωττιστή GNU και μια γεννήτρια κώδικα βασισμένη στο LLVM. Στο Xcode 3.2 και αργότερα, περιλάμβανε τον μεταγλωττιστή Clang C / C ++ / Objective-C, με πρόσφατα γραμμένα front ends και μια γεννήτρια κώδικα βασισμένη στο LLVM και τον στατικό αναλυτή Clang. Από το Xcode 4.2, ο μεταγλωττιστής Clang έγινε ο προεπιλεγμένος μεταγλωττιστής. Από το Xcode 5.0, ο Clang ήταν ο μόνος μεταγλωττιστής που παρείχε.

Παλαιότερα, το Xcode υποστήριζε τη διανομή μιας διαδικασίας δημιουργίας προϊόντων σε πολλά συστήματα. Μία σχετική τεχνολογία ονομαζόταν Shared Workgroup Build, η οποία χρησιμοποίησε το πρωτόκολλο Bonjour για να ανακαλύψει αυτόματα τα συστήματα που παρέχουν υπηρεσίες μεταγλωττιστή και μια τροποποιημένη έκδοση του προϊόντος distcc του ελεύθερου λογισμικού για να διευκολύνει τη διανομή του φόρτου εργασίας. Οι παλαιότερες εκδόσεις του Xcode παρείχαν ένα σύστημα με το όνομα Dedicated Network Builds. Αυτές οι δυνατότητες δεν υπάρχουν στις υποστηριζόμενες εκδόσεις του Xcode.

Το Xcode 10 παρουσίασε υποστήριξη για το Dark Mode που ανακοινώθηκε για το MacOS Mojave, τις πλατφόρμες συνεργασίας Bitbucket και GitLab (εκτός από το GitHub), μοντέλα μηχανικής εκπαίδευσης από περιβάλλοντα εκμάθησης που ονομάζονται “Playgrounds”. [5]

1.5 UIKit

Το πλαίσιο UIKit παρέχει την απαιτούμενη υποδομή για εφαρμογές iOS ή tvOS. Παρέχει την αρχιτεκτονική παραθύρων και προβολών για την υλοποίηση της διασύνδεσης, του χειρισμού της υποδομής και των τύπων εισόδου πολλών επαφών και άλλων τύπων στην εφαρμογή. Άλλα χαρακτηριστικά που προσφέρονται περιλαμβάνουν υποστήριξη κινούμενων εικόνων, υποστήριξη εγγράφων, υποστήριξη σχεδίασης και εκτύπωσης, πληροφορίες σχετικά με την τρέχουσα συσκευή, διαχείριση κειμένου και εμφάνιση, υποστήριξη αναζήτησης, υποστήριξη προσβασιμότητας, υποστήριξη επέκτασης εφαρμογών και διαχείριση πόρων. [6]

1.5.1 Storyboard

Ένα Storyboard είναι μια οπτική αναπαράσταση του περιβάλλοντος χρήστη μιας εφαρμογής iOS, που δείχνει οθόνες περιεχομένου και τις συνδέσεις μεταξύ αυτών των οθονών. Ένα storyboard αποτελείται από μια σειρά σκηνών, καθένα από τα οποία αντιπροσωπεύει έναν ελεγκτή προβολής και τα view του. Οι ελεγκτές προβολής συνδέονται με αντικείμενα segue, τα οποία αντιπροσωπεύουν μια μετάβαση μεταξύ δύο ελεγκτών προβολής.

Το Xcode παρέχει ένα πρόγραμμα επεξεργασίας για storyboards, όπου γίνεται ο σχεδιασμός του περιβάλλοντος της εφαρμογής προσθέτοντας αντικείμενα του πλαισίου UIKit χωρίς τη χρήση κώδικα. Επιπλέον, ένα storyboard δίνει τη δυνατότητα να συνδυάζεται ένα view με τον ελεγκτή προβολής και να διαχειρίζεται τη μεταφορά δεδομένων μεταξύ ελεγκτών προβολής. Η χρήση storyboard είναι ο συνιστάμενος τρόπος για τον σχεδιασμό του περιβάλλοντος χρήστη της εφαρμογής, επειδή επιτρέπει να απεικονίζεται η εμφάνιση και η ροή του περιβάλλοντος χρήστη σε έναν καμβά.

Στο iPhone, κάθε σκηνή αντιστοιχεί σε περιεχόμενο πλήρους οθόνης. στο iPad, μπορούν να εμφανιστούν ταυτόχρονα πολλαπλές σκηνές στην οθόνη - για παράδειγμα, χρησιμοποιώντας ελεγκτές αναδυόμενων προβολών. Κάθε σκηνή έχει μία βάση, η οποία εμφανίζει εικονίδια που αντιπροσωπεύουν τα αντικείμενα ανώτατου επιπέδου της σκηνής. Η βάση χρησιμοποιείται κυρίως για τη δημιουργία συνδέσεων δράσης και εξόδου μεταξύ του ελεγκτή προβολή και των view που έχει .

Storyboard

Ένα Storyboard είναι μια οπτική αναπαράσταση του περιβάλλοντος χρήστη μιας εφαρμογής iOS, που δείχνει οθόνες περιεχομένου και τις συνδέσεις μεταξύ αυτών των οθονών. Ένα storyboard αποτελείται από μια σειρά σκηνών, καθένα από τα οποία αντιπροσωπεύει έναν ελεγκτή προβολής και τα view του. Οι ελεγκτές προβολής συνδέονται με αντικείμενα segue, τα οποία αντιπροσωπεύουν μια μετάβαση μεταξύ δύο ελεγκτών προβολής.

Το Xcode παρέχει ένα πρόγραμμα επεξεργασίας για storyboards, όπου γίνεται ο σχεδιασμός του περιβάλλοντος της εφαρμογής προσθέτοντας αντικείμενα του πλαισίου UIKit χωρίς τη χρήση κώδικα. Επιπλέον, ένα storyboard δίνει τη δυνατότητα να συνδυάζεται ένα view με τον ελεγκτή προβολής και διαχειρίζεται τη μεταφορά δεδομένων μεταξύ ελεγκτών προβολής. Η χρήση storyboard είναι ο συνιστάμενος τρόπος για τον σχεδιασμό του περιβάλλοντος χρήστη της εφαρμογής, επειδή επιτρέπει να απεικονίζεται η εμφάνιση και η ροή του περιβάλλοντος χρήστη σε έναν καμβά.

Στο iPhone, κάθε σκηνή αντιστοιχεί σε περιεχόμενο πλήρους οθόνης. στο iPad, μπορούν να εμφανιστούν ταυτόχρονα πολλαπλές σκηνές στην οθόνη - για παράδειγμα, χρησιμοποιώντας ελεγκτές αναδυόμενων προβολών. Κάθε σκηνή έχει μία βάση, η οποία εμφανίζει εικονίδια που αντιπροσωπεύουν τα αντικείμενα ανώτατου επιπέδου της σκηνής. Η βάση χρησιμοποιείται κυρίως για τη δημιουργία συνδέσεων δράσης και εξόδου μεταξύ του ελεγκτή προβολή και των view που έχει. [7]

1.5.2 View Controller

Οι ελεγκτές προβολής “View Controllers” αποτελούν τη βάση της εσωτερικής δομής της εφαρμογής. Κάθε εφαρμογή έχει τουλάχιστον έναν ελεγκτή προβολής και οι περισσότερες εφαρμογές έχουν πολλούς. Κάθε ελεγκτής προβολής διαχειρίζεται ένα τμήμα της διεπαφής χρήστη της εφαρμογής καθώς και τις αλληλεπιδράσεις μεταξύ αυτής της διασύνδεσης και των δεδομένων. Οι ελεγκτές προβολής διευκολύνουν επίσης τις μεταβάσεις μεταξύ διαφορετικών τμημάτων του περιβάλλοντος χρήστη.

Επειδή διαδραματίζουν έναν τόσο σημαντικό ρόλο στην εφαρμογή, οι ελεγκτές προβολής βρίσκονται στο επίκεντρο. Η κλάση UIViewController καθορίζει τις μεθόδους και τις ιδιότητες για τη διαχείριση των μεθόδων, το χειρισμό συμβάντων, τη μετάβαση από έναν ελεγκτή προβολής σε άλλο και το συντονισμό με άλλα μέρη της εφαρμογής.

Υπάρχουν δύο τύποι ελεγκτών προβολής:

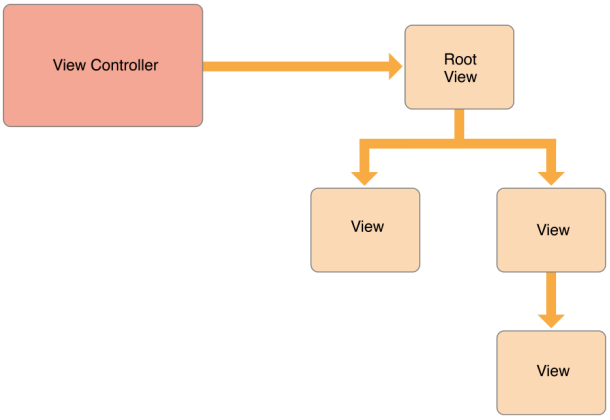
Οι Content View Controller διαχειρίζονται ένα διακριτό κομμάτι του περιεχομένου της εφαρμογής και αποτελούν τον κύριο τύπο ελεγκτή προβολής που δημιουργείται.

Οι Container View Controller συλλέγουν πληροφορίες από άλλους ελεγκτές προβολής (γνωστοί ως child view controllers) και παρουσιάζονται με τρόπο που διευκολύνει την πλοήγηση ή παρουσιάζει διαφορετικά το περιεχόμενο αυτών των ελεγκτών προβολής.

Οι περισσότερες εφαρμογές είναι ένα μείγμα και των δύο τύπων ελεγκτών προβολής. [8]

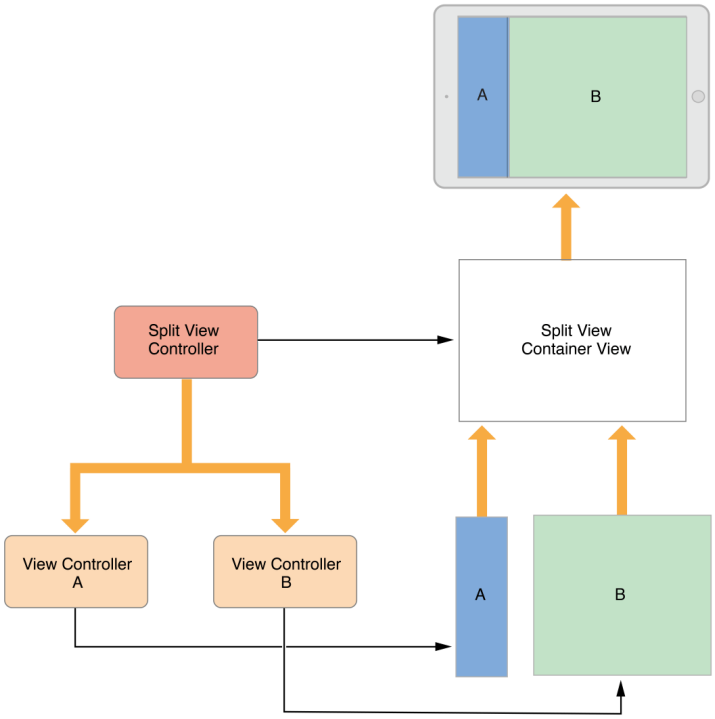
1.5.3 Διαχείριση προβολής

Ο σημαντικότερος ρόλος ενός ελεγκτή προβολής είναι η διαχείριση μιας ιεραρχίας των view. Κάθε ελεγκτής προβολής έχει ένα μοναδικό root view που περικλείει όλο το περιεχόμενο του ελεγκτή. Στη ριζική αυτή προβολή, προστίθενται οι προβολές που χρειάζονται για την εμφάνιση περιεχομένου. Το σχήμα 1.1 απεικονίζει την ενσωματωμένη σχέση μεταξύ του ελεγκτή προβολής και των view του. Ο ελεγκτής προβολής έχει πάντοτε αναφορά στην ριζική του προβολή και κάθε προβολή έχει ισχυρές αναφορές στις υποπροβολές του.



Σχήμα 1.1: Σχέση μεταξύ ενός ελεγκτή προβολής και των view του

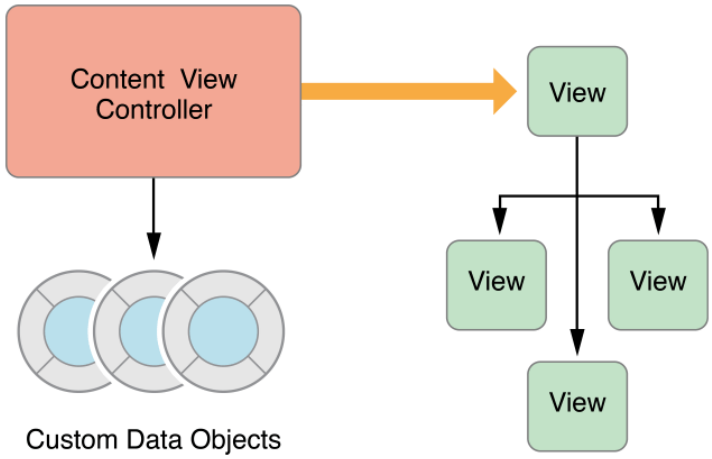
Ένας content view controller διαχειρίζεται όλα τα view του από μόνος του. Ένας container view controller διαχειρίζεται τα δικά του view και root view από έναν ή περισσότερους child view controllers. Το container δεν διαχειρίζεται το περιεχόμενο των παιδιών του. Διαχειρίζεται μόνο τη ριζική όψη, το μέγεθος και την τοποθέτηση σύμφωνα με το σχεδιασμό του container. Το σχήμα 1.2 απεικονίζει τη σχέση μεταξύ ενός split view controller και των παιδιών του. Ο split view controller διαχειρίζεται το συνολικό μέγεθος και τη θέση των παιδικών view του, αλλά οι child view controllers διαχειρίζονται το πραγματικό περιεχόμενο αυτών των προβολών.



