**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ**
**ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ**

**ΤΜΗΜΑ**
**ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# Κατασκευή δικτυακής εφαρμογής στην αρχιτεκτονική iOS iPhone που υλοποιεί ένα παιχνίδι ερωτοαπαντήσεων

Παρτώνας Αλέξανδρος

Επιβλέπων καθηγητής: Δασυγένης Μηνάς

Κοζάνη

Μάρτιος 2014

# Περιεχόμενα

# Κατάλογος Εικόνων

# Κατάλογος Πινάκων

# Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Δασυγένη Μηνά για την ανάθεση της εργασίας, την εμπιστοσύνη και ειδικότερα, για την πολύτιμη βοήθεια που μου προσέφερε σε κάθε φάση της υλοποίησης της.

Επίσης θα ήθελα να ευχαριστήσω τη γυναίκα μου και την οικογένεια μου για την αμέριστη υποστήριξη και συμπαράσταση κατά τη διάρκεια των σπουδών μου.

**Παρτώνας Αλέξανδρος**

# Περίληψη

Τα έξυπνα τηλέφωνα (Smart Phone) και οι υπολογιστές παλάμης (Tablet PC) έχουν γίνει πλέον κομμάτι της καθημερινότητας μας, κερδίζοντας ολοένα και περισσότερο έδαφος, τόσο σε επαγγελματική όσο και σε προσωπική χρήση. Η γκάμα των εφαρμογών για τις εν λόγω συσκευές μεγαλώνει μέρα με την μέρα.

Για τη δημιουργία της παρούσας διπλωματικής εργασίας απαιτήθηκε αρκετή προετοιμασία και μελέτη, τόσο μέσω διαδικτύου όσο και σχετικών συγγραμμάτων, καθώς προ της ενασχόλησής μου με το παρόν γνωστικό αντικείμενο δεν υπήρχε προηγούμενη επαφή με το περιβάλλον εργασίας (Mac OS X Mountain Lion), το λογισμικό ανάπτυξης εφαρμογών (Xcode) ή με τη γλώσσα προγραμματισμού (Objective-C). Ως εκ τούτου αυξήθηκε ο χρόνος έναρξης εκπόνησης της διπλωματικής εργασίας και της εφαρμογής.

Η mobile εφαρμογή που δημιουργήθηκε στα πλαίσια της διπλωματικής εργασίας πραγματεύεται ένα διαδικτυακό παιχνίδι ερωτοαπαντήσεων μέσω φορητών συσκευών iPhone και iPad, οι οποίες χρησιμοποιούν το λειτουργικό σύστημα iOS. Ο χρήστης θα έχει τη δυνατότητα να επιλέξει τον τρόπο του παιχνιδιού μεταξύ απλού παιχνιδιού (single player mode) ή διαδικτυακής μάχης (web based battle) μεταξύ περισσοτέρων του ενός χρήστες.

Σκοπός της εργασίας είναι να γίνει μία ορθολογική προσέγγιση του λειτουργικού μοτίβου που θα πρέπει να ακολουθείται όταν σχεδιάζονται mobile εφαρμογές, καθότι αυτές διαφέρουν ουσιαστικά από τον τρόπο σχεδιασμού των Desktop εφαρμογών.

**Λέξεις κλειδιά:** Σχεδιασμός εφαρμογής, παιχνίδι γνώσεων, iOS, iPhone, xCode, Application, Smartphone, SQLite Database, Bluetooth, WiFi.

# Abstract

Smart phones and tablet PCs have become a part of our everyday life, gaining more and more ground, both in professional and personal use. The range of applications for these devices is growing day by day.

For the creation of this thesis a lot of preparation and study has been required, both through the Internet and related writings, as prior to my involvement with this subject area there had been no previous contact with the working environment (Mac OS X Mountain Lion), the software of applications development (Xcode) or the programming language (Objective-C). Therefore, the onset time preparing the thesis and the application was increased.

The mobile application, which was created as a part of the thesis, deals with an online "question & answer" game via iPhone and iPad mobile devices that use the operating system iOS. The users will be able to select the game mode they prefer, choosing between simple gameplay (single player mode) and online battle (web based battle) amongst several users.

The objective of the thesis is to rationally approach the functional pattern that should be followed when designing mobile applications, because they substantially differ from the way Desktop applications are designed.

**Keywords**: Application design, quiz game, iOS, iPhone, xCode, Application, Smartphone, SQLite Database, Bluetooth, WiFi.

# Διάρθρωση κειμένου

Στο 1ο κεφάλαιο γίνεται μία εισαγωγή στο θέμα της διπλωματικής. Περιγράφεται συνοπτικά η λειτουργία της εφαρμογής και οι τεχνολογίες οι οποίες βοηθούν ώστε να πραγματοποιηθεί.

Στο 2ο κεφάλαιο περιγράφονται οι απαιτήσεις για τον σχεδιασμό του συστήματος και αναπτύσσεται η βάση δεδομένων της εφαρμογής. Επίσης περιγράφονται λεπτομερώς οι απαιτήσεις για τον σχεδιασμό της βάσης καθώς και των δεδομένων αυτής.

Το 3ο κεφάλαιο παρουσιάζει το υλικό, τις συσκευές και τις τεχνολογίες που χρησιμοποιήθηκαν για την επίτευξη της εργασίας.

Στο 4ο κεφάλαιο περιγράφεται η διαδικασία υλοποίησης της ιδέας στην πράξη και αναλύονται τα σημαντικότερα σημεία του κώδικα.

Στο 5ο κεφάλαιο γίνεται παρουσίαση του δικτυακού μέρους της εφαρμογής καθώς και ανάλυση της κατασκευής και λειτουργίας του.

Στο 6ο κεφάλαιο γίνεται μια σύνοψη για την εργασία. Αναφέρονται οι μελλοντικές επεκτάσεις της εφαρμογής και τα συμπεράσματα τα οποία προέκυψαν μετά την δημιουργία της.

# ΚΕΦΑΛΑΙΟ 1$^{ο}$

## Εισαγωγή

Το κεφάλαιο αυτό αποτελεί την εισαγωγή στο σύστημα της εφαρμογής του παιχνιδιού ερωτοαπαντήσεων καθώς και την γνωριμία με τις έξυπνες συσκευές, τις κινητές πλατφόρμες και τα διαφορετικά λειτουργικά συστήματα που υπάρχουν για αυτές.

## 1.1 Διαδικτυακό παιχνίδι ερωτοαπαντήσεων

Η εφαρμογή που δημιουργούμε πραγματεύεται ένα παιχνίδι γνώσεων, μεταξύ ενός ή περισσότερων παικτών, η οποία βασίστηκε στην ψυχαγωγία του χρήστη και πάνω από όλα στο να μπορεί ο χρήστης να αποκομίσει περεταίρω γνώσεις.

Η εφαρμογή αναπτύσσεται στο λειτουργικό σύστημα Mac OS X [1] για τις συσκευές iPhone και iPad οι οποίες χρησιμοποιούν το λειτουργικό σύστημα iOS.

### 1.1.1 Αρχιτεκτονική εφαρμογής

Η εφαρμογή έχει δύο επιλογές παιχνιδιού. Η πρώτη αφορά το δικτυακό παιχνίδι μεταξύ δύο ως τεσσάρων συσκευών (παικτών) και βασίζεται στο μοντέλο πελάτη – διακομιστή με χρήση της αρχιτεκτονικής peer to peer. Η δεύτερη επιλογή αφορά την απλή χρήση της εφαρμογής χωρίς δικτυακό μοντέλο.

Για την λειτουργία του συστήματος της εφαρμογής αρχικά δημιουργούμε μία βάση δεδομένων η οποία θα φιλοξενεί και θα εξάγει τις ερωτήσεις και τις απαντήσεις του παιχνιδιού.

Η εφαρμογή βασίζεται κυρίως στη διασύνδεση χρηστών μέσω των τεχνολογιών Bluetooth και Wi-Fi και αναπτύσσεται στην πλατφόρμα iOS SDK.

## 1.1.2 Bluetooth

Το Bluetooth [2] είναι ένα βιομηχανικό πρότυπο για ασύρματα προσωπικά δίκτυα υπολογιστών (Wireless Personal Area Networks, WPAN). Πρόκειται για μία ασύρματη τηλεπικοινωνιακή τεχνολογία μικρών αποστάσεων, η οποία μπορεί να μεταδώσει σήματα μέσω μικροκυμάτων σε ψηφιακές συσκευές. Επομένως το Bluetooth είναι ένα πρωτόκολλο το οποίο παρέχει προτυποποιημένη, ασύρματη επικοινωνία ανάμεσα σε PDA, κινητά τηλέφωνα, φορητούς υπολογιστές, προσωπικοί υπολογιστές, εκτυπωτές, καθώς και ψηφιακές φωτογραφικές μηχανές ή ψηφιακές κάμερες, μέσω μίας ασφαλούς, φθηνής και παγκοσμίως διαθέσιμης χωρίς ειδική άδεια ραδιοσυχνότητας μικρής εμβέλειας.

## 1.1.3 Wi-Fi

Το Wi-Fi [3] είναι μία δημοφιλής τεχνολογία που επιτρέπει σε μία ηλεκτρονική συσκευή την ανταλλαγή δεδομένων ή την σύνδεση της στο διαδίκτυο ασύρματα μέσω ραδιοκυμάτων. Η Wi-Fi Alliance ορίζει ως Wi-Fi όλα τα προϊόντα ασύρματου τοπικού δικτύου (WLAN) που βασίζονται στα πρότυπα 802.11 του Ινστιτούτου Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών (IEEE). Εντούτοις, δεδομένου ότι τα περισσότερα σύγχρονα δίκτυα WLAN βασίζονται σε αυτά τα πρότυπα, ο όρος "Wi-Fi" («ασύρματη πιστότητα» στα ελληνικά) έχει επικρατήσει γενικά ως συνώνυμο των ασύρματων τοπικών δικτύων «WLAN». Αυτή η οικογένεια πρωτοκόλλων αποτελεί το καθιερωμένο πρότυπο της βιομηχανίας στο χώρο των ασύρματων τοπικών δικτύων.

Ο όρος WiFi (Wireless Fidelity, κατά την ορολογία High Fidelity η οποία αφορά την εγγραφή ήχου) χρησιμοποιείται για να προσδιορίσει τις συσκευές που βασίζονται στην προδιαγραφή IEEE 802.11 b/g/n και εκπέμπουν σε συχνότητες 2.4GHz. Συνήθεις εφαρμογές του είναι η παροχή ασύρματων δυνατοτήτων πρόσβασης στο διαδίκτυο, τηλεφωνίας μέσω διαδικτύου (VoIP) [4] και διασύνδεσης μεταξύ ηλεκτρονικών συσκευών όπως τηλεοράσεις, ψηφιακές κάμερες, DVD Player και ηλεκτρονικούς υπολογιστές. Σε φορητές ηλεκτρονικές συσκευές το 802.11 βρίσκει εφαρμογές ασύρματης μετάδοσης, όπως στη μεταφορά φωτογραφιών από ψηφιακές κάμερες σε υπολογιστές για περαιτέρω

επεξεργασία και εκτύπωση, αν και στον τομέα αυτόν έχει υποσκελιστεί από το πρωτόκολλο Bluetooth για τα πολύ μικρότερης εμβέλειας ασύρματα προσωπικά δίκτυα.

## 1.2 Τι ονομάζεται κινητή πλατφόρμα

Κινητή πλατφόρμα ονομάζεται ένας υπολογιστής μεγέθους τσέπης, που συνήθως αποτελείται από μία οθόνη με δυνατότητα αφής ή ένα μικροσκοπικό πληκτρολόγιο. Επίσης είναι γνωστή ως κινητή συσκευή, συσκευή χειρός, υπολογιστής τσέπης ή απλά κινητό.

Σε ένα προσωπικό ψηφιακό υπολογιστή (PDA) ολόκληρη η αλληλεπίδραση μεταξύ του χρήστη και της συσκευής γίνεται μέσω μίας οθόνης αφής. Τα έξυπνα κινητά (smartphone) και τα PDA είναι πολύ διαδεδομένα στη σημερινή εποχή, κατά κύριο λόγο σε όσους έχουν ανάγκη τη βοήθεια και την ευκολία μερικών πτυχών των κανονικών ηλεκτρονικών υπολογιστών, όμως κινούνται συνήθως σε περιβάλλον όπου λόγω αυξημένου όγκου δεν θα ήταν δυνατό να χρησιμοποιηθούν ή να μεταφερθούν με ευκολία.

Οι κινητές πλατφόρμες είναι μία ποικιλία συσκευών που επιτρέπουν στο χρήστη να έχει πρόσβαση σε δεδομένα και πληροφορίες όπου και αν αυτός βρίσκεται. Σε αυτές τις συσκευές περιλαμβάνονται τα κινητά τηλέφωνα και οι κινητές συσκευές.

## 1.3 Είδη κινητής πλατφόρμας

Όπως γίνεται κατανοητό από τον παραπάνω ορισμό σήμερα υπάρχουν πολλά είδη κινητής πλατφόρμας. Η συνοπτική παρουσίαση εκείνων που συνετέλεσαν εξελικτικά στη σημερινή τους μορφή περιγράφεται παρακάτω.

### 1.3.1 Προσωπικός ψηφιακός βοηθός (PDA)

Ένας προσωπικός ψηφιακός υπολογιστής (PDA) [5], γνωστός και ως υπολογιστής παλάμης (Palmtop Computer) είναι μία κινητή πλατφόρμα που λειτουργεί σαν προσωπικός διαχειριστής πληροφοριών. Τα σημερινά

PDA έχουν την δυνατότητα να συνδέονται στο διαδίκτυο μέσω ασύρματων δικτύων (Wi-Fi) και μέσω της οθόνης τους μπορούν να περιλαμβάνουν πρόγραμμα περιήγησης σε αυτό. Τα σημερινά μοντέλα προσφέρουν πολλές άλλες δυνατότητες, όπως για παράδειγμα το ότι φέρουν μικρόφωνο, ακουστικό αλλά και ηχείο, επιτρέποντας τη χρήση τους ως κινητά τηλέφωνα αλλά και φορητά ηχοσυστήματα. Τα περισσότερα PDA χρησιμοποιούν οθόνη με τεχνολογία αφής.

Ο όρος PDA χρησιμοποιήθηκε για πρώτη φορά τον Ιανουάριο του 1992 από τον πρόεδρο της εταιρίας Apple, John Sculley σε μία έκθεση ηλεκτρονικών συσκευών, την CES (Consumer Electronics Show) στο Las Vegas, και αναφερόταν στη συσκευή Apple Newton. Το Apple Newton ήταν βασισμένο σε επεξεργαστή ARM και υποστήριζε αναγνώριση γραφής με μία ειδική γραφίδα. Το 1996 η Nokia δημιούργησε το πρώτο κινητό τηλέφωνο με όλες τις λειτουργίες ενός PDA, το 9000 Communicator, το οποίο έγινε το πρώτο σε πωλήσεις παγκοσμίως. Με την είσοδο αυτής της συσκευής στην αγορά δημιουργήθηκε ουσιαστικά ο όρος PDA τηλέφωνο που στην εποχή μας ονομάζουμε Smartphone.


Εικόνα 1.1: Το PDA Acer n10

## 1.3.2 Έξυπνα κινητά (Smartphone)

Έξυπνο κινητό (Smartphone) [6] ονομάζεται ένα κινητό τηλέφωνο που προσφέρει σαφώς πιο ανεπτυγμένες πληροφοριακές δυνατότητες αλλά και συνδεσιμότητας σε σύγκριση με ένα κοινό τηλέφωνο. Τα smartphone επιτρέπουν στο χρήστη να χρησιμοποιεί πολλές εφαρμογές ταυτόχρονα, εφόσον αυτές είναι σχεδιασμένες έτσι ώστε να αξιοποιούν

το υλικό (Hardware) της κάθε συσκευής. Τα smartphone χρησιμοποιούν ολοκληρωμένα λειτουργικά συστήματα, παρέχοντας στους προγραμματιστές μία πλατφόρμα στην οποία μπορούν να αναπτύξουν τις εφαρμογές τους. Αποτέλεσμα των παραπάνω είναι η δυνατότητα συνδυασμού όλων των δυνατοτήτων ενός κινητού τηλεφώνου εξοπλισμένου με κάμερα, με τις δυνατότητες ενός PDA.

Τα έξυπνα κινητά γίνονται όλο και ελκυστικότερα για τους καταναλωτές όπως φανέρωσαν έρευνες που έγιναν στις αρχές του 2011, καθώς στο Ηνωμένο Βασίλειο το 22% αυτών κατέχει ήδη ένα, με το ποσοστό να αυξάνεται στο 31% για τις ηλικίες μεταξύ 24-35 ετών. Η αύξηση της ζήτησης για περισσότερο εξελιγμένες κινητές συσκευές, οι οποίες θα χρησιμοποιούν δυνατότερους επεξεργαστές, περισσότερη μνήμη, μεγαλύτερες οθόνες αλλά και πιο 'ελεύθερα' (Open Source) λειτουργικά συστήματα, έχει ως αποτέλεσμα τα smartphone να κατέχουν το μεγαλύτερο ποσοστό της αγοράς κινητών τηλεφώνων τα τελευταία χρόνια. Σύμφωνα με άλλη πρόσφατη έρευνα στις Ηνωμένες Πολιτείες, από τα 234 εκατομμύρια συνδρομητών κινητής τηλεφωνίας πάνω από 45 εκατομμύρια άνθρωποι χρησιμοποιούν smartphone. Αν και αυτό αφορά μόνο το 20% των συνδρομητών, είναι αποκαλυπτικό πως στον τελευταίο χρόνο παρατηρήθηκε αύξηση των πωλήσεων smartphone περίπου 74%.


Εικόνα 1.2: Το smartphone Lg LS860 Cayenne

## 1.3.3 Ταμπλέτες (Tablet PC)

Ο όρος ταμπλέτα (tablet) [7] αναφέρεται στους μικρούς φορητούς υπολογιστές, που δε διαθέτουν πληκτρολόγιο και ποντίκι - αλλά οθόνη αφής. Χωρίζονται σε δύο κατηγορίες: τα Tablet PC, δηλαδή ταμπλέτες που διαθέτουν κάποιο λειτουργικό σύστημα και λειτουργούν ως μικρά

Laptop και τα Web Tablet, δηλαδή ταμπλέτες για αποκλειστική χρήση πλοήγησης στο διαδίκτυο και εφαρμογών πολυμέσων.

Η ταμπλέτα (Tablet PC) είναι ένας ηλεκτρονικός υπολογιστής μεγέθους συνήθως από 6 έως 10 ίντσες με τα βασικά χαρακτηριστικά αυτά ενός κανονικού Η/Υ και συνήθως περιλαμβάνει ένα πλήρες λειτουργικό σύστημα. Για την εισαγωγή δεδομένων χρησιμοποιεί κυρίως την οθόνη σαν εικονικό πληκτρολόγιο, μία παθητική γραφίδα - ή μία ψηφιακή πένα. Όλες οι ταμπλέτες έχουν δυνατότητα ασύρματης σύνδεσης στο διαδίκτυο μέσω δικτύων Wi-Fi, αλλά και ενσύρματης. Το λογισμικό τους περιλαμβάνει εφαρμογές γραφείου, προγράμματα περιήγησης στο διαδίκτυο, παιχνίδια, αλλά και μία τεράστια γκάμα εφαρμογών τις οποίες υποστηρίζει το λειτουργικό τους σύστημα. Ωστόσο στην περίπτωση απαιτητικότερων εφαρμογών τα συστήματα αυτά δεν μπορούν συνήθως να ανταπεξέλθουν λόγο της περιορισμένης υπολογιστικής τους δύναμης.

Ο όρος Tablet PC έγινε γνωστός από τη Microsoft το 2001 με την παρουσίαση προϊόντων με το όνομα Microsoft Tablet Pc όπου και χρησιμοποιήθηκε για πρώτη φορά. Όμως ο όρος πλέον χρησιμοποιείται από μία ευρεία γκάμα μοντέλων και προϊόντων προσωπικών ηλεκτρονικών υπολογιστών με αυτό το μέγεθος, ακόμα και αν δεν χρησιμοποιούν το λειτουργικό σύστημα της Microsoft.


Εικόνα 1.3: Η ταμπλέτα Microsoft Surface Tablet

## 1.4 Λειτουργικά συστήματα για κινητές πλατφόρμες

Για τον έλεγχο μίας κινητής πλατφόρμας απαιτείται ένα λειτουργικό σύστημα γνωστό και ως Mobile Operating System [8], το οποίο έχει τα

ίδια χαρακτηριστικά με ένα λειτουργικό σύστημα ηλεκτρονικών υπολογιστών όπως τα Windows, Mac OS και Linux. Τα λειτουργικά συστήματα για κινητές πλατφόρμες είναι κατασκευασμένα με τρόπο ώστε να απαιτούν λιγότερους υπολογιστικούς πόρους σε σχέση με ένα ηλεκτρονικό υπολογιστή, ενώ σχετίζονται περισσότερο με ασύρματες επικοινωνίες, τοπικά δίκτυα και διαφορετικά αρχεία πολυμέσων με διαφορετικούς τρόπους εισαγωγής εντολών.

## 1.5 Τα πρώτα λειτουργικά συστήματα

Θα αναφέρουμε πρώτα τα λειτουργικά συστήματα που χρονικά θεωρούνται τα πρώτα για κινητές πλατφόρμες, καθώς πολλά στοιχεία τους βοήθησαν στη σημερινή εξέλιξη των λειτουργικών συστημάτων για αυτές. Η ανάπτυξη και εξέλιξη ορισμένων από αυτά έχει πλέον σταματήσει, εφόσον αντικαταστάθηκαν από καινούργια λειτουργικά.

### 1.5.1 Palm OS

Ίσως το πρώτο λειτουργικό σύστημα, με όλα τα χαρακτηριστικά που ένα τέτοιο πρέπει να έχει - για κινητή πλατφόρμα, το Palm OS [9] έκανε τα πρώτα του βήματα στην αγορά το 1996. Αρχικά αναπτύχθηκε από την Palm και χρησιμοποιήθηκε σε pda. Το Palm OS σχεδιάστηκε με βάση την ευκολία χρήσης μίας οθόνης αφής και βασίστηκε στην αλληλεπίδραση του χρήστη με αυτή μέσω ενός γραφικού περιβάλλοντος. Προσέφερε μία σουίτα βασικών εφαρμογών με σκοπό τη διαχείριση προσωπικών πληροφοριών. Με την πάροδο των ετών, νέες εκδόσεις του λειτουργικού συστήματος υποστήριξαν και smartphone.

Υπήρξαν διάφορες εκδόσεις μέχρι και το 2004, με καθεμία από αυτές να προσθέτει όλο και περισσότερες λειτουργίες στοχεύοντας στην προσαρμογή του λειτουργικού συστήματος στις νέες απαιτήσεις της αγοράς. Η τελευταία σταθερή έκδοση λειτουργικού είναι η Garnet OS 5.6. Το 2009 το Palm OS άλλαξε και μετονομάστηκε σε Palm webOS το οποίο υιοθέτησε τις λειτουργίες του web OS, ένα λειτουργικό σύστημα βασισμένο στο Linux.


Εικόνα 1.4: Το Palm webOs στην έκδοση 1.4.5

## 1.5.2 Windows Mobile

Το Windows Mobile [10] είναι ένα λειτουργικό σύστημα για κινητές πλατφόρμες που αναπτύχθηκε από την Microsoft βασισμένο στα Windows CE και χρησιμοποιείται σε smartphone και pda. Έκανε την εμφάνιση του το 2000 ως το λειτουργικό σύστημα των υπολογιστών τσέπης.

Τα Windows Mobile 6.5, μία από τις τελευταίες εκδόσεις των Windows Mobile, περιλαμβάνουν μία σουίτα με βασικές εφαρμογές που αναπτύχθηκαν από την Microsoft και εφαρμογές τρίτων εταιρειών που υποστήριξαν το λειτουργικό αυτό σύστημα. Ήταν σχεδιασμένα έτσι ώστε να μοιάζουν όσο το δυνατόν περισσότερο με την έκδοση για ηλεκτρονικούς υπολογιστές των Windows, τόσο από άποψη λειτουργιών όπως και από αισθητικής πλευράς. Οι περισσότερες συσκευές που έτρέχαν Windows Mobile είχαν ένα μικρό στυλό (stylus) έτσι ώστε να είναι ευκολότερη η χρήση της οθόνης αφής.

Η τελευταία έκδοση των Windows Mobile ήρθε από τη Microsoft το 2010, η 6.5.5. Η εταιρεία ανακοίνωσε τότε και ένα νέο λειτουργικό σύστημα για smartphone, τα Windows Phone 7, σταματώντας παράλληλα την υποστήριξη των κινητών τηλεφώνων που χρησιμοποιούν το λειτουργικό Windows Mobile.


Εικόνα 1.5: Η τελευταία έκδοση των Windows Mobile 6.5

## 1.6 Τα λειτουργικά συστήματα νέας γενιάς

Τα τελευταία χρόνια τα λειτουργικά συστήματα για κινητές πλατφόρμες έχουν σημειώσει βελτίωση σε πολλούς τομείς όπως είναι η ευκολία χρήσης, το φιλικότερο περιβάλλον προς το χρήστη και η υποστήριξη πολλών διαφορετικών εφαρμογών. Η κατηγορία λογισμικού (software) για κινητές πλατφόρμες έχει γίνει πλέον πλήρως ανταγωνιστική, με αποτέλεσμα να κινήσει το ενδιαφέρον εταιρειών κολοσσών ανάπτυξης λογισμικού, των Google, Microsoft και Apple, αλλά και των εταιρειών που προηγήθηκαν στην κατασκευή κινητών συσκευών όπως των Nokia, Research in Motion και Palm.

Με την είσοδο του iPhone στην αγορά στο 2007, η Apple άλλαξε σε μεγάλο βαθμό την αγορά κινητών συσκευών και ως ένα βαθμό προκάλεσε την γέννηση νέων λειτουργικών συστημάτων για κινητές πλατφόρμες. Το Νοέμβριο του 2007 η Google ανακοίνωσε την δημιουργία του λειτουργικού συστήματος Android [11]. Με το λανσάρισμα και του λειτουργικού συστήματος της Google, η αγορά των smartphone έχει αυξηθεί σημαντικά. Αυτό οδήγησε του καταναλωτές στη γνωριμία με τα διάφορα λειτουργικά συστήματα και τους κατασκευαστές, όπως και τις εταιρείες τηλεπικοινωνιών στη διαφήμιση των πλεονεκτημάτων του κάθε λειτουργικού συστήματος.

Η νέα γενιά λειτουργικών συστημάτων υποστηρίζεται από εφαρμογές τρίτων εταιρειών αλλά και από εφαρμογές που κατασκευάζουν ανεξάρτητοι προγραμματιστές. Πλέον όλα τα νέα λειτουργικά συστήματα συνοδεύονται από δικό τους λογισμικό ανάπτυξης εφαρμογών, ώστε να μπορεί οποιοδήποτε θέλει να ασχοληθεί και να αναπτύξει εφαρμογές για το εκάστοτε λειτουργικό σύστημα.

Παρακάτω θα παρουσιάσουμε τα βασικά νέα λειτουργικά συστήματα βάσει της ευρύτητας χρήσης τους, καθώς και το λογισμικό ανάπτυξης εφαρμογών που διαθέτουν.

## 1.6.1 Blackberry OS (RIM)

Ως κινητό λειτουργικό σύστημα το Blackberry OS [12] έκανε την εμφάνισή του πρώτη φορά το 2005. Για την ανάπτυξή του υπεύθυνη είναι η εταιρεία Research in Motion και χρησιμοποιείται στα smartphone Blackberry.

Το λειτουργικό αυτό σύστημα δίνει την δυνατότητα χρήσης πολλαπλών εφαρμογών ταυτόχρονα και είναι ειδικά φτιαγμένο ώστε να υποστηρίζει τις συγκεκριμένες συσκευές εισόδου δεδομένων που χρησιμοποιεί η Research in Motion στα κινητά της τηλέφωνα. Παράδειγμα τέτοιας συσκευής είναι το trackwheel, το οποίο με την ανάπτυξη και βελτίωση των κινητών τηλεφώνων επίσης αντικαταστάθηκε στην πορεία, πρώτα με το trackball και έπειτα με το trackpad.

Αρχικός στόχος της πλατφόρμας Blackberry ήταν η υποστήριξη εταιρικών εφαρμογών όπως Email και για αυτό το λόγο χρησιμοποιείται κυρίως σε εταιρικό επίπεδο.

Τα τελευταία χρόνια έχει αυξηθεί η υποστήριξη του λειτουργικού από τρίτες εταιρίες ανάπτυξης λογισμικού με αποτέλεσμα την πληθώρα εφαρμογών του. Το Blackberry OS διαθέτει δική του εφαρμογή για την φιλοξενία των εφαρμογών το Blackberry App World , το οποίο με την ανακοίνωση της έκδοσης BlackBerry 10 μετονομάστηκε πιο απλά σε Blackberry World.

Για την ανάπτυξη εφαρμογών στη συγκεκριμένη πλατφόρμα χρησιμοποιείται το ολοκληρωμένο περιβάλλον ανάπτυξης QNX Momentics IDE (intergrated development environment) ενώ οι υποστηριζόμενες γλώσσες προγραμματισμού είναι οι Java, C/C++, HTML5/JAVASCRIPT/CSS .


Εικόνα 1.6: Το Blackberry OS 10

## 1.6.2 Symbian OS

Το Symbian OS [13] αποτελεί λειτουργικό σύστημα για φορητές συσκευές και είναι απόγονος του λειτουργικού συστήματος EPOC (γραφικό λειτουργικό σύστημα για PDA) της εταιρείας Psion. Η μεγαλύτερη επιτυχία του λειτουργικού ήρθε με την πλατφόρμα S60 από τη Nokia το 2002. Παράλληλα χρησιμοποιούνταν μία ακόμη πλατφόρμα με το όνομα UIQ (User Interface Quartz) η οποία όμως δεν ήταν συμβατή με την S60.

Με την δημιουργία του Symbian Platform το 2009, τα τρία περιβάλλοντα χρήστη ενώθηκαν σε ένα, το οποίο εξαγοράστηκε από την Nokia και στην συνεχεία μετατράπηκε σε λογισμικό ανοικτού κώδικα. Τον Αύγουστο του 2011 ήρθε η τελευταία αναβάθμιση Nokia Belle feature pack 2 του Symbian OS, ενώ τον Φεβρουάριο του ίδιου έτους ανακοινώθηκε από την Nokia ότι τα Windows Phone θα είναι το βασικό λειτουργικό σύστημα των συσκευών της.

Για την ανάπτυξη εφαρμογών χρησιμοποιείται το Symbian SDK με γλώσσα προγραμματισμού την C++ σε συνδυασμό με το Qt, ένα Framework εφαρμογών που χρησιμοποιείται από πολλές πλατφόρμες. Μπορεί να χρησιμοποιηθεί είτε με το Qt Creator είτε με το Carbide.C++, ένα παλιότερο IDE που χρησιμοποιείται για ανάπτυξη εφαρμογών Symbian. Επίσης υπάρχει ένας εξομοιωτής για τη δοκιμή των

εφαρμογών, που τρέχει τον κώδικα απευθείας αντί να προσομοιώνει ολόκληρη την λειτουργία του κινητού τηλεφώνου.


Εικόνα 1.7: Η οθόνη υποδοχής του Symbian Belle

## 1.6.3 Windows Phone

Το 2010 η Microsoft ανακοίνωσε το νέο λειτουργικό της σύστημα για κινητές πλατφόρμες, διάδοχο των Windows Mobile [10], τα Windows Phone [14]. Το νέο λειτουργικό σύστημα περιλαμβάνει ένα πλήρως διαφορετικό και νέο περιβάλλον χρήσης που δημιουργήθηκε με τη γλώσσα σχεδίασης της ίδιας της εταιρείας, της Metro (Microsoft design language). Υποστηρίζει πλήρως τις υπηρεσίες της Microsoft όπως το Windows Live, το SkyDrive και το Office, το Xbox Live, Xbox Music, Xbox Video και το Bing. Επίσης υποστηρίζει και υπηρεσίες τρίτων εταιριών όπως το Facebook, το Twitter και τα Google Accounts. Η νεότερη έκδοση του λειτουργικού είναι η έκδοση Windows Phone 8.

Στα Windows Phone έχει δημιουργηθεί και το Windows Phone Store για την ψυχαγωγία των χρηστών και την φιλοξενία των εφαρμογών, το οποίο χρησιμοποιείται για την ψηφιακή διανομή μουσικής, βίντεο και εφαρμογές τρίτων. Το κατάστημα διαχειρίζεται η Microsoft η οποία παρέχει και την έγκριση των εφαρμογών που φιλοξενούνται.

Στην πλατφόρμα αυτή οι εφαρμογές πρέπει να βασίζονται στο framework XNA ή σε μία συγκεκριμένη έκδοση του Silverlight που υποστηρίζει τα Windows Phone 7. Για να υπάρχει η δυνατότητα

σχεδίασης και δοκιμής εφαρμογών με το Visual Studio 2010 ή το Visual Studio 2010 Express, η Microsoft προσφέρει τα Windows Phone Developer Tools ως επέκταση. Αυτό το σετ εργαλείων υποστηρίζει υπολογιστές που χρησιμοποιούν Windows Vista SP2 ή νεότερα.


Εικόνα 1.8: Τα Windows Phone 8

## 1.6.4 Google Android

Το Android [11] είναι λειτουργικό σύστημα ανοιχτού κώδικα βασισμένο στο λειτουργικό Linux. Αρχικά σχεδιάστηκε για συσκευές με οθόνες αφής όπως κινητά τηλέφωνα και tablet. Η παραγωγή λογισμικού γίνεται με την χρήση της γλώσσας προγραμματισμού Java ελέγχοντας την συσκευή μέσω βιβλιοθηκών λογισμικού ανεπτυγμένων από την Google.

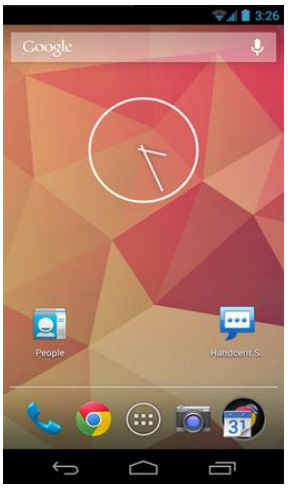
Το Android αρχικά αναπτύχθηκε από μία μικρή εταιρία λογισμικού η οποία εξαγοράστηκε από την Google το 2005. Η πρώτη παρουσίαση της πλατφόρμας Android έγινε τον Νοέμβριο του 2007, παράλληλα με την ανακοίνωση της ίδρυσης του οργανισμού Open Handset Alliance, μίας κοινοπραξίας 84 τηλεπικοινωνιακών εταιρειών παραγωγής λογισμικού και κατασκευής hardware.

Η Google δημοσιεύει το μεγαλύτερο μέρος του κώδικα του Android υπό τους όρους μίας ελεύθερης άδειας λογισμικού της Apache Licence. Αυτό επιτρέπει στους κατασκευαστές να μπορούν να διαμορφώσουν το λειτουργικό σύμφωνα με τις δυνατότητες των συσκευών. Επίσης υπάρχει μία μεγάλη κοινότητα προγραμματιστών που ασχολείται με τον προγραμματισμό στο Android και με αυτό τον τρόπο αυξάνει τις

δυνατότητες των συσκευών που το χρησιμοποιούν. Το ηλεκτρονικό κατάστημα που έχει φτιάξει η Google με το όνομα Google Play, φιλοξενεί αυτή την στιγμή περισσότερες από 700.000 εφαρμογές. Η πιο πρόσφατη έκδοση του Android είναι η 4.3 Jelly Bean.