Σχήμα 1.2: Οι ελεγκτές προβολής μπορούν να διαχειριστούν περιεχόμενο από άλλους ελεγκτές προβολής

1.5.4 Διαχείριση δεδομένων

Ένας ελεγκτής προβολής ενεργεί ως ενδιάμεσος μεταξύ των view που διαχειρίζεται και των δεδομένων της εφαρμογής. Οι μέθοδοι και οι ιδιότητες της κλάσης UIViewController επιτρέπουν την διαχείριση της οπτικής παρουσίασης της εφαρμογής. Όταν γίνεται υποκλάση της κλάσης UIViewController, προστίθενται οι μεταβλητές που χρειάζονται για τη διαχείριση των δεδομένων της υποκλάσης. Η προσθήκη προσαρμοσμένων μεταβλητών δημιουργεί μια σχέση όπως η εικόνα στο σχήμα 1.3, όπου ο ελεγκτής προβολής έχει αναφορές στα δεδομένα και στις προβολές που χρησιμοποιούνται για την παρουσίαση αυτών των δεδομένων.



Σχήμα 1.3: Ένας ελεγκτή προβολής διαμεσολαβεί μεταξύ Custom Data Object και View

Πρέπει πάντα να διατηρείται ένας καθαρός διαχωρισμός ευθυνών στους ελεγκτές προβολής και τα αντικείμενα δεδομένων. Το μεγαλύτερο μέρος της λογικής για την εξασφάλιση της ακεραιότητας των δομών δεδομένων ανήκει στα ίδια τα αντικείμενα δεδομένων. Ο ελεγκτής προβολής μπορεί να επικυρώσει την είσοδο που προέρχεται από views και στη συνέχεια να πακετάρει εκείνη την είσοδο με τη μορφή που απαιτούν τα αντικείμενα δεδομένων, αλλά πρέπει να ελαχιστοποιείται ο ρόλος του στη διαχείριση των πραγματικών δεδομένων.

Ένα αντικείμενο UIDocument είναι ένας τρόπος για τη διαχείριση των δεδομένων ξεχωριστά από τους ελεγκτές προβολής. Ένα αντικείμενο εγγράφου είναι ένα αντικείμενο ελέγχου που γνωρίζει πώς να διαβάζει και να γράφει δεδομένα σε μόνιμη αποθήκευση. Όταν γίνει υποκλάση αυτού, προστίθεται οποιαδήποτε λογική και μέθοδος που χρειάζεται για να εξαχθούν αυτά τα δεδομένα και να μεταβιβαστούν σε έναν ελεγκτή προβολής ή σε άλλα μέρη της εφαρμογής. Ο ελεγκτής προβολής μπορεί να αποθηκεύσει ένα αντίγραφο των δεδομένων που λαμβάνει για να διευκολύνει την ενημέρωση των προβολών, αλλά το έγγραφο εξακολουθεί να κατέχει τα πραγματικά δεδομένα.

1.5.5 UITabBarController

Η διεπαφή Tab bar εμφανίζει καρτέλες στο κάτω μέρος του παραθύρου για επιλογή μεταξύ των διαφορετικών τρόπων λειτουργίας και εμφάνιση των προβολών για τη συγκεκριμένη λειτουργία. Αυτή η κατηγορία χρησιμοποιείται γενικά ως-είναι, αλλά μπορεί επίσης να είναι υποκλάση.

Κάθε καρτέλα διεπαφής UITabBarController συνδέεται με έναν προσαρμοσμένο ελεγκτή προβολής. Όταν ο χρήστης επιλέξει μια συγκεκριμένη καρτέλα, ο tab bar controller εμφανίζει το root view του αντίστοιχου ελεγκτή προβολής, αντικαθιστώντας οποιεσδήποτε προηγούμενες προβολές. Επειδή η επιλογή μιας καρτέλας αντικαθιστά τα περιεχόμενα της διεπαφής, ο τύπος διεπαφής που διαχειρίζεται σε κάθε η καρτέλα δεν πρέπει να είναι παρόμοιος με κανέναν τρόπο. Στην πραγματικότητα, οι διασυνδέσεις στη γραμμή καρτών χρησιμοποιούνται συνήθως είτε για την παρουσίαση διαφορετικών τύπων πληροφοριών είτε για την παρουσίαση των ίδιων πληροφοριών χρησιμοποιώντας ένα εντελώς διαφορετικό στυλ διεπαφής.[9]

1.5.6 Αλληλεπιδράσεις χρηστών

Οι ελεγκτές προβολής είναι αντικείμενα απόκρισης και είναι σε θέση να χειριστούν συμβάντα που κατεβαίνουν στην αλυσίδα απόκρισης. Παρόλο που είναι σε θέση να το πράξουν, οι ελεγκτές προβολής σπάνια χειρίζονται άμεσα συμβάντα αφής. Αντίθετα, τα views συνήθως χειρίζονται τα δικά τους συμβάντα αφής και αναφέρουν τα αποτελέσματα σε μια μέθοδο ενός συνδεδεμένου αντικειμένου ή ενός αντικειμένου στόχου, που είναι συνήθως ο ελεγκτής προβολής. Επομένως, τα περισσότερα συμβάντα σε έναν ελεγκτή προβολής αντιμετωπίζονται χρησιμοποιώντας μεθόδους ή μεθόδους ανάθεσης.

1.5.7 Διαχείριση πόρων

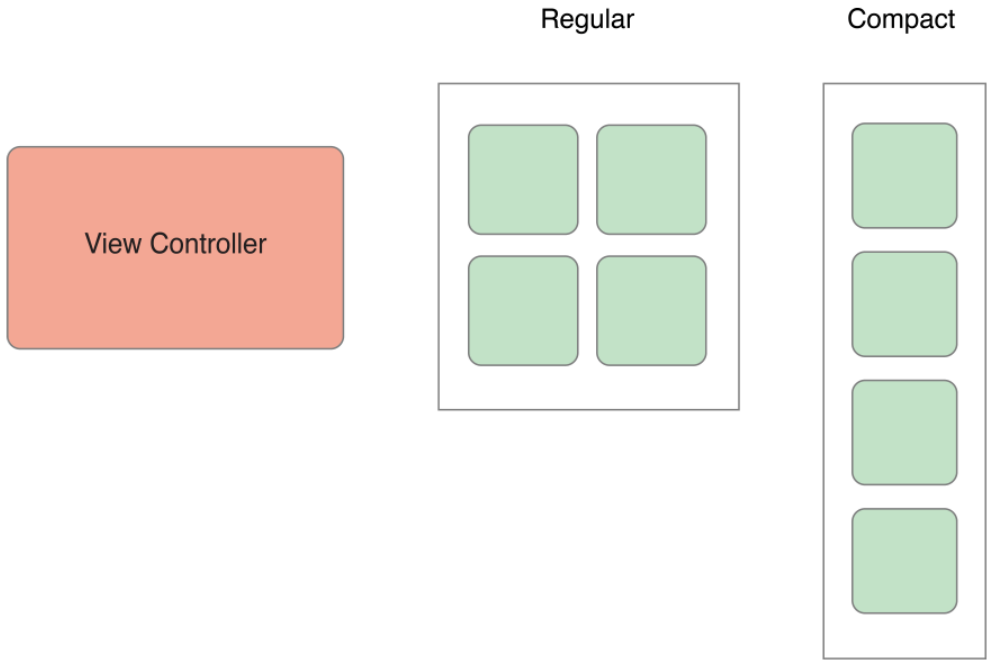
Ένας ελεγκτής προβολής αναλαμβάνει κάθε ευθύνη για τα view και τα αντικείμενα που δημιουργεί. Η κλάση UIViewController χειρίζεται αυτόματα τις περισσότερες πτυχές της διαχείρισης προβολής. Για παράδειγμα, το UIKit απελευθερώνει αυτόματα πόρους που σχετίζονται με την προβολή και δεν χρειάζονται πλέον. Στις υποκλάσεις UIViewController, είναι υπεύθυνος ο developer για τη διαχείριση των αντικειμένων που δημιουργούνται.

Όταν η διαθέσιμη ελεύθερη μνήμη είναι χαμηλή, το UIKit ζητά από τις εφαρμογές να απελευθερώσουν πόρους που δεν χρειάζονται πλέον. Ένας τρόπος με τον οποίο γίνεται αυτό είναι να κληθεί η μέθοδος didReceiveMemoryWarning των ελεγκτών προβολής. Αυτή η μέθοδος χρησιμοποιείται για την κατάργηση αναφορών σε αντικείμενα που δεν χρειάζονται πλέον ή μπορούν να αναδημιουργηθούν εύκολα αργότερα. Για παράδειγμα, γίνεται να χρησιμοποιηθεί αυτή η μέθοδος για την κατάργηση προσωρινών δεδομένων. Είναι σημαντικό να απελευθερωθεί όσο το δυνατόν περισσότερη μνήμη όταν συμβαίνει μια κατάσταση χαμηλής μνήμης. Οι εφαρμογές που καταναλώνουν υπερβολική μνήμη μπορεί να τερματιστούν απότομα από το σύστημα για να ανακτήσουν τη μνήμη.

1.5.8 Προσαρμοστικότητα

Οι ελεγκτές προβολής είναι υπεύθυνοι για την παρουσίαση των view τους και για την προσαρμογή αυτής της παρουσίασης ώστε να ταιριάζουν με το υποκείμενο περιβάλλον. Κάθε εφαρμογή iOS θα πρέπει να μπορεί να λειτουργεί σε iPad και σε πολλά διαφορετικά μεγέθη iPhone. Αντί να παρέχουν διαφορετικούς ελεγκτές προβολής και να προβάλλουν ιεραρχίες για κάθε συσκευή, είναι πιο εύκολο να χρησιμοποιείται ένας ελεγκτής προβολής που προσαρμόζει τα view του στις μεταβαλλόμενες απαιτήσεις χώρου.

Στο iOS, οι ελεγκτές προβολής πρέπει να χειρίζονται coarse-grained αλλαγές και fine-grained αλλαγές. Coarse-grained αλλαγές συμβαίνουν όταν αλλάζουν τα χαρακτηριστικά ενός ελεγκτή προβολής. Τα χαρακτηριστικά περιγράφουν το συνολικό περιβάλλον, όπως η κλίμακα προβολής. Δύο από τα πιο σημαντικά χαρακτηριστικά είναι οι οριζόντιες και κατακόρυφες κλάσεις μεγέθους του ελεγκτή προβολής, οι οποίες δείχνουν πόση απόσταση έχει ο ελεγκτής προβολής στη δεδομένη διάσταση. Μπορούν να χρησιμοποιηθούν αλλαγές κατηγορίας μεγέθους για τον τρόπο με τον οποίο σχεδιάζονται τα Views, όπως φαίνεται στο σχήμα 1.4. Όταν η οριζόντια τάξη μεγέθους είναι κανονική, ο ελεγκτής προβολής εκμεταλλεύεται τον επιπλέον οριζόντιο χώρο για να οργανώσει το περιεχόμενό του. Όταν η οριζόντια κλάση μεγέθους είναι συμπαγής, ο ελεγκτής προβολής οργανώνει το περιεχόμενό του κάθετα.



Σχήμα 1.4: Αλλαγές μεγέθους στα View

1.5.9 Προσαρμογή προβολών σε αλλαγές τάξης μεγέθους

Μέσα σε μια δεδομένη τάξη μεγέθους, είναι πιθανό να εμφανιστούν ανά πάσα στιγμή περισσότερες αλλαγές μεγέθους. Όταν ο χρήστης περιστρέφει ένα iPhone από portrait σε lanscape, η κατηγορία μεγέθους μπορεί να μην αλλάζει, αλλά οι διαστάσεις της οθόνης συνήθως αλλάζουν. Όταν χρησιμοποιείται το Auto Layout, το UIKit προσαρμόζει αυτόματα το μέγεθος και τη θέση των προβολών ώστε να ταιριάζει με τις νέες διαστάσεις. Οι ελεγκτές προβολής μπορούν να κάνουν πρόσθετες ρυθμίσεις ανάλογα με τις ανάγκες.

1.5.10 Auto Layout

Το Auto Layout υπολογίζει δυναμικά το μέγεθος και τη θέση όλων των προβολών στην ιεραρχία του view, βάσει των περιορισμών που έχουν τεθεί. Για παράδειγμα, γίνεται να περιοριστεί ένα κουμπί έτσι ώστε να είναι οριζόντια κεντραρισμένο με μια προβολή εικόνας και έτσι ώστε το πάνω άκρο του κουμπιού να παραμένει πάντα 8 pixel κάτω από το κάτω μέρος της εικόνας. Εάν το μέγεθος ή η θέση της προβολής εικόνων αλλάξει, η θέση του κουμπιού προσαρμόζεται αυτόματα για να ταιριάζει.

Αυτή η προσέγγιση σχεδιασμού βάσει περιορισμών επιτρέπει την δημιουργία διεπαφών χρήστη που ανταποκρίνονται δυναμικά σε εσωτερικές και εξωτερικές αλλαγές. [10]

1.5.11 Foundation

Το πλαίσιο Foundation παρέχει ένα βασικό επίπεδο λειτουργικότητας για εφαρμογές και πλαίσια, όπως αποθήκευση, επεξεργασία κειμένου, υπολογισμοί ημερομηνίας και ώρας, ταξινόμηση, φιλτράρισμα και δικτύωση. Οι κλάσεις, τα πρωτόκολλα και οι τύποι δεδομένων που ορίζονται από το Foundation χρησιμοποιούνται σε όλα τα SDK των macOS, iOS, watchOS και tvOS. [11]

1.5.12 UIView

Τα αντικείμενα της κλάσεις UIView είναι οι θεμελιώδεις δομικές μονάδες της διεπαφής χρήστη μίας εφαρμογής και η κλάση UIView καθορίζει τις συμπεριφορές που είναι κοινές σε όλα τα αντικείμενα UIView καθώς και των υποκλάσεων της. Ένα αντικείμενο UIView καθιστά το περιεχόμενο εντός των ορίων του ορθογωνίου και χειρίζεται οποιεσδήποτε αλληλεπιδράσεις με αυτό το περιεχόμενο. Η κλάση UIView είναι μια συγκεκριμένη κλάση που μπορεί να εμφανίζεται και να χρησιμοποιείται για να εμφανιστεί ένα σταθερό χρώμα φόντου. Μπορεί επίσης να υποκατασταθεί για να σχεδιαστεί πιο περίπλοκο περιεχόμενο. Για την εμφάνιση ετικετών, εικόνων, κουμπιών και άλλων στοιχείων διεπαφής που βρίσκονται συνήθως σε εφαρμογές, χρησιμοποιούνται οι υποκλάσεις της κλάσης UIView που παρέχονται από το πλαίσιο UIKit.