Για την ανάπτυξη εφαρμογών χρησιμοποιείται το Android Software Development Kit το οποίο περιλαμβάνει ένα μεγάλο σύνολο εργαλείων ανάπτυξης. Σε αυτό περιλαμβάνονται ένας debugger, βιβλιοθήκες, ένας εξομοιωτής, βιβλιογραφία καθώς και δείγματα κώδικα. Αυτή τη στιγμή οι πλατφόρμες που υποστηρίζονται περιλαμβάνουν υπολογιστές που χρησιμοποιούν Linux (οποιαδήποτε νέα έκδοση), Mac OS X 10.5.8 ή νεότερο, Windows XP ή νεότερο. Το επίσημο περιβάλλον ανάπτυξης είναι το Eclipse με ταυτόχρονη χρησιμοποίηση των Android Development Tools αν και δίνεται η δυνατότητα χρησιμοποίησης οποιουδήποτε κειμενογράφου για την σύνταξη κώδικα Java ή XML. Με κάθε νέα έκδοση του λειτουργικού συστήματος δημιουργείται και μία νέα έκδοση του Android SDK, χωρίς να σταματά την υποστήριξη ανάπτυξης εφαρμογών για την προηγούμενη έκδοση του λειτουργικού.

Υπάρχουν επίσης και άλλοι τρόποι δημιουργίας εφαρμογών για το Android με τη χρήση του Native Development Kit το οποίο μπορεί να συντάξει βιβλιοθήκες γραμμένες σε C ή C++ κώδικα που χρησιμοποιούν οι επεξεργαστές ARM. Επίσης η Google παρέχει για τη δημιουργία εφαρμογών το App Invertor, ένα γραφικό περιβάλλον ανάπτυξης προγραμμάτων το οποίο βασίζεται σε Web τεχνολογίες και προορίζεται για νέους προγραμματιστές. Αυτό δείχνει τα προτερήματα ενός λειτουργικού που έχει τόσο ανοικτή αρχιτεκτονική.




Εικόνα 1.9: Το Google Android 4.3 Jelly Bean

## 1.6.5 Apple iOS

Οι κινητές πλατφόρμες της Apple χρησιμοποιούν το λειτουργικό σύστημα iOS (γνωστό και ως iPhone OS) [15]. Αρχικά αναπτύχθηκε μόνο για το iPhone ενώ στην πορεία επεκτάθηκε ώστε να υποστηρίζει και άλλες φορητές συσκευές της Apple όπως τα iPod Touch, το iPad και το Apple TV. Αντίθετα από το Windows Phone της Microsoft και το Android της Google, η Apple δε δίνει την άδεια εγκατάστασης του λογισμικού iOS σε συσκευές που δεν είναι κατασκευής Apple. Ένα από τα μεγάλα πλεονεκτήματα του είναι το App Store το οποίο περιέχει περισσότερες από 900.000 εφαρμογές εκ των οποίων οι 375.000 περίπου είναι πλέον συμβατές με το iPad σύμφωνα με την τελευταία μέτρηση που έχει γίνει τον Αύγουστο του 2013.

Το περιβάλλον χρήσης του βασίζεται στην άμεση αλληλεπίδραση του χρήστη με την οθόνη αφής της συσκευής. Τα στοιχεία ελέγχου που απαρτίζουν το γραφικό περιβάλλον είναι διακόπτες, κουμπιά και ολισθητές (sliders). Για παράδειγμα ο χρήστης μέσω της οθόνης αφής πολλαπλών σημείων μπορεί να χρησιμοποιεί διάφορες κινήσεις των δακτύλων του και να παίρνει άμεσα τα αποτελέσματα στην οθόνη. Μπορεί να εστιάσει σε μία σελίδα κειμένου μεγεθύνοντας τη με το άνοιγμα των δυο δακτύλων του ή μπορεί να αλλάζει φωτογραφίες με μία απλή κίνηση του δακτύλου του από δεξιά προς τα αριστερά. Αυτός ο απλός τρόπος χρήσης έκανε το λειτουργικό σύστημα να ξεχωρίζει σε σχέση με τον ανταγωνισμό, ειδικά την περίοδο που παρουσιάστηκε στις αρχές του 2007. Είναι βασισμένο στο λειτουργικό σύστημα Mac OS X το οποίο βασίζεται στο UNIX. Για την εγκατάσταση του το λειτουργικό σύστημα δεσμεύει περίπου 500 MB μνήμης από τον αποθηκευτικό χώρο της συσκευής.

Για την ανάπτυξη εφαρμογών στο περιβάλλον iOS χρησιμοποιείται το λογισμικό ανάπτυξης εφαρμογών iOS SDK, το οποίο αναπτύχθηκε από την Apple και δόθηκε στους προγραμματιστές τον Φεβρουάριο του 2008. Οι προγραμματιστές έχουν την δυνατότητα να δημιουργήσουν εφαρμογές και να τις δοκιμάσουν σε ένα εξομοιωτή που ονομάζεται iPhone Simulator. Για την εγκατάσταση μίας εφαρμογής σε πραγματική συσκευή καθώς και για την πώληση της μέσω του App Store, κάθε προγραμματιστής θα πρέπει να είναι εγγεγραμμένος στο πρόγραμμα των

προγραμματιστών iPhone της Apple με κόστος 99 ευρώ το χρόνο. Κάθε δημιουργός εφαρμογής έχει τη δυνατότητα πώλησής της σε οποιαδήποτε τιμή πάνω από την μικρότερη επιτρεπτή τιμή λαμβάνοντας ως κέρδος το 70% του ποσού και με το υπόλοιπο 30% να αντιστοιχεί στο κέρδος της Apple. Εναλλακτικά, μπορεί να διαθέτει την εφαρμογή δωρεάν και να μη ζημιώνεται καθόλου από τα έξοδα κυκλοφορίας και διανομής, πέραν των εξόδων εγγραφής.

Το iOS SDK χρησιμοποιεί το ίδιο πρόγραμμα γραφής κώδικα που χρησιμοποιεί και το Mac OS X, το Xcode, και περιλαμβάνει τον iPhone Simulator και τον iPad Simulator, ένα πρόγραμμα που μπορεί να χρησιμοποιεί ο προγραμματιστής για να εξομοιώσει το πως θα φαίνονταν οι εφαρμογές και το πως θα δούλευαν αν έτρεχαν στο iPhone ή στο iPad. Η γλώσσα προγραμματισμού είναι η Objective C και οι απαιτήσεις συστήματος για την χρήση του SDK της Apple είναι ένας υπολογιστής Intel Mac με λειτουργικό σύστημα Mac OS X Leopard ή νεότερο. Παλαιότερες εκδόσεις του λειτουργικού Mac OS X δεν υποστηρίζονται, όπως άλλωστε και άλλα λειτουργικά συστήματα όπως για παράδειγμα τα Windows.


Εικόνα 1.10: Η οθόνη υποδοχής του iOS 7

## 1.7 Εφαρμογές κινητών πλατφορμών

Με την αλλαγή στον τρόπο χρήσης των κινητών πλατφορμών και κινητών συσκευών που έχουν φέρει τα νέα λειτουργικά συστήματα τα τελευταία χρονιά ένα από τα πιο σημαντικά πράγματα που έχουν

καταφέρει, είναι η δημιουργία μέσω αυτών, πολλών νέων και εντυπωσιακών εφαρμογών για τέτοιου είδους συστήματα.

Οι εφαρμογές για κινητές πλατφόρμες [16] είναι ένα μέρος της παγκόσμιας αγοράς κινητών συσκευών που μεγαλώνει και αναπτύσσεται ραγδαία. Αποτελούνται από λογισμικό που 'τρέχει' σε μία κινητή πλατφόρμα και εκτελεί συγκεκριμένες λειτουργίες για τον χρήστη του. Αυτές οι κινητές (mobile) εφαρμογές χρησιμοποιούνται σε πλήθος μοντέλων κινητών τηλεφώνων, ακόμα και σε συσκευές χαμηλού κόστους στην αγορά.

Στα νέα λειτουργικά συστήματα, μπορεί κάποιος να τις προμηθευτεί κατεβάζοντας τες από συγκεκριμένα ηλεκτρονικά καταστήματα εφαρμογών. Η ευρεία χρησιμοποίησή τους υπάρχει λόγω των πολλών λειτουργιών που μπορούν να πραγματοποιούν, περιλαμβάνοντας από απλά περιβάλλοντα χρήσης για βασικές υπηρεσίες τηλεφωνίας και μηνυμάτων, μέχρι εξελιγμένες υπηρεσίες όπως τα βιντεοπαιχνίδια και εφαρμογές πολυμέσων. Οι κατηγορίες των εφαρμογών αυτών είναι πολλές. Εφαρμογές σαν αυτές που χρησιμοποιούνται για την αποστολή και λήψη SMS/MMS, προγράμματα περιήγησης στο διαδίκτυο και εφαρμογές αναπαραγωγής πολυμέσων όπως mp3 players, έρχονται εγκατεστημένες στα λειτουργικά συστήματα των συσκευών ενώ οι υπόλοιπες μπορούν να εγκατασταθούν μετά την αγορά της συσκευής. Για παράδειγμα ο χρήστης μπορεί να κατεβάσει εφαρμογές μέσω του ασύρματου δικτύου και να τις εγκαταστήσει ο ίδιος ή μπορεί να την φορτώσει και να εγκατασταθεί από το ηλεκτρονικό κατάστημα που έρχεται μαζί με το λειτουργικό.

Μπορούμε να χωρίσουμε τις εφαρμογές από τεχνικής άποψης σε κατηγορίες σε σχέση με το προγραμματιστικό περιβάλλον στο οποίο εκτελούνται:

- Εφαρμογές που τρέχουν στο περιβάλλον του λειτουργικού συστήματος όπως εφαρμογές που τρέχουν σε iOS, Android, Symbian OS, Windows Phone και Blackberry OS
- Εφαρμογές που τρέχουν σε περιβάλλον περιήγησης στο διαδίκτυο όπως τα Webkit, Mozilla Firefox, Opera Mini κα RIM
- Άλλες πλατφόρμες και εικονικά συστήματα όπως τα Java/J2ME, BREW, Flash Lite και Silverlight

Επίσης μπορούμε να τις χωρίσουμε με τις λειτουργίες τους για κινητές πλατφόρμες ως εξής:

## *1.7 Εφαρμογές κινητών πλατφορμών*

- Εφαρμογές επικοινωνιών όπως email, μηνυμάτων, περιήγησης στο διαδίκτυο, πληροφοριών και κοινωνικών δικτύων
- Εφαρμογές παράγωγης όπως αριθμομηχανές, ημερολόγια, υπενθυμίσεων, σημειώσεων, επεξεργασίας λέξεων, λογιστικών φύλλων, υπηρεσιών GPS και τραπεζικών υπηρεσιών
- Εφαρμογές πολυμέσων όπως αναπαραγωγής ήχου και ροής δεδομένων ήχου και εικόνας, γραφικών και εικόνας, αναπαραγωγής βίντεο και παρουσίασης
- Εφαρμογές παιχνιδιών όπως στρατηγικής, τράπουλας και καζίνο, δράσης και περιπέτειας, πάζλ, αθλητικές και χόμπι.

Τα τελευταία χρόνια οι εφαρμογές για κινητές πλατφόρμες έχουν εξελιχθεί ως ένα σημείο που προσφέρουν στον χρήστη μία πλούσια και γρήγορη εμπειρία χρήσης. Από αυτή την άποψη αυτές οι εφαρμογές έχουν χαρακτηριστικές διάφορες από την πλοήγηση σε ιστοσελίδες φτιαγμένες για κινητά συστήματα (Mobile Web) οπού ακόμα χαρακτηρίζονται από προβλήματα πρόσβασης αλλά και χαμηλές ταχύτητες στο δίκτυο κινητής τηλεφωνίας.

# ΚΕΦΑΛΑΙΟ 2ο

## Ανάλυση και σχεδίαση βάσης δεδομένων

Στο προηγούμενο κεφάλαιο δώσαμε έναν ορισμό στις κινητές πλατφόρμες και στο νέο έξυπνο κινητό τηλέφωνο. Στη συνέχεια αναλύσαμε τα πρώτα λειτουργικά συστήματα που συνετέλεσαν στην σημερινή μορφή τους και τον τρόπο αξιοποίησης των χαρακτηριστικών ενός έξυπνου κινητού τηλεφώνου ανάλογα με το λειτουργικό το οποίο διαθέτει. Παρουσιάσαμε τα δημοφιλέστερα λειτουργικά συστήματα έξυπνων κινητών τηλεφώνων και αναφέραμε τις γλώσσες προγραμματισμού των εφαρμογών που τρέχουν σε αυτά και τέλος τα προγράμματα ανάπτυξης εφαρμογών έξυπνων κινητών τηλεφώνων ανάλογα με το λειτουργικό.

Στο κεφάλαιο αυτό γίνεται η ανάλυση του συστήματος της βάσης δεδομένων. Για την αποτελεσματική υλοποίηση της βάσης αυτής γίνεται η αναγνώριση των στοιχείων δεδομένων που απαιτούνται για την λειτουργία της εφαρμογής, καθώς των ρόλων και των απαιτήσεων. Ζητούμενο εδώ είναι η απλοποίηση της βάσης δεδομένων ως προς την χρηστικότητα.

## 2.1 Αρχιτεκτονική

Το σύστημα μας βασίζεται σε δύο διαφορετικές κατηγορίες υλοποίησης. Στην πρώτη έχουμε την λογική client-server με αμφίδρομη επικοινωνία. Δηλαδή έχουμε επικοινωνία και μετάδοση πληροφοριών μεταξύ όλων των συσκευών για την δημιουργία του παιχνιδιού. Στη δεύτερη περίπτωση έχουμε την απλή χρήση του συστήματος κατά την οποία γίνεται απλή εξαγωγή δεδομένων προς το χρήστη.

Στην πρώτη περίπτωση το σύστημα υποστηρίζει μέχρι τέσσερις χρήστες, εκ των οποίων ο ένας θα φιλοξενεί το παιχνίδι. Για την λειτουργία αυτή δημιουργείται μία σύνδεση μέσω ασύρματων δικτύων από το διακομιστή και ξεκινά η αποστολή πληροφοριών στους υπόλοιπους διαθέσιμους πελάτες. Ολόκληρη η αρχιτεκτονική του

παιχνιδιού βασίζεται στα πακέτα μηνυμάτων μεταξύ του διακομιστή και των πελατών.

Στην δεύτερη περίπτωση έχουμε μία διαφορετική προσέγγιση του παιχνιδιού. Έχουμε τον απλό χρήστη που λαμβάνει στην οθόνη τις πληροφορίες που αφορούν το παιχνίδι. Η αρχιτεκτονική εδώ δεν βασίζεται στη λογική των πακέτων πληροφοριών μεταξύ των συσκευών.

## 2.2 Ρόλοι του συστήματος

Η υλοποίηση της βάσης δεδομένων έγινε αρχικά με βάση τον προσδιορισμό των χρηστών και των μεταξύ των αλληλεπιδράσεων. Έτσι εντοπίστηκαν οι παρακάτω χρήστες:

- Διαχειριστής
- Πελάτης
- Απλός χρήστης

Ο ρόλος του διαχειριστή είναι ο σημαντικότερος από τους παραπάνω. Έχει τη δυνατότητα να διαχειρίζεται και να αναπτύσσει (προσθαφαιρεί) λειτουργίες και δεδομένα του συστήματος.

Ο ρόλος του πελάτη μπορεί μόνο να συνδεθεί και να συμμετάσχει σε ένα παιχνίδι που έχει δημιουργήσει κάποιος διαχειριστής.

Ο ρόλος του απλού χρήστη έχει εντελώς διαφορετικό ρόλο στο σύστημα. Λειτουργεί ως αποδέκτης των δεδομένων της βάσης ανάλογα με τις προσωπικές του επιλογές (single user).

## 2.3 Απαιτήσεις

Έπειτα αφού έχουμε καθορίσει τους ρόλους των χρηστών μπορούμε να ορίσουμε τις απαιτήσεις τους από το σύστημα (εφαρμογή).

Έτσι ο διαχειριστής έχει τη δυνατότητα :

- Να προσθαφαιρεί δεδομένα.
- Να καθορίζει τον τρόπο και τη διάρκεια του παιχνιδιού.
- Να επιλέγει τον αριθμό των ερωτήσεων που θα αναμετρηθεί μεταξύ των άλλων χρηστών.
- Να διαχειρίζεται όλους τους συνδεδεμένους χρήστες.

Ο απλός χρήστης έχει τη δυνατότητα:

- Να συνδεθεί ως πελάτης στο σύστημα.
- Να επιλέγει τον τρόπο και την διάρκεια του παιχνιδιού.
- Να επιλέγει τον αριθμό των ερωτήσεων που θέλει να παίξει.

# 2.4 Περιπτώσεις χρήσης

## 2.4.1 Ρόλος διαχειριστή

Ως διαχειριστής ορίζεται ο χρήστης ο οποίος θα επιλέξει να χρησιμοποιήσει την εφαρμογή στα πρωτόκολλα επικοινωνίας IEEE **802.11 (Wi-Fi) – Bluetooth, δημιουργώντας και τον τοπικό διακομιστή** (server) ο οποίος θα φιλοξενήσει τους υπόλοιπους παίκτες.

**Βασική ροή**

- Σε αυτή την περίπτωση χρήσης ο διαχειριστής ξεκινάει το host game επιλέγοντας το αντίστοιχο κουμπί.
- Στη συνέχεια εισαγάγει το όνομα της προτίμησης του για την δημιουργία του τοπικού διακομιστή server που θα φιλοξενήσει το παιχνίδι.
- Αναμένει την σύνδεση των υπόλοιπων παικτών που επιθυμούν να συνδεθούν στο δίκτυο του διαχειριστή.
- Αφού καλυφθεί ο αριθμός των διαθέσιμων παικτών επιλέγει "Έναρξη".
- Έπειτα εμφανίζεται η φόρμα επιλογής των κατηγοριών ερωτήσεων.
- Ο διαχειριστής έχει την ικανότητα να επιλέξει μία ή περισσότερες από αυτές εφόσον το επιθυμεί.
- Τέλος πατώντας το κουμπί "Έναρξη", ορίζει και τον επιθυμητό χρόνο διάρκειας του παιχνιδιού όπως και τον αριθμό των ερωτήσεων που θα διαγωνιστεί με τους υπόλοιπους συνδεδεμένους χρήστες.

**Εναλλακτική ροή – Ακύρωση δημιουργίας του τοπικού διακομιστή.**

Ο διαχειριστής έχει τη δυνατότητα όποτε θελήσει να ακυρώσει την επικείμενη έναρξη του παιχνιδιού που έχει δημιουργήσει πατώντας την επιλογή "Χ" που βρίσκεται στο κάτω αριστερά μέρος της φόρμας παιχνιδιού.

## 2.4.2 Ρόλος πελάτη

Ως πελάτης ορίζεται ο χρήστης ο οποίος επιθυμεί να συνδεθεί και να λάβει μέρος σε ένα ήδη υπάρχον παιχνίδι που έχει δημιουργηθεί από κάποιον διαχειριστή.

**Βασική ροή**

- Σε αυτή την περίπτωση χρήσης ο πελάτης ξεκινάει την συμμετοχή του επιλέγοντας το κουμπί join game.
- Στη συνέχεια εισάγει το όνομα της προτίμησης του.
- Από κάτω βλέπει τα διαθέσιμα παιχνίδια και επιλέγει αυτό που επιθυμεί.
- Έπειτα αναμένει την έναρξη του συγκεκριμένου παιχνιδιού από τον αντίστοιχο διαχειριστή.

**Εναλλακτική ροή – Έξοδος του πελάτη από το παιχνίδι.**

Ο πελάτης έχει τη δυνατότητα εξόδου από το παιχνίδι στο οποίο έχει επιλέξει να συμμετάσχει πατώντας την επιλογή "Χ" που βρίσκεται στο κάτω αριστερά μέρος της φόρμας παιχνιδιού.

## 2.4.3 Ρόλος απλού χρήστη

Απλός χρήστης θεωρείται εκείνος που πρόκειται να χρησιμοποιήσει την εφαρμογή καθαρά για προσωπική χρήση, μη συμμετέχοντας ή δημιουργώντας κάποιο παιχνίδι πολλαπλών συμμετοχών.

**Βασική ροή**

- Στην περίπτωση αυτή ο απλός χρήστης ξεκινάει το single game επιλέγοντας το αντίστοιχο κουμπί.
- Στη συνέχεια εμφανίζεται η φόρμα επιλογής των κατηγοριών ερωτήσεων που διατίθενται βάσει της θεματικής τους.
- Ο χρήστης επιλέγει μία από αυτές εφόσον το επιθυμεί.
- Τέλος πατώντας το κουμπί "Έναρξη" ορίζει τον επιθυμητό χρόνο διάρκειας του παιχνιδιού καθώς και τον αριθμό των ερωτήσεων με τις οποίες θέλει να παίξει.

**Εναλλακτική ροή – Ακύρωση – έξοδος του παιχνιδιού.**

Ο απλός χρήστης έχει την δυνατότητα να σταματήσει το παιχνίδι έπειτα του πέρατος του χρόνου που έχει ορίσει ή εναλλακτικά μπορεί να

περιμένει να περάσει ο αριθμός των ερωτήσεων που έχει θέσει κατά την έναρξη.

Τέλος αν θελήσει να αφήσει την εφαρμογή πριν το πέρας του χρόνου ή των ερωτήσεων μπορεί να βγει από αυτό πατώντας την επιλογή "X" που βρίσκεται στο κάτω αριστερά μέρος της φόρμας παιχνιδιού.

# 2.5 Βάση δεδομένων

Για τη δημιουργία μίας σωστά σχεδιασμένης και άνευ προβλημάτων βάσης δεδομένων απαιτείται αρκετός χρόνος για την εκμάθηση των σχετικών αρχών. Για το βέλτιστο σχεδιασμό και την ευχρηστία της θα πρέπει να αποφεύγονται πεδία που δε χρησιμοποιούνται ή, επαναλαμβάνονται, όπως και πεδία που καταλαμβάνουν μεγάλο χώρο. Για να επιτευχθεί το παραπάνω θα πρέπει να ληφθούν υπόψη όλες οι πληροφορίες που συγκεντρώθηκαν στα προηγούμενα στάδια.

Έτσι και σε συνδυασμό με τη βελτιστοποιημένη κανονικοποίηση των δεδομένων δημιουργήθηκαν οι δύο ακόλουθοι πίνακες:

- Categories
- Questions

## 2.5.1 Πίνακας 'Categories'

Σκοπός του πίνακα Categories είναι να αποθηκεύει τις κατηγορίες του παιχνιδιού.

| **Categories** | | | | |
|---|---|---|---|---|
| **Όνομα** | **Τύπος** | **Όχι Κενό** | **Προεπιλεγμένη Τιμή** | **Πρωτεύων Κλειδί** |
| ID | Int (2) | 0 | Καμία | 1 |
| Names | Varchar (20) | 0 | Καμία | 0 |

Πίνακας 2.1: Πίνακας κατηγοριών

**ID:**

Το ID είναι το πρωτεύον κλειδί για κάθε κατηγορία που εισαγάγεται στον πίνακα και έχει οριστεί ώστε να αυξάνεται αυτόματα. Στο πεδίο

"**Τύπος**" δηλώσαμε μεταβλητή Int με περιορισμό μήκους 2 χαρακτήρων, διότι έχουμε τιμή ακέραιου για να καλύψουμε μέχρι 99 μοναδικές εγγραφές στον πίνακα Categories. Στην περίπτωση μας όμως είναι αδύνατο να χρησιμοποιήσουμε 99 διαφορετικές κατηγορίες ερωτήσεων όμως έτσι έχουμε την δυνατότητα να χρησιμοποιήσουμε τουλάχιστον 10 διαφορετικές.

**Names:**

Στο πεδίο αυτό αποθηκεύονται τα ονόματα των κατηγοριών. Εδώ δηλώσαμε μεταβλητή τύπου Varchar και μήκος 20 χαρακτήρων για να έχουμε τη δυνατότητα αλφαριθμητικών χαρακτήρων. Οι 20 χαρακτήρες είναι αρκετοί για τον προσδιορισμό του ονόματος μίας κατηγορίας.

## 2.5.2. Πίνακας 'Questions'

Ο πίνακας Questions χρησιμοποιείται για την καταχώρηση των ερωτήσεων και των αντίστοιχων πιθανών απαντήσεων τους.

| **Questions** | | | | |
|---|---|---|---|---|
| **Όνομα** | **Τύπος** | **Όχι Κενό** | **Προεπιλεγμένη τιμή** | **Πρωτεύων Κλειδί** |
| Category | Int (2) | 0 | Καμία | 0 |
| Question | Varchar (150) | 0 | Καμία | 0 |
| Answer 1 | Varchar (50) | 0 | Καμία | 0 |
| Answer 2 | Varchar (50) | 0 | Καμία | 0 |
| Answer 3 | Varchar (50) | 0 | Καμία | 0 |
| Answer 4 | Varchar (50) | 0 | Καμία | 0 |
| Used | Varchar (4) | 0 | Καμία | 0 |

Πίνακας 2.2: Πίνακας ερωτήσεων

**Category:**

Πρόκειται για ξένο κλειδί (Foreign Key) το οποίο αντιστοιχεί στο πρωτεύον κλειδί ID του πίνακα των κατηγοριών (Categories). Είναι τύπου Int το οποίο εξαρτάται από το ID και χρησιμοποιείται για την εισαγωγή των καταχωρήσεων ερωτήσεων. Δηλαδή όταν καταχωρούμε

μία ερώτηση δηλώνουμε σε ποια κατηγορία ανήκει με βάση το ID της κάθε κατηγορίας.

**Question:**

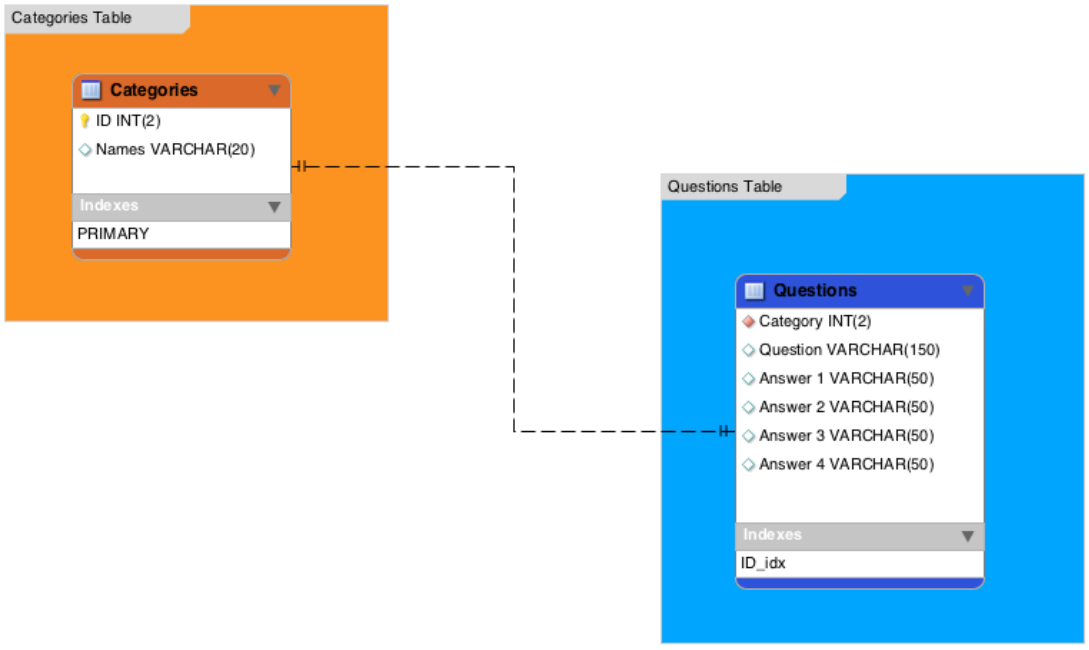
Χρησιμεύει για να αποθηκεύει τον τίτλο της κάθε ερώτησης. Χρησιμοποιήσαμε μεταβλητή τύπου Varchar γιατί μία ερώτηση μπορεί να περιέχει κάποια σύμβολα, αριθμούς και γράμματα. Περιορίσαμε το μήκος σε 150 χαρακτήρες διότι θεωρείται ικανοποιητικό για την περιγραφή μίας ερώτησης,

**Answer 1, 2, 3, 4:**

Αυτά τα πεδία αποθηκεύουν τις τέσσερις πιθανές απαντήσεις για την κάθε ερώτηση. Οι απαντήσεις είναι επίσης τύπου Varchar με μήκος χαρακτήρων τους 50, το οποίο είναι αρκετό για την εισαγωγή των απαντήσεων.

**Used**

Το πεδίο αυτό χρησιμοποιείται για να κρατάει η βάση μας την κάθε ερώτηση που εξάγει. Είναι τύπου Varchar με μήκος τέσσερα γιατί η λέξη που έχει οριστεί να γράφει κάθε φορά που εξάγει μια ερώτηση είναι τριών χαρακτήρων.




Εικόνα 2.1: Οι πίνακες από το πρόγραμμα mySQLWorkbench

## 2.6 Λειτουργία βάσης δεδομένων

Η βάση δεδομένων του συστήματος δημιουργήθηκε με τη χρήση της SQLite και είναι μία κλειστή βάση δεδομένων. Αυτό σημαίνει ότι οι τρείς χρήστες που ορίσαμε παραπάνω δεν έχουν τη δυνατότητα να προσθέσουν ή να αφαιρέσουν ερωτήσεις μέσω της εφαρμογής. Ο μόνος τρόπος για να ανανεωθεί η βάση δεδομένων είναι μέσω διαδικτύου ή μίας νέας ανανέωσης έκδοσης ολόκληρης της εφαρμογής.

Κατά την εκκίνηση της εφαρμογής η βάση δεδομένων αρχικά έχει οριστεί να ανανεώνεται μετά την επιλογή της κατηγορίας, δηλαδή πηγαίνει και γράφει στο πεδίο «Used» την τιμή 'NO' για όλες τις ερωτήσεις που υπάρχουν στην βάση. Αυτό συμβαίνει για την ορθή χρήση της βάσης κατά τη διαδικασία του παιχνιδιού. Επιλέγοντας την κατηγορία ερωτήσεων που θέλει, είτε ο διακομιστής είτε ο απλός χρήστης ξεκινά το παιχνίδι.

```
-(void)insertCheck {
    NSArray * paths = NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory, NSUserDomainMask, YES);
    NSString * documentsDirectory = [paths objectAtIndex:0];
    NSString * path = [documentsDirectory
        stringByAppendingPathComponent:@"Categories.sqlite"];
    if (sqlite3_open([path UTF8String], & db) == SQLITE_OK) {
       const char * sql = "UPDATE Questions SET Used='NO'";
        sqlite3_stmt * inputUsedVariable;
        if (sqlite3_prepare_v2(db, sql, -1, &
            inputUsedVariable, NULL)==SQLITE_OK) {
            while (sqlite3_step(inputUsedVariable)==
                SQLITE_OK) {
                NSString * name = [NSString
                    stringWithUTF8String:(char *)
                    sqlite3_column_text(inputUsedVariable, 6)
                    ];
                NSLog(@"Must Changed: %@", name);
            }
        }
        sqlite3_finalize(inputUsedVariable);
    }
    sqlite3_close(db);
}
```

Εικόνα 2.2: Η συνθήκη ανανέωσης της βάσης δεδομένων

Η βάση δεδομένων ξεκινά να εξάγει ερωτήσεις από την κατηγορία που επέλεξε ο χρήστης τυχαία και με κριτήριο επιλογής το πεδίο «Used» να έχει την τιμή 'NO'. Για κάθε ερώτηση που εμφανίζει, αυτόματα γίνεται η αλλαγή της μεταβλητής του πεδίου «Used» με την τιμή 'YES' για τη συγκεκριμένη ερώτηση. Με αυτό τον τρόπο μπορούμε να αποφύγουμε τη διπλή εμφάνιση κάποιας ερώτησης.

## *2.6 Λειτουργία βάσης δεδομένων*

Τέλος για να μην υπάρχει κατάχρηση της μνήμης της συσκευής έχουμε ορίσει τη βάση δεδομένων κάθε φορά που εκτελεί κάποιο ερώτημα να κλείνει.

```
-(void)insertCheckToYes {
    NSArray * paths = NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory, NSUserDomainMask, YES);
    NSString * documentsDirectory = [paths objectAtIndex:0];
    NSString * path = [documentsDirectory
        stringByAppendingPathComponent:@"Categories.sqlite"];
    if (sqlite3_open([path UTF8String], & db) == SQLITE_OK) {
        NSString * sql = [NSString stringWithFormat:@"UPDATE
            Questions SET Used='YES' WHERE Question=\"%@\"",
            question.text];
        const char *query_stmt = [sql UTF8String];
        sqlite3_stmt * inputUsedVariable;
        if (sqlite3_prepare_v2(db, query_stmt, -1, &
            inputUsedVariable, NULL)==SQLITE_OK) {
            while (sqlite3_step(inputUsedVariable)==
                SQLITE_ROW) {
                NSString * name = [NSString
                    stringWithUTF8String:(char *)
                    sqlite3_column_text(inputUsedVariable, 0)
                    ];
                NSLog(@"Must Changed: %@", name);
            }
        }
        sqlite3_finalize(inputUsedVariable);
    }
    sqlite3_close(db);
}
```

Εικόνα 2.3: Η συνθήκη που καλείται μετά την εμφάνιση ερώτησης

# ΚΕΦΑΛΑΙΟ 3ο

## Υλικό και λογισμικό που χρησιμοποιήθηκε

Στο προηγούμενο κεφάλαιο έγινε ο σχεδιασμός του συστήματος της εφαρμογής και ορίστηκε η αρχιτεκτονική του. Αντικείμενο του κεφαλαίου αυτού είναι η παρουσίαση και η περιγραφή του υλικού και του λογισμικού που χρησιμοποιήθηκε για την ανάπτυξή της εφαρμογής.

## 3.1 Το υλικό (Hardware)

### 3.1.1 Ηλεκτρονικός υπολογιστής

Για να υπάρχει η δυνατότητα προγραμματισμού μίας εφαρμογής για την οικογένεια των συσκευών της Apple [17] iPhone, iPod και iPad ο προγραμματιστής πρέπει να διαθέτει και τον ανάλογο ηλεκτρονικό υπολογιστή της ίδιας οικογενείας. Η σειρά ηλεκτρονικών υπολογιστών που παράγει η Apple είναι τα:

- iMac ένας πλήρης ηλεκτρονικός υπολογιστής που διαθέτει όλο το υλικό του σε μία οθόνη
- το Macbook Pro που είναι ένας υπολογιστής τύπου Laptop
- το Macbook Air επίσης ένα Laptop που ξεχωρίζει για την λεπτή γραμμή σχεδίασης και ελαφριά κατασκευή
- το Mac Mini που είναι ένα ολοκληρωμένο υπολογιστικό σύστημα με τις διαστάσεις ενός εξωτερικού σκληρού δίσκου.

Υπάρχει όμως και η δυνατότητα να φτιάξουμε τον δικό μας προσωπικό υπολογιστή x86 αρχιτεκτονικής, το οποίο όμως τρέχει Mac OS X λειτουργικό, με λίγα λόγια ένα Mac, μόνο που σε αυτή την περίπτωση το υλικό (hardware) είναι επιλογής μας και μπορεί να φτάσει σε ακόμα υψηλότερο επίπεδο ταχυτήτων και υπολογιστικής δύναμης από αυτή ενός αυθεντικού Mac Pro! Με αυτό τον τρόπο έχουμε στα χέρια μας ένα πολύ δυνατό μηχάνημα που μπορεί να τρέξει το λειτουργικό της Apple σε ορισμένες περιπτώσεις ίσως και καλύτερα από έναν Mac με

πολύ μικρότερο κόστος από τα γνήσια Apple Mac των οποίων οι τιμές (όπως γνωρίζουμε) είναι συγκριτικά πολύ υψηλότερες.

Θα πρέπει βέβαια κάπου εδώ να πούμε, πως από τη στιγμή που κάποιος εγκαταστήσει Mac OS X σε υπολογιστή εκτός των αυθεντικών Apple υπολογιστών, αυτόματα σπάει και τους όρους χρήσης (Terms Of Use) της Apple που λέει :

"Συμφωνείτε να μην εγκαταστήσετε, να χρησιμοποιήσετε ή να εκτελέσετε το λογισμικό της Apple για κάθε μη-Apple-επισημασμένο υπολογιστή, ή να επιτρέψετε σε άλλους να το κάνουν."

"You agree not to install, use, or run the Apple software on any non-Apple-labeled computer, or to enable others to do so."

## 3.1.2 Οι συσκευές iPhone, iPod Touch, iPad

Η σειρά κινητών πλατφορμών της Apple [17] αποτελείται από τρία προϊόντα που ως κύριο χαρακτηριστικό τους παρέχουν στο χρήστη σχεδόν όλες τις λειτουργίες που αφορούν το διαδίκτυο, και περιλαμβάνουν μία μεγάλη γκάμα εφαρμογών πολυμέσων όπως μουσική, βίντεο και φωτογραφία. Ο λόγος που ξεχωρίζει στην αγορά είναι οι διάφορες καινοτομίες που λανσάρει, όπως το φιλικότερο προς το χρήστη περιβάλλον αλλά και ένα πλήρες και ελαφρύ λειτουργικό σύστημα.

Το περιβάλλον χρήστη είναι σχεδιασμένο έτσι ώστε να χρησιμοποιεί την οθόνη αφής πολλαπλών σημείων η οποία όταν ανακοινώθηκε ξεχώριζε καθώς για να χρησιμοποιηθεί δεν χρειαζόταν τίποτα παραπάνω από τα δάκτυλα μας, και η απόκριση της είναι ίσως ακόμα η καλύτερη στην αγορά. Για την εισαγωγή δεδομένων και κειμένου χρησιμοποιήθηκε ένα εικονικό πληκτρολόγιο αντί για κουμπιά το οποίο αν και στην αρχή προκάλεσε τον προβληματισμό, τώρα έχει γίνει κανόνας στην αγορά.

Στις συσκευές αυτές έχουμε τρείς διαφορετικούς αισθητήρες. Ο πρώτος χρησιμοποιείται για να απενεργοποιεί την οθόνη όταν ο χρήστης μιλάει στο τηλέφωνο, ο δεύτερος είναι αισθητήρας φωτός και ρυθμίζει την φωτεινότητα της οθόνης αυτόματα έτσι ώστε να εξοικονομεί μπαταριά και τέλος υπάρχει ένας αισθητήρας τριών αξόνων οποίος αντιλαμβάνεται οποιαδήποτε αλλαγή στην γωνία κλίσης της συσκευής και με αυτό τον τρόπο επιτυγχάνεται η αλλαγή από κάθετη (portrait) σε οριζόντια (landscape) προβολή των δεδομένων στην οθόνη.

## ***3.1 Το υλικό (Hardware)***

Το πρώτο iPhone ανακοινώθηκε από τον πρόεδρο της Apple Steve Jobs τον Ιανουάριο του 2007 και κυκλοφόρησε στην αγορά τον Ιούνιο του ίδιου έτους. Η συγκεκριμένη συσκευή-κινητό τηλέφωνο διαθέτει βίντεο κάμερα, Bluetooth και Wi-Fi. Χρησιμοποιεί εφαρμογές πλοήγησης στο διαδίκτυο όπως το Safari και το Mail. Επίσης περιλαμβάνει λειτουργίες φορητού Media player και πολλαπλών άλλων εφαρμογών. Το iPhone 3G πρόσθεσε την δυνατότητα χρησιμοποίησης του δικτύου κινητής τηλεφωνίας 3G και δέκτη GPS. Το iPhone 3GS πρόσθεσε μία πυξίδα, πιο γρήγορο επεξεργαστή και κάμερα υψηλότερης ανάλυσης που περιλαμβάνει και εγγραφή βίντεο στα 480p. Το iPhone 4 πρόσθεσε μία κάμερα στο μπροστινό μέρος του κινητού για χρήση βιντεοκλήσης με προγράμματα όπως το Skype, καλύτερο επεξεργαστή αλλά και οθόνη με υψηλότερη ανάλυση. Το iPhone 4S με λίγες διαφορές σε σχέση με τον προκάτοχο του πρόσθεσε κάμερα 8MP με νέες λειτουργίες σε αυτό το στοιχείο. Το πιο πρόσφατο iPhone 5 πρόσθεσε οθόνη υψηλής ευκρίνειας Retina τεσσάρων ιντσών και νέο επεξεργαστή ισχύος δύο πυρήνων.

Το iPod εμφανίστηκε τον Οκτώβριο του 2001 σαν ένα φορητό σύστημα αναπαραγωγής πολυμέσων. Σήμερα το iPod Touch είναι ουσιαστικά ένα iPhone χωρίς τις δυνατότητες ενός κινητού τηλεφώνου. Τα βασικά χαρακτηριστικά και τον δύο είναι σχεδόν ίδια και χρησιμοποιούν το ίδιο λειτουργικό σύστημα, το iOS. Το iPod Touch, λόγω της αδυναμίας του να συνδέεται στο δίκτυο κινητής τηλεφωνίας, δεν υποστηρίζει εφαρμογές που βασίζονται σε αυτή την λειτουργία. Είναι πιο λεπτό, καταναλώνει λιγότερη ενέργεια άρα και η μπαταρία του κρατάει πολύ περισσότερο, και κοστίζει λιγότερο από τα αντίστοιχα μοντέλα iPhone.

Τα iPad είναι η τελευταία προσθήκη στη σειρά κινητών πλατφορμών της Apple καθώς το πρώτο μοντέλο τους ανακοινώθηκε τον Ιανουάριο του 2010. Παρόμοιο στη λειτουργία του με το μικρότερο iPhone ή iPod Touch, τρέχει μία τροποποιημένη έκδοση του ίδιου λειτουργικού συστήματος , με ένα επανασχεδιασμένο περιβάλλον για να εκμεταλλευτεί τη μεγαλύτερη οθόνη. Το iPad έχει μία αναδρομικά φωτισμένη οθόνη αφής πολλαπλών σημείων 9.7 ιντσών. Η κύρια λειτουργία του iPad είναι η ανάγνωση ηλεκτρονικών βιβλίων (e-books) και εφημερίδων. Όμως υπάρχει πλήρης υποστήριξη παιχνιδιών και εφαρμογών που έχουν

επανασχεδιαστεί για την συσκευή όπως και διαχείριση των φωτογραφιών μας μέσω διασύνδεσης στο διαδίκτυο.

### 3.1.3 Το Bluetooth


Εικόνα 3 1: Το σήμα του (μπλε δοντιού) Bluetooth

Μέχρι τα τέλη της δεκαετίας του 1990 δεν υπήρχε κάποιο ευρέως αποδεκτό πρότυπο για ασύρματα προσωπικά δίκτυα WPAN (Wireless personal area network), ούτε φυσικά ανάλογες εμπορικές εφαρμογές / πομποδέκτες. Όμως τότε η Ericsson έθεσε τις βάσεις για την ανάπτυξη μίας τεχνολογίας η οποία θα επέτρεπε τον σχηματισμό τοπικών δικτύων πολύ μικρής εμβέλειας με σκοπό την ασύρματη και ad hoc δικτύωση ετερογενών φορητών συσκευών. Το πρότυπο που προέκυψε υιοθετήθηκε στη συνέχεια από την IEEE ως το πρότυπο 802.15 για WPAN. Το πρότυπο ονομάστηκε Bluetooth [2] από τον Βασιλιά της Δανίας Harald “Blatand ” Gormsson τον πρώτο. Το όνομα πρότειναν Σκανδιναβοί επιστήμονες επειδή, ο Βασιλιάς αυτός ήταν φιλειρηνικός και κατάφερε να ενώσει τη Δανία με την Νορβηγία. Αρχικός σκοπός του Bluetooth ήταν να ενώσει-δικτυώσει συσκευές, γι’ αυτό και ονομάστηκε όπως το παρατσούκλι αυτού του Βασιλιά. Ο λόγος που δόθηκε το παρατσούκλι μπλε-δόντης στον Βασιλιά Harald ήταν επειδή είχε ένα δόντι ελαφρά πιο γαλάζιο από τα υπόλοιπα.

Οι προδιαγραφές του Bluetooth καθορίζουν την «ασύρματη» τεχνολογία χαμηλού κόστους και χαμηλής ισχύος, που εξαλείφει τα καλώδια μεταξύ των κινητών συσκευών και επιτρέπει τη διασύνδεσή τους. Το Bluetooth λειτουργεί στο «αδέσμευτο» φάσμα συχνοτήτων των 2,4 GHz, ώστε οι συσκευές που το ενσωματώνουν να μπορούν να λειτουργήσουν απροβλημάτιστα σε οποιοδήποτε σημείο του πλανήτη. Για να περιοριστούν στο ελάχιστο οι παρεμβολές από παρεμφερείς συσκευές, το Bluetooth εκμεταλλεύεται την αμφίδρομη επικοινωνία και τη μέθοδο μετάδοσης με διασπορά φάσματος Frequency Hopping (έως

και 1600 εναλλαγές συχνότητας ανά δευτερόλεπτο). Από φυσική άποψη επίσης το Bluetooth λειτουργεί περίπου στα 2.4 GHz, προδιαγράφει τρία επίπεδα ισχύος της εκπομπής από τα οποία εξαρτάται και η εμβέλεια επικοινωνίας (πάντα μικρότερη των 10 μέτρων σε PAN), ενώ η τακτική αλλαγή της συχνότητας εκπομπής λόγω της αξιοποίησης του FHSS καθορίζεται ψευδοτυχαία από έναν κεντρικό κόμβο, τον Master.

Το Bluetooth επιτρέπει τις απευθείας συνδέσεις από συσκευή προς συσκευή (point to point), καθώς και την ταυτόχρονη σύνδεση έως και επτά συσκευών με τη χρήση μίας μοναδικής συχνότητας. Τις προδιαγραφές της συγκεκριμένης τεχνολογίας ανέπτυξε και υποστηρίζει το Bluetooth Special Interest Group, ενώ η τελευταία «δημόσια» έκδοσή τους είναι η 4, η οποία ενσωματώνεται πλέον στις περισσότερες συμβατές συσκευές μέσω κατάλληλων πομποδεκτών και καρτών δικτύου. Ένα πρόβλημα των προδιαγραφών του Bluetooth είναι ότι, λόγω της μετάδοσης στην ελεύθερη ζώνη συχνοτήτων των 2,4 GHz, οι συσκευές που το υποστηρίζουν αδυνατούν να χρησιμοποιήσουν ταυτόχρονα τα περισσότερα πρωτόκολλα της οικογένειας IEEE 802.11, καθώς τότε θα υπήρχαν σοβαρά προβλήματα παρεμβολών.

Οι βασικότερες προδιαγραφές του Bluetooth αφορούν το φυσικό επίπεδο και το υποεπίπεδο MAC, όπου έχουν δημιουργηθεί διαφορετικά πρωτόκολλα για διαφορετικές εφαρμογές και τα οποία ονομάζονται προφίλ. Το Bluetooth SIG έχει ήδη παρουσιάσει τέτοιες παραμετροποιημένες εκδοχές του προτύπου για διάφορες «αγορές» (π.χ. προφίλ ασύρματου τηλεφώνου, προφίλ πρόσβασης σε LAN, προφίλ εκτύπωσης, φωτογραφίας, αυτοκινήτου κλπ). Κάθε προφίλ περιλαμβάνει πρότυπα για όλα τα επίπεδα και προσφέρει λύσεις για τη διασύνδεση με διαφορετικά δίκτυα μεγαλύτερης κλίμακας.

## 3.1.4 Το Wi-Fi


Εικόνα 3.2: Το σήμα του Wi-Fi

## *3.1 Το υλικό (Hardware)*

Η πρώτη έκδοση του WiFi [3] εισήχθη το 1997 και στο φυσικό επίπεδο του μοντέλου αναφοράς Ανοικτής Διασύνδεσης Συστημάτων (OSI) περιελάμβανε δύο μεθόδους διασποράς φάσματος για τη μετάδοση στη ζώνη συχνοτήτων 2.4GHz, η εκπομπή στην οποία δεν απαιτεί άδεια. Η πρώτη μέθοδος λειτουργούσε με Frequency Hopping (FHSS) και υποστήριζε ρυθμό μετάδοσης 1 Mbps, ενώ η δεύτερη λειτουργούσε με Direct Sequence (DSSS) και υποστήριζε ρυθμό μετάδοσης 1-2 Mbps. Περιλαμβανόταν επίσης και μία υπέρυθρη εκδοχή (IR). Πριν από την εμφάνιση του 802.11 δεν υπήρχε κάποιο ευρέως αποδεκτό πρότυπο για ασύρματα τοπικά δίκτυα υπολογιστών, ούτε ανάλογες εμπορικές εφαρμογές, καθώς η τεχνολογία ασύρματης δικτύωσης δεν ήταν ακόμα αρκετά ώριμη.

Το 1999 το 802.11b ώθησε την ταχύτητα στα 11 Mbps χρησιμοποιώντας DSSS. Οι ρυθμοί λειτουργίας 1-2 Mbps με DSSS ισχύουν ακόμα, έτσι ώστε οι συσκευές να μπορούν να πέσουν σε χαμηλότερες ταχύτητες για να διατηρήσουν μία σύνδεση όταν τα σήματα είναι αδύνατα. Με την έκδοση αυτή ο όρος WiFi άρχισε να χρησιμοποιείται ευρέως και οι ασύρματες κάρτες δικτύου 802.11 να εξαπλώνονται ταχέως.

Χρησιμοποιώντας τη μέθοδο μετάδοσης Orthogonal Frequency Division Multiplexing (OFDM), δύο πρότυπα υψηλής ταχύτητας ακολούθησαν το 802.11b τα οποία παρέχουν μέχρι 54 Mbps: το 802.11a εκπέμπει στη ζώνη συχνοτήτων των 5GHz αλλά δεν είναι συμβατό με τις ασύρματες κάρτες δικτύου οι οποίες υποστηρίζουν 802.11b, ενώ το 802.11g εκπέμπει στη ζώνη συχνοτήτων των 2.4GHz και είναι συμβατό με το 802.11b. Η επικοινωνία μεταξύ συσκευών εξοπλισμένων με κάρτες 802.11b και 802.11g γίνεται στην υψηλότερη δυνατή κοινή ταχύτητα, αυτήν του 802.11b.

Με τη διάδοση του WiFi κατά τις αρχές της δεκαετίας του 2000 εμφανίστηκε μία νέα μέθοδος πρόσβασης στο Internet: μία ψηφιακή συσκευή με κάρτα ασύρματης δικτύωσης WiFi, π.χ. ένας ηλεκτρονικός υπολογιστής ή ένα PDA, μπορεί να συνδεθεί στο Διαδίκτυο όταν βρίσκεται σε ακτίνα κάλυψης ασύρματου δικτύου ήδη συνδεδεμένου στο Internet, το οποίο ονομάζεται σημείο πρόσβασης (Access Point). Μία περιοχή που καλύπτεται από ένα ή περισσότερα σημεία πρόσβασης συνδεδεμένα μεταξύ τους λέγεται hotspot. Ένα hotspot μπορεί να

καλύπτει έναν χώρο έκτασης δωματίου ή και πολλών τετραγωνικών μέτρων, με εναλλασσόμενα σημεία πρόσβασης.

Έτσι η τεχνολογία WiFi επιτρέπει τη σύνδεση μεταξύ δύο συσκευών μεταξύ τους, τη σύνδεση ενός προσωπικού υπολογιστή με ένα τοπικό δίκτυο και άλλους υπολογιστές και, στη συνέχεια, μέσω αυτών στο Internet. Ένας φορητός υπολογιστής μπορεί να συνδεθεί οπουδήποτε υπάρχει σημείο πρόσβασης (π.χ. σε πάρκα ή πλατείες μεγάλων πόλεων, καφετέριες, βιβλιοθήκες κλπ).

# 3.2 Το λογισμικό (software)

## 3.2.1 Mac OS X

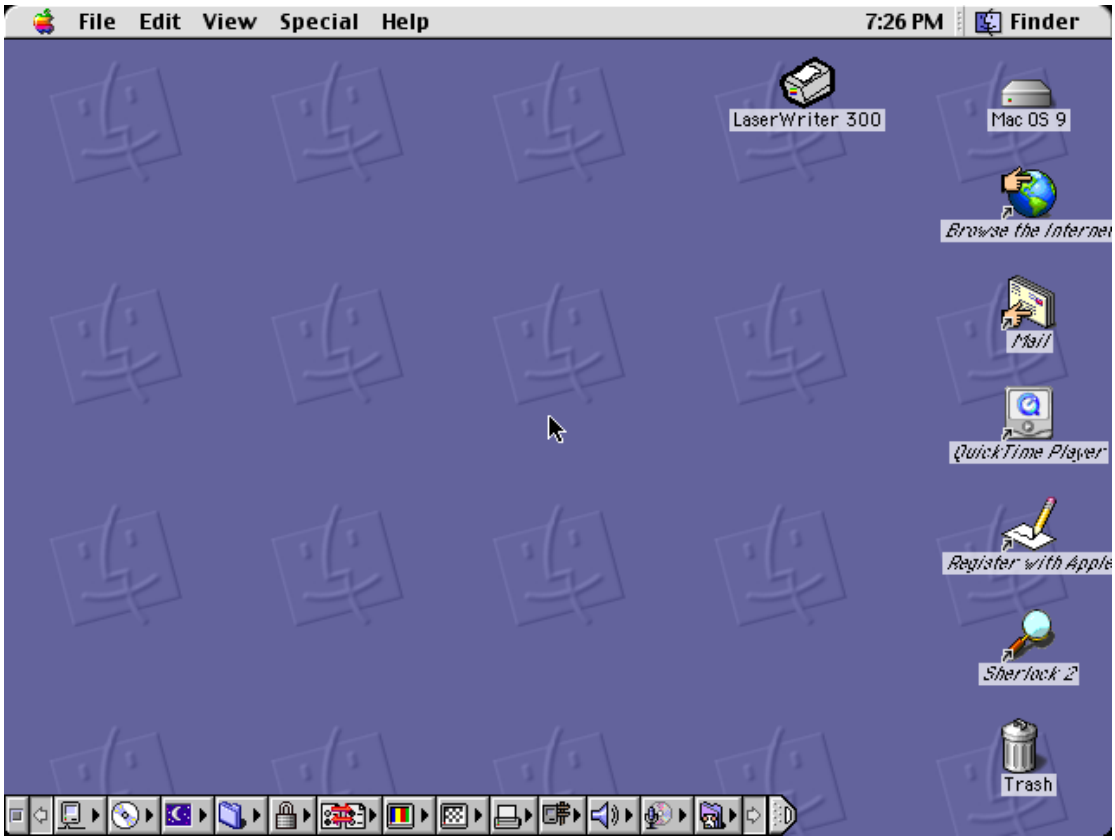
Το Mac OS [1] είναι ένα λειτουργικό σύστημα με γραφικό περιβάλλον που αναπτύχθηκε από την Apple Inc (πρώην Apple Computer, Inc) για τους ηλεκτρονικούς υπολογιστές Mackintosh που η εταιρία εισήγαγε στην αγορά. Το Mac OS, πριν ακόμη η Apple το ονομάσει έτσι, εφαρμόσθηκε για πρώτη φορά το 1984 στον πρωτότυπο υπολογιστή Macintosh ως μη διακριτό τμήμα του συστήματος και συνήθως αναφερόταν απλά ως το "λογισμικό συστήματος".

Η Apple εσκεμμένα υποβάθμιζε την ύπαρξη του λειτουργικού συστήματος κατά τα πρώτα χρόνια του Macintosh για να δώσει έμφαση στην εύκολη χρήση του υπολογιστή της αλλά και για να διαφοροποιήσει το προϊόν της από άλλα λειτουργικά συστήματα όπως το MS-DOS, το οποίο είχε την φήμη του δύσχρηστου και τεχνικά απαιτητικού συστήματος, και το UNIX. Η ευκολία επικεντρωνόταν στο ότι το Mac OS ήταν από τα πρώτα λειτουργικά που χρησιμοποίησαν γραφικό περιβάλλον χρήστη αντί για γραμμή εντολών για την επικοινωνία ανθρώπου και υπολογιστή. Μεγάλο μέρος του αρχικού λογισμικού συστήματος ήταν αποθηκευμένο στη μνήμη ROM καθώς οι πρώτοι Macintosh δεν διέθεταν σκληρό δίσκο, με ενημερώσεις να παρέχονται συνήθως δωρεάν στους εμπόρους από την Apple, μέσω δισκετών.

Το 2000 η Apple κυκλοφόρησε τη δέκατη έκδοση του Mac OS, το Mac OS X, ουσιαστικά ένα ριζικά διαφορετικό λειτουργικό σύστημα βασισμένο στη διανομή BSD του UNIX και τον μικροπυρήνα Mach. Στο νέο πλέον λειτουργικό η Apple πρόσθεσε αρκετά συστατικά όπως ο "Finder" και το γραφικό περιβάλλον "Aqua", ολοκληρώνοντας το

βασισμένο σε γραφικό περιβάλλον λειτουργικό σύστημα που είναι το Mac OS X.

Η χρήση κουμπιών σαν χρωματιστές σταγόνες, οι διαφάνειες και φωτορεαλιστικά εικονίδια έφεραν υφή και χρώμα σε σχέση με τα προηγούμενα λειτουργικά. Πολλοί χρήστες εξέφρασαν την αρνητική άποψη ότι ήταν πολύ "χαριτωμένο" χωρίς επαγγελματικό ερέθισμα. Άλλοι πίστεψαν ότι το Aqua ήταν ένα γενναίο και πρωτοποριακό βήμα σε μία εποχή που τα γραφικά περιβάλλοντα ήταν απλά βαρετά. Παρόλο το διχασμό, η εμφάνισή του ήταν άμεσα αναγνωρίσιμη, ακόμα και πριν την πρώτη έκδοσή του Mac OS X, άλλοι προγραμματιστές άρχισαν να προσπαθούν να αντιγράψουν την εμφάνιση του Aqua.




Εικόνα 3.3: Το λειτουργικό σύστημα Mac OS στην έκδοση 9

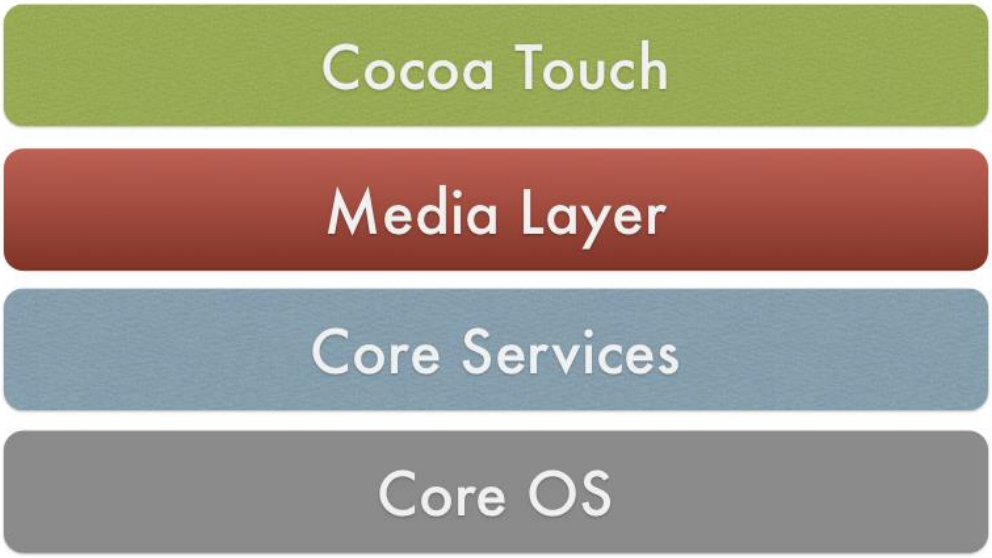
## 3.2.2 iOS SDK

Το iOS SDK (Software Development Kit) [18] διακρίνεται από την αρχιτεκτονική στρωμάτων. Είναι χωρισμένο σε τέσσερα στρώματα τα οποία από το χαμηλότερο ως προς το υψηλότερο είναι:

- Core OS

- Core Services
- Media
- Cocoa Touch

Οι θεμελιώδεις υπηρεσίες στις οποίες βασίζονται όλες οι εφαρμογές παρέχονται στα χαμηλότερα στρώματα. Στα υψηλότερα στρώματα περιέχονται υπηρεσίες και τεχνολογίες σχετικές με γραφικά και διεπαφές χρήστη.




Εικόνα 3.4: Τα στρώματα του iOS SDK

Θα αναπτύξουμε παρακάτω επιγραμματικά το κάθε στρώμα.

**Core OS**

Στο χαμηλότερο στρώμα υπάρχει το περιβάλλον των βασικών διεπαφών του λειτουργικού συστήματος όπως οι οδηγοί (drivers) των συσκευών, το σύστημα αρχείων (file system) και το περιβάλλον του πυρήνα (kernel). Το στρώμα διαχειρίζεται τα νήματα (threads), τη δικτύωση (networking), την εικονική μνήμη (virtual memory) και την ενδοδιεργασιακή επικοινωνία (interprocess communication).

Η βιβλιοθήκη LibSystem είναι γραμμένη στη γλώσσα προγραμματισμού C και παρέχει συναρτήσεις για πρόσβαση σε πολλά στοιχεία χαμηλού επιπέδου του συστήματος. Περιλαμβάνει πλήθος από χαρακτηριστικά, όπως πρόσβαση στο σύστημα αρχείων, υπηρεσίες δικτύωσης Bonjour και DNS, δέσμευση μνήμης, μαθηματικούς υπολογισμούς, τοπικές ρυθμίσεις και πληροφορίες για την τρέχουσα γλώσσα.

**Core Services**

Είναι το στρώμα που περιλαμβάνει βιβλιοθήκες και πλαίσια (framework) που χρησιμοποιούν οι εφαρμογές του λειτουργικού συστήματος.

Το σημαντικότερο framework είναι το Core Foundation το οποίο αποτελείται από διεπαφές σε γλώσσα C, οι οποίες παρέχουν θεμελιώδη διαχείριση δεδομένων και υπηρεσιών όπως διαχείριση ημερομηνίας και ώρας, διαχείριση προτιμήσεων χρήστη και δικτυακή ροή.

Υπάρχει επίσης το CFNetwork framework επίσης γραμμένο σε C, είναι υψηλής απόδοσης και παρέχει μία συλλογή αντικειμενοστραφών αφαιρέσεων για αλληλεπίδραση με δικτυακά πρωτόκολλα. Μπορούμε να το χρησιμοποιήσουμε για να κάνουμε απλούστερη τη δικτυακή επικοινωνία μέσω FTP/ HTTP.

Το Security framework μας παρέχει τους μηχανισμούς προστασίας των δεδομένων των εφαρμογών μας. Επίσης παρέχει διαχείριση πιστοποιητικών ασφαλείας και δημοσίων ιδιωτικών κλειδιών.

Η ενσωματωμένη βιβλιοθήκη SQLite [19] επιτρέπει στις εφαρμογές τη δημιουργία και τη χρήση αρχείων βάσεων δεδομένων. Η συγκεκριμένη βιβλιοθήκη ενώ είναι γενικής χρήσης εδώ έχει βελτιστοποιηθεί για αποθήκευση δεδομένων σε φορητές συσκευές.

**Media**

Σε αυτό το στρώμα περιλαμβάνονται οι τεχνολογίες πολυμέσων και γραφικών. Περιλαμβάνει υψηλού επιπέδου framework που επιτρέπουν προηγμένα γραφικά και animations, όπως και χαμηλού επιπέδου συναρτήσεις που επιτρέπουν υψηλότερο έλεγχο των επιθυμητών λειτουργιών.

Για τα γραφικά παρέχεται η τεχνολογία Quartz η οποία δίνει δυνατότητες ζωγραφικής σε δύο διαστάσεις, γραμμών και σχημάτων, μοτίβων και εικόνων. Επίσης το Core Animation αποτελείται από ένα API που μας επιτρέπει τη δημιουργία animation. Η τεχνολογία OpenGL ES η οποία είναι η mobile εκδοχή της OpenGL μας παρέχει τη δυνατότητα για animation με γραφικά επιπέδου 3D.

Για τον ήχο παρέχονται τα Core Audio και Audio Toolbox framework. Το Core Audio μας δίνει τη δυνατότητα για αναπαραγωγή και καταγραφή ροών ήχου, ενώ το Audio Toolbox χρησιμοποιείται για

την ενεργοποίηση δόνησης στις συσκευές που το υποστηρίζουν. Επίσης στο iOS υποστηρίζεται η βιβλιοθήκη OpenAL (Open Audio Library) για την διαχείριση του ήχου τριών διαστάσεων.

**Cocoa Touch**

Κάθε εφαρμογή του iOS χρησιμοποιεί το υψηλότερο στρώμα για την βασική διεπαφή του χρήστη με τη συσκευή. Αποτελείται από το UIKit framework το οποίο για την διεπαφή με το χρήστη περιλαμβάνει παράθυρα, στοιχεία ελέγχου (controls), απόψεις (views), διαχείριση κειμένου (text management) κλπ.

Επίσης το Cocoa Touch περιλαμβάνει τα framework Addressbook και Addressbook UI για την διαχείριση και πρόσβαση των εφαρμογών στον τηλεφωνικό κατάλογο της συσκευής.

Το Core Location framework προσδιορίζει την θέση του χρήστη εντοπίζοντας το τρέχον γεωγραφικό μήκος και πλάτος. Αυτό γίνεται με την χρήση του υλικού εντοπισμού (GPS) της συσκευής ή μέσω του δικτύου κινητής τηλεφωνίας και του διαδικτύου.

## 3.2.3 Objective C

Η γλώσσα προγραμματισμού Objective C [20] είναι η κύρια γλώσσα προγραμματισμού που χρησιμοποιεί η Apple για τα λειτουργικά συστήματα Mac OSX και iOS. Είναι αντικειμενοστρεφής γλώσσα (object oriented programming language) προγραμματισμού που σημαίνει ότι υποστηρίζει διαφορετικές κλάσεις, αντικείμενα, κληρονομικότητα, αφαιρετικές διεπαφές (πρωτόκολλα) και άλλα χαρακτηριστικά όπως άλλες αντικειμενοστρεφείς γλώσσες. Αποτελεί υπερσύνολο της ANCI C, άρα υποστηρίζει πλήρως την σύνταξη της C δηλαδή τύπους μεταβλητών, δομές επαναλήψεις όπως επίσης μας επιτρέπει να χρησιμοποιούμαι βιβλιοθήκες γραμμένες σε C.

Το περιβάλλον χρόνου εκτέλεσης (runtime) της Objective C αποτελείται από μία βιβλιοθήκη δυναμικής σύνδεσης με συναρτήσεις της C. Η βασική ιδιότητα του runtime είναι να θέσει σε λειτουργία το σύστημα μηνυματοδοσίας της Objective C.

Το σύστημα μηνυματοδοσίας (messaging system) χρησιμοποιείται για την αποστολή μηνυμάτων στα αντικείμενα. Εδώ έχουμε την

απευθείας κλήση των μεθόδων ενός αντικειμένου όπως στις άλλες αντικειμενοστραφείς γλώσσες με μία μικρή διαφοροποίηση.

Επίσης η γλώσσα αποτελείται από ένα σύστημα διαχείρισης μνήμης που ονομάζεται καταμέτρηση αναφορών (reference counting) [21]. Με το σύστημα διαχείρισης μνήμης ο προγραμματιστής πρέπει να στέλνει μηνύματα διατήρησης (retain) και απελευθέρωσης (release) για τα αντικείμενα που έχει θέσει, προκειμένου να επιτυγχάνεται ανακατανομή ή όχι της μνήμης.

Πρέπει να παρατηρήσουμε ότι όταν προγραμματίζουμε στην Objective C οι πιο πολλές κλάσεις που δηλώνουμε ξεκινάνε με τα γράμματα "NS" όπως NSString. Αυτό συμβαίνει λόγω του Cocoa Touch framework που αρχικά αναπτύχθηκε στις αρχές του 1980 από την Next για το λειτουργικό σύστημα NeXTStep [22] από το οποίο προέρχεται το Mac OS. Έτσι προέκυψαν τα αρχικά "NS".

## 3.2.4 Xcode

Το Xcode [23] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού που αποτελείται από μία σουίτα εργαλείων για την ανάπτυξη εφαρμογών στα λειτουργικά συστήματα Mac OS και iOS. Αναπτύχθηκε από την Apple το 2003, η πιο πρόσφατη έκδοση του είναι η 4.6 και παρέχεται δωρεάν μέσω του App Store.

Στο Xcode ουσιαστικά αναπτύσσεται ολόκληρος ο κώδικας της εφαρμογής μας. Εδώ δημιουργούμε τις κλάσεις του προγράμματος μας, δηλώνουμε τις μεταβλητές, και γενικότερα προγραμματίζουμε όλη την συμπεριφορά της εφαρμογής μας.