Επειδή τα αντικείμενα UIView είναι ο κύριος τρόπος αλληλεπίδρασης της εφαρμογής με τον χρήστη, έχουν ορισμένες ευθύνες όπως:

<u>Σχεδιασμός και Animation</u>

- Τα αντικείμενα κλάσης UIView σχεδιάζουν περιεχόμενο στην ορθογώνια περιοχή τους χρησιμοποιώντας το UIKit ή Core Graphics.
- Ορισμένες ιδιότητες μπορούν να γίνονται animated σε νέες τιμές.

<u>Διαχείριση διάταξης και των subviews</u>

- Οι προβολές ενδέχεται να περιέχουν και περισσότερες υπό προβολές.
- Οι προβολές μπορούν να προσαρμόσουν το μέγεθος και τη θέση των υπό προβολών τους.
- Χρησιμοποιείται το Auto Layout για να οριστούν οι κανόνες για την αλλαγή μεγέθους και την επανατοποθέτηση των προβολών σε σχέση με αλλαγές στην ιεραρχία των προβολών.

<u>Διαχείριση συμβάντων</u>

- Μία προβολή είναι μια υποκλάση της κλάσης UIResponder και μπορεί να ανταποκριθεί σε αγγίγματα στην οθόνη και άλλα είδη συμβάντων.
- Οι προβολές μπορούν να εγκαταστήσουν αναγνωριστές χειρονομιών για να χειριστούν τις κοινές χειρονομίες.

Οι προβολές μπορούν να ενσωματωθούν μέσα σε άλλες προβολές για να δημιουργηθούν ιεραρχίες προβολών, οι οποίες προσφέρουν έναν βολικό τρόπο για την οργάνωση σχετικού περιεχομένου. Η

ανίχνευση μίας προβολής δημιουργεί μια σχέση γονέα-παιδιού μεταξύ της εμφάνισης παιδιού που είναι ενωμένη (subview) και του γονέα (superview). Μια γονική προβολή μπορεί να περιέχει οποιονδήποτε αριθμό υπό προβολών αλλά κάθε υποπροβολή έχει μόνο ένα γονέα. Από προεπιλογή, όταν η ορατή περιοχή μίας υποπροβολής εκτείνεται εκτός των ορίων του, δεν υπάρχει αποκοπή του περιεχομένου της υποπροβολής. Υπάρχει η δυνατότητα να χρησιμοποιηθεί η ιδιότητα clipsToBounds για να αλλάξει η συμπεριφορά αυτή.

Η γεωμετρία κάθε προβολής ορίζεται από τις ιδιότητες frame και bounds. Η ιδιότητα frame ορίζει την προέλευση και τις διαστάσεις της προβολής στο σύστημα συντεταγμένων της προβολής γονέα του. Η ιδιότητα bounds ορίζει τις εσωτερικές διαστάσεις του view. Η ιδιότητα center παρέχει έναν βολικό τρόπο για να επανατοποθετείται μια προβολή χωρίς να αλλάζει απευθείας η ιδιότητα frame ή bounds. [12]

1.5.13 Animations

Τα Animation παρέχουν οπτικές μεταβάσεις μεταξύ διαφορετικών καταστάσεων της διεπαφής χρήστη. Στο iOS, οι κινούμενες εικόνες χρησιμοποιούνται εκτεταμένα για να επανατοποθετήσουν τις προβολές, να αλλάξουν το μέγεθός τους, να τις αφαιρέσουν από ιεραρχίες προβολής και να τις αποκρύψουν. Μπορούν να χρησιμοποιηθούν κινούμενα σχέδια για την εμφάνιση σχολίων προς το χρήστη ή για την εφαρμογή οπτικών εφέ.

Στο iOS, η δημιουργία animation δεν απαιτεί γραφή κώδικα σχεδίασης. Όλες οι τεχνικές κινούμενης εικόνας που χρησιμοποιούν την ενσωματωμένη υποστήριξη που παρέχεται από το Core Animation. Το μόνο που χρειάζεται είναι να ενεργοποιείται το animation και το Core Animation διαχειρίζεται την απόδοση μεμονωμένων frame. Αυτό κάνει πολύ εύκολη τη δημιουργία έξυπνων κινούμενων εικόνων με λίγες μόνο γραμμές κώδικα.

Και το UIKit και το Core Animation παρέχουν υποστήριξη για animation, αλλά το επίπεδο υποστήριξης που παρέχεται από κάθε τεχνολογία ποικίλλει. Στο UIKit, τα animation πραγματοποιούνται χρησιμοποιώντας αντικείμενα UIView. Τα αντικείμενα UIView υποστηρίζουν ένα βασικό σύνολο κινούμενων σχεδίων που καλύπτουν πολλές κοινές εργασίες. Για παράδειγμα, μπορούν να γίνουν αλλαγές στις ιδιότητες των προβολών ή να χρησιμοποιηθούν animation για την αντικατάσταση ενός συνόλου από προβολές με ένα άλλο.

1.6 Επίλογος

Αυτό το κεφάλαιο προσφέρει τις απαραίτητες γνώσεις που χρειάζεται κάποιος developer για να καταλάβει πως γίνεται ο προγραμματισμός της εφαρμογής που θα εξηγηθεί παρακάτω καθώς και κάθε βασική γνώση που χρειάζεται κάποιος για να κατανοήσει τον τρόπο με τον οποίο προγραμματίζονται οι εφαρμογές για συσκευές με λειτουργικό σύστημα iOS.

Κεφάλαιο 2ο: Εργαλεία βελτιστοποίησης εμπειρίας

2.1 Εισαγωγή

Για την δημιουργία της τελικής εφαρμογής χρησιμοποιήθηκαν εργαλεία, κάποια από τα οποία ήταν απαραίτητα για την ολοκλήρωση της εφαρμογής και κάποια από τα οποία προσθέτουν κάποια χαρακτηριστικά για καλύτερη εμπειρία χρήστη. Τα εργαλεία αυτά είναι το Firebase, τα Cocoapods, η χρήση εντοπισμού γλώσσας χρήστη, τα Core Data, τα User Defaults, το MessageUI και το Adobe Photoshop

2.2 Firebase

Η Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών για κινητά και web που αναπτύχθηκε από την Firebase, Inc. το 2011, και στη συνέχεια εξαγοράστηκε από την Google το 2014. Από τον Οκτώβριο του 2018, η πλατφόρμα Firebase διαθέτει 18 προϊόντα που χρησιμοποιούνται από περισσότερες από 1,5 εκατομμύρια εφαρμογές.

Παρέχει στους προγραμματιστές ένα API που επιτρέπει την ενσωμάτωση της online δυνατότητας συνομιλίας στις ιστοσελίδες τους. Μετά την έκδοση της υπηρεσίας συνομιλίας, οι Tamplin και Lee διαπίστωσαν ότι χρησιμοποιείται για να διαβιβάζει δεδομένα εφαρμογών που δεν ήταν μηνύματα συνομιλίας. Οι προγραμματιστές χρησιμοποίησαν το Envolve για να συγχρονίσουν δεδομένα εφαρμογών, όπως η κατάσταση του παιχνιδιού, σε πραγματικό χρόνο στους χρήστες τους. Οι Tamplin και Lee αποφάσισαν να χωρίσουν το σύστημα συνομιλίας και την αρχιτεκτονική πραγματικού χρόνου που την τροφοδοτούσαν. Ίδρυσαν τη Firebase ως ξεχωριστή εταιρεία τον Απρίλιο του 2012. [14]

Firebase Υπηρεσίες

•Analytics

Το Firebase Analytics είναι μια δωρεάν λύση μέτρησης εφαρμογών που παρέχει πληροφορίες για τη χρήση της εφαρμογής και την αφοσίωση του χρήστη.

•Firebase Cloud Messaging

Παλαιότερα γνωστό ως Google Cloud Messaging (GCM), το Firebase Cloud Messaging (FCM) είναι μια λύση μεταξύ των πλατφόρμων για μηνύματα και ειδοποιήσεις για εφαρμογές Android, iOS και web, οι οποίες από το 2016 μπορούν να χρησιμοποιηθούν χωρίς κόστος

•Firebase Auth

Το Firebase Auth είναι μια υπηρεσία που μπορεί να πιστοποιήσει τους χρήστες χρησιμοποιώντας μόνο τον κωδικό πελάτη. Υποστηρίζει παροχή κοινωνικής σύνδεσης Facebook, GitHub, Twitter και Google (και Παιχνίδια Google Play). Επιπλέον, περιλαμβάνει ένα σύστημα διαχείρισης χρηστών, στο οποίο οι προγραμματιστές μπορούν να ενεργοποιήσουν τον έλεγχο ταυτότητας χρήστη με σύνδεση μέσω ηλεκτρονικού ταχυδρομείου και κωδικού πρόσβασης που είναι αποθηκευμένη με το Firebase.

•Realtime Database

Το Firebase παρέχει μια βάση δεδομένων σε πραγματικό χρόνο ως υπηρεσία. Η υπηρεσία παρέχει στους προγραμματιστές εφαρμογών ένα API που επιτρέπει στα δεδομένα εφαρμογών να συγχρονίζονται μεταξύ των πελατών και να αποθηκεύονται στο cloud του Firebase. Η εταιρεία παρέχει βιβλιοθήκες πελατών που επιτρέπουν την ενσωμάτωση με εφαρμογές Android, iOS, JavaScript, Java, Objective-C, Swift και Node.js. Η βάση δεδομένων είναι επίσης προσβάσιμη μέσω ενός API REST και συνδέσεων για διάφορα πλαίσια JavaScript, όπως AngularJS, React, Ember.js και Backbone.js. Το API REST χρησιμοποιεί το πρωτόκολλο Server-Sent Events, το οποίο είναι ένα API για τη δημιουργία συνδέσεων HTTP και τη λήψη push notifications από ένα διακομιστή. Οι προγραμματιστές που χρησιμοποιούν τη βάση δεδομένων σε πραγματικό χρόνο μπορούν να εξασφαλίσουν τα δεδομένα τους, χρησιμοποιώντας τους κανόνες ασφαλείας που εφαρμόζει η εταιρεία από πλευράς διακομιστή.

•Firebase Storage

Το Firebase Storage παρέχει ασφαλή updload και download αρχείων για εφαρμογές Firebase, ανεξάρτητα από την ποιότητα του δικτύου. Ο προγραμματιστής μπορεί να το χρησιμοποιήσει για την αποθήκευση εικόνων, ήχου, βίντεο ή άλλου περιεχομένου που δημιουργεί ο χρήστης. Η αποθήκευση Firebase υποστηρίζεται από το Google Cloud Storage.

•Firebase Hosting

Το Firebase Hosting είναι μια στατική και δυναμική υπηρεσία φιλοξενίας ιστοσελίδων που ξεκίνησε στις 13 Μαΐου 2014. Υποστηρίζει φιλοξενία στατικών αρχείων όπως CSS, HTML, JavaScript και άλλα αρχεία, καθώς και υποστήριξη μέσω των λειτουργιών Cloud. Η υπηρεσία παραδίδει αρχεία μέσω δικτύου παροχής περιεχομένου (CDN) μέσω της κρυπτογράφησης HTTP Secure (HTTPS) και της κρυπτογράφησης Secure Sockets Layer (SSL). Η εταιρεία Firebase συνεργάζεται με το Fast, ένα CDN, για να παρέχει το υποστηρικτικό CDN Firebase Hosting. Η εταιρεία δηλώνει ότι η Firebase Hosting εξελίχθηκε από αιτήματα πελατών. Οι προγραμματιστές χρησιμοποιούσαν τη βάση δεδομένων Firebase για ανάκτηση δεδομένων σε πραγματικό χρόνο, αλλά χρειάζονταν ένα μέρος για να φιλοξενήσουν το περιεχόμενό τους.

•ML Kit

Το ML Kit είναι ένα mobile σύστημα machine learning για προγραμματιστές που ξεκίνησε στις 8 Μαΐου 2018 σε beta κατά τη διάρκεια του Google I / O 2018. Τα εργαλεία AP API του ML Kit διαθέτουν μια ποικιλία χαρακτηριστικών όπως αναγνώριση κειμένου, ανίχνευση προσώπων, σάρωση γραμμικών κωδικών, ετικετών εικόνων και αναγνώριση ορόσημων. Αυτήν τη στιγμή διατίθεται για προγραμματιστές iOS ή Android. Επίσης, μπορούν να εισαχθούν μοντέλα TensorFlow Lite, αν τα API δεν είναι αρκετά. Τα API μπορούν να χρησιμοποιηθούν σε συσκευή ή σε cloud.

•Crashlytics

Τα Crashlytics δημιουργούν λεπτομερείς αναφορές για τα σφάλματα στην εφαρμογή. Τα σφάλματα ομαδοποιούνται σε ομάδες παρόμοιων ιχνών στοίβας και ταξινομούνται βάσει της σοβαρότητας των επιπτώσεων στους χρήστες εφαρμογών. Εκτός από τις αυτόματες αναφορές, ο προγραμματιστής μπορεί να καταγράφει προσαρμοσμένα συμβάντα για να βοηθήσει να καταγράψει τα βήματα που οδηγούν σε αναγκαστικό τερματισμό της εφαρμογής.

•Performance

Το Firebase Performance παρέχει πληροφορίες σχετικά με την απόδοση μιας εφαρμογής και τις λανθάνουσες περιόδους λειτουργίας των χρηστών της εφαρμογής.

•Firebase Test Lab για Android και iOS

Το Firebase Test Lab για Android και iOS παρέχει υποδομή βασισμένη σε cloud για δοκιμή εφαρμογών Android και iOS. Με μία ενέργεια, οι προγραμματιστές μπορούν να ξεκινήσουν τη δοκιμή των εφαρμογών τους σε μια μεγάλη ποικιλία συσκευών. Τα αποτελέσματα των δοκιμών - συμπεριλαμβανομένων των αρχείων καταγραφής, των βίντεο και των στιγμιότυπων οθόνης - διατίθενται στο έργο στην κονσόλα Firebase. Ακόμα κι αν ένας προγραμματιστής δεν έχει γράψει κανένα κώδικα δοκιμής για την εφαρμογή του, το Lab Test μπορεί να τρέξει αυτόματα την εφαρμογή, αναζητώντας crashes. Το Test Lab για iOS βρίσκεται αυτή τη στιγμή σε beta.