Παρακάτω γίνεται μία συνοπτική παρουσίαση των σημαντικότερων χαρακτηριστικών [24] των εργαλείων προγραμματισμού που μας προσφέρει το Xcode:

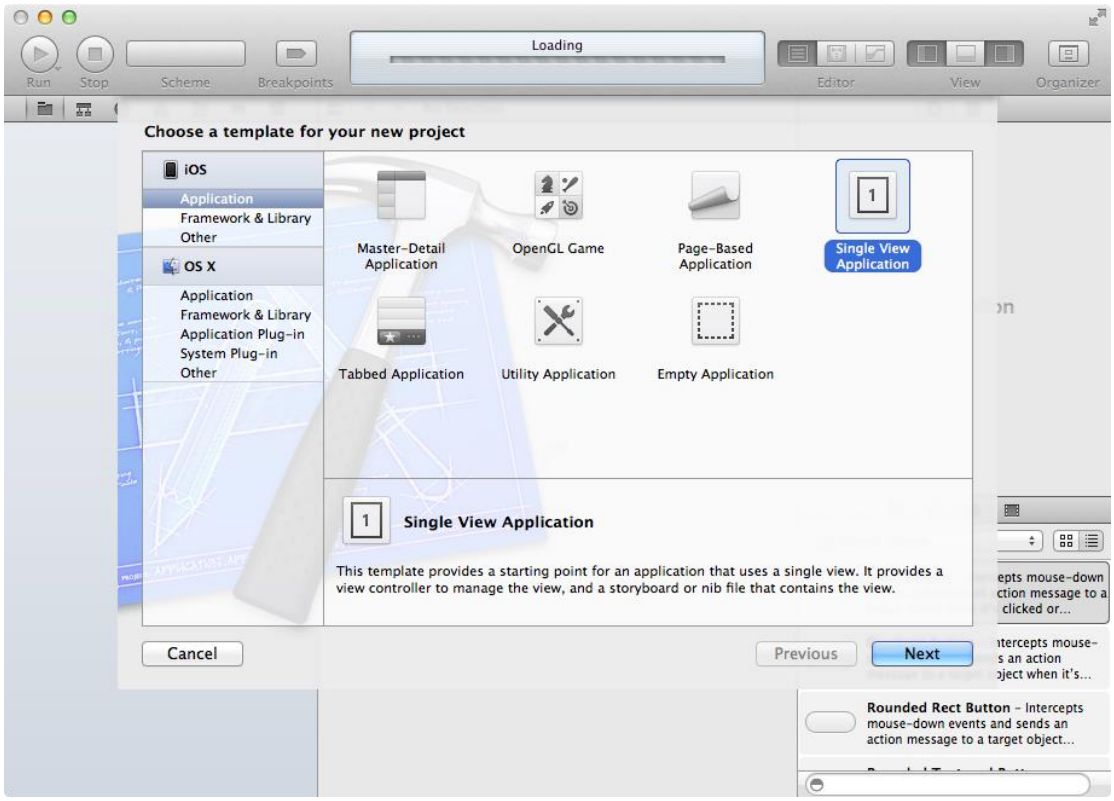
- **Assistant Editor (Βοηθός συντάκτη)**
  Το κουμπί του Βοηθού συντάκτη που παρέχεται από το Xcode χωρίζει τον επεξεργαστή του κώδικα στα δύο, δημιουργώντας ένα δευτερεύον παράθυρο που εμφανίζει αυτόματα τα αρχεία που είναι πιο χρήσιμα για σας με βάση τον κώδικα που είναι ενεργός προς επεξεργασία.
- **Source Editor (Συντάκτης κώδικα)**

Ένας επαγγελματικός συντάκτης που χρησιμοποιείται για την υλοποίηση του πηγαίου κώδικα με τη δυνατότητα επισήμανσης σύνταξης, καθώς και μηνυμάτων προειδοποίησης για τα λάθη, τύπου φυσαλίδας (Bubble), καθώς και άλλων θεματικών πληροφοριών για τον κώδικα.

- **Static Analysis (Στατική ανάλυση)**
  Το Xcode μας δίνει την δυνατότητα εύρεσης σφαλμάτων στον κώδικά μας πριν ακόμη ξεκινήσουμε την προσομοίωση της εφαρμογής μας. Αυτό επιτυγχάνεται με την στατική αναλυτή που παρέχει, η οποία δοκιμάζει χιλιάδες πιθανές γραμμές κώδικα μέσα σε λίγα δευτερόλεπτα, αναφέροντας πιθανά σφάλματα που θα μπορούσαν να έχουν παραμείνει κρυφά.

- **Interface Builder (Γραφικό περιβάλλον σχεδιασμού)**
  Ο interface builder είναι ενσωματωμένο πρόγραμμα σχεδιασμού γραφικών και εμφάνισης της εφαρμογής. Μέσω αυτού μπορούμε να τοποθετήσουμε κουμπιά, ετικέτες, εικόνες και γενικότερα να δούμε την εμφάνιση της εφαρμογής μας χωρίς να γράψουμε ούτε μία γραμμή κώδικα. Τέλος μπορούμε να συνδέσουμε τα στοιχεία του interface builder με τις μεθόδους που έχουμε γράψει στον συντάκτη του XCode έτσι ώστε να δέχονται τα δεδομένα που θα προβάλουν στην οθόνη.

- **iOS Simulator (Εξομοιωτής iOS)**
  Με το iOS SDK , το Xcode μπορεί να χτίσει, να εγκαταστήσει και να τρέξει την εφαρμογή μας χάρη του ενσωματωμένου εξομοιωτή για το iOS. Αυτός μας δίνει τη δυνατότητα προσομοίωσης της λειτουργίας της εφαρμογής μας με την χρήση εικονικού τηλεφώνου iPhone.




Εικόνα 3.5: Η οθόνη δημιουργίας ενός νέου project του Xcode

## 3.2.5 Wireframing (Balsamiq Mockups)

Το Wireframing [25] είναι ένα πολύ σημαντικό βήμα στο σχεδιασμό και την ανάπτυξη μίας εφαρμογής. Είναι ένα χαμηλού επιπέδου σχεδιάγραμμα (προσχέδιο, σκελετός, blueprint) της σχεδίασης μας (design), το οποίο βοηθάει τόσο εμάς όσο και τους πελάτες μας, στην απεικόνιση της δομής αλλά και της αίσθησης που αφήνει ο σχεδιασμός μας. Στα wireframes αναπαράγουμε τα στοιχεία όλης της εφαρμογής στις πραγματικές τους διαστάσεις και μεγέθη, αλλά παρόλα αυτά δεν περιέχει γραφιστικά στοιχεία, απλά τη θέση που θα πάρουν στον τελικό σχεδιασμό πάνω στην οθόνη. Μέσα από το wireframe μπορούμε να κάνουμε δομικές αλλαγές, αλλαγές σε κουμπιά, κείμενα, τίτλους και στοιχεία της εφαρμογής, σε πολύ μικρό χρόνο και έτσι μπορούμε να επιτύχουμε ευκολότερα μία πιο ξεκάθαρη εικόνα της εφαρμογής μας.

Μεταξύ σχεδιαστών ακολουθούνται διαφορετικές τακτικές κατά τη διαδικασία του σχεδιασμού, ωστόσο ο βέλτιστος και παραγωγικότερος πιθανότατα τρόπος ακολουθεί στο παρακάτω γράφημα.

## *3.2 Το λογισμικό (software)*


Εικόνα 3.6: Βέλτιστος τρόπος σχεδιασμού μίας εφαρμογής

Μετά την σύλληψη μίας ιδέας, σχεδιάζεται ένα πρόχειρο σκίτσο σε χαρτί και αφού τελειοποιηθεί περνάμε σε wireframe. Εδώ υπάρχει όλο το περιθώριο για αλλαγές, διορθώσεις και προσθήκες. Αφού κλείσουμε και συμφωνήσουμε στο wireframe, τότε έρχεται η σειρά του mockup, που ουσιαστικά είναι το "ντύσιμο" του wireframe με γραφικά, χρώματα, εικονίδια και οτιδήποτε άλλο συνοδεύει το σχέδιο της εφαρμογής μας. Τέλος το mockup περνάει σε επίπεδο development. Είναι πολύ συνηθισμένο για ορισμένους σχεδιαστές να παραλείπουν τελείως κάποιο ή κάποια από τα ενδιάμεσα παραπάνω βήματα.

Στο παραπάνω διάγραμμα, μετά την ιδέα ακολουθούν τα βήματα σχεδιασμού μέχρι την περαιτέρω ανάπτυξη (development) της εφαρμογής. Θα αναπτύξουμε λίγο αναλυτικότερα τί είναι κάθε βήμα και πώς μας βοηθάει στη βέλτιστη δημιουργία των σχεδίων (design) μας:

**Sketch (Σκίτσο)**

Το σκίτσο γίνεται συνήθως με χαρτί και μολύβι ή με κάποιο tabler και μας βοηθά να βρούμε γρήγορα και εύκολα τη βάση του σχεδίου μας. Μας βοηθά επίσης να επικοινωνήσουμε τον αρχικό σχεδιασμό της εφαρμογής και τη δομή της. Ακόμα στο σκίτσο μπορούμε εύκολα να επεξεργαστούμε το σχέδιο μας χρησιμοποιώντας γόμα.

**Wireframe (Σκελετός)**

Το wireframe, που ακολουθεί, είναι ουσιαστικά το "ντύσιμο" των σκίτσων με πιο καθαρά στοιχεία, γραμμές ακόμα και χρώματα. Το wireframe περιέχει την αληθινή διάσταση των πραγμάτων, με σωστές αποστάσεις, και σωστά μεγέθη στα διάφορα στοιχεία. Μέσα από το wireframe μπορούμε να δούμε ξεκάθαρα την δομή της εφαρμογής, και πλέον οι όποιες αλλαγές χρειαστούν να μπορούν να γίνουν πολύ γρήγορα.

**Mockup (Το πραγματικό σχέδιο)**

Τέλος το mockup είναι το πραγματικό look and feel της εφαρμογής. Χρώματα, γραμματοσειρές, στοιχεία, εικονίδια και ότι άλλο συνοδεύει το σχέδιο έρχονται και δένουν με τον σκελετό της εφαρμογής.

Υπάρχουν πολλά διαφορετικά δωρεάν και επί πληρωμή προγράμματα για την επίτευξη του wireframing. Επιλέχθηκε το Balsamiq Mockups [26] γιατί δίνει την αίσθηση της ζωγραφικής σε χαρτί στον προγραμματιστή, αλλά επειδή είναι ψηφιακό μπορούμε εύκολα να σβήσουμε και να αναδιατάξουμε τα στοιχεία πάνω στο σκίτσο μας. Είναι ένα πολύ εύχρηστο πρόγραμμα που σε επαγγελματικό επίπεδο δίνει τη δυνατότητα σε προγραμματιστές και σε πελάτες τους να καταλήξουν σε ένα προσχέδιο της εφαρμογής σε πραγματικό χρόνο. Τέλος μπορούμε να συντάξουμε μία πλήρη εικόνα της εφαρμογής μας πριν την έναρξη γραφής κώδικα.


Εικόνα 3.7: Το λογότυπο του λογισμικού Balsamiq Mockups

## 3.2.6 SQLite


Εικόνα 3.8: Το σήμα της SQlite

Η βιβλιοθήκη SQLite [19] μας προσφέρει διάφορα εργαλεία ανοιχτού κώδικα για την δημιουργία ερωτημάτων σε βάσεις δεδομένων. Σε αντίθεση με άλλα συστήματα διαχείρισης βάσεων δεδομένων, η SQLite δεν αποτελεί ξεχωριστή διαδικασία που γίνεται προσβάσιμη από μία εφαρμογή-πελάτη (client), αλλά αναπόσπαστο τμήμα της.

Η SQLite συμφωνεί με το πρότυπο ACID ( Atomicity , Consistency , Isolation , Durability) [27], που στην επιστήμη της πληροφορικής είναι

ένα σύνολο ιδιοτήτων που εγγυώνται ότι οι συναλλαγές της βάσης δεδομένων επεξεργάζονται με αξιοπιστία. Στο πλαίσιο των βάσεων δεδομένων, μία ενιαία λογική λειτουργία σχετικά με τα δεδομένα ονομάζεται μία συναλλαγή.

Σε αντίθεση με τις περισσότερες άλλες βιβλιοθήκες SQL , η SQLite δεν έχει ξεχωριστό διακομιστή (server). Διαβάζει και γράφει απευθείας σε συνηθισμένα αρχεία στο δίσκο . Μία πλήρης βάση δεδομένων SQL με πολλούς πίνακες ,δείκτες (triggers) και δεδομένα, περιέχεται σε ένα ενιαίο μικρό αρχείο στο δίσκο μεγέθους μερικών kilobyte. Υποστηρίζει την πλειοψηφία των οδηγιών SQL και γενικότερα δε χρειάζεται εγκατάσταση που σημαίνει ότι μπορούμε να την χρησιμοποιήσουμε σε οποιοδήποτε λειτουργικό σύστημα. Η βιβλιοθήκη έκανε την εμφάνιση της το 2000 και σήμερα είναι πλέον πολύ διαδομένη και χρησιμοποιείται από δημοφιλείς εφαρμογές όπως οι περιηγητές Mozilla Firefox και Google Chrome.

Επιλέχθηκε διότι κρίθηκε ο καλύτερος τρόπος αποθήκευσης των δεδομένων της εφαρμογής μας, αποφεύγοντας την χρήση αρχείων XML. Ένα ακόμη θετικό στοιχείο για την επιλογή της είναι το ότι δε χρειάζεται να ανησυχούμε για την πλατφόρμα ανάπτυξης της βάσης μας, καθώς μπορούμε εύκολα να τη μεταφέρουμε ανάμεσα στα διάφορα λειτουργικά.

Τέλος, υπάρχει μία αντίστροφη σχέση μεταξύ της χρήσης της μνήμης και της ταχύτητας. Γενικά, όσο περισσότερη μνήμη δώσουμε στην SQLite τόσο γρηγορότερα τρέχει. Παρ 'όλα αυτά, οι επιδόσεις της είναι συνήθως αρκετά καλές, ακόμη και σε περιβάλλοντα χαμηλής μνήμης. Λόγω του μεγέθους που καταλαμβάνει, η χρήση της είναι ιδανική για συσκευές όπου η μνήμη και η επεξεργαστική ισχύς είναι περιορισμένες, όπως κινητά τηλέφωνα, ταμπλέτες και συσκευές αναπαραγωγής μουσικής (MP3 Player).

# ΚΕΦΑΛΑΙΟ 4ο

## Σχεδίαση και ανάπτυξη εφαρμογής

Στο προηγούμενο κεφάλαιο έγινε η παρουσίαση του υλικού και του λογισμικού που χρησιμοποιήθηκε. Αντικείμενο του κεφαλαίου αυτού είναι η παρουσίαση της σχεδίασης της εφαρμογής και η χρήση των προηγούμενων τεχνολογιών ως προς αυτή.

## 4.1 Η προσέγγιση του θέματος ως προγραμματιστής

Για να εκμεταλλευτούμε στο έπακρο τα εργαλεία που μας δίνονται πρέπει αρχικά να ξεκαθαρίσουμε τον τύπο της εφαρμογής. Η δική μας εφαρμογή εντάσσεται στην κατηγορία των ψυχαγωγικών εφαρμογών - παιχνιδιών με στόχο τη δοκιμή των γνώσεων των χρηστών.

Για την ανάπτυξη μιας επιτυχημένης εφαρμογής στο iOS θα πρέπει να γνωρίζουμε συγκεκριμένα θέματα. Αυτά αφορούν τη σχεδίαση, τη σωστή διαχείριση της μνήμης, την ανταπόκριση, την κατανάλωση ενέργειας και τέλος την ασφάλεια τα οποία θα αναπτύξουμε παρακάτω συνοπτικά.

### 4.1.2 Η σχεδίαση (design)

Οι αποφάσεις που παίρνουμε για τη σχεδίαση έχουν μεγάλο αντίκτυπο στην απόδοση της εφαρμογής μας. Κατά τη σχεδίαση της βέλτιστης παρουσίασης μιας πληροφορίας στην οθόνη θα πρέπει να λάβουμε υπόψη την περιορισμένη έκταση οθόνης που έχουμε διαθέσιμη. Στόχος μας είναι να κατασκευάσουμε μια απλή και καλά οργανωμένη διεπαφή χρήστη. Διατηρώντας απλή τη διεπαφή χρήστη πετυχαίνουμε μια ελαφριά και ευχάριστη στη χρήση εφαρμογή.

### 4.1.3 Η διαχείριση μνήμης

Η εικονική μνήμη (swap) της iOS συσκευής περιορίζεται από τη φυσική μνήμη. Συγκεκριμένα δεν υπάρχει αρχείο swap. Οπότε όταν εξαντλείται όλη η διαθέσιμη μνήμη, η συσκευή επανεκκινεί τη λειτουργία της. Για να μην οδηγηθούμε ποτέ σε τέτοια περίπτωση το ίδιο το xCode μας παρέχει κάποιες λειτουργίες βελτιστοποίησης που μειώνουν σημαντικά την χρήση της φυσικής μνήμης.

Στόχος μας είναι να μην έχουμε διαρροές μνήμης, να απελευθερώνουμε τους πόρους το συντομότερο δυνατό μετά την ολοκλήρωση χρήσης και τέλος να επιβάλλουμε όρια στις προβολές. Αυτό σημαίνει ότι σε ένα κυλιόμενο μενού πρέπει να εμφανίζονται μόνο τα απαραίτητα στοιχεία χωρίς κενές γραμμές.

### 4.1.4 Ανταπόκριση

Για την σωστή λειτουργία η εφαρμογή θα πρέπει να ξεκινά γρήγορα και να παρέχει άριστη απόκριση στο χρήστη. Σαν προγραμματιστές αποφασίζουμε ποιές πληροφορίες χρειάζεται να προβάλλονται γρήγορα και ορίζουμε να φορτώνονται μόνο αυτές. Οι πληροφορίες που δεν χρειάζονται φροντίζουμε να φορτώνονται μετέπειτα. Η εφαρμογή μας εμφανίζει το κυρίως μενού στο χρήστη σε χρόνο λιγότερο των 0.2 δευτερολέπτων.

Τέλος η εφαρμογή μας θα πρέπει να μπορεί να τερματίσει ή να διακόψει οποιαδήποτε στιγμή χωρίς να χάνονται τα δεδομένα. Φροντίζουμε στην περίπτωση που ο χρήστης δεχτεί τηλεφώνημα η εφαρμογή να διακόψει «παγώνει», χωρίς να χρειάζεται να αποθηκεύσει μεγάλο όγκο δεδομένων.

### 4.1.5 Κατανάλωση ενέργειας

Η βελτίωση κατανάλωσης ενέργειας της συσκευής εξαρτάται συνήθως από τις λειτουργίες που χρησιμοποιεί η εφαρμογή μας. Στην περίπτωση σχεδίασης μιας εφαρμογής δικτύου (internet) ή χρήσης εύρεσης τοποθεσίας πρέπει να δηλώσουμε μια ανοχή ως προς τη

συχνότητα που θέλουμε να λαμβάνουμε την ενημέρωση της υπηρεσίας. Περιστασιακή ανανέωση των δεδομένων σημαίνει και μεγαλύτερη διάρκεια της μπαταρίας.

### 4.1.6 Ασφάλεια

Για να προστατέψουμε την ίδια την εφαρμογή, καταφεύγουμε σε πρακτικές συγγραφής ασφαλούς κώδικα με τη χρήση πιστοποιητικών ασφαλείας, υπηρεσιών εμπιστευτικότητας ή υπηρεσιών σύνταξης κρυπτογραφικών μηνυμάτων. Επίσης η Apple έχει χρησιμοποιήσει πολλαπλά επίπεδα ασφάλειας στο iOS. Ένα από τα μέτρα ασφαλείας είναι το sandbox. Το Sandboxing περιορίζει όλες τις εφαρμογές και τις κάνει να έχουν το δικό τους χώρο στο σύστημα. Οποιαδήποτε εφαρμογή είναι πολύ περιορισμένη για το πώς μπορεί να αλληλεπιδρά με άλλες εφαρμογές και τα δεδομένα τους.

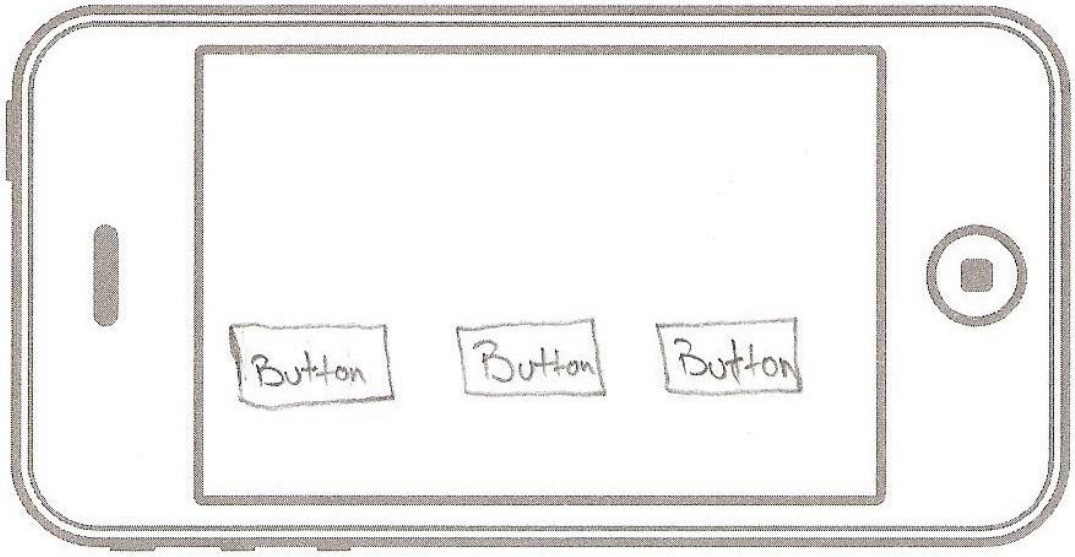
## 4.2 Η συγγραφή στην πράξη

Αφού ξεκαθαρίσαμε αρχικά τη πρέπει να προσέξουμε για την δημιουργία μιας εφαρμογής πρέπει να ξεκινήσουμε και να κάνουμε τη σκέψη μας πράξη.

Αρχικά για να δώσουμε εικόνα στην ιδέα μας πρέπει να δημιουργήσουμε ένα σκίτσο στο χαρτί σχετικά με το πώς φανταζόμαστε τη διεπαφή. Αυτή η διαδικασία είναι πολύ σημαντική γιατί μας βοηθά να καταλήξουμε στο επιθυμητό αποτέλεσμα με λιγότερο κόπο. Επίσης σχεδιάζοντας στο χαρτί, είναι εύκολο να δείξουμε τις ιδέες μας σε άλλους ώστε να δεχτούμε παρατηρήσεις και σχόλια για αυτές. Έπειτα, το τελικό αποτέλεσμα μπορεί να διαφέρει τελείως από το αρχικό σκίτσο.

Παρακάτω φαίνεται η αρχική σχεδίαση του παιχνιδιού ερωτοαπαντήσεων σε χαρτί η οποία και ακολουθήθηκε με κάποιες μικρές αλλαγές.

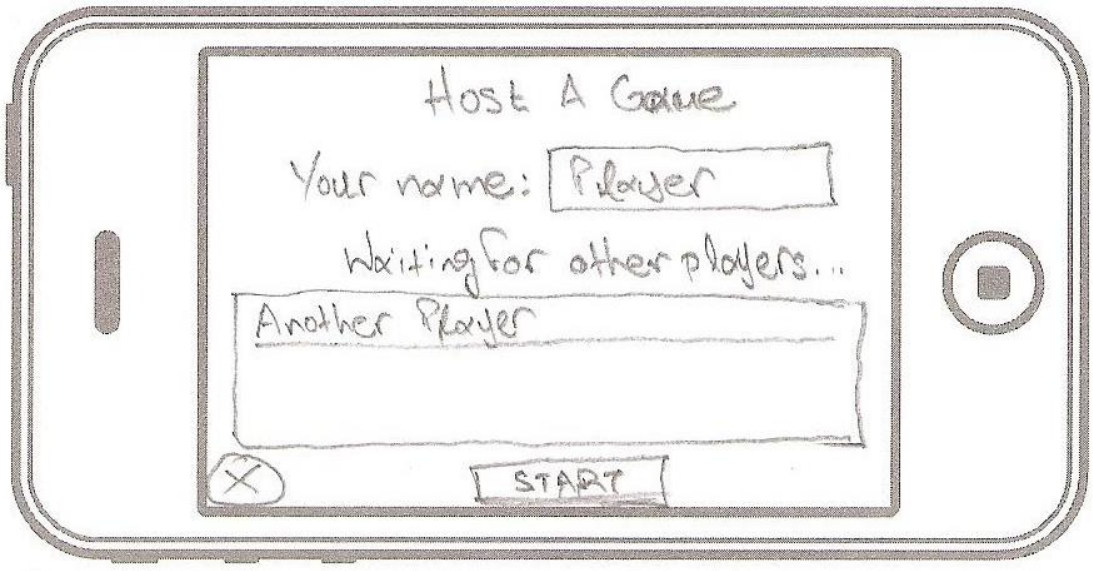
## *4.2 Η συγγραφή στην πράξη*




Εικόνα 4.1: Η οθόνη (view) υποδοχής της εφαρμογής

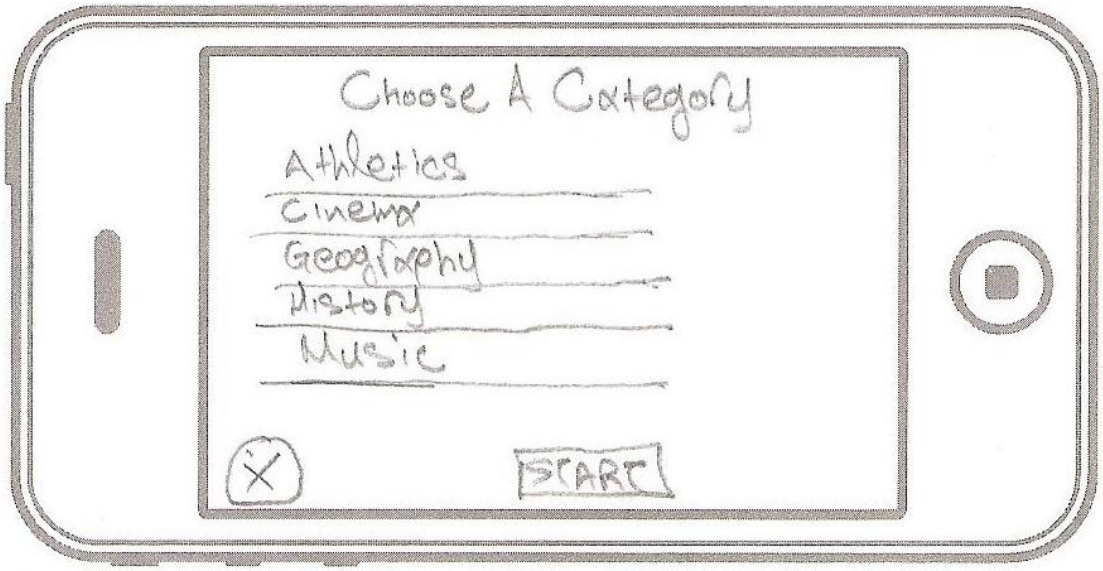
Στην παραπάνω εικόνα τοποθετούμε στο χαρτί τη σκέψη μας για το μενού επιλογών που θα προβάλλεται στο χρήστη όταν η εφαρμογή ξεκινά. Υπάρχουν τρία κουμπιά για τις επιλογές χρήσης της εφαρμογής, μία για το single player, μία για το host game και μία για την επιλογή join game.

Παρακάτω έχουμε την όψη της οθόνης του διακομιστή που θα φιλοξενεί το δικτυακό παιχνίδι. Έχουμε την ετικέτα της οθόνης στο επάνω μέρος και ακολουθούν, ένα πλαίσιο για την προσθήκη του ονόματος διακομιστή ενώ στο κάτω μέρος έχουμε ένα κυλιόμενο πίνακα ο οποίος θα προβάλει τα ονόματα των υπολοίπων παικτών που συνδέονται στο παιχνίδι. Στην επιλογή του πελάτη έχουμε ακριβώς την ίδια οθόνη η οποία λειτουργεί ανάστροφα.




Εικόνα 4.2: Η οθόνη (view) του διακομιστή

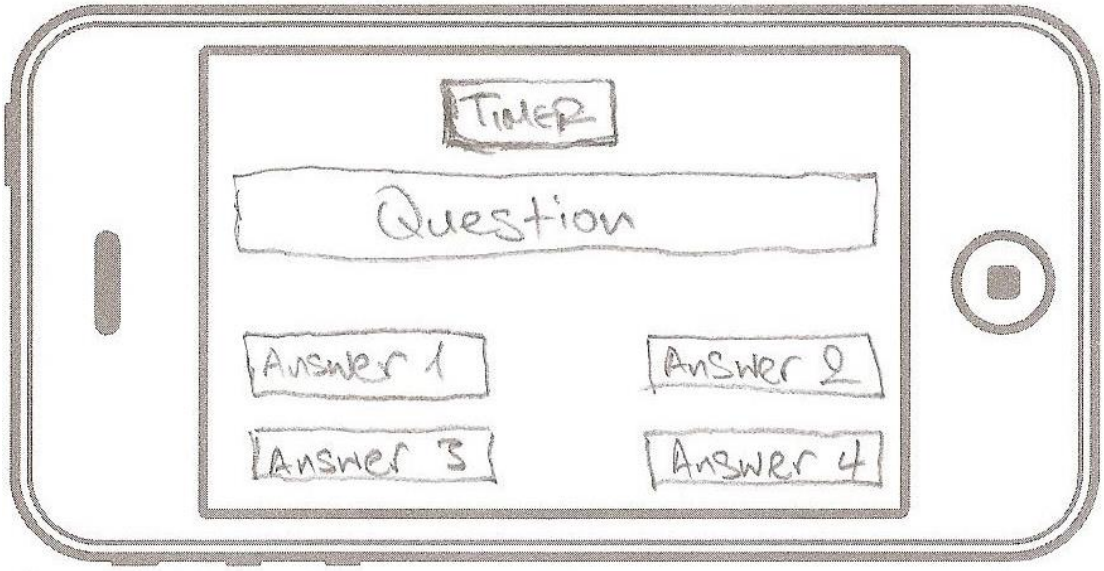
## *4.2 Η συγγραφή στην πράξη*




Εικόνα 4.3: Η οθόνη επιλογής κατηγορίας

Εδώ βλέπουμε την οθόνη επιλογής κατηγορίας η οποία αποτελείται από ένα κυλιόμενο μενού το οποίο μελλοντικά μπορεί να αναπτυχθεί προσθέτοντας νέες κατηγορίες στο παιχνίδι. Επίσης έχουμε δύο κουμπιά, αυτό της εξόδου και αυτό της έναρξης.

Τέλος ακολουθεί η οθόνη του παιχνιδιού η οποία αποτελείται από ετικέτες και κουμπιά. Η πάνω ετικέτα φιλοξενεί το χρόνο του παιχνιδιού. Έπειτα ακολουθεί η ερώτηση και τα κουμπιά για τις απαντήσεις των ερωτήσεων.



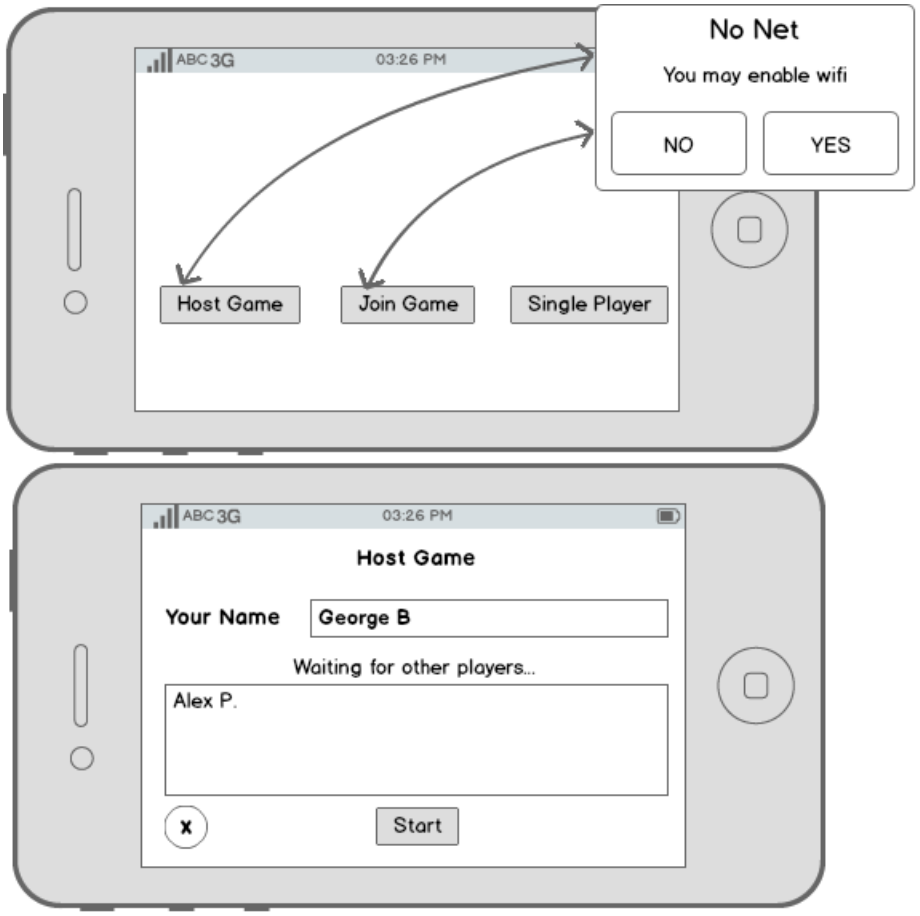

Εικόνα 4.4: Η οθόνη του παιχνιδιού

### 4.2.1 Από το σκίτσο στον υπολογιστή

Όπως αναφέραμε η σχεδίαση είναι πολύ σημαντική. Αφού λοιπόν έχουμε δώσει μορφή στην εφαρμογή μας στο χαρτί τη σχεδιάζουμε και στον υπολογιστή με τη χρήση του προγράμματος Balsamiq Mockups.

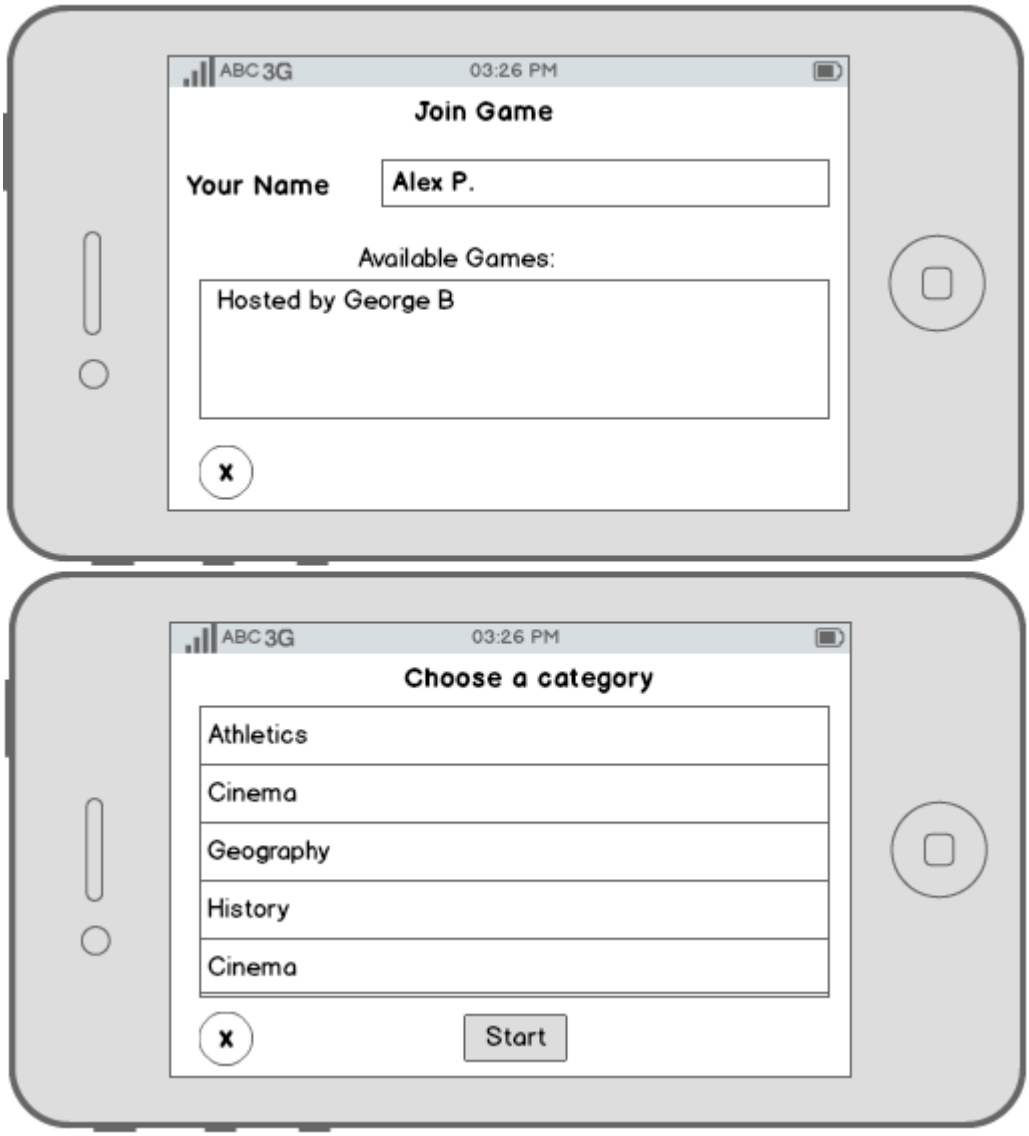
Μέσω αυτού του προγράμματος μας παρέχεται η δυνατότητα εύκολων αλλαγών των όψεων (οθονών) της εφαρμογής. Επίσης μπορούμε να πετύχουμε ακριβείς διαστάσεις των στοιχείων που θα τοποθετηθούν στην οθόνη με ένα ποιο ικανοποιητικό αποτέλεσμα για την όψη της εφαρμογής μας. Αυτό το αποτέλεσμα είναι πολύ σημαντικό στην αγορά εργασίας γιατί έτσι μπορούμε να δώσουμε στους πελάτες μας μια πιο ξεκάθαρη εικόνα της δουλειά που θα ετοιμάσουμε.

Ακολουθούν παρακάτω συνολικά όλα τα σκίτσα (mockup) που απαρτίζουν τις όψεις του παιχνιδιού ερωτοαπαντήσεων.

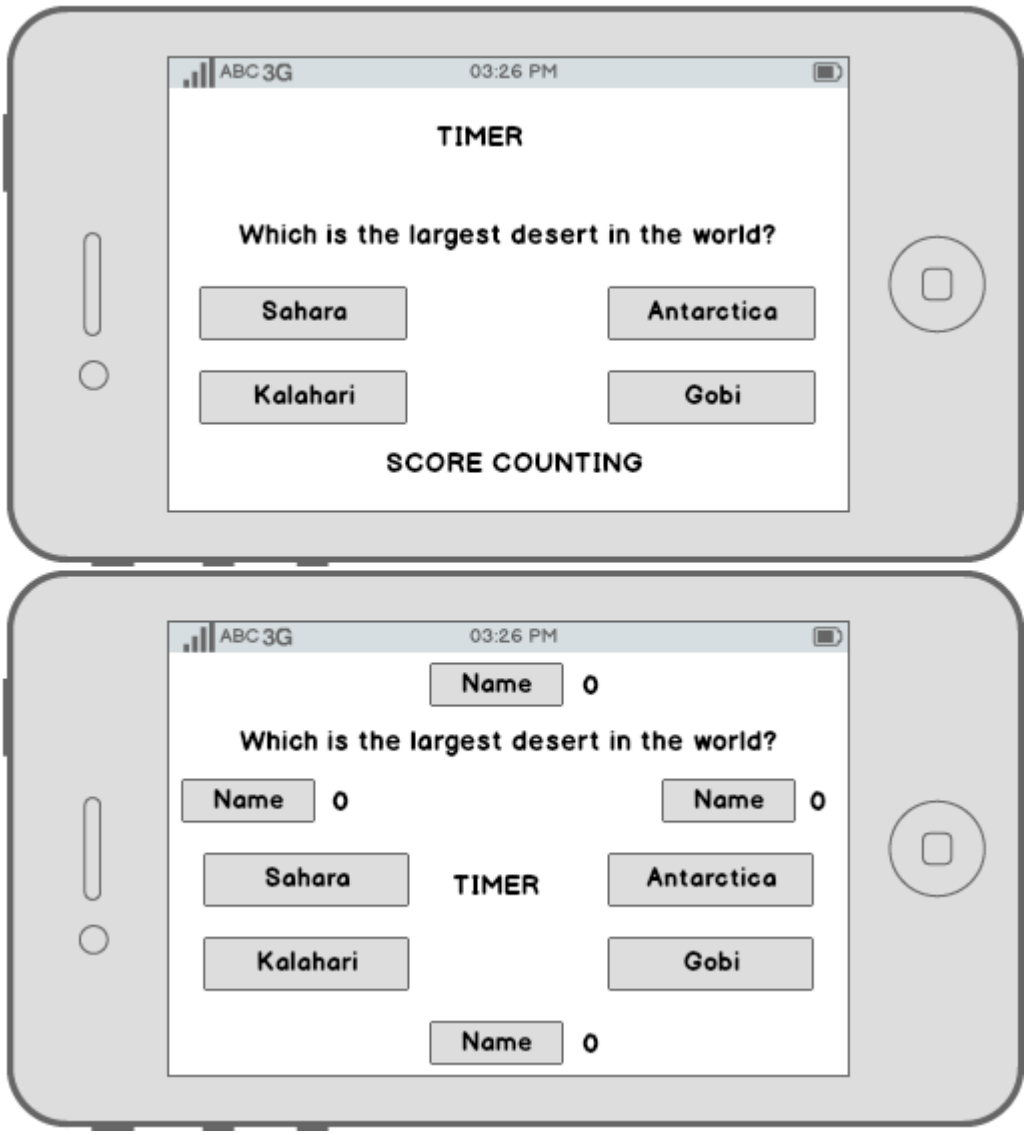



Εικόνα 4.5: Mockup του μενού υποδοχής και διακομιστή

## *4.2 Η συγγραφή στην πράξη*




Εικόνα 4.6: Mockup πελάτη και επιλογής κατηγορίας

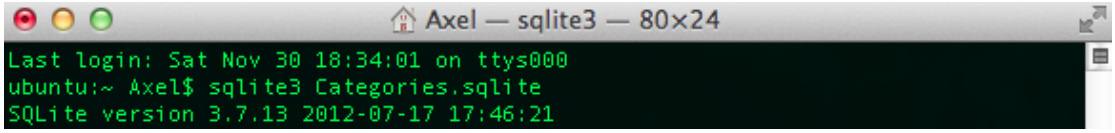



Εικόνα 4.7: Mockup παιχνιδιού απλού χρήστη και δικτυακού

## 4.3 Δημιουργία της βάσης δεδομένων

Ξεκινάμε με τη δημιουργία της βάσης γιατί είναι ένα ξεχωριστό αρχείο το οποίο έπειτα θα εισαχθεί στην εφαρμογή μας και θα αλληλεπιδρά με αυτή.

Για τη δημιουργία της βάσης δεδομένων ανοίγουμε το τερματικό (terminal) στον υπολογιστή μας. Για να δημιουργήσουμε το αρχείο πληκτρολογούμε την εντολή “sqlite3”:



## *4.3 Δημιουργία της βάσης δεδομένων*

Έπειτα δημιουργούμε τον πίνακα Categories που περιέχει τις κατηγορίες του παιχνιδιού με την εντολή “create table”:

```
sqlite> create table Categories (ID INTEGER PRIMARY KEY AUTOINCREMENT, Names VAR
CHAR(20));
```

Συνεχίζουμε με το δεύτερο πίνακα ο οποίος περιέχει τις ερωτήσεις με τις απαντήσεις τους:

```
sqlite> create table Questions (Category INTEGER(2), Question VARCHAR(150), Answ
er1 VARCHAR(50), Answer2 VARCHAR(50), Answer3 VARCHAR(50), Answer4 VARCHAR(50),
Used VARCHAR(4), FOREIGN KEY (Category) REFERENCES Categories (ID));
```

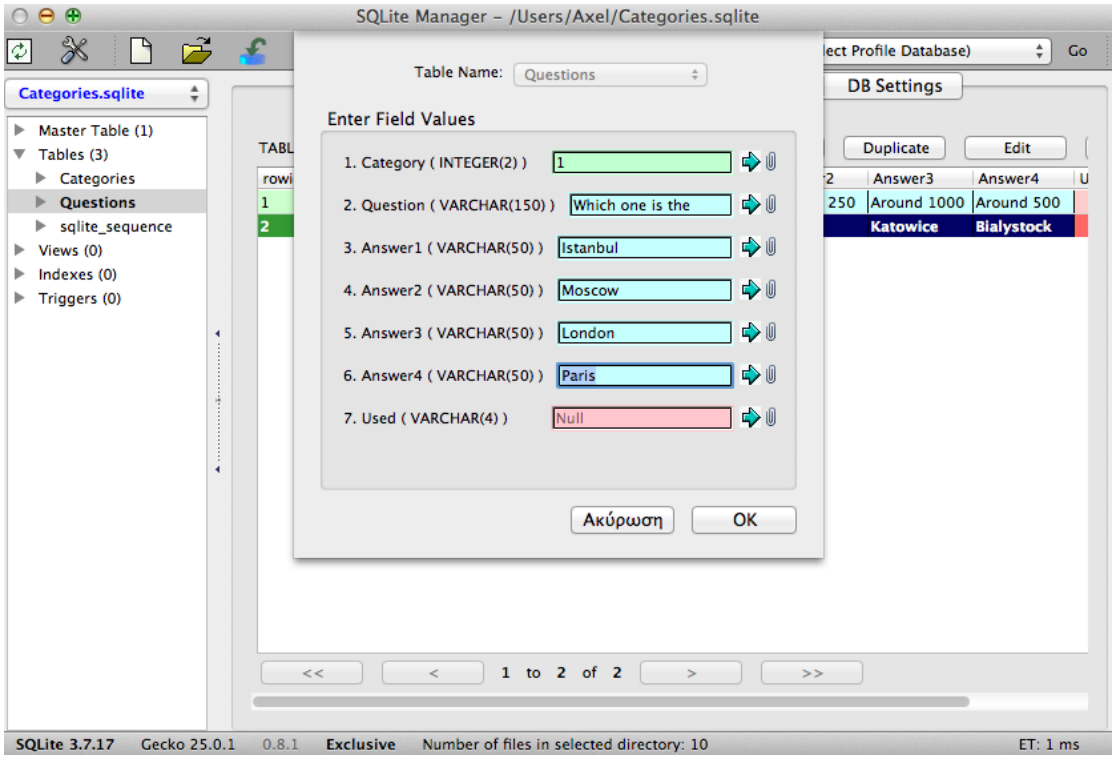
Και τέλος εισάγουμε στον πίνακα Categories τα στοιχεία με την εντολή “insert into”:

```
sqlite> insert into Categories(ID, Names) values ("1" , "Geography");
sqlite> insert into Categories(ID, Names) values ("2" , "Athletics");
sqlite> insert into Categories(ID, Names) values ("3" , "Cinema");
sqlite> insert into Categories(ID, Names) values ("4" , "History");
sqlite> insert into Categories(ID, Names) values ("5" , "Music");
```

Έτσι έχουμε δημιουργήσει με επιτυχία τη βάση μας και έχουμε προσθέσει σε αυτή τις κατηγορίες μας. Βγαίνουμε κλείνοντας το τερματικό ή πληκτρολογώντας σε αυτό την εντολή “.exit”.

Με τον ίδιο τρόπο μπορούμε να προσθέσουμε στον πίνακα των ερωτήσεων στοιχεία αλλά επειδή ο όγκος γραμμών κώδικα είναι μεγαλύτερος προτιμούμε να διαχειριστούμε τη βάση μας με το γραφικό περιβάλλον του διαχειριστή SQLite του περιηγητή Firefox. Αυτό είναι ένα πρόσθετο που μας διευκολύνει την προσθήκη των ερωτήσεων στη βάση δεδομένων κερδίζοντας έτσι κατασκευαστικό χρόνο.

Για την ορθή λειτουργία της βάσης η σωστή απάντηση για κάθε ερώτηση τοποθετείται στο πεδίο Answer 1 του πίνακα Questions. Αυτό είναι ένα σημείο το οποίο πρέπει να προσέξουμε, γιατί αν τοποθετηθεί σε άλλο πεδίο η σωστή απάντηση κατά τη λειτουργία του παιχνιδιού θα είναι λάθος.

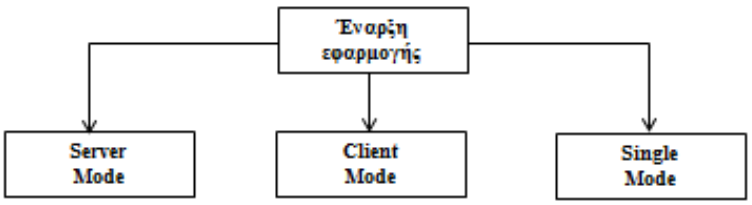



Εικόνα 4.8: Ο διαχειριστής βάσεων δεδομένων του Firefox

## 4.4 Διαγράμματα ροής εκτέλεσης

Πριν ξεκινήσουμε να αναπτύσσουμε την εφαρμογή μας προγραμματιστικά πρέπει να ξεκαθαρίσουμε τον τρόπο λειτουργίας αυτής. Αυτό συνήθως αναλύεται από τις ενέργειες του χρήστη σε σχέση με την εφαρμογή.

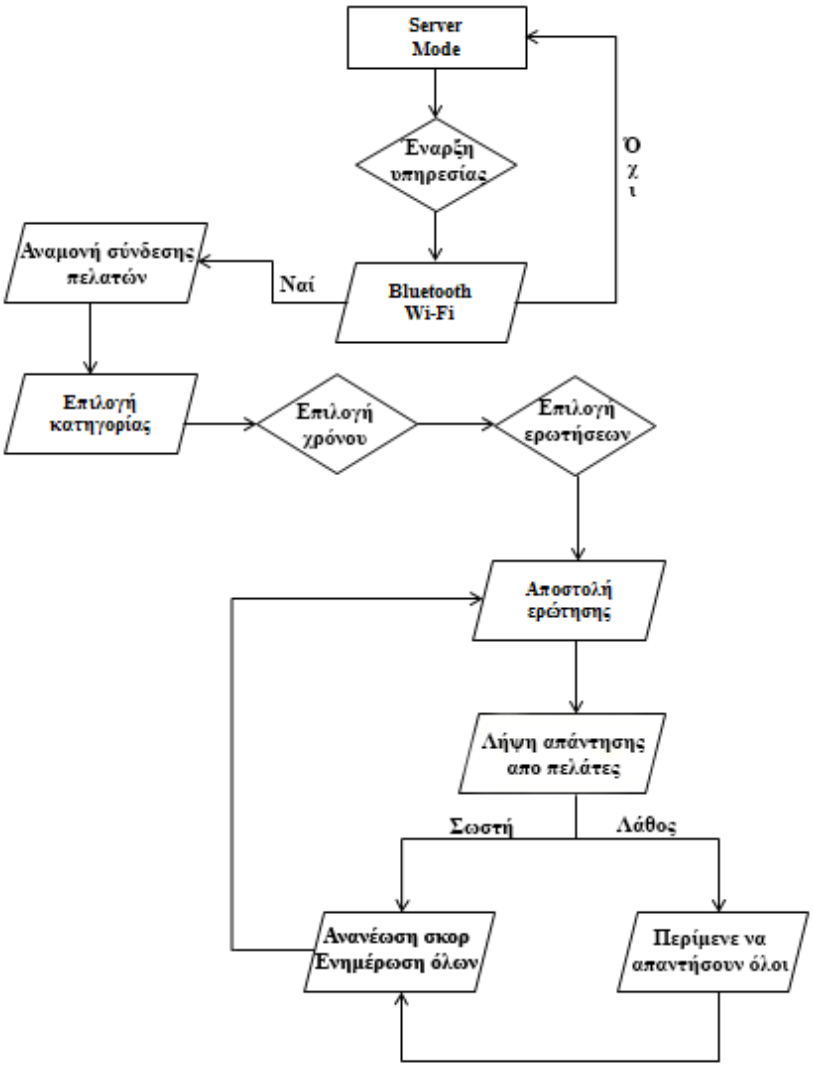
Αρχικά ξεκινώντας την εφαρμογή ο χρήστης μπορεί να επιλέξει τον τρόπο συμπεριφοράς της συσκευής του (Εικόνα 4.9). Υπάρχουν τρείς επιλογές. Η πρώτη αφορά τη φιλοξενία ενός δικτυακού παιχνιδιού (διακομιστής) και η δεύτερη ως πελάτης σε ένα δικτυακό παιχνίδι. Αυτές οι δύο επιλογές χρησιμοποιούν τις τεχνολογίες δικτύωσης (Bluetooth Wi-fi) για τη χρήση της εφαρμογής. Η τρίτη επιλογή αφορά την απλή χρήση η οποία δεν απαιτεί κάποια τεχνολογία.




Εικόνα 4.9: Επιλογές χρήστη κατά την έναρξη

## *4.4 Διαγράμματα ροής εκτέλεσης*

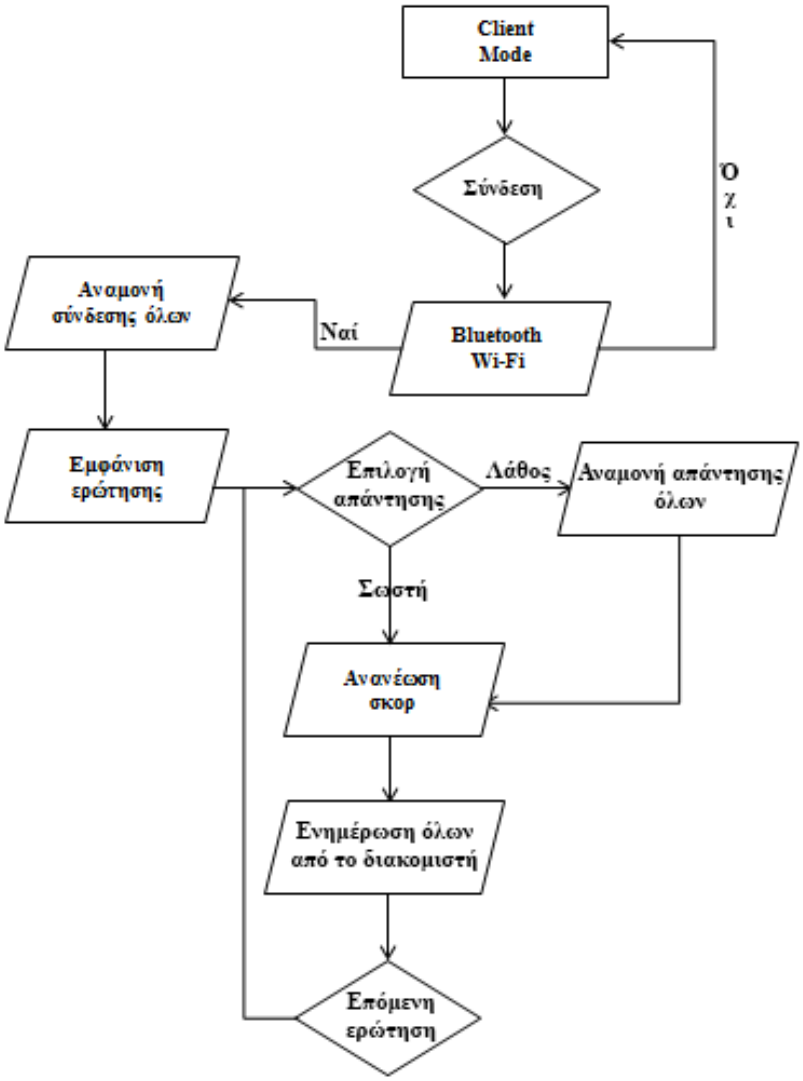
Στη συνέχεια (Εικόνα 4.10) αναλύονται οι επιλογές που έχει ο χρήστης όταν συνδέεται ως διακομιστής. Με την επιλογή του διακομιστή η συσκευή ξεκινά την εκπομπή σήματος που βλέπουν οι πελάτες. Όταν οι τεχνολογίες δικτύωσης είναι κλειστές ο χρήστης δέχεται ένα μήνυμα για να ανοίξει μία από τις δύο. Έπειτα περιμένει τη σύνδεση των πελατών με μέγιστο αριθμό τους τρείς. Όταν ολοκληρωθεί η σύνδεση επιλέγει το χρόνο και τις ερωτήσεις που θα παίξει με τους υπόλοιπους παίκτες.




Εικόνα 4.10: Διάγραμμα ροής διακομιστή

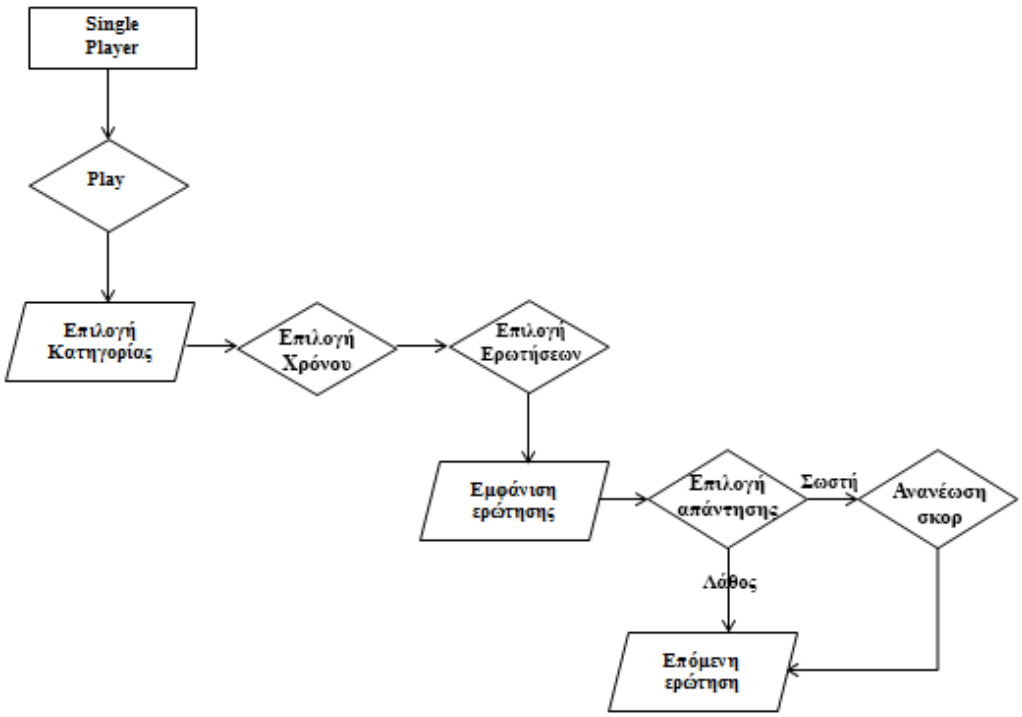
Στην περίπτωση του πελάτη (Εικόνα 4.11) ο χρήστης βλέπει τους διαθέσιμους διακομιστές και συνδέεται σε ένα από αυτούς. Όταν συνδεθούν όλοι, ο διακομιστής θέτει το παιχνίδι και έπειτα εμφανίζεται η πρώτη ερώτηση.

## *4.4 Διαγράμματα ροής εκτέλεσης*




Εικόνα 4.11: Το διάγραμμα ροής πελάτη

Τέλος, (Εικόνα 4.12), έχουμε το διάγραμμα του απλού χρήστη. Επιλέγοντας ο χρήστης να παίξει μόνος του μεταβαίνει στην επιλογή της κατηγορίας και έπειτα θέτει χρόνο και ερωτήσεις. Εδώ δεν χρειάζονται οι τεχνολογίες δικτύωσης.



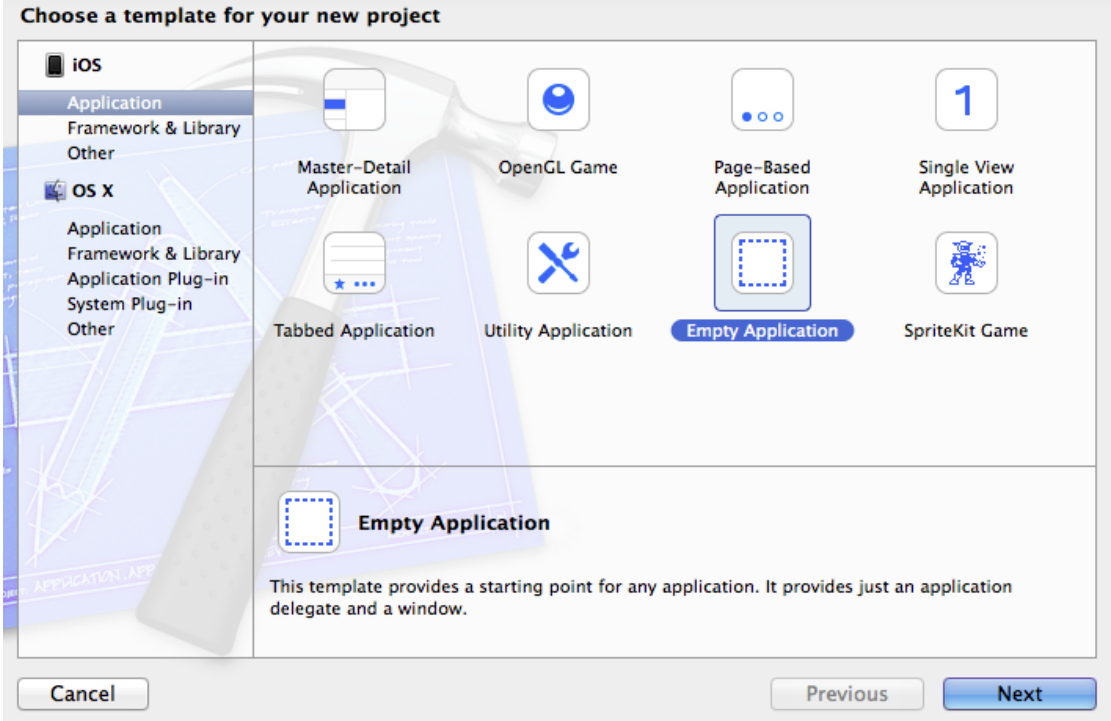

Εικόνα 4.12: Διάγραμμα ροής απλού χρήστη

## 4.5 Δημιουργία της εφαρμογής (project) στο xCode

Επιλέξαμε να μην επικολλήσουμε όλο των κώδικα αναφέροντας τι κάνει, αλλά να εστιάσουμε στον τρόπο σκέψης κατά το σχεδιασμό, γιατί η εφαρμογή που πρόκειται να δημιουργήσουμε είναι εκτενής σε κώδικα. Θα παρουσιάσουμε επιλεκτικά την ανάπτυξη της εφαρμογής ώστε να αναφερθούν όλα τα σημαντικά κομμάτια του κώδικα χωρίς να γίνουμε κουραστικοί.

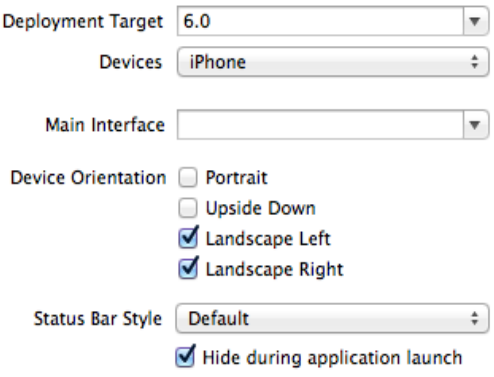
Εκτελώντας το xCode και επιλέγοντας “New Project” βλέπουμε κάποια έτοιμα πρότυπα εφαρμογής. Για την εφαρμογή μας θα δημιουργήσουμε ένα “Empty Application”. Αφού επιλέξουμε τον τύπο εφαρμογής δίνουμε ένα όνομα στο project και ολοκληρώνουμε την δημιουργία του. Για την παρουσίαση των δεδομένων στην οθόνη της έξυπνης συσκευής θα μπορούσε να χρησιμοποιηθεί κάποιος από τους έτοιμους διαθέσιμους διαχειριστές όψης (view controller) του XCode, αναλύοντας όμως την ιδέα στο μυαλό μας προτιμήθηκε η χρήση κενών view controller ώστε να δοθεί στην εφαρμογή ένας ποιό “παιχνιδιάρικος” χαρακτήρας με custom οθόνες.

## *4.5 Δημιουργία της εφαρμογής (project) στο xCode*




Εικόνα 4.13: Δημιουργία του Empty Application

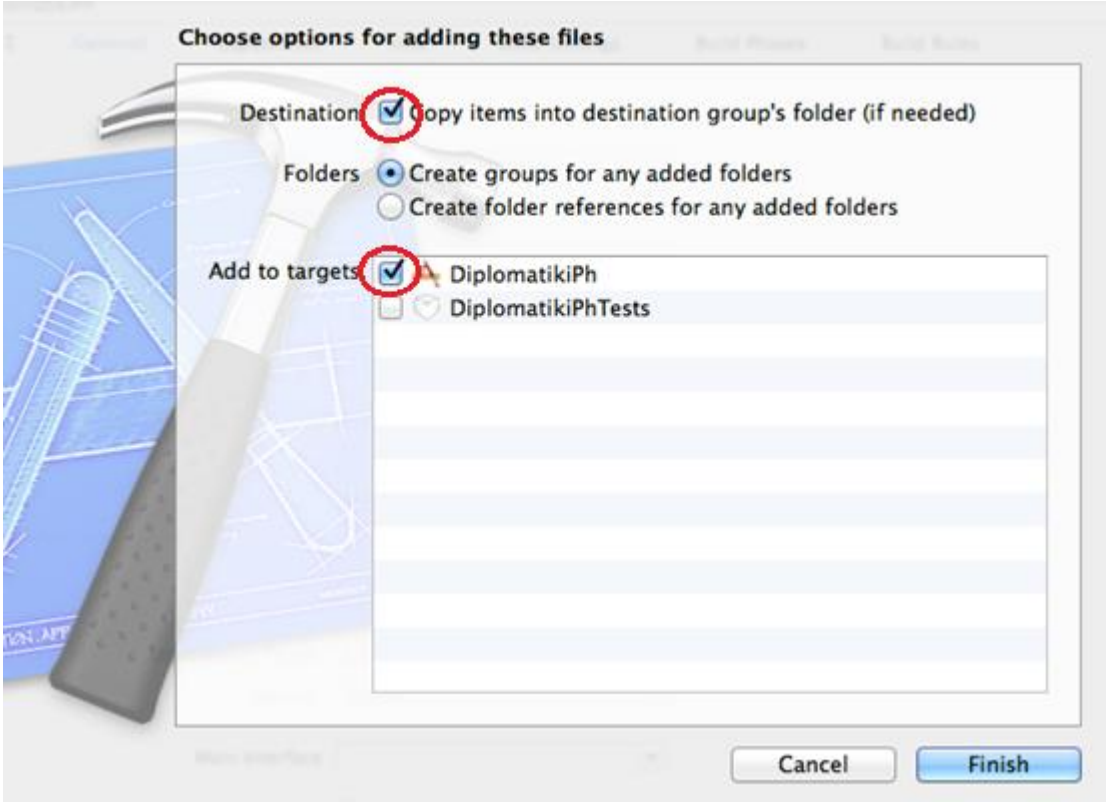
Με τη χρήση κενών view controller μας παρέχεται η δυνατότητα να συνδυάσουμε σε μια οθόνη διάφορα στοιχεία όπως για παράδειγμα κουμπιά, ετικέτες και μια προβολή πίνακα (table view). Όταν είναι έτοιμο το project, επιλέγουμε από το αριστερό μενού του xCode το αρχικό αρχείο της εφαρμογής που περιέχει όλα τα υποαρχεία που θα δημιουργήσουμε και στο στην καρτέλα "General" του μεσαίου μενού δηλώνουμε το "Deployment Target", δηλαδή από ποιο iOS SDK και έπειτα να είναι συμβατή η εφαρμογή μας. Επίσης, ενεργοποιούμε την επιλογή "Device Orientation" στις δύο επιλογές προσανατολισμού που θέλουμε να εμφανίζεται η εφαρμογή μας "Landscape Left" και "Landscape Right".




Εικόνα 4.14: Επιλογές εμφάνισης εφαρμογής

Έπειτα, προσθέτουμε τα Frameworks που μας χρειάζονται για τη λειτουργία της βάσης δεδομένων “libsqlite3.0.dylib” και για το δικτυακό μέρος της εφαρμογής που θα αναπτύξουμε στο επόμενο κεφάλαιο το “Gamekit.framework”. Τέλος, τραβάμε (drag and drop) από το φάκελο του υπολογιστή μας που έχουμε δημιουργήσει τη βάση δεδομένων στο φάκελο “Supporting Files” του project μας και τσεκάρουμε να αντιγραφεί η βάση στο φάκελο.

Σε αυτό το σημείο έχουμε προσθέσει όλα τα πρόσθετα αρχεία που χρειαζόμαστε για να αρχίσουμε να προγραμματίζουμε την εφαρμογή μας (Εικόνα 4.15).