•Firebase Ειδοποιήσεις

Η ειδοποίηση Firebase είναι μια υπηρεσία που επιτρέπει ειδοποιήσεις στοχευμένων χρηστών για προγραμματιστές εφαρμογών για κινητά χωρίς κόστος.

2.3 Cocoapods

Το CocoaPods είναι ένας διαχειριστής εξάρτησης επιπέδου εφαρμογών για τις γλώσσες Objective-C, Swift και οποιεσδήποτε άλλες γλώσσες που εκτελούνται στο χρόνο εκτέλεσης της Objective-C, όπως το RubyMotion, το οποίο παρέχει μια τυπική μορφή διαχείρισης εξωτερικών βιβλιοθηκών. Έχει πάνω από 56.000 βιβλιοθήκες και χρησιμοποιείται σε πάνω από 3 εκατομμύρια εφαρμογές [16]. Αναπτύχθηκε από τους Eloy Durán και Fabio Pelosin, οι οποίοι συνεχίζουν να διαχειρίζονται το έργο με τη βοήθεια και τη συμβολή πολλών άλλων. Ξεκίνησαν την ανάπτυξη τον Αύγουστο του 2011 και πραγματοποίησαν την πρώτη δημόσια κυκλοφορία την 1η Σεπτεμβρίου 2011.

Το CocoaPods επικεντρώνεται στην κατανομή των βιβλιοθηκών τρίτων με βάση την πηγή και στην αυτόματη ενσωμάτωση σε έργα Xcode. Το CocoaPods είναι χτισμένο με Ruby και εγκαθίσταται με τη προεπιλεγμένη Ruby διαθέσιμη στο macOS. Συνιστάται να χρησιμοποιείται η προεπιλεγμένη ruby.

Το CocoaPods τρέχει από τη γραμμή εντολών και είναι επίσης ενσωματωμένο στο ολοκληρωμένο περιβάλλον ανάπτυξης του AppCode του JetBrains. Εγκαθιστά εξαρτήσεις για μια εφαρμογή με εξειδίκευση εξαρτήσεων αντί για χειροκίνητη αντιγραφή αρχείων προέλευσης. Εκτός από την εγκατάσταση πολλών διαφορετικών βιβλιοθηκών, παρέχονται επίσης τα master repositories για πολλές βιβλιοθήκες Open Source και διατηρούνται αποθηκευμένα στο git και φιλοξενούνται στο

GitHub. Το σύστημα ανάλυσης εξάρτησης CocoaPods τροφοδοτείται από το Molinillo, το οποίο χρησιμοποιείται επίσης από άλλα μεγάλα έργα όπως τα Bundler, RubyGems και Berkshelf. [15]

2.4 Internationalization and Localization

Ο εντοπισμός της περιοχής είναι η διαδικασία μετάφρασης της εφαρμογής σε πολλές γλώσσες. Αλλά προτού μπορέσει να εντοπιστεί η περιοχή της εφαρμογής πρέπει πρώτα να υποστηρίζει διεθνοποίηση. Η διεθνοποίηση είναι η διαδικασία της προσαρμογής της εφαρμογής σε διαφορετικές γλώσσες, περιοχές και πολιτισμούς. Επειδή μια ενιαία γλώσσα μπορεί να χρησιμοποιηθεί σε πολλά μέρη του κόσμου, η εφαρμογή πρέπει να προσαρμοστεί στις περιφερειακές και πολιτιστικές συμβάσεις του τόπου κατοικίας ενός ατόμου. Μια διεθνοποιημένη εφαρμογή εμφανίζεται σαν να είναι μια εγγενής εφαρμογή σε όλες τις γλώσσες και περιοχές που υποστηρίζει.

Το App Store είναι διαθέσιμο σε περισσότερες από 150 διαφορετικές χώρες και η διεθνοποίηση της εφαρμογής είναι το πρώτο βήμα για την επίτευξη αυτής της παγκόσμιας αγοράς. Στο App Store Connect, καθορίζεται αν η εφαρμογή είναι διαθέσιμη σε όλες τις περιοχές ή συγκεκριμένες περιοχές. Οι χρήστες σε άλλες χώρες μπορούν να χρησιμοποιήσουν την εφαρμογή σε οποιαδήποτε από τις διαθέσιμες της εφαρμογής.

Το Xcode παρέχει ένα εργαλείο για την δημιουργία διαφορετικών λεκτικών για κάθε γλώσσα που θέλει ο προγραμματιστής να υποστηρίζει η εφαρμογή του. Πρώτα διεθνοποιείται η διεπαφή χρήστη και ο κώδικας κατά την ανάπτυξη. Αυτό έχει σαν αποτέλεσμα την δημιουργία αρχείων τόσα όσες και οι γλώσσες που υποστηρίζει η εφαρμογή. Σε κάθε αρχείο κάθε γραμμή υποστηρίζει την μετάφραση ενός μοναδικού κλειδιού, για παράδειγμα «“key_helloWorld” = “Καλημέρα κόσμε”» . Κάθε κλειδί που δημιουργείται πρέπει να υπάρχει σε όλα τα αρχεία γλωσσών. Ανάλογα με την γλώσσα που παίρνει η εφαρμογή κατά την εκτέλεση διαβάζεται και η αντίστοιχη μετάφραση για να παρουσιαστεί στον χρήστη. Στον κώδικα αυτό γράφεται σαν « “Lebel.text = Localized(“key_helloWorld ”)». Με αυτόν τον τρόπο η εφαρμογή μεταφράζεται σε διαφορετικές γλώσσες. [16]

2.5 Core Data

Τα Core Data είναι ένα πλαίσιο που χρησιμοποιείται για τη διαχείριση των αντικειμένων του στρώματος μοντέλου της εφαρμογής. Παρέχει γενικευμένες και αυτοματοποιημένες λύσεις σε κοινές εργασίες που σχετίζονται με τον κύκλο ζωής αντικειμένων και τη διαχείριση γραφικών αντικειμένων. [18]

Τα Core Data συνήθως μειώνουν κατά 50 έως 70% του ποσού του κώδικα που γράφεται για την υποστήριξη του επιπέδου μοντέλου. Αυτό οφείλεται κυρίως στις ακόλουθες ενσωματωμένες λειτουργίες που δεν χρειάζεται να εφαρμόζονται, να δοκιμάζονται ή να βελτιστοποιούνται:

•Αλλαγή της παρακολούθησης και της ενσωματωμένης διαχείρισης της αναίρεσης.

•Διατήρηση της διάδοσης των αλλαγών, συμπεριλαμβανομένης της διατήρησης της συνέπειας των σχέσεων μεταξύ αντικειμένων.

•Λανθασμένη φόρτωση αντικειμένων, μερική υλοποίηση συμβολαίων μελλοντικής εκπλήρωσης (σφάλμα) και ανταλλαγή δεδομένων αντιγραφής-εγγραφής για μείωση των γενικών εξόδων.

•Αυτόματη επικύρωση των τιμών ιδιοκτησίας. Τα διαχειριζόμενα αντικείμενα επεκτείνουν τις τυπικές μεθόδους επικύρωσης κωδικοποίησης-κλειδιού για να εξασφαλίσουν ότι οι μεμονωμένες τιμές βρίσκονται μέσα σε αποδεκτές περιοχές, έτσι ώστε οι συνδυασμοί τιμών να έχουν νόημα.

•Εργαλεία μετεγκατάστασης σχήματος που απλοποιούν τις αλλαγές σχήματος και επιτρέπουν την εκτέλεση αποτελεσματικής μετανάστευσης σχήματος επιτόπου.

•Προαιρετική ενσωμάτωση με το επίπεδο ελεγκτή της εφαρμογής για υποστήριξη του συγχρονισμού διεπαφής χρήστη.

•Ομαδοποίηση, φιλτράρισμα και οργάνωση δεδομένων στη μνήμη και στο περιβάλλον χρήστη.

•Αυτόματη υποστήριξη για την αποθήκευση αντικειμένων σε αποθήκες εξωτερικών δεδομένων.

•Σύνθετη συλλογή ερωτημάτων. Εναλλακτικός τρόπος γραφής SQL, υποστήριξη δημιουργίας πολύπλοκων ερωτημάτων, συσχετίζοντας ένα αντικείμενο της κλάσης NSPredicate με ένα αίτημα λήψης.

•Παρακολούθηση της έκδοσης και ασφάλεια για την υποστήριξη της αυτόματης επίλυσης συγκρούσεων πολλών εγγράφων.

•Αποτελεσματική ενσωμάτωση με τις αλυσίδες εργαλείων macOS και iOS. [17]

2.6 Adobe Photoshop

Το Adobe Photoshop είναι ένας επεξεργαστής γραφικών raster που αναπτύχθηκε και δημοσιεύθηκε από την Adobe Inc. για macOS και Windows.

Το πρόγραμμα δημιουργήθηκε το 1988 από τους Thomas και John Knoll. Από τότε, έχει γίνει το πιο γνωστό και πιο πολυχρησιμοποιημένο βιομηχανικό πρότυπο στην επεξεργασία raster γραφικών. Μπορεί να επεξεργαστεί και να συνθέσει εικόνες raster σε πολλαπλά στρώματα και να υποστηρίξει μάσκες, σύνθετα alpha και διάφορα έγχρωμα μοντέλα, όπως RGB, CMYK, CIELAB, spot color και duotone. Το Photoshop χρησιμοποιεί τις δικές του μορφές αρχείων PSD και PSB για να υποστηρίξει αυτές τις λειτουργίες.

Εκτός από τα ράστερ γραφικά, έχει περιορισμένες δυνατότητες επεξεργασίας ή εκτύπωσης κειμένου, διανυσματικών γραφικών (ειδικά μέσω διαδρομής αποκοπής), 3D γραφικών και βίντεο. Το σύνολο χαρακτηριστικών του προγράμματος μπορεί να επεκταθεί με plug-ins, προγράμματα που αναπτύσσονται και διανέμονται ανεξάρτητα από το ίδιο και μπορούν να λειτουργήσουν μέσα σε αυτό και να προσφέρουν νέες ή βελτιωμένες λειτουργίες.

Το σχήμα ονομασίας του Photoshop βασίστηκε αρχικά σε αριθμούς έκδοσης. Ωστόσο, τον Οκτώβριο του 2002 κάθε νέα έκδοση του Photoshop χαρακτηρίστηκε με "CS" συν ένα αριθμό. Τον Ιούνιο του 2013, με την εισαγωγή του Creative Cloud, το πρόγραμμα αδειοδότησης του Photoshop άλλαξε σε αυτό του λογισμικού ως μοντέλο μίσθωσης υπηρεσίας και τα επιθέματα "CS" αντικαταστάθηκαν με το "CC". Ιστορικά, το Photoshop συνδυάστηκε με πρόσθετο λογισμικό όπως το Adobe ImageReady, το Adobe Fireworks, το Adobe Bridge, το Adobe Device Central και το Adobe Camera RAW.

2.7 Επίλογος

Η τελική εφαρμογή που δημιουργήθηκε έπρεπε να είναι όσο το δυνατόν πιο επίκαιρη στη σύγχρονη εποχή και για την επίτευξη αυτού δεν αρκούσε μόνο η χρήση των framework της apple έπρεπε να ενσωματωθούν κι άλλα εργαλεία. Όλα ήταν εξίσου σημαντικά.

Κεφάλαιο 3ο: Εφαρμογή

3.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται η εφαρμογή. Τα παιχνίδια που έχει η εφαρμογή. Η δομή της, ο τρόπος που λειτουργεί και ο κώδικας της. Περιγράφονται αναλυτικά οι κλάσεις και η χρήση τους.

3.2 Τα παιχνίδια

Στην εφαρμογή υπάρχουν τρία παιχνίδια από τα οποία μπορείς να επιλέξεις. Όλα τους είναι παιχνίδια ρόλων και σκοπός τις εφαρμογής είναι να καθορίσει τους ρόλους ανάλογα το πλήθος των παικτών και να εξηγήσει στον κάθε παίκτη ξεχωριστά τον ρόλο και τον σκοπό που έχει σε κάθε γύρο.

3.2.1 Παλέρμο

Το Παλέρμο είναι το πρώτο παιχνίδι της εφαρμογής βασισμένο στο παιχνίδι «μια νύχτα στο παλέρμο», όπου ο κάθε παίκτης έχει έναν κρυφό ξεχωριστό ρόλο με διαφορετικό κίνητρο. Οι βασικοί χαρακτήρες του παιχνιδιού είναι οι δολοφόνοι, ο αστυνόμος και οι πολίτες. Οι δολοφόνοι κάθε νύχτα που «πέφτει» στο Παλέρμο (φάση της Νύχτας) και ενώ όλοι οι παίκτες έχουν τα μάτια κλειστά, δολοφονούν έναν από τους πολίτες. Σκοπός της υπόλοιπης ομάδας είναι να βρουν ποιοι είναι οι δολοφόνοι και να τους διώξουν από την πόλη μέσω ψηφοφορίας (φάση της ημέρας) πριν πεθάνουν όλοι. Το ενδιαφέρων στο παιχνίδι αυτό είναι ότι οι παίκτες δεν γνωρίζουν τους ρόλους των υπόλοιπων, εκτός από κάποιες εξαιρέσεις, και στην φάση της ψηφοφορίας πρέπει να υπερασπιστούν τον εαυτό τους αλλά και την ψήφο τους στους υπόλοιπους. Τα μπερδέματα και οι αλληλοκατηγορίες δίνουν και παίρνουν σε αυτό το συναρπαστικό παιχνίδι παρέας. Στο βασικό παιχνίδι έχουν προστεθεί επιπλέον ρόλοι αντί για τους απλούς πολίτες με αποτέλεσμα να αποκτά περισσότερο ενδιαφέρον για μεγαλύτερη αλληλεπίδραση των παικτών στην φάση της ημέρας.

Χαρακτήρες - Ρόλοι:

Ο κόκκινος κλέφτης γνωρίζει τον μαύρο κλέφτη. Σκοπός του είναι μαζί με τον μαύρο κλέφτη να μείνουν ζωντανοί μέχρι το τέλος. Τον κόκκινο (φανερό) κλέφτη τον γνωρίζει και το καρφί. Ο κόκκινος κλέφτης ΔΕΝ γνωρίζει το καρφί.