Εικόνα 4.15: Προσθήκη αρχείων στο project

## 4.6 Δημιουργία των επιμέρους controllers και views

Κάθε διεπαφή (όψη) της εφαρμογής θα φορτώνει και ένα διαφορετικό ViewController ο οποίος με τη σειρά του θα προβάλλει ένα ξεχωριστό View. Στη συγκεκριμένη εφαρμογή θα πρέπει να κατασκευάσουμε συνολικά δέκα νέους ViewController. Για να δημιουργήσουμε κάθε controller κάνουμε τα εξής βήματα:

- Από το project επιλέγουμε File -> New File -> Objective-C class τύπου "UIViewController" με συνοδευτικό αρχείο ".xib" και προσθέτουμε ένα όνομα για τον κάθε controller. Για κάθε κλάση δημιουργούνται τρία αρχεία.
- Οι κλάσεις χωρίζονται σε δύο κατηγορίες: το interface (διεπαφή ή προβολή δεδομένων) και το implementation (εφαρμογή). Το interface περιέχει τη δήλωση της κλάσης και βρίσκεται στο αρχείο με την κατάληξη ".h". Το implementation περιέχει τον κώδικα που καθορίζει μια κλάση και συνήθως βρίσκεται στο αρχείο με την κατάληξη ".m"
- Το αρχείο ".xib" αφορά τον interface builder για τη συγκεκριμένη διεπαφή και χρησιμοποιείται για την προσθήκη στοιχείων και τη σύνδεση αυτών με τις κλάσεις στο αντίστοιχο αρχείο ".h".
- Για κάθε διεπαφή που χρησιμοποιεί στοιχεία άλλης κλάσης τη δηλώνουμε με τη μεταβλητή "#import filename.h".
- Για κάθε διεπαφή πρέπει να δηλώσουμε στα αρχεία ".m" την υποστήριξη μόνο οριζόντιου προσανατολισμού τοποθετώντας τον κώδικα:

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    return UIInterfaceOrientationIsLandscape(interfaceOrientation);
}
```

## 4.7 Δημιουργία του μενού υποδοχής

Έχουμε δημιουργήσει ένα νέο "UIViewController" με το όνομα "ViewController" και ξεκινάμε με το "ViewController.xib" αρχείο. Προσθέτουμε στη διεπαφή τρία κουμπιά από το μενού του interface builder και τα ονομάζουμε αντίστοιχα "Host Game", "Join Game" και "Single Player", τα όποια όμως δεν μπορούν να χρησιμοποιηθούν ακόμη. Για να χρησιμοποιήσουμε τα κουμπιά θα πρέπει να δηλώσουμε τις μεθόδους τους στο αρχείο "ViewController.h" χρησιμοποιώντας τον παρακάτω κώδικα:

## *4.7 Δημιουργία του μενού υποδοχής*

```
@interface ViewController : UIViewController

@property (nonatomic, weak) IBOutlet UIButton *hostGameButton;
@property (nonatomic, weak) IBOutlet UIButton *joinGameButton;
@property (nonatomic, weak) IBOutlet UIButton *singlePlayerGameButton;

@end
```

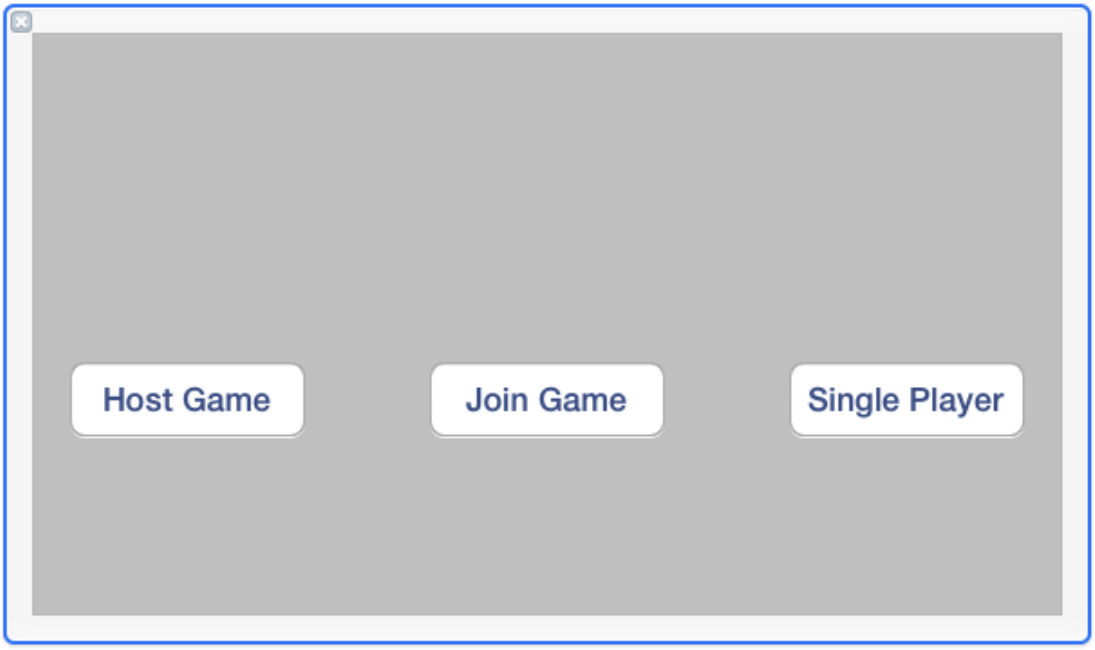
Για να ολοκληρώσουμε τη δημιουργία των κουμπιών και να εφαρμόσουμε τις μεθόδους που δηλώσαμε παραπάνω προσθέτουμε στο αρχείο "ViewController.m" τον κώδικα:

```
@implementation ViewController

@synthesize hostGameButton = _hostGameButton;
@synthesize joinGameButton = _joinGameButton;
@synthesize singlePlayerGameButton = _singlePlayerGameButton;
```

Και τέλος με την παρακάτω μέθοδο καλούμε ξεχωριστά για το κάθε κουμπί ένα νέο controller:

```
- (IBAction)singlePlayerGameAction:(id)sender
{
    if (_buttonsEnabled)
    {
        ChooseViewController *controller = [[ChooseViewController alloc]
                                            initWithNibName:@"ChooseViewController" bundle:nil];
        controller.delegate = self;
        [self presentViewController:controller animated:YES completion:nil];
        }
}
```

Στο παράδειγμα αυτό βλέπουμε ότι πατώντας το κουμπί "Single Player" οδηγούμαστε σε μια νέα κατάσταση. Καλούμε δηλαδή το νέο μας controller με όνομα "ChooseViewController" ο οποίος είναι υπεύθυνος για την επιλογή της κατηγορίας ερωτήσεων που θα επιλέξει ο χρήστης. Πρέπει να τονίσουμε ότι στον interface builder υπάρχει το αρχείο File's Owner, το οποίο συλλέγει όλες τις μεθόδους και τα στοιχεία της διεπαφής για να τα φορτώνει όταν καλέσουμε τον controller.




Εικόνα 4.16: Το μενού υποδοχής από τον interface builder

## 4.7.1 Δημιουργία της διεπαφής επιλογής κατηγορίας

Δημιουργούμε ένα νέο "UIViewController" με το όνομα "ChooseViewController" και ξεκινάμε με το αρχείο "ChooseViewController.xib". Σε αυτή τη διεπαφή θα μπορούσαμε να χρησιμοποιήσουμε κουμπιά, τα οποία θα ονοματίζαμε τραβώντας τα ονόματα των κατηγοριών από τη βάση δεδομένων. Αυτό όμως αποδείχτηκε μη λειτουργικό κατά την εξέλιξη της εφαρμογής. Προτιμήθηκε η επιλογή ενός κυλιόμενου πίνακα γραμμών (table view), γιατί δίνει τη δυνατότητα προσθήκης περεταίρω δεδομένων.

Αρχικά εισάγουμε στο "ChooseViewController.xib" ένα table view από τον interface builder και δύο κουμπιά ένα για την «έναρξη» του παιχνιδιού και ένα για την «εξαγωγή» από τη διεπαφή. Εν συνεχεία δηλώνουμε τις μεθόδους για τα στοιχεία που προσθέσαμε στο αρχείο "ChooseViewController.h".

Τέλος πρέπει να φορτώσουμε τη βάση δεδομένων. Γεμίζουμε τον πίνακα με τις κατηγορίες και δημιουργούμε ένα περιστασιακό πίνακα, που φορτώνει τις εγγραφές και τις παρουσιάζει με αλφαβητική σειρά. Στη συνέχεια ακολουθεί ο κώδικας που εκτελεί αυτή την μέθοδο:

## *4.7 Δημιουργία του μενού υποδοχής*

```
-(NSMutableArray *) categoriesList{
    categories = [[NSMutableArray alloc] initWithCapacity:10];
    @try {
        NSFileManager *fileMgr = [NSFileManager defaultManager];
        NSString *dbPath = [[[NSBundle mainBundle] resourcePath ]
                            stringByAppendingPathComponent:@"Categories.sqlite"];
        BOOL success = [fileMgr fileExistsAtPath:dbPath];
        if(!success)
        {
            NSLog(@"Cannot locate database file '%@'.", dbPath);
        }
        if(!(sqlite3_open([dbPath UTF8String], &db) == SQLITE_OK))
        {
            NSLog(@"An error has occured: %s", sqlite3_errmsg(db));
        }
        const char *sql = "SELECT * FROM Categories ORDER BY Names";
        sqlite3_stmt *sqlStatement;
        if(sqlite3_prepare(db, sql, -1, &sqlStatement, NULL) != SQLITE_OK)
        {
            NSLog(@"Problem with prepare statement:  %s", sqlite3_errmsg(db));
        }else{
            while (sqlite3_step(sqlStatement)==SQLITE_ROW) {
                Categories * choise = [[Categories alloc] init];
                choise.name = [NSString stringWithUTF8String:(char *) sqlite3_column_text(sqlStatement,1)];
                [categories addObject:choise];
            }
        }
        sqlite3_finalize(sqlStatement);
    }
    @catch (NSException *exception) {
        NSLog(@"Problem with prepare statement:  %s", sqlite3_errmsg(db));
    }
    @finally {
        sqlite3_close(db);
        return categories;
    }
}
```

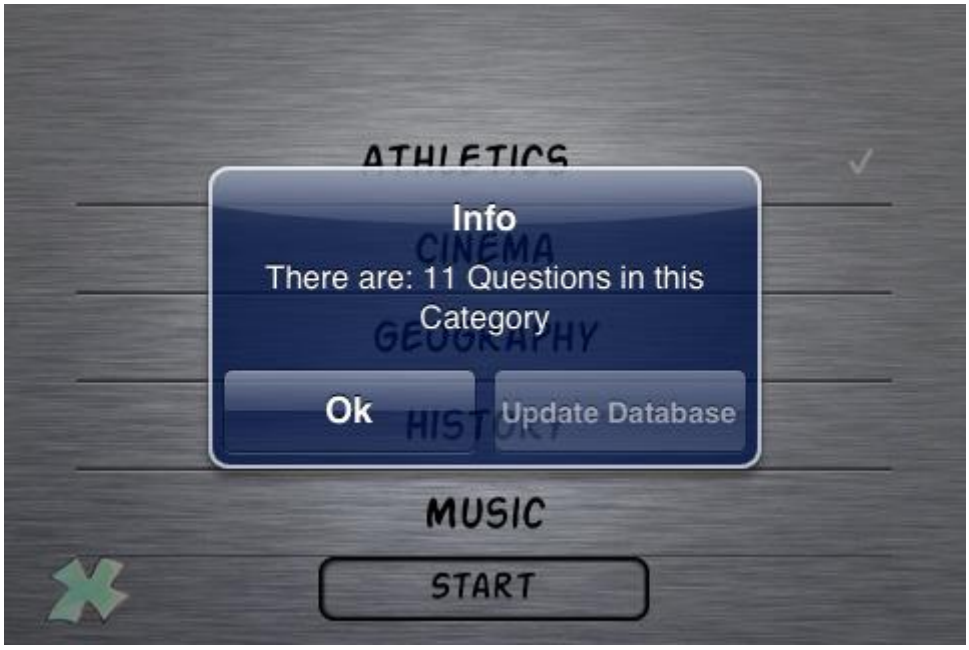
Υλοποιώντας των κώδικα και με βάση τη χρηστικότητα της εφαρμογής στη συγκεκριμένη διεπαφή προσθέσαμε και τη δυνατότητα ο χρήστης να γνωρίζει το σύνολο των ερωτήσεων που είναι εγγεγραμμένες στη βάση δεδομένων. Δημιουργούμε λοιπόν ένα “pop up” μήνυμα (Εικόνα 4.18) ειδοποίησης με τις απαραίτητες πληροφορίες για το χρήστη. Αυτό είναι ένα σημαντικό σημείο για τη διεπαφή αυτή, γιατί ξεκινώντας ο χρήστης το παιχνίδι πρέπει να δηλώσει το χρόνο του παιχνιδιού και τον αριθμό των ερωτήσεων. Για να εξασφαλίσουμε τη δυνατότητα να μην μπορούν να δηλωθούν περισσότερες από τις καταχωρημένες ερωτήσεις στην κατηγορία που επιλέγει ο χρήστης, δημιουργούμε μια μεταβλητή τύπου “NSInteger” με όνομα “numCount” την οποία θα περάσουμε στον επόμενο controller που θα εκτελεί το παιχνίδι.

```
if (indexPath.row == 0) {
    NSFileManager *fileMgr = [NSFileManager defaultManager];
    NSString *dbPath = [[[NSBundle mainBundle] resourcePath ]
                        stringByAppendingPathComponent:@"Categories.sqlite"];
    BOOL success = [fileMgr fileExistsAtPath:dbPath];
    if(!success)
    {
        NSLog(@"Cannot locate database file '%@'.", dbPath);
    }
    if(!(sqlite3_open([dbPath UTF8String], &db) == SQLITE_OK))
    {
        NSLog(@"An error has occured: %s", sqlite3_errmsg(db));

    }
    const char *sql = "SELECT COUNT (*) FROM Questions WHERE Category IS 2";
    sqlite3_stmt *sqlStatement;
    if(sqlite3_prepare(db, sql, -1, &sqlStatement, NULL) != SQLITE_OK)
    {
        NSLog(@"Problem with prepare statement:  %s", sqlite3_errmsg(db));
    }else{

        while (sqlite3_step(sqlStatement)==SQLITE_ROW) {
            self.count = sqlite3_column_int(sqlStatement, 0);
        }
    }
    sqlite3_finalize(sqlStatement);
    sqlite3_close(db);
    self.numCount = count;

    [self questionCountAlertView];
}
```

Εικόνα 4.17: Μέτρηση των ερωτήσεων μιας κατηγορίας

Στην παραπάνω εικόνα βλέπουμε τον κώδικα που εκτελεί την καταμέτρηση των ερωτήσεων μιας κατηγορίας. Ο κώδικας υλοποιείται στη μέθοδο tableView:didSelectRowAtIndexPath. Αυτή η μέθοδος καλείται κάθε φορά που διαλέγουμε ένα αντικείμενο στην οθόνη και επιστρέφει την τιμή του αντικειμένου, ώστε να χρησιμοποιηθεί για περαιτέρω επεξεργασία. Για τη μεταφορά της τιμής που χρειαζόμαστε για τον επόμενο controller στο τέλος της μεθόδου, έχουμε τη μεταβλητή "numCount" την οποία εξισώνουμε κάθε φορά με τη μεταβλητή τύπου "Int" με το όνομα "count". Τέλος περνάμε τη μεταβλητή μέσω της συνθήκης –(IBAction)startAction:(id)sender που αφορά το πλήκτρο έναρξης του παιχνιδιού με τον παρακάτω κώδικα:

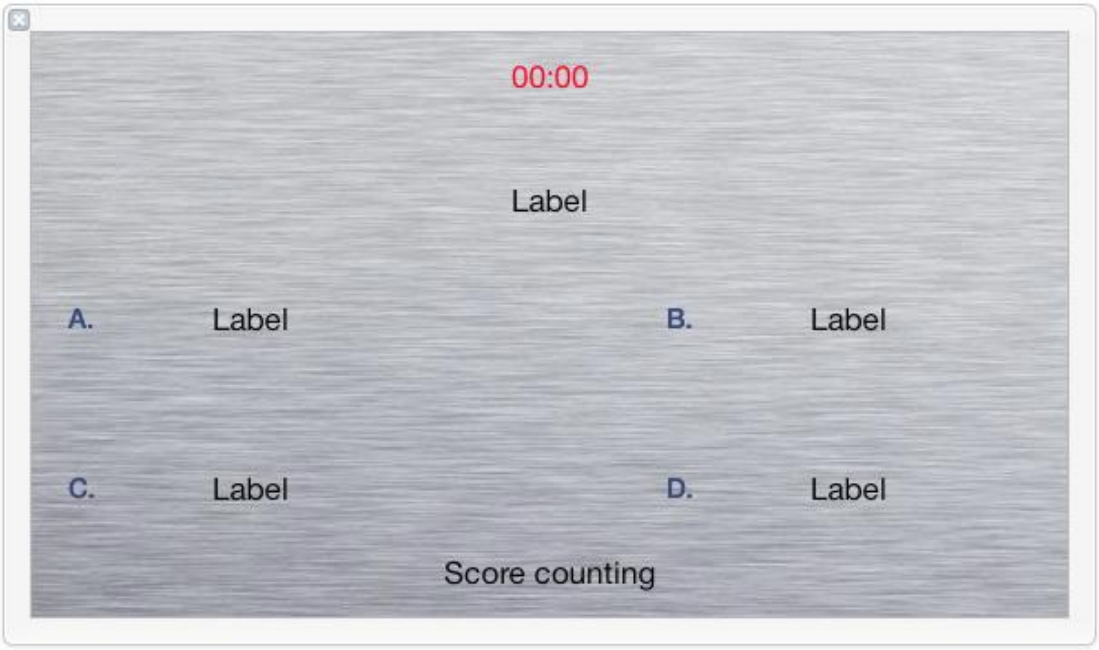
```
AthleticsViewController * controller = [[AthleticsViewController alloc]
    initWithNibName:@"AthleticsViewController" bundle:nil];
controller.numCount2 = numCount;
[self presentViewController:controller animated:YES completion:nil];
```




Εικόνα 4.18: Μήνυμα ειδοποίησης του επιλογέα κατηγορίας

## 4.8 Δημιουργία της διεπαφής παιχνιδιού

Δημιουργούμε ένα νέο "UIViewController" με το όνομα "AthleticsViewController" και ξεκινάμε με το αρχείο "AthleticsViewController.xib". Εισάγουμε στοιχεία από τον interface builder τύπου "Label", τα οποία θα χρησιμοποιήσουμε για να προβάλουμε την ερώτηση με τις αντίστοιχες απαντήσεις όπως και το χρόνο του παιχνιδιού. Επίσης προσθέτουμε και τέσσερα κουμπιά (Button), τα οποία θα επικαλύψουν τις τέσσερις ετικέτες των απαντήσεων και θα χρησιμοποιηθούν για την μέθοδο που θα ελέγχει την σωστή απάντηση. Τέλος, χρησιμοποιούμε και μια ετικέτα που θα προβάλει τους πόντους στο χρήστη για κάθε σωστή ερώτηση. Για την έναρξη του παιχνιδιού θέλουμε όπως έχουμε αναφέρει, ο χρήστης να έχει τη δυνατότητα να εισάγει το συνολικό χρόνο που θέλει να παίξει και τον αριθμό των ερωτήσεων. Για την αποφυγή δημιουργίας ενός νέου controller που θα εκτελεί αυτή τη μέθοδο, προτιμήθηκε η δημιουργία ενός μηνύματος ειδοποίησης στον ίδιο controller , αυτή τη φόρα όμως με τη δυνατότητα εισαγωγής δεδομένων.




Εικόνα 4.19: Η όψη του παιχνιδιού από τον interface builder

## 4.8.1 Αναλύοντας τον κώδικα του παιχνιδιού

Ξεκινώντας προβάλλεται στο χρήστη πάντα ο αντίστοιχος controller της κατηγορίας που επιλέγει. Το πρώτο πράγμα που πρέπει να κάνουμε είναι να "καθαρίσουμε" την οθόνη και να προβάλουμε το μήνυμα ειδοποίησης. Για να το πετύχουμε αυτό σε κάθε controller που δημιουργούμε, υπάρχει η μέθοδος (void) WillAppear: (Bool) animated, στην οποία δηλώνουμε τις μεθόδους που θα ενεργοποιούνται κατά την έναρξη του controller. Εδώ δημιουργήσαμε μια μέθοδο για να πετύχουμε τον καθαρισμό της οθόνης και να απενεργοποιήσουμε τη λειτουργία των κουμπιών, στην περίπτωση που ο χρήστης ακουμπήσει την οθόνη. Η ίδια μέθοδος καλείται επίσης κάθε φορά που ο χρήστης επιλέγει μια απάντηση.

```
- (void)prepareForIntroAnimation
{
    self.question.hidden = YES;
    self.scoreLabel.hidden = YES;
    self.answer1.alpha = 0.0f;
    self.answer2.alpha = 0.0f;
    self.answer3.alpha = 0.0f;
    self.answer4.alpha = 0.0f;
    self.answer1btn.alpha = 0.0f;
    self.answer2btn.alpha = 0.0f;
    self.answer3btn.alpha = 0.0f;
    self.answer3btn.alpha = 0.0f;
    self.answer4btn.alpha = 0.0f;
    answer1btn.enabled = NO;
    answer2btn.enabled = NO;
    answer3btn.enabled = NO;
    answer4btn.enabled = NO;

    _buttonsEnabled = NO;
}
```

Επάνω: Καθαρισμός οθόνης

Η μέθοδος που είναι υπεύθυνη να καλέσει αρχικά άλλες μεθόδους είναι η (void)viewDidLoad. Η παρακάτω εικόνα παρουσιάζει τη μέθοδο

για την εμφάνιση του μηνύματος ειδοποίησης, καθώς και τις μεθόδους για τη φόρτωση της βάσης και τον μηδενισμό αυτής.

```
-(void)showAlertWithTextField{
    UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Set Timer And Questions"
        message:nil delegate:self cancelButtonTitle:@"Cancel" otherButtonTitles:@"Ok", nil];
    [alertView setTag:101];
    alertView.alertViewStyle=UIAlertViewStyleLoginAndPasswordInput;
    [[alertView textFieldAtIndex:0] setKeyboardType:UIKeyboardTypeNumberPad];
    [[alertView textFieldAtIndex:0] becomeFirstResponder];
    [[alertView textFieldAtIndex:0] setPlaceholder:@"Time in seconds"];
    [[alertView textFieldAtIndex:1] setKeyboardType:UIKeyboardTypeNumberPad];
    [[alertView textFieldAtIndex:1] setSecureTextEntry:NO];
    [[alertView textFieldAtIndex:1] setPlaceholder:@"Number of questions"];

    [alertView show];

}
```

Εικόνα 4.20: Μέθοδος μηνύματος ειδοποίησης

Έχουμε καθαρίσει την οθόνη και για την έναρξη του παιχνιδιού πρέπει να δηλώσουμε το χρόνο και τις ερωτήσεις. Για να ολοκληρωθεί το μήνυμα ειδοποίησης δημιουργούμε δύο μεταβλητές τύπου "Int" που χρειαζόμαστε. Ο χρήστης αφού έχει θέσει τα στοιχεία για το χρόνο και τις ερωτήσεις, πατώντας το πλήκτρο "OK" του μηνύματος ειδοποίησης, ενεργοποιούνται η μέθοδος εμφάνισης του χρόνου και έναρξης του χρονομέτρου, καθώς και των ερωτήσεων. Ταυτόχρονα κάνουμε (get) παίρνουμε τα στοιχεία και τα τοποθετούμε στις αντίστοιχες ετικέτες (Label). Βέβαια, για να ενεργοποιηθεί το πλήκτρο αυτό θα πρέπει να πληρούνται κάποιες προϋποθέσεις.

Για το χρόνο ο χρήστης θα πρέπει να εισάγει πάνω από δύο αριθμητικά ψηφία, ενώ ο αριθμός των ερωτήσεων δεν πρέπει να ξεπερνά τις εγγεγραμμένες ερωτήσεις. Ο παρακάτω κώδικας δείχνει την υλοποίηση αυτών των προϋποθέσεων με τη χρήση δύο μεταβλητών τύπου "NSString" καθώς και τις μεταβλητής "NSInteger" "numCount2" που εισάγουμε από τον προηγούμενο controller.

```
- (BOOL)alertViewShouldEnableFirstOtherButton:(UIAlertView *)alertView
{
    if (alertView.tag == 101){
        NSString * inputText = [[alertView textFieldAtIndex:0] text];
        NSString * questions = [[alertView textFieldAtIndex:1] text];
        int test = [questions integerValue];
        if([inputText length] >= 2 && [questions length] >=1
           && test !=0 && [questions integerValue] <= numCount2){
            return YES;
        }

        else
        {
            return NO;
        }
    }else{
        return YES;
    }
}
```

Για να γράψουμε πιο ξεκάθαρα κώδικα και να πετύχουμε ένα πιο εύχρηστο αποτέλεσμα με αναγνωσιμότητα, επιλέγουμε να χρησιμοποιήσουμε ετικέτες (tags) όταν έχουμε πολλά κοινά στοιχεία σε ένα controller.

Όταν παρουσιαστεί μία ερώτηση στο χρήστη πρέπει να επιλέξει μια πιθανή απάντηση. Δεν θα ήταν σωστό να εξάγουμε τις απαντήσεις με τη σειρά που είναι γραμμένες στη βάση δεδομένων, γιαυτό δημιουργούμε μια συνθήκη που ανακατεύει τις απαντήσεις τυχαία "arc4random" και τις αποθηκεύει προσωρινά σε ένα πίνακα τύπου "NSMutableArray".

```
NSMutableArray  *arary = [[NSMutableArray alloc]init];
while (arary.count < 4) {
    int value = arc4random()%4+2;
    BOOL isFound = [[arary filteredArrayUsingPredicate:[NSPredicate predicateWithFormat:
        [NSString stringWithFormat:@"intValue == %d",value]]] count];
    if(!isFound)
        [arary addObject:[NSNumber numberWithInteger:value]];
}
```

Όταν ο πίνακας γεμίσει με τα τέσσερα ανακατεμένα νούμερα των απαντήσεων, προβάλουμε τις απαντήσεις ανακατεμένες στο χρήστη θέτοντας το κείμενο της κάθε ετικέτας με τον ακόλουθο κώδικα:

```
answer1.text = [NSString stringWithUTF8String:(char *)
    sqlite3_column_text(sqlStatement, [[arary objectAtIndex:0] intValue])];
answer2.text = [NSString stringWithUTF8String:(char *)
    sqlite3_column_text(sqlStatement, [[arary objectAtIndex:1] intValue])];
answer3.text = [NSString stringWithUTF8String:(char *)
    sqlite3_column_text(sqlStatement, [[arary objectAtIndex:2] intValue])];
answer4.text = [NSString stringWithUTF8String:(char *)
    sqlite3_column_text(sqlStatement, [[arary objectAtIndex:3] intValue])];
```

Για την επίτευξη της σωστής λειτουργίας της μεθόδου απάντησης, θέτουμε tags σε κάθε ετικέτα που δέχεται τη σωστή απάντηση με αριθμούς «999» και «0» για τη λάθος.

Επίσης εξισώνουμε τις ετικέτες στα αντίστοιχα κουμπιά για να υλοποιήσουμε τη μέθοδο σωστής απάντησης. Δημιουργούμε μία μέθοδο τύπου –(IBAction) την οποία συνδέουμε με όλα τα κουμπιά στον

interface builder. Ανάλογα με την επιλογή του χρήστη, αν η απάντηση είναι σωστή εκτελούνται με τη σειρά η μέθοδος που αυξάνει τους πόντους για κάθε σωστή απάντηση και η μέθοδος της επόμενης ερώτησης, ενώ αν είναι λάθος εκτελείται η μέθοδος της επόμενης ερώτησης (Εικόνα 4.22).

```
- (IBAction)answerButtonclicked:(id)sender {

    if([sender tag] == 999)
    {
        [self Score];
        [self gameGetQuestion];
        NSLog(@"Correct");
    }else{
        [self gameGetQuestion];
        NSLog(@"Incorrect");
```

Εικόνα 4.21: Μέθοδος επιλογής απάντησης

Τέλος για να κεντρίσουμε το ενδιαφέρον του χρήστη, τη στιγμή που δίνει κάποια απάντηση, ταυτόχρονα «πυροδοτείται» ένα κρυφό δεύτερο χρονόμετρο διάρκειας δέκα δευτερολέπτων με αντίστροφη μέτρηση. Με το χρονόμετρο αυτό δημιουργήσαμε μια μέθοδο που αφορά τους συνολικούς πόντους του χρήστη τύπου –(void). Όταν η μέθοδος αυτή εκτελείται ενημερώνει την ετικέτα που δείχνει τους πόντους για κάθε σωστή απάντηση. Έχουμε δηλώσει το μέγιστο αριθμό πόντων τους εκατό. Ο χρόνος των δέκα δευτερολέπτων ξεκινά να μετρά τη στιγμή που εμφανίζονται πλήρως οι απαντήσεις. Αυτό σημαίνει ότι όσο πιο γρήγορα απαντήσει ο χρήστης τόσους πιο πολλούς πόντους παίρνει.

```
-(void)Score {
    int maxScore = 100;
    int userScore = maxScore - (secondsCount *10);
    runningSore = runningSore + userScore;
    NSLog(@"%d", userScore);
    NSLog(@"%d", runningSore);
    scoreLabel.text= [NSString stringWithFormat:
                      @"Your score is: %2d", runningSore];

}
```

Εικόνα 4.22: Μέθοδος συνολικών πόντων

## *4.8 Δημιουργία της διεπαφής παιχνιδιού*


Εικόνα 4.23: Η όψη παιχνιδιού φορτώνοντας τις απαντήσεις

# ΚΕΦΑΛΑΙΟ 5ο

## Κατασκευή δικτυακού μέρους

Στο προηγούμενο κεφάλαιο έγινε η ανάλυση του σχεδιασμού της εφαρμογής. Επίσης έγινε εκτενής παρουσίαση της δημιουργίας μιας εφαρμογής από το χαρτί στην πράξη και αναλύσαμε τα σημαντικότερα σημεία του κώδικα που αφορούν τη λειτουργία της εφαρμογής. Σε αυτό το κεφάλαιο γίνετε παρουσίαση του δικτυακού μέρους της εφαρμογής, αναλύοντας τις καταστάσεις (statements) με τα ποιό σημαντικά σημεία κώδικα.

## 5.1 Συνδεσιμότητα Bluetooth και Wi-Fi

Για το δικτυακό μέρος της εφαρμογής χρησιμοποιήθηκαν οι τεχνολογίες του Bluetooth και του Wi-Fi. Σε προηγούμενο κεφάλαιο αναλύσαμε αυτά τα δύο πρωτόκολλα επικοινωνίας αναλυτικά. Λόγω της διαφορετικότητας τους κρίθηκε απαραίτητη η χρήση και των δύο στην εφαρμογή.

Η χρήση του Bluetooth στις συσκευές της Apple είναι λίγο ιδιαίτερη. Αυτό σημαίνει ότι κατασκευαστικά το λειτουργικό σύστημα iOS έχει ένα interface για την αναζήτηση συσκευών μέσω bluetooth το οποίο όμως είναι περιορισμένο. Δηλαδή επιτρέπει τη σύνδεση συσκευών ακοής για τη χρήση τηλεφωνικών κλήσεων, ενώ δεν επιτρέπει το ζευγάρωμα (pair) δύο κινητών τηλεφώνων για την ανταλλαγή δεδομένων. Για το λόγο αυτό η ανταλλαγή δεδομένων πάντα γίνεται μέσω μιας εφαρμογής η οποία χρησιμοποιεί το GameKit.framework.

Στο παιχνίδι ερωτοαπαντήσεων, το bluetooth είναι ιδανικό για χρήση όταν οι παίκτες βρίσκονται σε απόσταση μικρότερη των δέκα μέτρων. Όταν χρησιμοποιούμε το bluetooth, οι συσκευές δεν χρειάζεται να συνδεθούν όπως με ένα ποντίκι ή ένα πληκτρολόγιο. Το GameKit είναι αυτό που επιτρέπει τους πελάτες να ανιχνεύσουν ένα διακομιστή μέσω της εφαρμογής. Έπειτα, μόλις η σύνδεση επιτευχθεί οι συσκευές μπορούν να στείλουν μηνύματα επικοινωνίας μέσω του τοπικού δικτύου. Στην

εφαρμογή μας ο μέγιστος αριθμών παικτών έχει οριστεί στους τέσσερις μαζί με το διακομιστή.

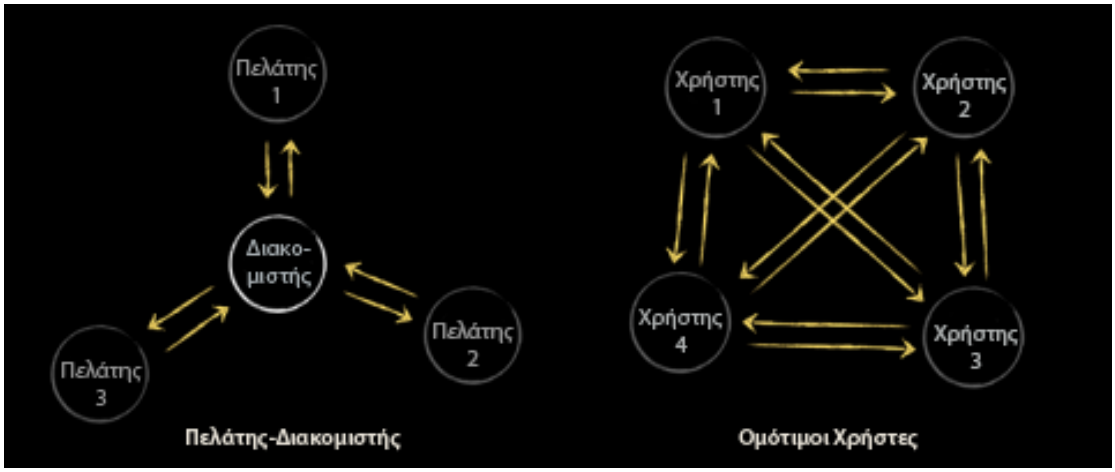
Με τη χρήση της τεχνολογίας Wi-Fi, οι χρήστες έχουν την δυνατότητα να παίξουν το παιχνίδι όταν είναι συνδεδεμένοι στο ίδιο τοπικό ασύρματο δίκτυο. Υπάρχει όμως και η δυνατότητα σύνδεσης με την τεχνολογία tethering, που επιτρέπει τη χρήση ενός κινητού τηλεφώνου ως ενδιάμεσου (access point) για την παροχή πρόσβασης στο διαδίκτυο σε μια άλλη συνδεδεμένη συσκευή, είτε μέσω καλωδίου είτε ασύρματα. Έτσι, όταν ένας χρήστης έχει πρόσβαση στο διαδίκτυο από το κινητό του τηλέφωνο (δηλαδή, συνδρομή στην ανάλογη υπηρεσία δεδομένων του δικτύου κινητής τηλεφωνίας), μπορεί να παίξει το παιχνίδι με τους υπόλοιπους παίκτες οι οποίοι θα συνδεθούν στο δικό του ασύρματο δίκτυο. Το tethering μπορεί να υποστηρίζεται εγγενώς από το κινητό τηλέφωνο (συνήθως smartphone), ενώ υποστήριξη απαιτείτε και από το δίκτυο κινητής τηλεφωνίας του συνδρομητή.

## 5.1.1 Επιλογή της αρχιτεκτονικής

Για τη δημιουργία δικτυακών παιχνιδιών απαιτείτε να γνωρίζουμε ότι επί της ουσίας έχουμε μία επιλογή μεταξύ δύο αρχιτεκτονικών (Εικόνα 5.1), οι οποίες είναι: πελάτη-διακομιστή (client- server) και ομότιμων χρηστών (peer-to-peer). Για την υλοποίηση του παιχνιδιού επιλέχθηκε η αρχιτεκτονική peer-to-peer, αλλά στην πραγματικότητα χρησιμοποιούμε το μοντέλο client-server. Ο χρήστης που φιλοξενεί το παιχνίδι είναι ο server ενώ όλοι οι άλλοι παίκτες είναι οι πελάτες.

Στο μοντέλο πελάτη-διακομιστή, ο διακομιστής είναι υπεύθυνος για τα πάντα και καθορίζει ποια είναι η «αλήθεια». Οι πελάτες στέλνουν συνέχεια μηνύματα ενημερώσεων στο διακομιστή και ο διακομιστής ενημερώνει όλους τους πελάτες αντίστοιχα. Οι πελάτες δεν έχουν τη δυνατότητα επικοινωνίας μεταξύ τους.

Στο μοντέλο peer-to-peer όλοι οι συμμετέχοντες είναι ίσοι και όλοι κάνουν την ίδια εργασία, αλλά πρέπει να φροντίσουμε ότι κάθε συμμετέχοντας βλέπει τα ίδια πράγματα με τους υπόλοιπους, μιας και δεν υπάρχει κεντρικός διαχειριστής.




Εικόνα 5.1: Οι αρχιτεκτονικές Client-Server και Peer-to-Peer

# 5.2 Δημιουργία της διεπαφής διακομιστή-πελάτη

Δημιουργούμε ένα νέο "UIViewController" με το όνομα "HostViewController" και ξεκινάμε με το αρχείο "HostViewController.xib". ". Εισάγουμε στοιχεία από τον interface builder τύπου "Label", τα οποία θα χρησιμοποιήσουμε για να προβάλουμε τα μηνύματα που ορίζουν τον controller. Ακόμη προσθέτουμε δύο κουμπιά (Button), ένα για την έναρξη και ένα για την έξοδο από τον controller. Τέλος, χρησιμοποιούμε ένα πλαίσιο για την προσθήκη του επιθυμητού ονόματος χρήστη και ένα πίνακα περιεχομένων ο οποίος θα προβάλει τα ονόματα των συνδεδεμένων πελατών στο παιχνίδι.

Αντίστοιχα για την επιλογή χρήσης της εφαρμογής ως πελάτης δημιουργούμε ένα ακριβώς ίδιο "UIViewController" που ονομάζουμε "JoinViewController". Έπειτα, προγραμματίζοντας την εφαρμογή μας διαχωρίζουμε τις λειτουργίες του κάθε controller χωριστά.


Εικόνα 5.2: Η όψη του διακομιστή

## 5.2.1 Έναρξη εκπομπής "υπηρεσίας" (broadcasting)

Για να διαχειριστούμε τις συνδέσεις μεταξύ συσκευών και να ξεκινήσουμε την εκπομπή της υπηρεσίας που θα "βλέπουν" οι πελάτες, δημιουργούμε μια νέα κλάση τύπου "NSObject".

Ο διακομιστής αρχικά κρατάει μια λίστα πελατών σε ένα προσωρινό πίνακα τύπου "Array" και ορίζει με μία μεταβλητή τύπου "Int" τον μέγιστο αριθμό πελατών που μπορούν να συνδεθούν ταυτόχρονα. Η εφαρμογή ερωτοαπαντήσεων έχει μέγιστο αριθμό παικτών τέσσερις, επομένως δηλώνουμε το μέγιστο αριθμό επιθυμητών συνδέσεων σε τρεις.

Για τη διαχείριση των συνδέσεων και το broadcasting είναι υπεύθυνο το "Gamekit.framework", το οποίο μας παρέχει αντικείμενα τύπου "GKSession". Για τη λειτουργία ως διακομιστή δημιουργούμε μια μέθοδο τύπου –(void)startAcceptingConnections η οποία ενημερώνει το αντικείμενο "GKSession" να λειτουργεί ως server. Αυτό σημαίνει ότι κάνει broadcasting το όνομα που έχει δώσει ο χρήστης, χωρίς όμως να ελέγχει αν εκπέμπετε η ίδια υπηρεσία όπως φαίνεται στον παρακάτω κώδικα.