Ο μαύρος κλέφτης γνωρίζει τον κόκκινο κλέφτη. Σκοπός του είναι μαζί με τον κόκκινο κλέφτη να μείνουν ζωντανοί μέχρι το τέλος.

Το καρφί γνωρίζει ποιος είναι ο κόκκινος κλέφτης και τον γνωρίζει ο αστυνόμος. Σκοπός του είναι να μείνουν ζωντανοί οι κλέφτες.

Ο αστυνόμος γνωρίζει το καρφί. Σκοπός του είναι να καταλάβει από τις απόψεις που έχει το καρφί ποιοί είναι οι κλέφτες.

Ο γιατρός έχει την δυνατότητα να κρατάει ζωντανό έναν τυχαίο παίχτη το βράδυ. Κερδίζει όταν οι κλέφτες βγουν από το παιχνίδι.

Οι συγκάτοικοι γνωρίζονται μεταξύ τους. Κερδίζουν το παιχνίδι όταν οι κλέφτες βγουν από το παιχνίδι.

Αν ο κόκκινος καμικάζι ψηφιστεί το πρωί, πρέπει να σκοτώσει έναν τυχαίο παίχτη της επιλογής του. Κερδίζει όταν οι κλέφτες βγουν από το παιχνίδι.

Αν ο μαύρος καμικάζι πεθάνει το βράδυ πρέπει να πάρει κάποιον μαζί του. Κερδίζει όταν οι κλέφτες βγουν από το παιχνίδι.

Οι πολίτες κερδίζουν όταν οι κλέφτες βγουν από το παιχνίδι.

3.2.2 Κατάσκοπος

Το επόμενο παιχνίδι ονομάζεται Κατάσκοπος. Όλοι οι παίκτες παίρνουν έναν κρυφό ρόλο. Όλοι οι ρόλοι εκτός από τους κατάσκοπους γνωρίζουν την περιοχή που βρίσκονται. Για ένα συγκεκριμένο χρονικό διάστημα (10 - 20 λεπτά), οι παίκτες κάνουν ο ένας στον άλλο εναλλάξ ερωτήσεις που μπορούν να απαντηθούν μονολεκτικά. Ο ερωτώμενος απαντά και ρωτάει αμέσως μετά έναν άλλο συμπαίκτη του μια διαφορετική ερώτηση. Οι πολίτες πρέπει να προσδιορίσουν τον κατάσκοπο που δεν έχει ιδέα ποια είναι η τοποθεσία και ο κατάσκοπος πρέπει να προσδιορίσει την τοποθεσία.

Παράδειγμα: Η τοποθεσία είναι ο ζωολογικός κήπος. Εάν ερωτηθεί ένας παίκτης αν θα συναντούσε ζώα εκεί μέσα η απάντηση για τους πολίτες είναι προφανείς. Για τον κατάσκοπο, που δεν γνωρίζει που βρίσκεται, όχι. Αλλά απαντώντας ναι ή όχι έχει πιθανότητες να κρυφτεί και συνεχίζοντας τις ερωτήσεις να καταλάβει που μπορεί να βρίσκεται.

Εάν οι κατάσκοποι μέχρι το τέλος του χρόνου πιστεύουν ότι ξέρουν την τοποθεσία, αποκαλύπτονται και αν είναι σωστοί κερδίζουν το παιχνίδι. Εάν δεν είναι σίγουροι οι πολίτες έχουν την ευκαιρία να εντοπίσουν τους κατασκόπους που λογικά έχουν απαντήσει λάθος σε ορισμένες από τις ερωτήσεις που τους έγιναν. Εάν οι πολίτες αναγνωρίσουν τους κατασκόπους κερδίζουν.

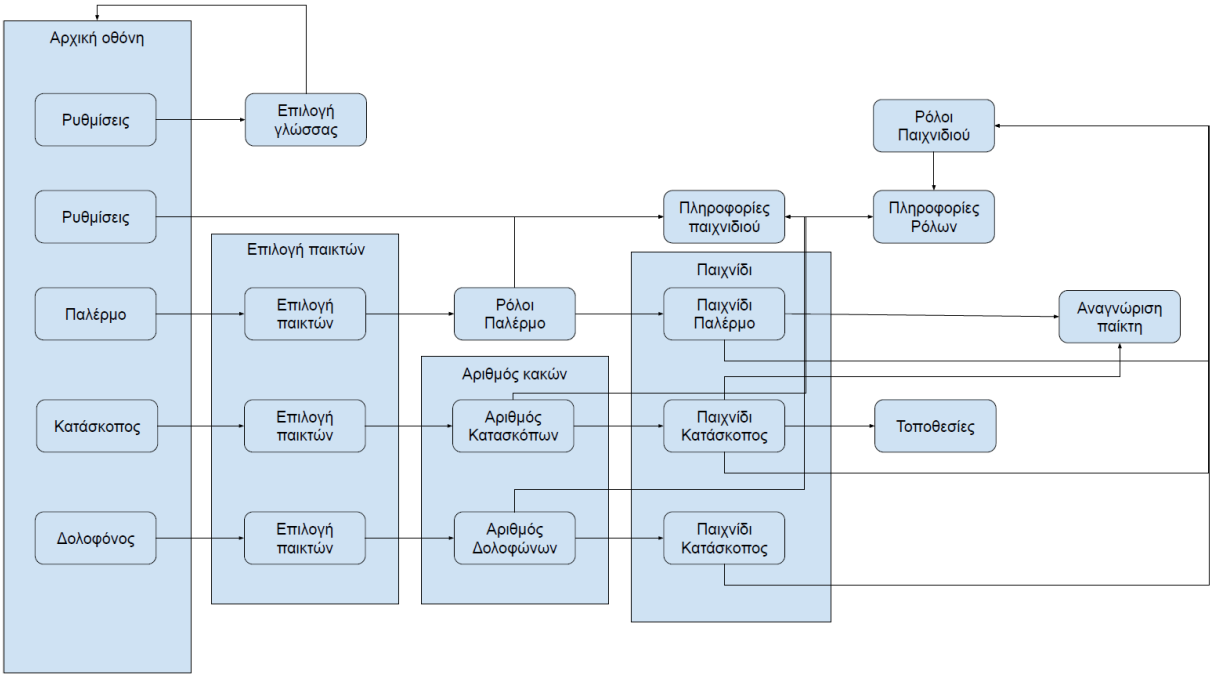
3.2.3 Δολοφόνος

Το τελευταίο παιχνίδι λέγεται Δολοφόνος. Είναι το πιο απλό και γρήγορο από τα παιχνίδια της εφαρμογής. Όλοι οι παίκτες παίρνουν έναν κρυφό ρόλο, πολίτη αστυνόμο ή δολοφόνο. Οι παίκτες κάθονται σε κύκλο ώστε να υπάρχει οπτική επαφή μεταξύ όλων. Ο δολοφόνος μπορεί να σκοτώσει τους πολίτες με όπλο το κλείσιμο του ματιού του. Κάθε φορά που ο δολοφόνος κλείνει το μάτι σε έναν πολίτη, ο πολίτης πεθαίνει έπειτα από 5 δευτερόλεπτα ανακοινώνοντας τον θάνατο του σε όλους. Ο δολοφόνος γνωρίζει ποιος είναι ο αστυνόμος και πρέπει να σκοτώσει όλους τους πολίτες χωρίς να τον αντιληφθεί ο αστυνόμος. Ο αστυνόμος πρέπει να πιάσει τον δολοφόνο και να σώσει τους πολίτες από βέβαιο θάνατο.

3.3 Λειτουργία της εφαρμογής

Ανοίγοντας την εφαρμογή βλέπουμε στο πρώτο παράθυρο τα τρία παιχνίδια από τα οποία μπορούμε να επιλέξουμε (Εικόνα 3.1), καθώς και το κουμπί των ρυθμίσεων (Εικόνα 3.2). Επιλέγοντας ένα από τα παιχνίδια, στο επόμενο παράθυρο θα ζητηθεί να προστεθούν οι παίκτες και να ορίσουν τα ονόματα τους (Εικόνα 3.1). Αναλόγως το παιχνίδι θα πρέπει να γίνει επιλογή αριθμού κατασκόπων, αριθμού δολοφόνων (Εικόνα 3.5) ή ρόλων (Εικόνα 3.4). Έπειτα ο κάθε παίκτης έχει την δυνατότητα να δει τον ρόλο (Εικόνα 3.6) του καθώς και πληροφορίες για αυτόν ή την περιοχή που βρίσκεται (Εικόνα 3.7).

Όταν όλοι οι παίκτες μάθουν τον ρόλο τους, το παιχνίδι ξεκινάει. Κατά την διάρκεια του παιχνιδιού οι παίκτες έχουν την δυνατότητα να ξαναδούν τον ρόλο τους.



Σχήμα 3.1: Διάγραμμα ροής

3.4 Αρχικοποίηση παραθύρου

SceneDelegate

Για την αρχικοποίηση της αρχικής οθόνης χρησιμοποιείται η κλάση SceneDelegete. Η μέθοδος **func scene(_ scene: UIScene, willConnectTo _: UISceneSession, options _: UIScene.ConnectionOptions)** καλείται αυτόματα από το σύστημα κατά την αρχικοποίηση της εφαρμογής και με τη χρήση της μπορούμε να ορίσουμε την αρχική οθόνη στο ενεργό παράθυρο.

```
if let windowScene = scene as? UIWindowScene {
    window = UIWindow(windowScene: windowScene)
    let storyborad = UIStoryboard.storyboard(.Home)
    let viewController: HomeViewController = storyborad.instantiateViewController(withIdentifier: HomeViewController.storyboardIdentifier) as! HomeViewController
    HomeModuleAssembly().configureModuleForViewInput(viewInput: viewController)
    let navigationController = BaseNavigationController(rootViewController: viewController)
    window = UIWindow(windowScene: windowScene)
    window!.rootViewController = navigationController
```

```
        window!.makeKeyAndVisible()
    }
```

Αφού ορίσουμε αρχική οθόνη στον ριζικό View Controller εμφανίζουμε το παράθυρο και φαίνονται στον χρήστη τα διαθέσιμα παιχνίδια καθώς και η επιλογή για να πάει στις ρυθμίσεις.

3.5 Base

Κάθε οπτική κλάση η οποία επαναχρησιμοποιείται στην εφαρμογή κληρονομεί μία native κλάση του συστήματος πχ. UIView – UIViewController – UINavigationController.

Για επαναχρησιμοποίηση δυνατοτήτων έχει δημιουργηθεί μία ενδιάμεση Base κλάση για κάθε μία επαναχρησιμοποιούμενη κλάση η οποία κληρονομεί τις native κλάσεις και οι οπτικές κλάσεις που χρησιμοποιούνται στην εφαρμογή κληρονομούν τις base κλάσεις.

```
import Foundation
class BaseViewController: UIViewController {
    let gradientBackgroundView = UIGradientView()
    var hasGradientView: Bool = true

    override func viewDidLoad() {
        super.viewDidLoad()
        self.localiseView()
        NotificationCenter.default.addObserver(self, selector: #selector(userDidUpdateLocale), name: N
SNotification.Name.Language.languageUpdate, object: nil)
    }

    @objc func userDidUpdateLocale() {
        self.localiseView()
    }

    func localiseView() {

    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        if hasGradientView {
            implementGradientView()
        }
    }

    func implementGradientView() {
        view.backgroundColor = Colors.lightBackGroundColor
```

```
        gradientBackgroundView.frame = UIScreen.main.bounds
        gradientBackgroundView.barGradientStartColor = Colors.lightBackGroundColor
        gradientBackgroundView.barGradientEndColor = Colors.strongBackGroundColor
        view.insertSubview(gradientBackgroundView, at: 0)
        gradientBackgroundView.reloadGradientView()
    }
}
```

BaseViewController.swift

Όλες οι οθόνες της εφαρμογής έχουν κοινό χρώμα παρασκηνίου. Για να αποφευχθεί η επαναλαμβανόμενη οριστικοποίηση του Gradient χρώματος στις οθόνες ο ενδιάμεσος BaseViewController αναλαμβάνει κατά τον κύκλο ζωής της εφαρμογής να προσθέσει το gradient χρώμα. Επίσης αναλαμβάνει να καλέσει την μέθοδο localize View την οποία κληρονομούν όλοι οι ViewController της εφαρμογής και είναι υπεύθυνη για να φαίνονται τα σωστά λεκτικά στην εφαρμογή.

3.6 Home Module

Στην αρχική οθόνη οι δυνατότητες του χρήστη είναι να επιλέξει ποιο παιχνίδι θα παίξει, να δει πληροφορίες για τα παιχνίδια και να μεταβεί στις ρυθμίσεις. Τα παιχνίδια τα οποία φαίνονται στον HomeViewController είναι αυτά που ορίζονται στη μέθοδο loadGames() του HomeInteractor.

```
func loadGames() {
        let palermo = Game(GameType.palermo)
        let spy = Game(GameType.spy)
        let killer = Game(GameType.killer)
        let games = [palermo, spy, killer]
        output.loadGamesCompleted(games: games)
    }
```

Με προσθαφαίρεση παιχνιδιών στον HomeInteractor μπορούμε να αλλάξουμε το οπτικό περιεχόμενο του HomeViewController χωρίς να αλλάξουμε κάτι στην ίδια την κλάση.

Από την αρχική ο χρήστης μπορεί να επιλέξει το σύμβολο i που βρίσκεται σε κάθε παιχνίδι και να μεταβεί στο GameInformationModule ή να επιλέξει ποιο παιχνίδι θέλει να παίξει και να μεταβεί στο GameDetailsModule.