```
- (void)startAcceptingConnectionsForSessionID:(NSString *)sessionID
{
    if (_serverState == ServerStateIdle)
    {
        _serverState = ServerStateAcceptingConnections;
    _connectedClients = [NSMutableArray arrayWithCapacity:self.maxClients];

    // Έναρξη του GKSession αντικειμένου και
        //εντολή διαθεσιμότητας λειτουργίας ως Server.
    _session = [[GKSession alloc] initWithSessionID:sessionID
                        displayName:nil sessionMode:GKSessionModeServer];
    _session.delegate = self;
    _session.available = YES;
    }
}
```

Για να κάνουμε χρήση του νέου αυτού αντικειμένου εισάγουμε στη μέθοδο -(void)viewDidAppear του "HostViewController" τον κώδικα:

```
if (_matchmakingServer == nil)
{
    _matchmakingServer = [[MatchmakingServer alloc] init];
    _matchmakingServer.maxClients = 3;
    [_matchmakingServer startAcceptingConnectionsForSessionID:SESSION_ID];
    _matchmakingServer.delegate = self;

    self.nameTextField.placeholder = _matchmakingServer.session.displayName;
    [self.tableView reloadData];
}
```

ο οποίος μόλις φορτωθεί ο controller του διακομιστή ενημερώνει την υπηρεσία και ξεκινά το broadcasting. Αν ο χρήστης δεν εισάγει κάποιο όνομα τότε αναγνωρίζεται με το όνομα της συσκευής του. Για να δουλέψουν όλα αυτά όμως, πρέπει να εισάγουμε στο κεντρικό αρχείο των ρυθμίσεων του project μας το αναγνωριστικό της υπηρεσίας. Ανοίγουμε το αρχείο "filename-Prefix.pch" και εισάγουμε το όνομα υπηρεσία με το κώδικα "#define SESSION_ID @΄΄Name"΄΄.

## 5.2.2 Εύρεση μιας υπηρεσίας

Για να διαχειριστούμε τον controller του πελάτη δημιουργούμε μια νέα κλάση για την εφαρμογή μας τύπου "NSObject". Η διαδικασία αντικατοπτρίζει κατά κάποιο τρόπο αυτό που κάναμε για το διακομιστή, αλλά αντί για συνδεμένους πελάτες δημιουργούμε μια λίστα διαθέσιμων διακομιστών.

Ο πελάτης αρχικά κρατάει μια λίστα διακομιστών σε ένα προσωρινό πίνακα τύπου "Array". Για τη λειτουργία αυτή δημιουργούμε μια μέθοδο τύπου –(void)startSearchingForServers η οποία ενημερώνει το αντικείμενο "GKSession" να λειτουργεί ως πελάτης. Αυτό σημαίνει ότι ψάχνει για διαθέσιμες υπηρεσίες όπως φαίνεται στον παρακάτω κώδικα:

```
- (void)startSearchingForServersWithSessionID:(NSString *)sessionID
{
    _availableServers = [NSMutableArray arrayWithCapacity:10];

    _session = [[GKSession alloc] initWithSessionID:sessionID
                displayName:nil sessionMode:GKSessionModeClient];
    _session.delegate = self;
    _session.available = YES;
}
```

Για να κάνουμε χρήση του νέου αυτού αντικειμένου εισάγουμε στη μέθοδο -(void)viewDidAppear του "JoinViewController" τον κώδικα:

```
if (_matchmakingClient == nil)
{
    _matchmakingClient = [[MatchmakingClient alloc] init];
    [_matchmakingClient startSearchingForServersWithSessionID:SESSION_ID];

    self.nameTextField.placeholder = _matchmakingClient.session.displayName;
    [self.tableView reloadData];
}
```

Τέλος, για να δείξουμε στον πελάτη πότε ένας server είναι διαθέσιμος δημιουργούμε μεταβλητές τύπου "NSString", οι οποίες αποθηκεύουν πληροφορίες για την ταυτότητα της συσκευής του παίκτη, καθώς και του ονόματος που θα προβάλλεται στον πίνακα. Κάθε στιγμή που γίνεται διαθέσιμη κάποια υπηρεσία, ανανεώνουμε τον πίνακα (tableview) με τη μέθοδο cellForRowAtIndexPath με τον κώδικα:

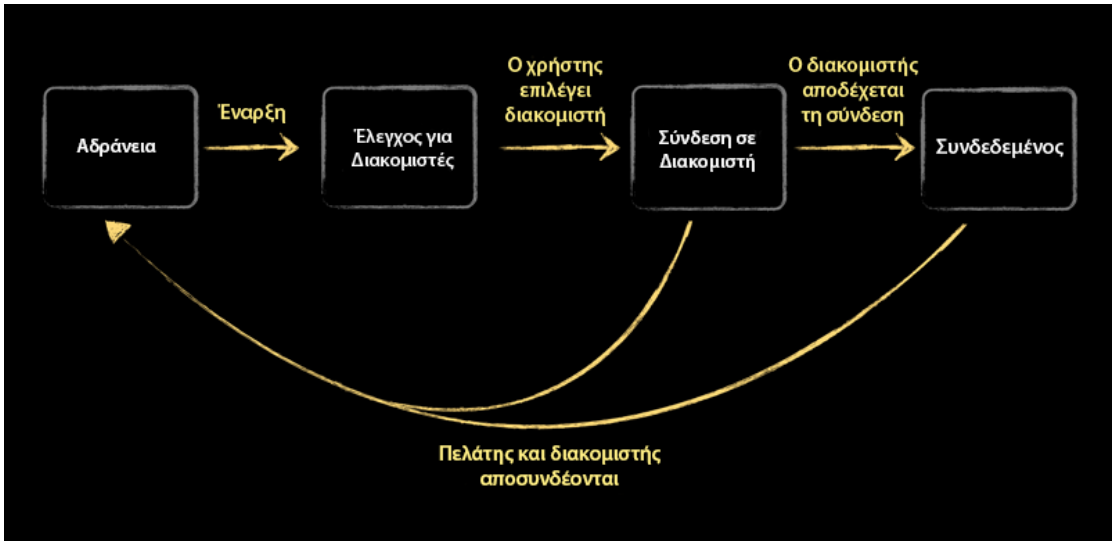
```
static NSString *CellIdentifier = @"CellIdentifier";

UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
if (cell == nil)
    cell = [[PeerCell alloc] initWithStyle:UITableViewCellStyleDefault
                           reuseIdentifier:CellIdentifier];
NSString *peerID = [_matchmakingClient peerIDForAvailableServerAtIndex:indexPath.row];
cell.textLabel.text = [_matchmakingClient displayNameForPeerID:peerID];

return cell;
```

## 5.2.3 Μηχανή καταστάσεων πελάτη

Έως τώρα για την έναρξη επικοινωνίας μεταξύ συσκευών ο διακομιστής δεν γνωρίζει κάτι. Ακόμη και ο πελάτης ενημερώνεται μόνο κάθε φορά που ο διακομιστής γίνεται διαθέσιμος ή μη.

Ξεκινώντας από τον πελάτη, αρχικά δημιουργούμε ένα διάγραμμα καταστάσεων (Εικόνα 5.3) που μας περιγράφει τις πιθανές καταστάσεις των αντικειμένων της κλάσης. Αυτό έχει ως αποτέλεσμα να είναι πιο ξεκάθαρη η δουλειά κάθε αντικειμένου σε διαφορετική περίπτωση.

## *5.2 Δημιουργία της διεπαφής διακομιστή-πελάτη*




Εικόνα 5.3: Διάγραμμα καταστάσεων πελάτη

Συνολικά δημιουργούμε τέσσερις καταστάσεις. Ξεκινώντας ο χρήστης τον "JoinViewController", καλείται η κατάσταση "Αδράνεια" κατά την οποία περιμένει χωρίς να κάνει τίποτα. Έπειτα αυτόματα μπαίνει στην κατάσταση "Έλεγχος για διακομιστές". Όταν ο χρήστης αποφασίσει να συνδεθεί σε ένα συγκεκριμένο διακομιστή, μπαίνει στην κατάσταση "Σύνδεση στο διακομιστή", όπου προσπαθεί να συνδεθεί στο διακομιστή και τέλος μπαίνει στην κατάσταση "Συνδεμένος" μόλις εδραιωθεί η σύνδεση. Αν οποιαδήποτε στιγμή κατά τις δύο τελευταίες καταστάσεις ο διακομιστής απορρίψει τη σύνδεση, ο πελάτης επιστρέφει στην αρχική κατάσταση.

Η κλάση που διαχειρίζεται τους πελάτες συμπεριφέρεται διαφορετικά ανάλογα με την κατάσταση που εκτελεί. Όταν βρίσκεται στην κατάσταση "Έλεγχος για διακομιστές", θα προσθέτει και θα αφαιρεί ονόματα διακομιστών στο πίνακα, ενώ όταν βρίσκεται στην κατάσταση "Σύνδεση στο διακομιστή" ή "Συνδεμένος" όχι.

Ξεκαθαρίζοντας λοιπόν τις καταστάσεις του πελάτη, συνεχίζουμε δημιουργώντας στην κλάση τύπου "NSObject" τη μηχανή καταστάσεων η οποία ορίζεται αριθμητικά. Τέλος για να ξεκινήσουμε τις συνδέσεις από τον πελάτη το "Gamekit.framework" μας παρέχει τη μέθοδο peerdid ChangeState τύπου "void" η οποία διαχειρίζεται τις καταστάσεις.

```
typedef enum
{
    //Μηχανή καταστάσεων του πελάτη.
    ClientStateIdle,
    ClientStateSearchingForServers,
    ClientStateConnecting,
    ClientStateConnected,
}
ClientState;
```

Όταν ο controller του πελάτη έρθει στην κατάσταση "Συνδεδεμένος" καλούμε μια δευτερεύουσα όψη (Εικόνα 5.4), για να επικαλύψουμε την κεντρική οθόνη μέχρι να ξεκινήσει το παιχνίδι από την πλευρά του διακομιστή. Ακολουθεί ένα παράδειγμα ορισμού της κατάστασης "Συνδεδεμένος".

```
case GKPeerStateConnected:
    if (_clientState == ClientStateConnecting)
    {
        _clientState = ClientStateConnected;
        [self.delegate matchmakingClient:self didConnectToServer:peerID];
    }
    break;
```


Εικόνα 5.4: Όψη αναμονής πελάτη

## 5.2.4 Μηχανή καταστάσεων διακομιστή

Έχουμε ολοκληρώσει επιτυχώς τη μηχανή καταστάσεων του πελάτη και μπορούμε να κάνουμε μια σύνδεση. Όμως για να ολοκληρωθεί επιτυχώς μια σύνδεση πρέπει να δημιουργήσουμε και τη μηχανή καταστάσεων του διακομιστή.

## *5.2 Δημιουργία της διεπαφής διακομιστή-πελάτη*


Εικόνα 5.5: Μηχανή καταστάσεων διακομιστή

Η μηχανή καταστάσεων του διακομιστή σε σχέση με τον πελάτη έχει μόνο τρείς καταστάσεις. Ξεκινώντας ο χρήστης τον "HostViewController" καλείται η κατάσταση "Αδράνεια", κατά την οποία περιμένει χωρίς να κάνει τίποτα. Έπειτα αυτόματα μπαίνει στην κατάσταση "Αποδοχή συνδέσεων". Όταν ο χρήστης πατήσει το πλήκτρο "Start" για να ξεκινήσει το παιχνίδι μεταξύ χρηστών, μεταφέρεται στην κατάσταση "Απόρριψη νέων συνδέσεων". Αυτό σημαίνει ότι σταματά να δέχεται νέες συνδέσεις από νέους πελάτες.

Όπως και στην περίπτωση του πελάτη έτσι και στο διακομιστή, δημιουργούμε στην κλάση τύπου "NSObject" τη μηχανή κατα-στάσεων η οποία ορίζεται αριθμητικά. Τέλος, για να ξεκινήσουμε τις συνδέσεις στο διακομιστή χρησιμοποιούμε τη μέθοδο peerdidChangeState τύπου "void" η οποία διαχειρίζεται τις καταστάσεις.

```
typedef enum
{
    //Μηχανή καταστάσεων του διακομιστή.
    ServerStateIdle,
    ServerStateAcceptingConnections,
    ServerStateIgnoringNewConnections,
}
ServerState;
```

Για να εδραιωθεί η σύνδεση στο διακομιστή το "Gamekit.framework" μας παρέχει τη μέθοδο acceptConnection FromPeer, η οποία όταν καλείται ελέγχει αν η κατάσταση στο διακομιστή είναι "Αποδοχή συνδέσεων". Επίσης η μέθοδος καλείται κάθε φορά που έχουμε φτάσει το μέγιστο αριθμό συνδέσεων πελατών που έχουμε ορίσει στους τρεις όπως φαίνεται παρακάτω:

```
if (_serverState == ServerStateAcceptingConnections &&
    [self connectedClientCount] < self.maxClients)
    //Έλεγχος για την κατάσταση του διακομιστή σε έναρξη αποδοχής συνδέσεων.
    //Αν ναι τότε αποδοχή πελατών με βάση το όριο.
{
    NSError *error;
    if ([session acceptConnectionFromPeer:peerID error:&error])
        NSLog(@"MatchmakingServer: Connection accepted from peer %@", peerID);
    else
        NSLog(@"MatchmakingServer: Error accepting connection from peer %@, %@", peerID, error);
}
else  // Δεν δέχεται περισσότερες συνδέσεις.
{
    [session denyConnectionFromPeer:peerID];
}
```

Ολοκληρώνοντας τη μηχανή καταστάσεων του διακομιστή, μπορούμε να γνωρίζουμε πότε έχει συνδεθεί ένας πελάτης στο διακομιστή επιτυχώς. Έπειτα δύο συσκευές μπορούν να ανταλλάσουν επιτυχώς μηνύματα μέσω του αντικειμένου "GKSession".


Εικόνα 5.6: Διακομιστής με συνδεδεμένο ένα πελάτη

## 5.2.5 Διαχείριση λαθών και αποσυνδέσεων

Όταν γράφουμε κώδικα για δικτυακές εφαρμογές θα πρέπει να λάβουμε σοβαρά υπόψη κάτι πολύ σημαντικό: Η δικτύωση είναι εξαιρετικά απρόβλεπτη. Οποιαδήποτε στιγμή η σύνδεση μπορεί να διακοπεί και πρέπει να μπορούμε να τη διαχειριστούμε, τόσο στον πελάτη όσο και στο διαχειριστή.

Για να διαχειριστούμε σωστά τις αποσυνδέσεις και τα τυχών λάθη επικοινωνίας μεταξύ συσκευών, δημιουργούμε ένα αριθμητικό

αντικείμενο και το τοποθετούμε στο κεντρικό αρχείο ρυθμίσεων του project μας "filename-Prefix.pch". Αυτοί είναι οι τέσσερις λόγοι τους οποίους αναγνωρίζει η εφαρμογή μας και προειδοποιεί τους χρήστες.

```
typedef enum
{
    QuitReasonNoNetwork,
    QuitReasonConnectionDropped,
    QuitReasonUserQuit,
    QuitReasonServerQuit,
}
QuitReason;
```

Για τη σωστή λειτουργία της εφαρμογής, επιλέχθηκε όταν ο πελάτης βρίσκεται στην κατάσταση "Συνδεδεμένος" και ο διακομιστής για οποιονδήποτε λόγο αποσυνδέεται, να επιστρέφει στο μενού υποδοχής. Δημιουργούμε μια μέθοδο με το όνομα disconnectFromServer. Η μέθοδος καλείται για να τοποθετήσει τον πελάτη σε κατάσταση "Αδράνεια" και να καθαρίσει το αντικείμενο "GKSession" μετά από αποσύνδεση του διακομιστή.

```
- (void)disconnectFromServer //Αποσύνδεση απο το διακομιστή.
{
    NSAssert(_clientState != ClientStateIdle, @"Wrong state");

    _clientState = ClientStateIdle; //Θέτουμε την κατάσταση του πελάτη σε αναμονή-ανενεργό.

    [_session disconnectFromAllPeers];  //Αποσύνδεση από όλους τους peers.
    _session.available = NO;  //Καθαρισμός του GKSession
    _session.delegate = nil;  //Καταστροφή του αντικειμένου GKSession.
    _session = nil;
    _availableServers = nil;
    [self.delegate matchmakingClient:self didDisconnectFromServer:_serverPeerID];
    _serverPeerID = nil;
}
```

Επίσης σωστό είναι να ενημερώσουμε το χρήστη για ποιο λόγο μεταφέρθηκε στο κεντρικό μενού υποδοχής. Για παράδειγμα, αν ο χρήστης εγκαταλείψει το παιχνίδι πατώντας το κουμπί εξόδου επίτηδες τότε δε χρειάζεται κάποια ειδοποίηση. Στην περίπτωση όμως σφάλματος του δικτύου είναι καλό να υπάρχει ενημέρωση. Δημιουργούμε στον αρχικό "viewController" ένα μήνυμα ειδοποίησης (alertView), που θα εμφανίζεται στους πελάτες κάθε φορά που ο διακομιστής θα σταματά το παιχνίδι με τον κώδικα:

```
- (void)showDisconnectedAlert
{
    UIAlertView *alertView = [[UIAlertView alloc]
            initWithTitle:NSLocalizedString(@"Disconnected",
                                @"Client disconnected alert title")
            message:NSLocalizedString(@"You were disconnected from the game.",
                                      @"Client disconnected alert message")
            delegate:nil
            cancelButtonTitle:NSLocalizedString(@"OK", @"Button: OK")
            otherButtonTitles:nil];
    [alertView show];
}
```

Ως προεπιλεγμένη τιμή για το μήνυμα αποσύνδεσης ορίζουμε την κατάσταση "Η σύνδεση διακόπηκε". Εκτός την περίπτωση που ο χρήστης διακόψει πατώντας το κουμπί εξόδου, όλες οι άλλες αποσυνδέσεις από το διακομιστή θα ορίζονται ως πρόβλημα του δικτύου.

Συνεχίζοντας την κατασκευή μηνυμάτων για την απώλεια δικτύου, κρίθηκε απαραίτητο όταν ο χρήστης έχει κλειστό το Bluetooth ή το Wi-Fi της συσκευής να δέχεται ένα μήνυμα προειδοποίησης. Κατασκευάζουμε ακόμη ένα μήνυμα ειδοποίησης, το οποίο ενημερώνει το χρήστη ότι πρέπει να ανοίξει μία εκ των δύο τεχνολογιών (Εικόνα 5.7). Το μήνυμα εμφανίζεται όταν ο χρήστης επιλέξει τη δικτυακή επιλογή και καμία από τις δύο τεχνολογίες δεν είναι διαθέσιμες.

Για την περίπτωση επιλογών αποσύνδεσης από το διακομιστή η διαδικασία είναι ίδια με αυτή του πελάτη. Για το διακομιστή δημιουργούμε μια μέθοδο τύπου -(void) η οποία καλείται κάθε φορά που υπάρχει σφάλμα δικτύου.

```
- (void)matchmakingServerSessionDidEnd:(MatchmakingServer *)server
{
    _matchmakingServer.delegate = nil;
    _matchmakingServer = nil;
    [self.tableView reloadData];
    [self.delegate hostViewController:self didEndSessionWithReason:_quitReason];
}
```

Η μέθοδος αυτή αποσυνδέει το διακομιστή από όλους τους πελάτες και καθαρίζει το αντικείμενο “GKSession”. Επίσης ανανεώνει τον πίνακα με τα διαθέσιμα ονόματα πελατών.


Εικόνα 5.7: Μήνυμα απώλειας δικτύου

## 5.3 Υλοποιώντας το δικτυακό παιχνίδι

Το δικτυακό παιχνίδι ξεκινά όταν ο διακομιστής πατά το κουμπί “Start” στην οθόνη του διακομιστή «Host». Έπειτα ο διακομιστής αρχίζει

να στέλνει μηνύματα προς όλους τους διαθέσιμους πελάτες για να ετοιμαστούν. Τα μηνύματα αυτά είναι πακέτα δεδομένων που "πετάνε" μέσω του δικτύου Bluetooth ή Wi-Fi των συσκευών.

Τα πακέτα δικτύου που λαμβάνονται στην Objective C, διαχειρίζονται από ένα αντικείμενο που ονομάζεται "χειριστής λαμβανόντων πακέτων" (data receive handler). Ο διακομιστής πρέπει να στείλει ένα μεγάλο όγκο μηνυμάτων στους πελάτες πριν ξεκινήσει το παιχνίδι, αλλά για να μην ορίσουμε ως "data receive handler" τον controller του πελάτη, δημιουργούμε μια νέα κλάση με το όνομα "Game" τύπου "NSObject". Αυτή θα είναι η κλάση που διαχειρίζεται τα πακέτα του "GKSession", καθώς και τη λογική του παιχνιδιού.

Αρχικά για να ξεκινήσει το παιχνίδι αλλάζουμε την κατάσταση του πελάτη σε "Συνδεδεμένος" και ενημερώνουμε το διακομιστή με το "peerID" του πελάτη.

```
    // Ο πελάτης είναι συνδεμένος με τον διακομιστή.
case GKPeerStateConnected:
    if (_clientState == ClientStateConnecting)
    {
        _clientState = ClientStateConnected;
        [self.delegate matchmakingClient:self didConnectToServer:peerID];
    }
    break;
```

Επίσης δημιουργούμε ένα πακέτο δεδομένων, το οποίο αφορά το όνομα που δηλώνει ο πελάτης καθώς και το "peerID" και ενημερώνουμε τον κεντρικό controller ότι ο πελάτης είναι έτοιμος να ξεκινήσει το παιχνίδι.

```
- (void)matchmakingClient:(MatchmakingClient *)client
      didConnectToServer:(NSString *)peerID
{
    NSString *name = [self.nameTextField.text
        stringByTrimmingCharactersInSet:[NSCharacterSet
            whitespaceAndNewlineCharacterSet]];
    // Επιλογή του ονόματος που έγραψε ο πελάτης για την έναρξη του παιχνιδιού
    if ([name length] == 0)
        name = _matchmakingClient.session.displayName;

    [self.delegate joinViewController:self startGameWithSession:
     _matchmakingClient.session playerName:name server:peerID];
    // Ενημέρωση του αρχικού Controller ότι ο πελάτης είναι έτοιμος
    //για την έναρξη του παιχνιδιού.
}
```

Δημιουργούμε μια μέθοδο τύπου –(void) στον κεντρικό controller η οποία θα καλείται όταν είναι έτοιμοι για το παιχνίδι οι πελάτες και ο διακομιστής. Η μέθοδος απευθύνεται στην κλάση που διαχειρίζεται τα πακέτα δεδομένων και καλεί τον controller που φιλοξενεί το δικτυακό παιχνίδι.

```
- (void)startGameWithBlock:(void (^)(Game *))block
{
    GameViewController *gameViewController = [[GameViewController alloc]
                        initWithNibName:@"GameViewController" bundle:nil];
    gameViewController.delegate = self;
    [self presentViewController:gameViewController animated:NO completion:^
     {
         Game *game = [[Game alloc] init];
         gameViewController.game = game;
         game.delegate = gameViewController;
         block(game);
     }];
}
```

## 5.3.1 Ο διαχειριστής δεδομένων

Όπως έχουμε ήδη αναφέρει, η κλάση που διαχειρίζεται τα εισερχόμενα πακέτα δικτύου του αντικειμένου "GKSession" είναι τύπου "NSObject". Σε αυτή την κλάση επίσης πρέπει να προσθέσουμε όλα τα αντικείμενα του μοντέλου δεδομένων που αφορούν το παιχνίδι.

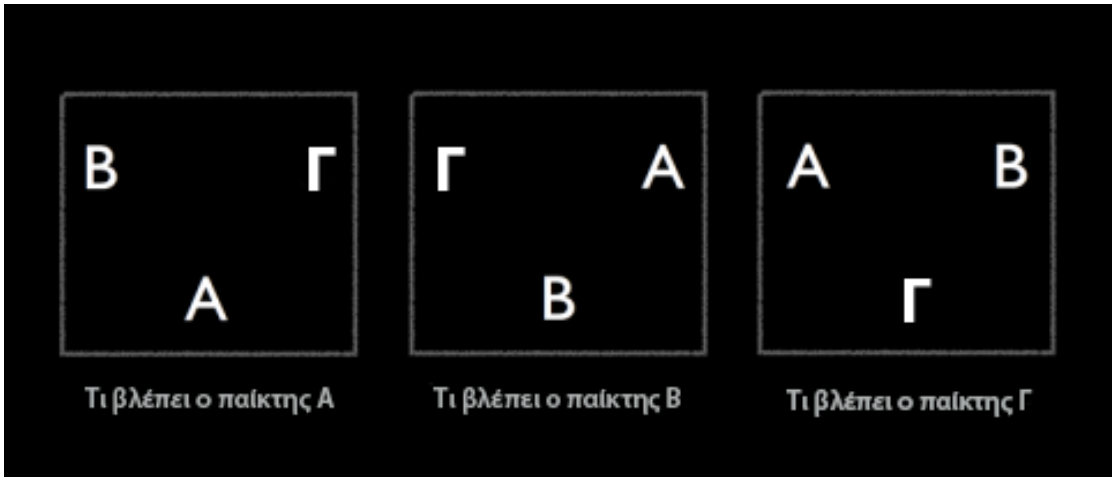
Αρχικά κατασκευάζουμε μια μηχανή καταστάσεων για το παιχνίδι. Η κάθε κατάσταση του παιχνιδιού έχει άμεση ανταπόκριση στο δικτυακό "View-Controller".

```
typedef enum
{
    GameStateWaitingForSignIn,
    GameStateWaitingForReady,
    GameStatePlaying,
    GameStateGameOver,
    GameStateQuitting,
}
GameState;
```

Για τους παίκτες που συμμετέχουν στο παιχνίδι, πρέπει να δημιουργήσουμε μια ξεχωριστή κλάση τύπου "NSObject". Την ονομάζουμε "Player" και δημιουργούμε τρείς ιδιότητες για κάθε παίκτη. Οι τρείς ιδιότητες αντιπροσωπεύουν τους διαφορετικούς τρόπους που μπορεί να αναγνωριστεί κάθε παίκτης:

1. Από το όνομα. Αυτό είναι που βλέπει κάθε χρήστης στον "GameViewController". Το όνομα είναι αυτό που πληκτρολογεί ο χρήστης στις οθόνες του διακομιστή και πελάτη. Αν ο χρήστης δεν γράψει κάποιο όνομα εμφανίζεται με το όνομα που έχει δώσει στη συσκευή του.
2. Από το "peerID". Το αναγνωριστικό αυτό είναι που χρησιμοποιεί το αντικείμενο "GKSession" εσωτερικά. Το χρειαζόμαστε για να αναγνωρίζουμε τους παίκτες όταν χρειάζεται να στέλνουμε μηνύματα μέσω του δικτύου.
3. Από τη θέση τους στην οθόνη. Κάθε παίκτης θα βλέπει τον εαυτό του στο κάτω μέρος της οθόνης (Εικόνα 5.8). Έτσι δημιουργούμε μια σειρά με τα ονόματα των παικτών στην οθόνη δεξιόστροφα.

## *5.3 Υλοποιώντας το δικτυακό παιχνίδι*




Εικόνα 5.8: Οι θέσεις των παικτών στην οθόνη

Όταν η κλάση "Game" που διαχειρίζεται το παιχνίδι έρθει στην κατάσταση "GameStateWaitingForSignIn", ο διακομιστής στέλνει ένα μήνυμα προς όλους τους πελάτες ζητώντας το όνομα τους. Δημιουργούμε ένα λεξικό "NSMutableDictionary" τεσσάρων θέσεων στο οποίο κρατάμε τα αντικείμενα των παικτών με κλειδί το "peerID". Στη μέθοδο startServerGameWithSession:playerName:Clients, δημιουργούμε το αντικείμενο του διαχειριστή αρχικά και το τοποθετούμε στο κάτω μέρος της οθόνης. Τέλος, κάνουμε ένα εσωτερικό έλεγχο στον πίνακα των πελατών και δημιουργούμε τα υπόλοιπα αντικείμενα των παικτών, τοποθετώντας τα δεξιόστροφα ανάλογα με το πόσοι παίκτες έχουν συνδεθεί με τον παρακάτω κώδικα:

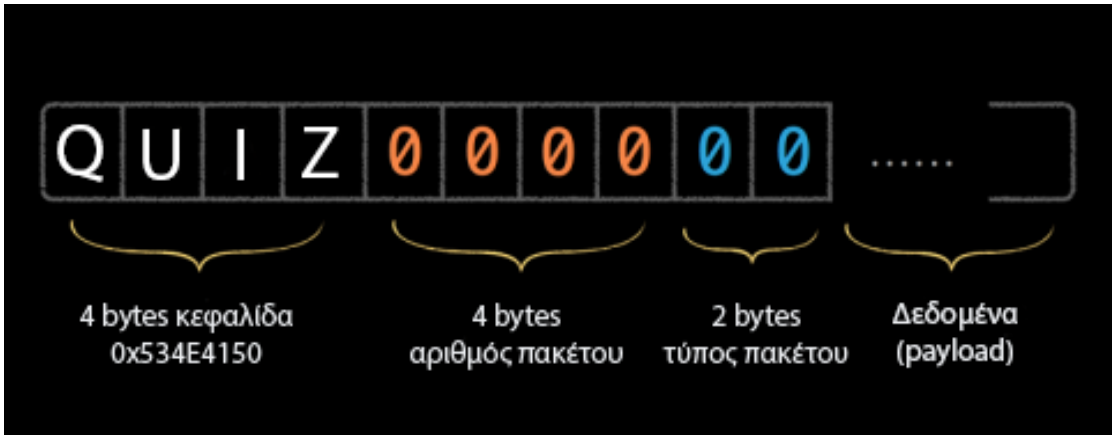
```
_state = GameStateWaitingForSignIn;
[self.delegate gameWaitingForClientsReady:self];
// Δημιουργία του αντικειμένου του παίκτη στο διακομιστή.
Player *player = [[Player alloc] init];
player.name = name;
player.peerID = _session.peerID;
player.position = PlayerPositionBottom;//Τοποθέτηση στο κάτω μέρος της οθόνης.
[_players setObject:player forKey:player.peerID];
// Πρόσθεση αντικειμένου παίκτη για κάθε πελάτη .
int index = 0;
for (NSString *peerID in clients)
{
    Player *player = [[Player alloc] init];
    player.peerID = peerID;
    [_players setObject:player forKey:player.peerID];
    if (index == 0)
        player.position = ([clients count] == 1) ?
        PlayerPositionTop : PlayerPositionLeft;
    else if (index == 1)
        player.position = PlayerPositionTop;
    else
        player.position = PlayerPositionRight;
    index++;
}
```

## 5.4 Αποστολή μηνυμάτων μεταξύ συσκευών

Έχουμε δημιουργήσει επιτυχώς το αντικείμενο για κάθε πελάτη και μπορούμε να στείλουμε το αίτημα σύνδεσης προς όλους τους πελάτες. Κάθε πελάτης πρέπει να απαντήσει ασύγχρονα με το όνομα του. Λαμβάνοντας το μήνυμα ο διακομιστής θέτει το όνομα για κάθε παίκτη.

Το αντικείμενο "GKSession" έχει μια μέθοδο που ονομάζεται sendDataToAllPeers:withDataMode:error. Τη μέθοδο αυτή χρησιμοποιεί ο διαχειριστής για να στείλει αντικείμενα τύπου "NSData" σε όλους τους συνδεδεμένους πελάτες. Τα αντικείμενα "NSData' ουσιαστικά είναι τα μηνύματα που ανταλλάσουν οι συσκευές μεταξύ τους.

Στην εφαρμογή μας όλα τα μηνύματα έχουν την ακόλουθη διάταξη:




Εικόνα 5.9: Διάταξη πακέτων μηνυμάτων

Κάθε πακέτο είναι τουλάχιστον δέκα byte. Οι διαφορετικοί τύποι πακέτων έχουν την ίδια κεφαλίδα αλλά διαφορετικό ωφέλιμο φορτίο (payload). Τα δέκα αυτά bytes αποτελούνται από:

- Τα τέσσερα πρώτα byte, από την κεφαλίδα με το όνομα QUIZ και επαληθεύουν ότι τα πακέτα είναι όντως δικά μας.
- Τα επόμενα τέσσερα χρησιμοποιούνται για να αναγνωρίζουν πότε ένα πακέτο φτάνει εκτός σειράς και αποτελούνται από ένα ακέραιο 32 bit.
- Τα τελευταία δύο byte αντιπροσωπεύουν το είδος του πακέτου. Επειδή έχουμε πολλά είδη μηνυμάτων που αποστέλλονται μεταξύ του διακομιστή και των πελατών, αυτός ο ακέραιος των 16 bit καθορίζει τι είδος πακέτο είναι.

## *5.4 Αποστολή μηνυμάτων μεταξύ συσκευών*

Τέλος για κάθε τύπο πακέτου, μπορούν να υπάρχουν περισσότερα δεδομένα. Το πακέτο που στέλνουν ο πελάτες στο διακομιστή με το όνομα, περιέχει για παράδειγμα, ωφέλιμο φορτίο τύπου "UTF-8 string" με το όνομα του παίκτη.

Για να κρατήσουμε τον κώδικα μας τακτοποιημένο δημιουργούμε μια κλάση τύπου "NSObject" την οποία ονομάζουμε "Packet". Η κλάση αυτή διαχειρίζεται τα παραπάνω bit-byte πίσω από το παιχνίδι μας. Αρχικά δημιουργούμε όλα τα διαφορετικά μηνύματα που θα αποστέλλονται και θα λαμβάνονται. Έπειτα δημιουργούμε τη μέθοδο δεδομένων "NSDdata". Αυτή διαθέτει ένα αντικείμενο τύπου "NSMutableData", στο οποίο τοποθετούνται οι δύο ακέραιοι των 32 bit και ένας 16 bit. Αυτά είναι τα δέκα byte που έχουμε αναφέρει.