Εικόνα 3.1

3.7 NavigateToGameDetailsHandler.swift

Επειδή ο χρήστης πρέπει να έχει τη δυνατότητα να μεταφέρεται από οποιοδήποτε σημείο στο παιχνίδι και για κάθε παιχνίδι να μπορεί να βλέπει τις πληροφορίες του παιχνιδιού και των ρόλων δημιουργήθηκε το πρωτόκολλο **NavigateToGameDetailsHandler.** Το πρωτόκολλο αυτό έχει 3 μεθόδους οι οποίες υλοποιούνται σε επέκταση του πρωτοκόλλου και οποιαδήποτε κλάση υλοποιεί αυτό το πρωτόκολλο έχει τις μεθόδους που προσφέρονται, οι οποίες είναι:

func navigateToGameDetails(from view: AnyObject, game: Game)

Μετακίνηση στο GameDetailsModule από οποιονδήποτε ViewController

func navigateToGameInformation(from view: AnyObject, game: Game)

Μετακίνηση στο GameInformationModule από οποιονδήποτε ViewController

func navigateToRolesInformation(from view: AnyObject, game: Game)

Μετακίνηση στο RolesInformationModule από οποιονδήποτε ViewController

3.8 SettingsModule

Η οθόνη των ρυθμίσεων χρησιμοποιείται μόνο για αλλάζει η γλώσσα της εφαρμογής. Οι επιλογές του χρήστη είναι Ελληνικά και Αγγλικά. Για την αλλαγή της γλώσσας ο Presenter της οθόνης καλεί την μέθοδο updateLocalizedLanguage από την κλάση Localization και για να δείξει στον χρήστη την επιλεγμένη γλώσσα χρησιμοποιείται η μέθοδος getCurrentLanguage.

Εικόνα 3.2

3.9 Localization

Για να μπορεί η εφαρμογή να έχει παραπάνω από μία γλώσσα πρέπει κάθε λεκτικό της εφαρμογής να αντιστοιχεί σε ένα μοναδικό κλειδί. Με βάση αυτό το κλειδί και τα αρχεία el.Iproj και en.Iproj γίνεται η αντιστοίχιση στην σωστή μετάφραση.

Για να γίνει η μετάφραση ενός κλειδού χρησιμοποιείται από την κλάση Localization η μέθοδος static func localized(_ key: String) -> String

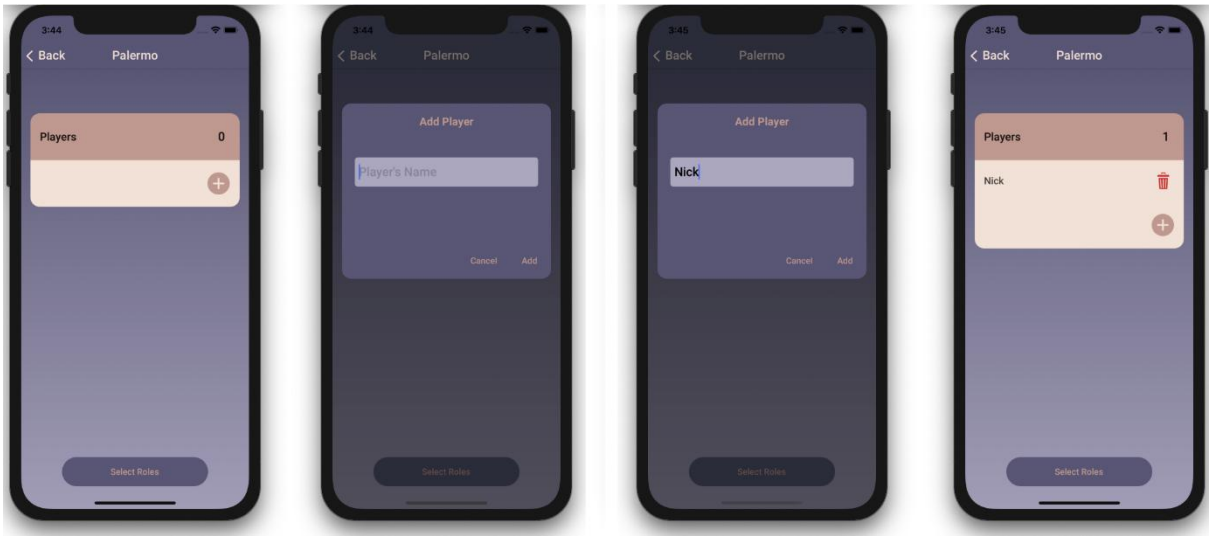
Παίρνει σαν παράμετρο ένα κλειδί και επιστρέφει το μεταφρασμένο λεκτικό ανάλογα με την αποθηκευμένη επιλογή γλώσσας του χρήστη.

Η μέθοδος static func getCurrentLanguage() -> Language επιστρέφει το λεκτικό “el” αν η επιλεγμένη γλώσσα είναι ελληνικά και “en” αν η επιλεγμένη γλώσσα είναι αγγλικά. Για την υλοποίηση των αλλαγών πάνω σε ότι έχει να κάνει με τις επιλογές του χρήστη χρησιμοποιείται η κλάση LocalizationDefaults. Στην περίπτωση που ο χρήστης μπαίνει για πρώτη φορά στην εφαρμογή ελέγχεται αν η γλώσσα του συστήματος του κινητού είναι τα ελληνικά. Αν ναι τότε ορίζεται η επιλεγμένη γλώσσα της εφαρμογής τα ελληνικά ενώ σε οποιαδήποτε άλλη περίπτωση τα αγγλικά. Όταν ο χρήστης θέλει να αλλάξει την επιλογή γλώσσας τότε η επιλογή του αποθηκεύεται στα UserDefaults με μοναδικό κλειδί "key_SpyRolePalermo_LocalizationDefaults_SelectedLanguage" και έτσι όταν ο χρήστης μελλοντικά ξανά ανοίξει την εφαρμογή θα έχει αποθηκευτεί η επιλογή του.

3.10 GameDetailsModule

Το game details module είναι υπεύθυνο για να οριστούν οι παίκτες στο παιχνίδι. Για κάθε επιλογή παιχνιδιού χρησιμοποιείται το GameDetailsModule για να οριστούν οι παίκτες. Αποτελείται από 2 UICollectionView την SavedPlayerNamesCollectionView και PlayerListCollectionView. Στο αντικείμενο της κλάσης SavedPlayerNamesCollectionView φαίνονται τα ονόματα των παικτών οι οποίοι είχαν παίξει στο τελευταίο παιχνίδι και ο χρήστης μπορεί να τα επιλέξει έτσι ώστε να μην χρειαστεί να τα ξαναγράψει. Στο αντικείμενο της κλάσης PlayerListCollectionView φαίνονται τα ονόματα των παικτών οι οποίοι θα παίξουν. Ο χρήστης έχει τη δυνατότητα είτε να προσθέσει έξτρα

άτομα είτε να διαγράψει. Τα δύο collection View βρίσκονται μέσα σε ένα UIScrollview και επιτρέπεται η δυναμική αυξομείωση του ύψους καθώς και η χρήση ενός ενιαίου scrolling.



Εικόνα 3.3

Από το GameDetailsModule ανάλογα με το παιχνίδι ο χρήστης μπορεί να προχωρήσει στην επόμενη οθόνη η οποία είναι διαφορετική για κάθε παιχνίδι. Το σε ποια οθόνη θα οδηγηθεί ο χρήστης ορίζεται από την μέθοδο func getModuleInput(vc: UIViewController) -> GameInputProtocol? της κλάσης Game η οποία επιστρέφει ένα αντικείμενο το οποίο πρέπει να υλοποιεί το GameInputProtocol. Με αυτό τον τρόπο το Module δεν χρειάζεται να ξέρει που πραγματικά θα πάει ο χρήστης και επιτυγχάνεται η συνθήκη ανοιχτό για επέκταση και κλειστό για τροποποίηση καθώς όσα παιχνίδια και να δημιουργηθούν καινούρια, το GameDetailsModule δε θα χρειαστεί να τροποποιηθεί και θα λειτουργεί κανονικά.

```
func getModuleInput(vc: UIViewController) -> GameInputProtocol? {
        switch self.type {
            case .palermo:
                return SelectAvailableRolesModuleAssembly().configureModuleForViewInput(viewInput:
vc, delegate: nil)
            case .killer:
                return SelectNumberOfBadModuleAssembly().configureModuleForViewInput(viewInput: v
c)
            case .spy:
                return SelectNumberOfBadModuleAssembly().configureModuleForViewInput(viewInput: v
c)
        }
    }
```

Η μεθοδος getModuleInput επιστρέφει το κατάλληλο Module για να προχωρήσει η διαδικασία καταχώρησης των πληροφοριών για το παιχνίδι. Για το Palermo επιστρέφεται το SelectAvailableRoles Module ενώ για το Killer και για το Spy to SelectNumberOfBad Module.

Η τεχνική του να λειτουργούν όλες οι οθόνες χωρίς να εξαρτώνται από το παιχνίδι εφαρμόζεται σε πολλά σημεία και για αυτό η κλάση Game έχει την ευθύνη να τροφοδοτεί τις κλάσεις με abstract και μη πληροφορίες οι οποίες μπορούν να είναι τροποποιήσιμες κατά την κληρονομικότητα ή να είναι διαφορετικές ανάλογα με τον τύπο του παιχνιδιού (GameType). Για παράδειγμα κάθε παιχνίδι διαθέτει ένα InformationString το οποίο είναι μία get μεταβλητή η οποία δεν μπορεί να τεθεί στο Game object παρά μόνο να αναγνωστεί και υπολογίζεται με βάση την τιμή της μεταβλητής type της κλάσης Game.

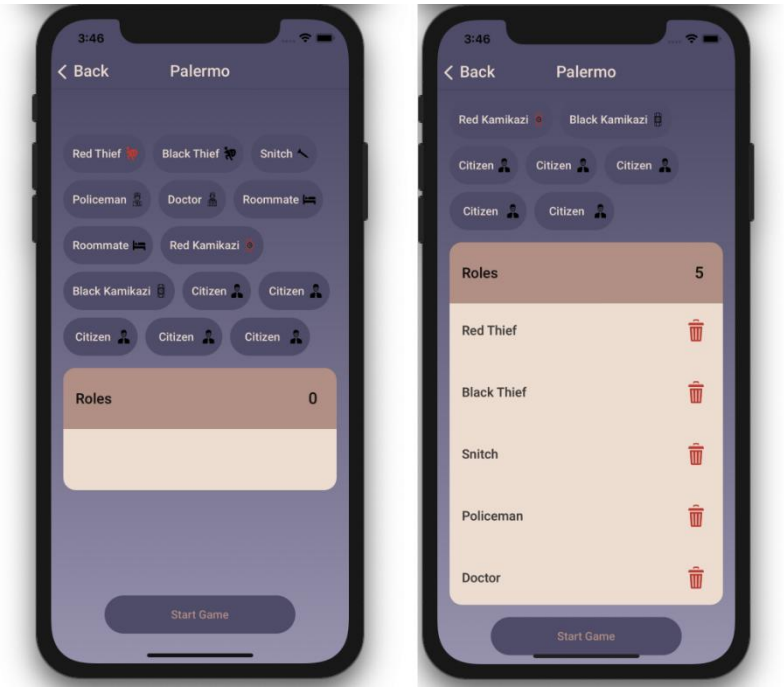
```
var informationString: String {
      get {
         switch self.type {
         case .killer:
            return Localization.localized("information_killer")
         case .palermo:
            return Localization.localized("information_palermo")
         case .spy:
            return Localization.localized("information_spy")
         }
      }
   }
```

Για κάθε καινούριο τύπο παιχνιδιού που προστίθεται στον enumerator GameType πρέπει να παρέχονται και οι αντίστοιχες πληροφορίες για κάθε μεταβαλλόμενη μεταβλητή. Στην περίπτωση που γίνει η παροχή αυτή θα δημιουργηθεί compile error. Επίσης με βάση την τιμή της υπολογιζόμενης μεταβλητής minimumPlayers υπάρχει ένας ελάχιστος αριθμός παιχτών για κάθε παιχνίδι.

3.11 SelectAvailableRolesModule.

Το module SelectAvailableRoles επαναχρησιμοποιεί τον ViewController του GameDetails module καθώς και πάλι χρειάζονται 2 αντικείμενα της κλάσης UICollectionView για να εμφανίσουν τους διαθέσιμους ρόλους και τους επιλεγμένους ρόλους. Οι αλλαγές στην λογική γίνονται στα υπόλοιπα μέλη του module, presenter router interactor assembly. Ο χρήστης χρησιμοποιώντας τις ίδιες function μπορεί να επιλέξει τους ρόλους του παιχνιδιού και να ξεκινήσει το παιχνίδι με τον ίδιο τρόπο που θα επέλεγε τους χρήστες από τους αποθηκευμένους του προηγούμενου παιχνιδιού. Οι διαφορές που ορίζονται στον Presenter είναι ότι ο χρήστης δεν μπορεί να προσθέσει δικούς του ρόλους αλλά μπορεί να επιλέξει μόνο από τους διαθέσιμους καθώς και ότι όταν γίνεται διαγραφή ενός ρόλου τότε ο ρόλος επιστρέφει στους διαθέσιμους. Με την χρήση του enum PlayerListCollectionViewHeaderType αλλάζουν και τα γραφικά της κλάσης PlayerListCollectionView έτσι ώστε να είναι διαφορετικά το Header και το Footer του UICollectionView αντικειμένου.

Όταν ο χρήστης επιλέξει ίσο αριθμό ρόλων με τον αριθμό των παικτών τότε μπορεί να ξεκινήσει το παιχνίδι.

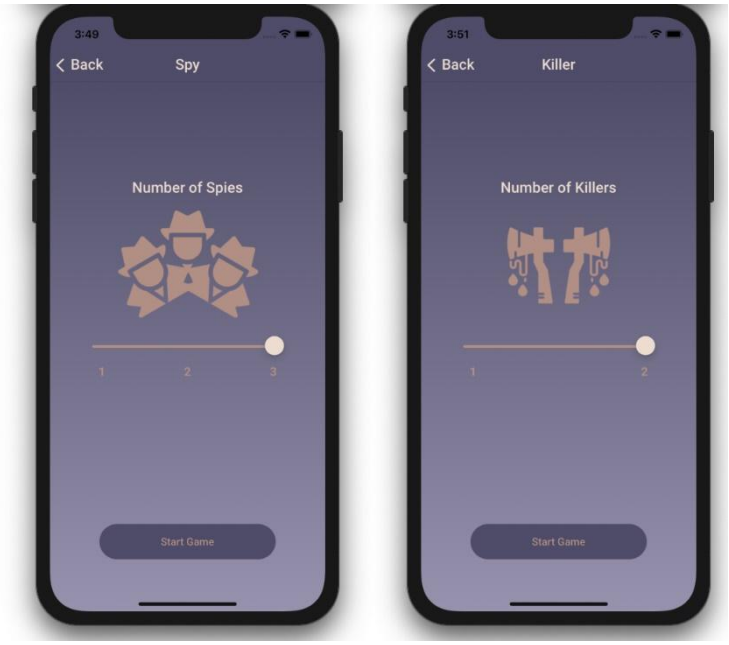


Εικόνα 3.4

3.12 SelectNumberOfBadModule

Το SelectNumberOfBad module αποτελείται από έναν slider κατά τον οποίο επιλέγεται ο αριθμός των κακών στην περίπτωση του Killer και ο αριθμός των κατασκόπων στην περίπτωση του Spy. Ο slider παίρνει την υψηλότερη πιθανή τιμή από την μεταβλητή maxNumberOfBad της κλάσης Game και οι εικόνες που απεικονίζονται με βάση τον αριθμό των κακών από την μέθοδο getImageBasedOnNumberOfBad(numberOfBad: Int) -> UIImage?. Κάθε φορά που αλλάζει τιμή ο slider ανανεώνεται το περιεχόμενο του UIImageView του SelectNumberOfBadViewController και εμφανίζει την αντίστοιχη εικόνα.

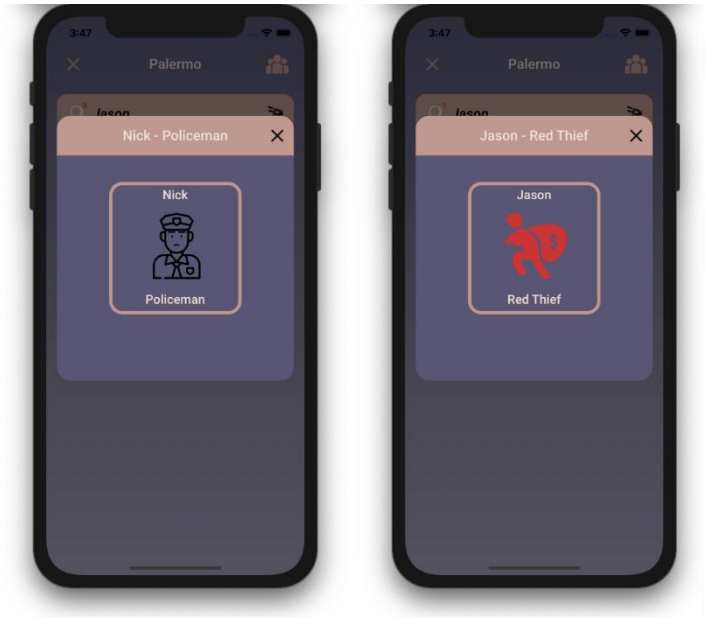
Όταν ο χρήστης επιλέξει τους διαθέσιμους ρόλους ή επιλέξει τον αριθμό των κακών για κάθε παιχνίδι αντίστοιχα ξεκινάει το παιχνίδι και οι χρήστες μπορούν να δουν τους ρόλους τους. Ο κάθε παίκτης λαμβάνει τυχαία έναν ρόλο.



Εικόνα 3.5

3.13 SeeYourRoleModule

Οι δυνατότητες που παρέχονται στην οθόνη του SeeYourRole module παραμετροποιούνται ανάλογα με το παιχνίδι. Σε κάθε παιχνίδι οι χρήστες έχουν τη δυνατότητα να δουν το ρόλο που τους έχει ανατεθεί. Κάθε φορά που ένας χρήστης βλέπει έναν ρόλο αυξάνεται ο αριθμός προβολής του ρόλου έτσι ώστε να καταγράφεται η προβολή και να αποφεύγονται τυχόν λάθη. Επιπλέον δυνατότητες είναι να δολοφονηθεί ένας παίκτης που στην προκειμένη περίπτωση διαχωρίζεται σε μέρα και νύχτα, να προβληθούν οι διαθέσιμοι ρόλοι του παιχνιδιού και να προβληθούν οι τοποθεσίες οι οποίες είναι διαθέσιμες. Στην περίπτωση που το παιχνίδι διαθέτει τοποθεσίες τότε κάθε χρήστης κατά την προβολή του ρόλου του μπορεί να δει και την τοποθεσία η οποία έχει ορισθεί. Όταν ο χρήστης επιλέξει να δει τους διαθέσιμους ρόλους, έχει τη δυνατότητα να δει και τις πληροφορίες για κάθε ρόλο. Στην προκειμένη περίπτωση επαναχρησιμοποιείται το πρωτόκολλο **NavigateToGameDetailsHandler.**



Εικόνα 3.6

3.14 PopUpHandler

Για την προσθήκη παραθύρων πάνω από τα αντικείμενα των UIViewController χρησιμοποιείται το Singleton PopUpHandler. Αυτό σημαίνει ότι υπάρχει μόνο μία αναφορά του αντικειμένου στην εφαρμογή κατά τη διάρκεια ζωής της κατά το runtime. Η προσθήκη των παραθύρων γίνεται με την χρήση αντικειμένων τα οποία είναι τύπου υπο-κλάσεως του UIView.

Για να γίνει εφικτή η ενθυλάκωση, κάθε υποκλάση η οποία μπορεί να προβληθεί μέσα από τον PopUpHandler υλοποιεί το PopUpViewable πρωτόκολλο που προσθέτει τη μεταβλητή type. Με βάση αυτή τη μεταβλητή τροποποιούνται οι παράμετροι για κάθε διαφορετικό UIView αντικείμενο. Για παράδειγμα το όνομα του αρχείου .XIB το οποίο πρόκειται να προστεθεί δίνεται με βάση τον τύπο που έχει η εκάστοτε υποκλάση.

```
static var viewIdentifier: String {
    switch type {
    case .addPlayerPopUpView:
        return "AddPlayerPopUpView"
    case .message:
        return "MessagePopUpView"
    case .yesNo:
        return "YesNoPopUpView"
    case .roleInformationPopUp:
        return "RoleInformationsPopUpView"
    case .customRoleView:
        return "CustomRoleView"
```

```
        case .peoplePopUpView:
            return "PeoplePopUpView"
        case .locationsPopUpView:
            return "LocationsPopUpView"
        }
    }
```

Ομοίως ορίζεται και το μέγεθος κάθε διαφορετικού αντικειμένου.

```
extension PopUpViewable {
    static func getRect() -> CGRect {
        let screenWidth = UIScreen.width
        switch type {
        case .addPlayerPopUpView:
            return CGRect(x: 16, y: 150, width: screenWidth - 32, height: 300)
        case .message:
            return CGRect(x: 16, y: 150, width: screenWidth - 32, height: 250)
        case .yesNo:
            return CGRect(x: 16, y: 150, width: screenWidth - 32, height: 150)
        case .roleInformationPopUp:
            return CGRect(x: 16, y: 150, width: screenWidth - 32, height: 400)
        case .customRoleView:
            return CGRect(x: 16, y: 150, width: 200, height: 200)
        case .peoplePopUpView:
            return CGRect(x: 16, y: 150, width: screenWidth - 32, height: UIScreen.height)
        case .locationsPopUpView:
            let availableScreenSize = UIScreen.height - 100 - 40
            let collectionViewContentSize = (APILocationsService.shared.getLocations().count / 2) * 50
            if (Int(availableScreenSize) - collectionViewContentSize) > 0 {
                return CGRect(x: 16, y: 100, width: Int(screenWidth) -
 32, height: 40 + (APILocationsService.shared.getLocations().count / 2) * 50)
            } else {
                return CGRect(x: 16, y: 100, width: Int(screenWidth) - 32, height: Int(UIScreen.height -
 116.0))
            }
        }
    }
}
```

Ο PopUpHandler για την προσθήκη ενός παραθύρου στην οθόνη του κινητού χρησιμοποιεί το παράθυρο οθόνης το οποίο είναι μία getOnly μεταβλητή μέσα στην κλάση

```
private var window: UIWindow? {
        return UIApplication.shared.connectedScenes
```

```
        .filter { $0.activationState == .foregroundActive }
        .map { $0 as? UIWindowScene }
        .compactMap { $0 }
        .first?.windows
        .filter { $0.isKeyWindow }.first
    }
```

Για την προσθήκη των παραθύρων χρησιμοποιείται πάντα ένα αντικείμενο της κλάσης UIView το οποίο καλύπτει όλη την επιφάνεια του παραθύρου και μέσα σε αυτό προστίθεται το παράθυρο το οποίο θέλουμε να εμφανιστεί στον χρήστη. Η έξτρα προσθήκη του πατέρα παραθύρου μας επιτρέπει να χρησιμοποιήσουμε gestureRecognizer αντικείμενα για να εντοπίσουμε αλληλεπιδράσεις του χρήστη με το παρασκήνιο έτσι ώστε να αποκρύψουμε το παράθυρο σε περίπτωση που ο χρήστης κάνει αφή στον πατέρα παράθυρο.

Κάθε φορά που γίνεται καινούργια προσθήκη παραθύρου αφαιρείται ο πατέρας παράθυρο και λόγο της πατέρα-παιδιού σχέσης των αντικειμένων και το προστιθέμενο παράθυρο και εμφανίζεται το καινούριο με καινούρια σχέση πατέρα-παιδιού. Ο πατέρας αποθηκεύεται στην μεταβλητή της κλάσης PopUpHandler έτσι ώστε να έχουμε αναφορά σε αυτόν και να μπορούμε να το αφαιρέσουμε είτε όταν ο χρήστης θέλει να κλείσει το παράθυρο είτε όταν θέλουμε να προσθέσουμε καινούριο παράθυρο.

```
private var containerView: ContainerView?
```

```
public func closePopUp() {
        containerView?.removeFromSuperview()
        containerView = nil
    }
```

3.15 PlayerCaser

Η κλάση PlayerCaser.swift έχει σαν σκοπό την αποθήκευση των χρηστών του προηγούμενου παιχνιδιού. Ο τρόπος με τον οποίο γίνεται είναι με την αποθήκευση ενός αλφαριθμητικού στα UserDefaults του συστήματος. Κάθε φορά που γίνεται η ανανέωση των ονομάτων των χρηστών από το Module GameDetails τότε χρησιμοποιείται η μέθοδος

```
func savePlayersToStorage(players: [String]) {
        let allPlayers = combinePlayerNames(Players: players)
        userDefault.set(allPlayers, forKey: allNamesCombinedKey)
    }
```

Ο τρόπος με τον οποίο από ένα πίνακα ονομάτων αποθηκεύεται στη μνήμη μόνο ένα αλφαριθμητικό είναι με την ένωση όλων των ονομάτων με ένα έξτρα μοναδικό αλφαριθμητικό το οποίο χρησιμοποιείται και για την ανάγνωση των ονομάτων από τη μνήμη.

```
private let prefix = " #"
```

```
private func combinePlayerNames(Players: [String]) -> String {
    return Players.joined(separator: prefix)
```

```
}

private func seperatePlayers(combinePlayerNames: String) -> [String] {
   return combinePlayerNames.components(separatedBy: prefix)
}
```

Ομοίως η ανάγνωση των ονομάτων γίνεται με τη χρήση της μεθόδου restoreAllPlayerNames όπου και πάλι από τα UserDefaults διαβάζονται τα ονόματα των χρηστών και διαχωρίζονται με βάση το μοναδικό λεκτικό.

```
func restoreAllPlayerNames() -> [String] {
      guard let combinePlayerNames = userDefault.value(forKey: allNamesCombinedKey) as? String,
combinePlayerNames.count > 0 else {
         return []
      }
      return seperatePlayers(combinePlayerNames: combinePlayerNames)
   }
```

3.16 ErrorHandling

Για τη κοινή αντιμετώπιση των σφαλμάτων χρήστη δημιουργήθηκε η κλάση Error.

```
class Error {
   private var type: ErrorType

   var title: String {
      return type.title
   }

   var description: String {
      return type.description
   }

   init(type: ErrorType) {
      self.type = type
   }
}
```

Κάθε αντικείμενο της κλάσης Error έχει ένα τίτλο, μία περιγραφή και έναν τύπο. Ο τύπος errorType είναι ένα enum αντικείμενο που προσφέρει πληροφορίες και με την προσθήκη ενός νέου τύπου επαναχρησιμοποιείται η υλοποίηση της κλάσης Error χωρίς αλλαγές.

Οι διαφορετικοί τύποι σφαλμάτων που μπορούν να προκύψουν στην κλάση ErrorType είναι

```
   case playerNameExists
```

```
    case emptyPlayerName
    case playerNumberError
    case rolesLimitError
    case playerCountToRolesCountInconsistencyError
    case incorrectNumberOfBad
    case locationsNotLoaded
```

Ανάλογα με τον τύπο του σφάλματος παρέχονται και οι αντίστοιχες πληροφορίες τίτλου και περιγραφής.

```
var title: String {
        switch self {
            case .playerNameExists, .emptyPlayerName:
                return Localization.localized("error_title_invalidPlayerName")
            case .playerNumberError:
                return Localization.localized("error_title_playerNumber")
            case .rolesLimitError:
                return Localization.localized("error_title_rolesLimit")
            case .playerCountToRolesCountInconsistencyError:
                return Localization.localized("error_title_playerCountInconsistency")
            case .incorrectNumberOfBad:
                return Localization.localized("error_title_incorrectNumberOfBad")
            case .locationsNotLoaded:
                return Localization.localized("error_title_locationsNotLoaded")
        }
    }
```

3.17 GameRolesService

Για την δημιουργία των ρόλων των παιχνιδιών χρησιμοποιείται η κλάση GameRolesService. Η κλάση αυτή δημιουργεί στατικά όλα τα αντικείμενα της κλάσης Role ανάλογα με το παιχνίδι και τα επιστρέφει σαν αποτέλεσμα μεθόδου. Όλα τα παιχνίδια χρειάζονται αντικείμενα της κλάσης Role για να μπορέσουν να παιχτούν σωστά.

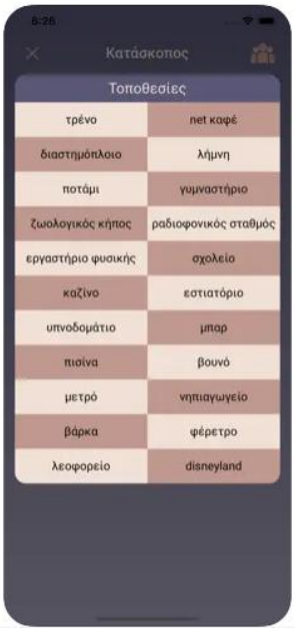
Κάθε αντικείμενο της κλάσης Role έχει 4 μεταβλητές. Το μοναδικό κλειδί του, τον τίτλο, το εικονίδιο και την περιγραφή του. Παράδειγμα δημιουργίας ρόλων Palermo.

```
static func getPalermoRoles() -> [Role] {
        let redThief = Role(id: 1, name: Localization.localized("palermo_red_thief"), icon: StyleKit.imag
eOfIcon_palermo(mainIconPictureColor: Colors.redColor), information: Localization.localized("role_
info_redThief"))
        let blackThief = Role(id: 2, name: Localization.localized("palermo_black_thief"), icon: StyleKit.i
mageOfIcon_palermo(mainIconPictureColor: Colors.blackColor), information: Localization.localized(
"role_info_blackThief"))
        redThief.setPayerThatIKnow(knownPlayer: blackThief)
```

```
        blackThief.setPayerThatIKnow(knownPlayer: redThief)
        let snitch = Role(id: 3, name: Localization.localized("palermo_snitch"), icon: StyleKit.imageOfIcon_snitch(roleColor: Colors.blackColor), information: Localization.localized("role_info_snitch"))
        snitch.setPayerThatIKnow(knownPlayer: redThief)
        let policeman = Role(id:4, name: Localization.localized("palermo_policeman"), icon: StyleKit.imageOfIcon_policeman(roleColor: Colors.blackColor), information: Localization.localized("role_info_policeman"))
        policeman.setPayerThatIKnow(knownPlayer: snitch)
        let doctor = Role(id: 5, name: Localization.localized("palermo_doctor"), icon: StyleKit.imageOfIcon_doctor(roleColor: Colors.blackColor), information: Localization.localized("role_info_doctor"))
        let roomMaiteL = Role(id: 6, name: Localization.localized("palermo_roomMate_1"), icon: StyleKit.imageOfIcon_roommaite(roleColor: Colors.blackColor), information: Localization.localized("role_info_roommateOne"))
        let roomMaiteR = Role(id: 7, name: Localization.localized("palermo_roomMate_2"), icon: StyleKit.imageOfIcon_roommaite(roleColor: Colors.blackColor), information: Localization.localized("role_info_roommateTwo"))
        roomMaiteL.setPayerThatIKnow(knownPlayer: roomMaiteR)
        roomMaiteR.setPayerThatIKnow(knownPlayer: roomMaiteL)
        let redKamikazi = Role(id: 8, name: Localization.localized("palermo_red_kamikazi"), icon: StyleKit.imageOfIcon_bomb(roleColor: Colors.blackColor, bombColor: Colors.redColor), information: Localization.localized("role_info_redKamikazi"))
        let blackKamikazi = Role(id: 9, name: Localization.localized("palermo_black_kamikazi"), icon: StyleKit.imageOfIcon_bomb(roleColor: Colors.blackColor, bombColor: Colors.blackColor), information: Localization.localized("role_info_blackKamicazi"))
        let citizenGeorge = Role(id: 10, name: Localization.localized("palermo_citizen_george"), icon: StyleKit.imageOfIcon_citizen(roleColor: Colors.blackColor), information: Localization.localized("role_info_citizen"))
        let citizenMichael = Role(id: 11, name: Localization.localized("palermo_citizen_michael"), icon: StyleKit.imageOfIcon_citizen(roleColor: Colors.blackColor), information: Localization.localized("role_info_citizen"))
        let citizenJohn = Role(id: 12, name: Localization.localized("palermo_citizen_john"), icon: StyleKit.imageOfIcon_citizen(roleColor: Colors.blackColor), information: Localization.localized("role_info_citizen"))
        let citizenJack = Role(id: 13, name: Localization.localized("palermo_citizen_jack"), icon: StyleKit.imageOfIcon_citizen(roleColor: Colors.blackColor), information: Localization.localized("role_info_citizen"))
        let citizenJames = Role(id: 14, name: Localization.localized("palermo_citizen_james"), icon: StyleKit.imageOfIcon_citizen(roleColor: Colors.blackColor), information: Localization.localized("role_info_citizen"))
        return [redThief, blackThief, snitch, policeman, doctor, roomMaiteR, roomMaiteL, redKamikazi, blackKamikazi, citizenGeorge, citizenMichael, citizenJack, citizenJames, citizenJohn]
    }
```

3.18 Τοποθεσίες

Για την προβολή των τοποθεσιών στην περίπτωση του παιχνιδιού κατασκόπου χρησιμοποιείται το Firebase ως Database και οι τοποθεσίες γίνονται λήψη από την απομακρυσμένη βάση. Η λήψη των τοποθεσιών γίνεται κατά την έναρξη της εφαρμογής. Στην περίπτωση που ο χρήστης δεν έχει σύνδεση στο διαδίκτυο λαμβάνει μήνυμα σφάλματος όταν επιλέξει να παίξει το παιχνίδι Κατάσκοπος. Αν συνδεθεί στο διαδίκτυο τότε γίνεται εκ νέου λήψη των τοποθεσιών. Οι τοποθεσίες λαμβάνονται και στα ελληνικά και στα αγγλικά και αποθηκεύονται στο Singleton APILocationsService από το οποίο μπορούν να αναγνωστούν και να χρησιμοποιηθούν αργότερα σε οποιαδήποτε φάση της εφαρμογής.



Εικόνα 3.7

```
func loadPlaces(completion: (()->()?)) {

    ref.child("placesEN").observeSingleEvent(of: .value, with: { (snapShot) in
        var locations: [String] = []
        if let value = snapShot.value as? Array<String> {
            print(value)
            locations.append(contentsOf: value)
            APILocationsService.shared.updateLocationsEN(locations: locations)
        } else {
            print("map didn't pass")
        }
    }) { (error) in
        print(error)
    }
```

```
        ref.child("placesEL").observeSingleEvent(of: .value, with: { (snapShot) in
            var locations: [String] = []
            if let value = snapShot.value as? Array<String> {
                print(value)
                locations.append(contentsOf: value)
                APILocationsService.shared.updateLocationsEL(locations: locations)
            } else {
                print("map didn't pass")
            }
        }) { (error) in
            print(error)
        }


    }
```

Η βάση του Firebase είναι μία NoSQL βάση και τα δεδομένα τα οποία είναι αποθηκευμένα έχουν τη μορφή JSON.

```
{
  "placesEL":[
    "τρένο",
    "net καφέ",
    "διαστημόπλοιο",
    "λίμνη",
    "ποτάμι",
    "γυμναστήριο",
    "ζωολογικός κήπος",
    "ραδιοφωνικός σταθμός",
    "εργαστήριο φυσικής",
    "σχολείο",
    "καζίνο",
    "εστιατόριο",
    "υπνοδωμάτιο",
  ],
  "placesEN":[
```

```
    "train",
    "net caffe",
    "spaceship",
    "lake",
    "river",
    "gym",
    "zoo",
    "radio station",
    "science lab",
    "school",
    "casino",
    "restaurant",
    "bedroom",

  ]
}
```

Με την χρήση της Online βάσης προστίθεται η αλλαγή των τοποθεσιών σε πραγματικό χρόνο χωρίς την ανανέωση της εφαρμογής στο store.

3.19 APILocationsService

Η κλάση Singleton APILocationsService διαθέτει δύο μεταβλητές

```
private var locationsEN: [String] = []
private var locationsEL: [String] = []
```

στις οποίες αποθηκεύονται οι τοποθεσίες. Για την ανάγνωση των τοποθεσιών από το Singleton χρησιμοποιείται η μέθοδος getLocations που ανάλογα με την επιλογή του χρήστη ως προς την τοποθεσία επιστρέφει και τον πίνακα των τοποθεσιών.

```
func getLocations() -> [String] {
    let lang = Localization.getCurrentLanguage()
    if lang == .english {
        return self.locationsEN
    } else {
        return self.locationsEL
    }
```

```
}
```

3.20 Επίλογος

Η εφαρμογή έχει ανεβεί στο app store και μπορεί να την κατεβάσει οποιοδήποτε συσκευή χρησιμοποιεί iOS 13.0 ή μεταγενέστερη έκδοση. Υπάρχουν πολλές ακόμα δυνατότητες και προσθήκες που μπορούν να γίνουν. Ο κώδικας της εφαρμογής δεν είναι περίπλοκος και είναι εύκολα επεκτάσιμος. Επίσης τα δεδομένα της εφαρμογής είναι εύκολα τροποποιήσιμα.

Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Η εφαρμογή έχει ήδη ανέβει στο app store με την ονομασία Palermo, έχει μέγεθος μόλις 13,6 MB και διατίθεται δωρεάν.

Μπορούν να γίνουν αρκετές προσθήκες για να βελτιωθεί όπως animation και αφήγηση με ήχο ώστε να γίνει πιο διαδραστικό το παιχνίδι. Επίσης και άλλα παιχνίδια ρόλων μπορούν να συμπληρωθούν στην συλλογή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] iOS 16 - Apple Developer

[2] Swift - Apple Developer

[3] Swift.org - Documentation

[4] Swift - Apple Developer

[5] Xcode 14 Overview - Apple Developer

[6] UIKit | Apple Developer Documentation

[7] Storyboard (apple.com)

[8] View controllers | Apple Developer Documentation

[9] UITabBarController | Apple Developer Documentation

[10] Auto Layout Guide: Understanding Auto Layout (apple.com)

[11] Foundation | Apple Developer Documentation

[12] UIView | Apple Developer Documentation

[14] Firebase Documentation (google.com)

[15] CocoaPods Guides - Home

[16] About Internationalization and Localization (apple.com)

[17] Core Data | Apple Developer Documentation

ΚΩΔΙΚΑΣ

```
import Foundation
enum GameType {
case palermo
case spy
case killer
 var title: String {
switch self{
 case .palermo:
 return Localization.localized("games_palermo")
case .spy:
return Localization.localized("games_spy")
case .killer:
return Localization.localized("games_killer")}
}
var rolesTitle: String {
switch self {
case .palermo:
return Localization.localized("games_roles_palermo")
case .spy:
return Localization.localized("games_roles_spy")
case .killer:
return Localization.localized("games_roles_killer")}
}
var icon: UIImage {
switch self {
case .palermo:
return Images.palermoImage
case .spy:
return Images.spyImage
```

```
case .killer:
return Images.killerImage }
}
var minimumPlayers: Int
{
switch self {
case .palermo:
return 5
case .spy:
return 3
case .killer:
return 3}
}
}
class Game
{
private var type: GameType
var title: String
{ return type.title}
var rolesTitle: String
{return self.type.rolesTitle}
var icon: UIImage
{return type.icon}
 var minimumPlayers: Int
{return type.minimumPlayers}
init(_ type: GameType)
{ self.type = type }
func getModuleInput(vc: UIViewController) -> GameInputProtocol? {
switch self.type {
case .palermo:
return SelectAvailableRolesModuleAssembly().configureModuleForViewInput(viewInput: vc, delegate: nil)
case .killer:
```

```
return SelectNumberOfBadModuleAssembly().configureModuleForViewInput(viewInput: vc)
case .spy:
return SelectNumberOfBadModuleAssembly().configureModuleForViewInput(viewInput: vc) }
}
var availableRoles: [Role] {
get {
switch self.type {
case .palermo:
return GameRolesService.getPalermoRoles()
case .killer:
return GameRolesService.getKillerRoles()
case .spy:
return GameRolesService.getSpyRoles()}}
}
var informationString: String {
get {
switch self.type {
case .killer:
return Localization.localized("information_killer")
case .palermo:
return Localization.localized("information_palermo")
case .spy:
return Localization.localized("information_spy") } }
}
func getRolesViewController() -> BaseViewController? {
switch self.type {
case .palermo:
let storyboard = UIStoryboard.storyboard(.Game)
return storyboard.instantiateViewController(withIdentifier: GameDetailsViewController.storyboardIdentifier)
as? GameDetailsViewController
case .killer:
let storyboard = UIStoryboard.storyboard(.NumberOfBad)
```

```
return storyboard.instantiateViewController(withIdentifier:
SelectNumberOfBadViewController.storyboardIdentifier) as? SelectNumberOfBadViewController

case .spy:

let storyboard = UIStoryboard.storyboard(.NumberOfBad)

return storyboard.instantiateViewController(withIdentifier:
SelectNumberOfBadViewController.storyboardIdentifier) as? SelectNumberOfBadViewController }

}

var badGuysSelectionTitle: String {

get {

switch self.type {

case .palermo:

return ""

case .spy:

return Localization.localized("games_numberOfSpies")

case .killer:

return Localization.localized("games_numberOfKillers")}

}

}

var hasCop: Bool

{ return self.type != .spy }

var badName: String {

get {

switch self.type {

case .killer:

return Localization.localized("games_killer")

case .spy:

return Localization.localized("games_spy")

case .palermo:

return "" }

}

}

var badImage: UIImage {

get {
```

Κώδικας

```
switch self.type {
case .killer:
return StyleKit.imageOfIcon_killer(mainIconPictureColor: Colors.fourthColor)
case .spy:
return StyleKit.imageOfIcon_spy(mainIconPictureColor: Colors.fourthColor)
case .palermo:
return StyleKit.imageOfIcon_killer()}
}
}
var calculatedDistanceForContraint: CGFloat {
switch self.type {
case .spy:
return 180.0
default:
return 100.0}
}
var showLocations: Bool {
get {
switch self.type {
case .spy:
return true
default:
return false}
}
}
var maxNumberOfBad: Int {
get {
switch self.type {
case .palermo:
return 0
case .spy:
return 3
```

```
case .killer:
return 2 }
}
}
func getImageBasedOnNumberOfBad(numberOfBad: Int) -> UIImage? {
switch self.type {
case .spy:
switch numberOfBad {
case 1:
return StyleKit.imageOfIcon_spies_1(mainBadGuysColor: Colors.fourthColor)
case 2:
return StyleKit.imageOfIcon_spies_2(mainBadGuysColor: Colors.fourthColor)
case 3:
return StyleKit.imageOfIcon_spies_3(mainBadGuysColor: Colors.fourthColor)
default:
return nil}
case .killer:
switch numberOfBad {
case 1:
return StyleKit.imageOfIcon_killers_1(mainBadGuysColor: Colors.fourthColor)
case 2:
return StyleKit.imageOfIcon_killers_2(mainBadGuysColor: Colors.fourthColor)
case 3:
return nil
default:
return nil
}
default:
return nil
}
}
```

Κώδικας

```
var isPalermo: Bool {
get {
return self.type == .palermo }
}
}
```