```
typedef enum
{
    // Διαφορετικοί τύποι μηνυμάτων
    PacketTypeSignInRequest = 0x64,
    PacketTypeSignInResponse,

    PacketTypeServerReady,
    PacketTypeClientReady,

    PacketTypeQuestions,
    PacketTypeClientAnswer,
    PacketTypeAnswerResponse,

    PacketTypeOtherClientQuit,
    PacketTypeServerQuit,
    PacketTypeClientQuit,

    PacketTypeGameOver,
}
PacketType;
```

```
- (NSData *)data
{
    NSMutableData *data = [[NSMutableData alloc]
                           initWithCapacity:100];

    [data rw_appendInt32:'QUIZ'];
    [data rw_appendInt32:0];
    [data rw_appendInt16:self.packetType];

    [self addPayloadToData:data];
    return data;
}
```

Για να καθορίσουμε το μοντέλο δεδομένων των μηνυμάτων δημιουργούμε μια νέα κλάση τύπου "NSData". Η κλάση αυτή καθορίζει τα διαφορετικά μεγέθη των ακεραίων. Στο τέλος κάθε μεθόδου καλούμε τη συνάρτηση [self appendBytes:length:], προκειμένου να προσθέσουμε στη μνήμη την τιμή της κάθε μεταβλητής. Πρέπει να τονίσουμε ότι πριν από αυτή τη συνάρτηση καλούμε τη συνάρτηση htonl() για την τιμή. Αυτό το κάνουμε για να διασφαλίσουμε ότι τα byte του ακεραίου μεταδίδονται με τη σωστή "δικτυακή σειρά" που είναι η "big endian" [28]. Οι επεξεργαστές ARM χρησιμοποιούν τη διάταξη "bi endian". Αυτό σημαίνει ότι μπορούν να λειτουργήσουν είτε ως "big endian" είτε ως "little endian", οπότε για να μην έχουμε πρόβλημα όταν

```
- (void)rw_appendInt32:(int)value
{
    value = htonl(value);
    [self appendBytes:&value length:4];
}
```

προγραμματίζουμε δικτυακές εφαρμογές χρησιμοποιούμε τη συνάρτηση htonl() πριν την αποστολή των byte.


Εικόνα 5.10: Διάταξη των byte στη μνήμη

Εφόσον αναλύσαμε την κατασκευή των πακέτων μηνυμάτων θα δώσουμε ένα παράδειγμα κώδικα. Δημιουργούμε τη μέθοδο clientReceivedPacket στην κλάση "Game". Στη μέθοδο αυτή ελέγχουμε το είδος του πακέτου που παραλαμβάνει ο πελάτης και αποφασίζουμε πως θα το χειριστούμε.

```
switch (packet.packetType)
{
    case PacketTypeSignInRequest:
        if (_state == GameStateWaitingForSignIn)
        {
            _state = GameStateWaitingForReady;

            Packet *packet = [PacketSignInResponse
                    packetWithPlayerName:_localPlayerName];

            [self sendPacketToServer:packet];
        }
        break;
```

Όταν το πακέτο είναι "Sign in request" αλλάζουμε την κατάσταση του παιχνιδιού για τον πελάτη σε "Waiting for ready" και στέλνουμε στο διακομιστή ένα πακέτο "Sign in response". Το πακέτο αυτό είναι μεγαλύτερο των δέκα byte γιατί περιέχει ωφέλιμο φορτίο με το όνομα του χρήστη. Η μέθοδος sendPacketToServer είναι σχεδόν ίδια με τη μέθοδο sendPacketToAllClients, με τη διαφορά ότι στέλνουμε το πακέτο μόνο στο χρήστη με αναγνωριστικό _serverPeerID.

## 5.4.1 Διαχειρίζοντας τις απαντήσεις στο διακομιστή

Χρησιμοποιώντας το "GameKit" και κατά συνέπεια τα αντικείμενα "GKSession" για μια δικτυακή εφαρμογή, ο διαχειριστής με τους πελάτες μοιράζονται το μεγαλύτερο μέρος του κώδικα.

Για να διαχειριστούμε σωστά τα πακέτα δεδομένων δικτύου, δημιουργούμε μια μεταβλητή τύπου "Bool" και την ονομάζουμε "isServer". Έτσι λαμβάνοντας το "Sign in response" πακέτο από τους πελάτες καλούμε τη μέθοδο serverReceivedPacket:fromPlayer με τα ονόματα των παικτών.

```
if (self.isServer)
    [self serverReceivedPacket:packet fromPlayer:player];
else
    [self clientReceivedPacket:packet];
```

```
switch (packet.packetType)
{
    case PacketTypeSignInResponse:
        if (_state == GameStateWaitingForSignIn)
        {
            player.name = ((PacketSignInResponse *)packet).playerName;
```

Τέλος κάθε φορά που προσθέτουμε ένα νέο τύπο πακέτου στον κώδικα μας, προσθέτουμε και μια νέα περίπτωση-κατάστασης στην κλάση "Packet". Αυτό το κάνουμε με τη λογική κάθε φορά που διαβάζουμε ένα πακέτο στη μέθοδο packetWithData, να παρακάμπτουμε την υποκλάση και να διαβάζουμε ένα συγκεκριμένο πακέτο.

## 5.5 Μηχανές καταστάσεων παιχνιδιού

Έως τώρα έχουμε κατασκευάσει τις μηχανές καταστάσεων δικτύου για τον πελάτη και το διακομιστή. Αντίστοιχα η κλάση "Game" που διαχειρίζεται τη λογική του παιχνιδιού έχει τις δικές τις μηχανές καταστάσεων, οι οποίες είναι περισσότερο πολύπλοκες.

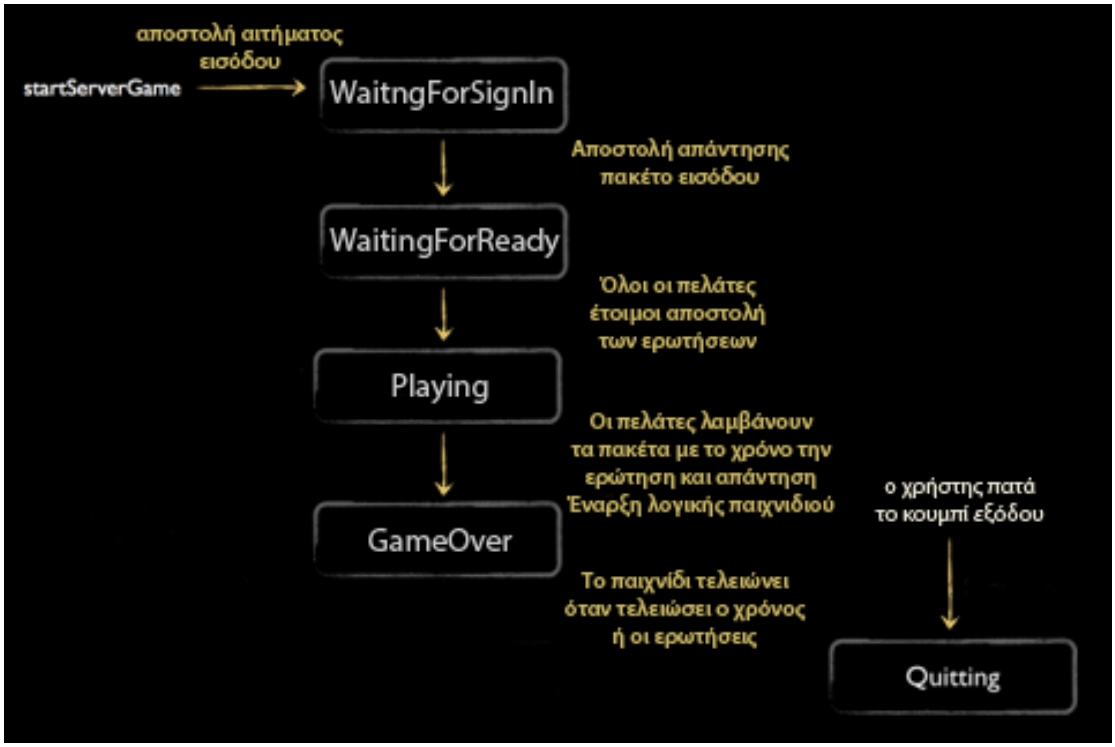
Για το διακομιστή όταν καλούμε τη μέθοδο startServerGame αποστέλλουμε το πακέτο "Sign in request" προς τους πελάτες. Έπειτα ακολουθούν οι καταστάσεις (Εικόνα 5.11):

- Waiting for Sign In. Αναμονή για σύνδεση των πελατών. Όλοι οι πελάτες στέλνουν το πακέτο "Sign in Response".
- Waiting for Ready. Όλοι οι πελάτες είναι έτοιμοι. Αποστολή της ερώτησης και των απαντήσεων.
- Playing. Όλοι οι πελάτες λαμβάνουν τα πακέτα με το χρόνο του παιχνιδιού, την ερώτηση και τις απαντήσεις. Εδώ ξεκινά η λογική του παιχνιδιού. Όποιος παίκτης προλάβει απαντά πρώτος και αν η απάντηση είναι σωστή παίρνει ένα πόντο αλλιώς δίνεται η δυνατότητα στους υπόλοιπους να απαντήσουν.

- Game Over. Όταν τελειώσει ο χρόνος ή οι ερωτήσεις που έχει θέσει ο διακομιστής τελειώνει το παιχνίδι.
- Quitting. Ο διακομιστής πατάει το πλήκτρο εξόδου.

Η μηχανή καταστάσεων για τον πελάτη είναι σχεδόν η ίδια με το διακομιστή. Καλούμε τη μέθοδο startClientGame και ακολουθούν οι καταστάσεις:

- Waiting for Sign In. Αποδοχή του πακέτου "Sign in Request" από το διακομιστή και αποστολή του "Sign in Response".
- Waiting for Ready. Ο διακομιστής και οι πελάτες είναι έτοιμοι.
- Playing. Όλοι οι πελάτες λαμβάνουν τα πακέτα με το χρόνο του παιχνιδιού, την ερώτηση και τις απαντήσεις. Εδώ ξεκινά η λογική του παιχνιδιού. Όποιος παίκτης προλάβει απαντά πρώτος και αν η απάντηση είναι σωστή παίρνει ένα πόντο αλλιώς δίνεται η δυνατότητα στους υπόλοιπους να απαντήσουν.
- Game Over. Όταν τελειώσει ο χρόνος ή οι ερωτήσεις που έχει θέσει ο διακομιστής τελειώνει το παιχνίδι.
- Quitting. Ο διακομιστής πατάει το πλήκτρο εξόδου.




Εικόνα 5.11:Μηχανή καταστάσεων διακομιστή

## 5.6 Η κατάσταση “Waiting for Ready”

Μέχρι τώρα είδαμε ότι ο διακομιστής στέλνει ένα μήνυμα σε κάθε πελάτη ζητώντας του να συνδεθεί και να στείλει το όνομα του. Αφού ο διακομιστής έχει λάβει την απάντηση από όλους τους συνδεδεμένους πελάτες το παιχνίδι μπορεί να ξεκινήσει. Ο διακομιστής γνωρίζει ποιοι είναι οι υπόλοιποι παίκτες, όμως οι πελάτες δεν γνωρίζουν τίποτα για τους υπόλοιπους παίκτες ακόμη.

Γι αυτό το λόγο δημιουργήσαμε την κατάσταση “Waiting for Ready” κατά την οποία ο διακομιστής στέλνει ένα μήνυμα σε κάθε πελάτη, με το οποίο τον ενημερώνει για τους άλλους παίκτες. Δημιουργούμε λοιπόν μια υποκλάση της κλάσης “Packet” με το όνομα “PacketTypeServerReady”.

```
- (void)addPayloadToData:(NSMutableData *)data
{
    [data rw_appendInt8:[self.players count]];

    [self.players enumerateKeysAndObjectsUsingBlock:^(id key, Player *player, BOOL *stop)
     {
         [data rw_appendString:player.peerID];
         [data rw_appendString:player.name];
         [data rw_appendInt8:player.position];
     }];
    NSString* sCategory = [NSString stringWithFormat:@"%i", _category];
    NSString* sSetTime = [NSString stringWithFormat:@"%i", _setTime];
    NSString* sSetQuestions = [NSString stringWithFormat:@"%i", _setQuestions];
        [data rw_appendString:sCategory];
        [data rw_appendString:sSetTime];
        [data rw_appendString:sSetQuestions];
}
```

Κάθε πελάτης χρησιμοποιεί αυτό το μήνυμα για να δημιουργήσει ένα λεξικό “NSMutableDictionary” με τα αντικείμενα του κάθε παίκτη. Έτσι μοιράζονται όλοι οι πελάτες τις ίδιες πληροφορίες με το διακομιστή. Επιπλέον στο ίδιο πακέτο μεταφέρονται τα στοιχεία του παιχνιδιού για κάθε πελάτη, δηλαδή ο συνολικός χρόνος η κατηγορία ερωτήσεων καθώς και ο συνολικός αριθμός αυτών.

Για να εξασφαλίσει ο διακομιστής ότι όλοι οι πελάτες έχουν λάβει το μήνυμα “Server Ready” και έχουν δημιουργήσει επιτυχώς το λεξικό των παικτών καθώς και τα υπόλοιπα στοιχεία του παιχνιδιού, περιμένει από κάθε πελάτη να στείλει πίσω ένα μήνυμα “Client Ready”. Το νέο μήνυμα “PacketTypeClientReady” δεν χρειάζεται επιπλέον δεδομένα, οπότε δε δημιουργούμε νέα υποκλάση. Προσθέτουμε απλώς στη μηχανή καταστάσεων τον παρακάτω κώδικα:

## 5.6 Η κατάσταση “Waiting for Ready”

```
case PacketTypeServerReady:
    if (_state == GameStateWaitingForReady)
    {  // Ο διακομιστής στέλνει σε όλους τους πελάτες όλα τα στοιχεία των υπολοίπων.
        _players = ((PacketServerReady *)packet).players;
        _category = ((PacketServerReady *)packet).category;
        _setTime = ((PacketServerReady *)packet).setTime;
        _setQuestions = ((PacketServerReady *)packet).setQuestions;
        [self changeRelativePositionsOfPlayers];

        Packet *packet = [Packet packetWithType:PacketTypeClientReady];
        [self sendPacketToServer:packet];
        [self beginGame];

    }
    break;
```

Τη στιγμή που οι πελάτες είναι έτοιμοι να ξεκινήσουν πραγματικά το παιχνίδι περνάνε στην κατάσταση “Playing”, όμως ο διακομιστής είναι αυτός που θα ξεκινήσει το παιχνίδι. Όπως φαίνεται στον παραπάνω κώδικα καλούμε δύο νέες μεθόδους τύπου (Void), αυτή που ξεκινά το παιχνίδι (beginGame) και την (changeRelativePositionOfPlayers). Με τη μέθοδο “beginGame” ξεκινάμε και αποστέλλουμε στους πελάτες την πρώτη ερώτηση και ξεκινά το παιχνίδι. Παρακάτω θα επισημάνουμε τον κώδικα για την αλλαγή της θέσης του κάθε παίκτη στην οθόνη. Η συνθήκη αυτή περιστρέφει δεξιόστροφα τους παίκτες στην οθόνη, ώστε ο τοπικός χρήστης να βλέπει το όνομα του στο κάτω μέρος της οθόνης. Για να το πετύχουμε αυτό χρησιμοποιούμε το “peerID” του κάθε παίκτη και όχι τη θέση ή το όνομα για να τον αναγνωρίσουμε.

```
- (void)changeRelativePositionsOfPlayers
{
    NSAssert(!self.isServer, @"Must be client");

    Player *myPlayer = [self playerWithPeerID:_session.peerID];
    int diff = myPlayer.position;
    myPlayer.position = PlayerPositionBottom;

    [_players enumerateKeysAndObjectsUsingBlock:^(id key, Player *obj, BOOL *stop)
     {
         if (obj != myPlayer)
         {
             obj.position = (obj.position - diff) % 4;
         }
     }];
}
```

Τέλος όταν ο διακομιστής λάβει απάντηση από όλους τους πελάτες ότι είναι έτοιμοι, καλεί τη μέθοδο (beginGame) για να ξεκινήσει το παιχνίδι από την πλευρά του διακομιστή. Δημιουργούμε λοιπόν μια νέα κατάσταση στη μέθοδο serverReceivedPacket:fromPlayer όπως φαίνεται παρακάτω:

```
case PacketTypeClientReady:
    if (_state == GameStateWaitingForReady &&
        [self receivedResponsesFromAllPlayers])
    {
        [self beginGame];
    }
    break;
```

# 5.7 Παίζοντας δικτυακά την εφαρμογή

Το δικτυακό κομμάτι της εφαρμογής βασίστηκε στη λογική του απλού χρήστη. Μέχρι τώρα είδαμε την επικοινωνία μεταξύ των συσκευών και τη δημιουργία των πιο σημαντικών πακέτων για την εφαρμογή.

Η βασική διαφορά στο δικτυακό παιχνίδι είναι ότι οι συσκευές μοιράζονται ένα κοινό controller. Επίσης αλλάζει και η ροή του παιχνιδιού ανάλογα με την απάντηση του χρήστη.

Φτάσαμε μέχρι τώρα στην κατάσταση "Playing" για το διακομιστή και τους πελάτες. Τώρα θα δούμε τη μέθοδο η οποία είναι υπεύθυνη για την εμφάνιση των στοιχείων στην οθόνη. Πηγαίνουμε στον "GameViewController" και προσθέτουμε μια μέθοδο τύπου -(void) με το όνομα gameDidBegin. Η μέθοδος εμφανίζει στην οθόνη τα ονόματα των παικτών, τις σωστές απαντήσεις, το χρόνο, τις ερωτήσεις και τις απαντήσεις. Επίσης έχουμε προσθέσει και μια κεντρική ετικέτα η οποία μας ενημερώνει κάθε φορά για την κατάσταση του παιχνιδιού.

Δημιουργούμε αρχικά μια υποκλάση της κλάσης "Packet" με το όνομα "PacketQuestions". Αυτή η υποκλάση χρησιμοποιείται για τη μεταφορά της ερώτησης και των απαντήσεων μεταξύ των συσκευών.

```
case PacketTypeQuestions:
    if (_state == GameStatePlaying)
    {
        _selectQuestion = ((PacketQuestions *)packet).question;
        [self.delegate gameShowQuestion:self question:_selectQuestion];
    }
    break;
```

Έπειτα, ο διακομιστής περιμένει να λάβει μια απάντηση από τους πελάτες για την επιλογή απάντησης. Δημιουργούμε μια νέα υποκλάση της κλάσης "Packet" με το όνομα "PacketAnswerResponse". Αυτό το μήνυμα αποστέλλεται σε όλους τους πελάτες από το διακομιστή, για να τους ενημερώσει ότι περιμένει μια επιλογή για την απάντηση της ερώτησης.

```
case PacketTypeAnswerResponse:
    if (_state == GameStatePlaying)
    {
        NSString *pID = ((PacketAnswerResponse *)packet).peerID;
        Player * player = [self playerWithPeerID:pID];
        player.gamesWon += 1;
        [self.delegate gameWhoAnswerCorrect:self playerName:player.name];
    }
    break;
```

Το αντίστοιχο μήνυμα απάντησης για τους πελάτες ονομάζεται "PacketClientAnswer". Όταν η κατάσταση είναι ακόμη "Playing"

ελέγχεται από το διακομιστή και αν ο πρώτος παίκτης απαντήσει σωστά, αυξάνεται το σκορ του κατά ένα βαθμό. Ταυτόχρονα ενημερώνεται η κεντρική ετικέτα ότι κάποιος παίκτης απάντησε σωστά και εμφανίζεται η επόμενη ερώτηση. Αν ο πρώτος παίκτης δεν απαντήσει σωστά τότε ενημερώνεται η κεντρική ετικέτα με το μήνυμα "Another player choose". Αν τέλος δεν απαντήσει κανείς σωστά εμφανίζεται η επόμενη ερώτηση. Για να λειτουργήσει σωστά το παιχνίδι ελέγχουμε κάθε επιλογή των παικτών από το "peerID" τους. Έτσι κάθε παίκτης έχει το δικαίωμα μόνο μιας επιλογής απάντησης.

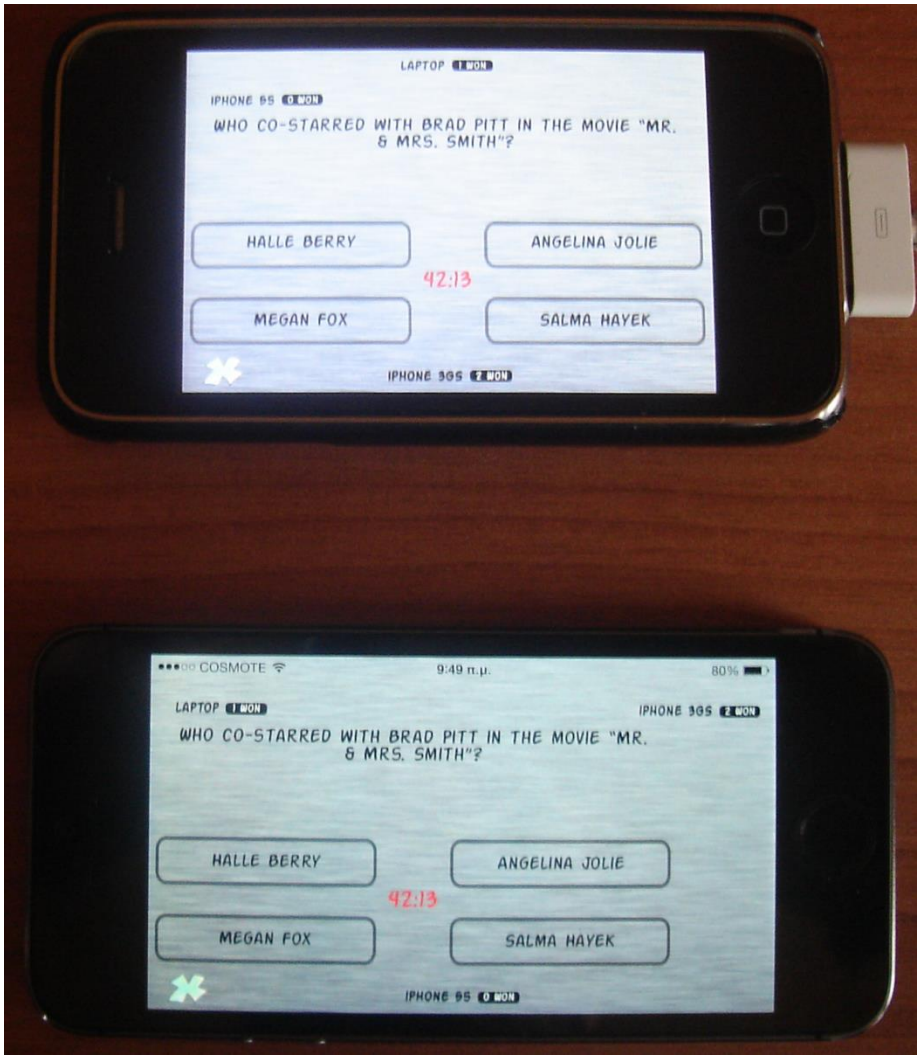
Τέλος, όπως και στην απλή μορφή της εφαρμογής, το παιχνίδι τελειώνει όταν περάσει ο επιθυμητός χρόνος ή όταν τελειώσουν οι ερωτήσεις που έθεσε ο διαχειριστής. Τότε οι χρήστες περνάνε στην κατάσταση "Game Over". Η κατάσταση αυτή αποτελείται από μια μέθοδο τύπου -(void) που ενημερώνει όλους του πελάτες με το όνομα του νικητή. Επειδή δεν χρειάζεται επιπλέον δεδομένα, δημιουργούμε ένα νέο τύπο πακέτου με το όνομα "PacketTypeGameOver" και καθαρίζουμε την οθόνη όλων των παικτών.

```
- (void)gameOver
{
    _state = GameStateGameOver;
    NSString *playerName = [self winsPlayer];
    if (self.isServer)
    {
        Packet *packet = [Packet packetWithType:PacketTypeGameOver];
        [self sendPacketToAllClients:packet];
    }
    [self.delegate gameOver:self playerName:playerName];
}
```

Ακόμη υπάρχει και το κουμπί εξόδου για το διαχειριστή και τους πελάτες. Αν κάποιος πελάτης επιλέξει να εγκαταλείψει το παιχνίδι, περνάει στην κατάσταση "Quitting" και το όνομα του δεν εμφανίζεται πλέον στον controller του παιχνιδιού. Δημιουργούμε μια καινούργια μέθοδο με το όνομα clientDidDisconnect στην κλάση "Game", κατά την οποία αφαιρούμε το αντικείμενο του παίκτη και το όνομα του από το λεξικό. Δημιουργούμε ένα νέο πακέτο με το όνομα "Packet-OtherClientQuit" . Στην ίδια μέθοδο, στέλνουμε αυτό το πακέτο στους υπόλοιπους πελάτες για να τους ενημερώσουμε ότι κάποιος εγκατέλειψε το παιχνίδι όπως φαίνεται στον παρακάτω κώδικα.

## *5.7 Παίζοντας δικτυακά την εφαρμογή*

```
- (void)clientDidDisconnect:(NSString *)peerID
{
    if (_state != GameStateQuitting)
    {
        Player *player = [self playerWithPeerID:peerID];
        if (player != nil)
        {
            [_players removeObjectForKey:peerID];

            if (_state != GameStateWaitingForSignIn)
            {
                if (self.isServer)
                {
                    PacketOtherClientQuit *packet = [PacketOtherClientQuit
                                                     packetWithPeerID:peerID];
                    [self sendPacketToAllClients:packet];
                }

                [self.delegate game:self playerDidDisconnect:player];
            }
        }
    }
}
```




Εικόνα 5.12: Δικτυακό παιχνίδι με τρείς παίκτες

# ΚΕΦΑΛΑΙΟ 6$^{ο}$

# Επίλογος

Στο κεφάλαιο αυτό γίνεται η σύνοψη της διπλωματικής εργασίας και της εφαρμογής που προέκυψε. Επίσης γίνεται αναφορά για την μελλοντική εξέλιξη της εφαρμογής και των δυνατοτήτων αυτής.

## 6.1 Σύνοψη και συμπεράσματα

Μέσα από την παρούσα διπλωματική εργασία μου δόθηκε η ευκαιρία να γνωρίσω τις κινητές πλατφόρμες και τα λειτουργικά τους συστήματα. Επίσης, προγραμματίζοντας για το λειτουργικό σύστημα iOS, γνώρισα τις ιδιαιτερότητες του προγραμματισμού για κινητές πλατφόρμες.

Με το πέρας της διπλωματικής εργασίας δημιουργήθηκε μια εφαρμογή γνώσεων στο λειτουργικό σύστημα iOS. Στόχος ήταν η εξοικείωση με τη γλώσσα προγραμματισμού Objective C, καθώς και η γνώση αυτής. Κατά την ανάπτυξη της εφαρμογής βρέθηκα αρκετές φορές αντιμέτωπος με προβλήματα που δεν περίμενα ότι θα συναντήσω. Όμως η διαδικασία εύρεσης λύσης στα προβλήματα αυτά, ήταν αύτη που με οδήγησε σε περεταίρω μελέτη του αντικειμένου. Η ολοκλήρωση της δημιουργίας της εφαρμογής που πραγματοποιεί διαφορετικές λειτουργίες, ήταν μια ολοκληρωμένη εμπειρία γύρω από τον προγραμματισμό.

Η εφαρμογή δημιουργήθηκε αρχικά για το σενάριο του απλού χρήστη. Έπειτα βασίστηκε στις τεχνολογίες των Bluetooth – Wi-Fi για τα σενάρια πολλαπλών χρηστών. Καθοριστικό ρόλο στη λειτουργία της εφαρμογής έπαιξε η χρήση της SQLite, η οποία χρησιμοποιήθηκε για την δημιουργία της βάσης δεδομένων της εφαρμογής. Κατανοήθηκε ο τρόπος λειτουργίας του μοντέλου πελάτη – διακομιστή που χρησιμοποιείται σε δικτυακές εφαρμογές.

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε ένας φορητός ηλεκτρονικός υπολογιστής με λειτουργικό σύστημα Mac OS X. Η

εφαρμογή αποτελείται από 76 διαφορετικά αρχεία, μαζί με τα αρχεία μορφοποίησης και ο συνολικός αριθμός γραμμών κώδικα, είναι 11.496.

Το αποτέλεσμα κρίθηκε ικανοποιητικό με βάση τους διαφορετικούς χρήστες που χρησιμοποίησαν την εφαρμογή. Η εφαρμογή τοποθετήθηκε στη συσκευή iPhone 3GS και δοκιμάστηκε εκτενώς. Δοκιμάζοντας πρακτικά την εφαρμογή προέκυψαν διάφορα σφάλματα τα οποία διορθώθηκαν στην πορεία.

## 6.2 Μελλοντικές εξελίξεις

Κύριος μελλοντικός στόχος είναι η διάθεση της εφαρμογής στο κοινό μέσω του App Store. Για τη διαδικασία αυτή πρέπει να εγγραφούμε στο πρόγραμμα των προγραμματιστών iOS της Apple με κόστος 99 δολάρια το χρόνο. Έτσι καθίσταται δυνατό κάθε κάτοχος συσκευών με λειτουργικό σύστημα iOS να κατεβάσει την εφαρμογή, να τη δοκιμάσει και να τη βαθμολογήσει.

Ακόμη σημαντικό είναι να εμπλουτίσουμε τη βάση δεδομένων με περισσότερες ερωτήσεις για να ανακαλύπτει πάντα κάτι νέο ο χρήστης. Έπειτα, από τις κριτικές των χρηστών θα μπορέσουμε να επιτύχουμε καλύτερα αποτελέσματα με ένα update της εφαρμογής, όπως το φιλικότερο προς το χρήστη περιβάλλον.

Παρακάτω ακολουθούν κάποιες από τις σημαντικότερες επεκτάσεις τις εφαρμογής:

- Ελληνική έκδοση της εφαρμογής και μετάφραση αυτής στις κυριότερες ξένες γλώσσες για την χρήση από επιπλέον χρήστες.
- Προσθήκη διάφορων ήχων για τη σωστή ή λάθος απάντηση καθώς και χρωμάτων στα κουμπιά των απαντήσεων.
- Προσθήκη σύνδεσης στα δημοφιλέστερα κοινωνικά δίκτυα και high score μεταξύ των φίλων του χρήστη.
- Φιλοξενία και υποστήριξη της βάσης δεδομένων μέσω διαδικτύου ώστε κάθε χρήστης να ανανεώνει τη βάση πριν την έναρξη της εφαρμογής.

# *Design of networking game in the iPhone iOS platform*

Partonas Alexandros
University of Western Macedonia
Department of Informatics and Telecommunications Engineering
Kozani, Greece
axelpart@hotmail.com

Dr. Dasygenis Minas
University of Western Macedonia
Department of Informatics and Telecommunications Engineering
Kozani, Greece
mdasyg@ieee.org

*Abstract*—**In our days everyone has the possibility to use a mobile device more than a desktop computer because of its portability. In this way every user can have direct and easy access to any information. The last five years we have witnessed a new revolution in information technology: The revolution of the mobile applications, especially on smart phones and tablet devices. It is estimated that soon people will use their mobile devices twice the time they work on desktop, because internet browsing optimized for mobile devices. Today, there are many different mobile platforms such as Android, iOS and Windows Mobile. Here, we present our design for the iOS platform, using as a test case a quiz application. Starting from the idea, we move to a hard copy design, coding, networking, design verification and testing. The main focus of this paper is to present the full design flow and to motivate other developers to select the iOS as their working platform.**

***Application design; quiz game; iOS; iPhone; xCode; Application; Smartphone; SQLite Database; Bluetooth; WiFi***

## I. INTRODUCTION

Nowadays companies are facing a lack of mobile development talents. Market demands developers for apps to run on iOS, Android and whatever operating system will come next. For young developers with programming skills, that is a promising opportunity on a career makeover.

The way we communicate, do business, and access news and entertainment changed with the latest mobile devices and their applications. Everyone has the ability to access their mobile device easier than their Desktop computer. Mobile application developers become one of the most demanding and fastest growing IT careers.

Applications for mobile platforms have evolved to a point that offers the user a rich and fast experience. Applications are technically categorized based on the programming environment in which they are executed, such as iOS, Android, or Symbian OS. They are also divided according to their functions for mobile platforms as follows: communication applications such as email, production applications such as calculators, multimedia applications such as audio and video streaming, and game applications.

Mobile developers, develop using the Objective C, C++, C# or Java programming languages. A mobile app developer chooses the mobile platform they will develop for, such as Google's Android or Apple's iOS. Then, they learn the programming languages and software development environment for that platform. As a mobile developer, must take into consideration that they have to deal with many different devices. They have to think about the available screen sizes, processor units and RAM, in order to execute an application on every different device. That is the main difference between developing for a Desktop computer and a mobile device. On Desktop computers we can choose the performance and RAM and have a powerful machine.

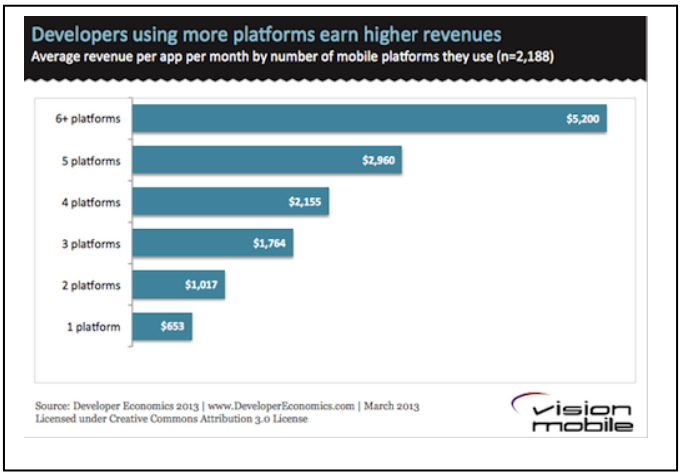



Figure 1. Developers revenues depending on their working platforms

Figure 1, demonstrates that the developers employing just in one platform are probably solo, amateur developers or have not yet had an expansion onto more platforms [1]. In order to earn more revenue, developers should expand their applications onto new platforms. So, developing in a multi-platform strategy it is likely to generate more revenue.

We all know that it is hard for someone to be a developer. However, we are here to motivate you by presenting our own mobile design flow from the scratch, for a quiz application in the iOS platform.

The rest of the paper is structured as follows. In section II, we initiate with the idea, where we analyze the steps of design and development of the application. In Section III, we describe the operations connected with the database. In Section IV and V, we are dealing with the creation of the quiz project and its networking function, respectively. In Section VI, we present our testing procedure and analyze the results. Finally, we give the concluding remarks in Section VII.

## II. DESIGNING AND DEVELOPMENT OF APPLICATION

It is well known, that most of the students who try to become developers give up in the middle of the road, because of high demands. It is hard for a new developer to deal the many different things that are required in order to achieve a functional application.

When we decided to create our application, we thought that it would be useful for someone to be able to test their knowledge while enjoying a coffee with their friends. It is also very fruitful for a teacher to use a knowledge game to intrigue his students and to help them grasp and retain knowledge about his course. Researchers have shown that learning using games has a much stronger impact than the steer typical teaching of a lecture [7].

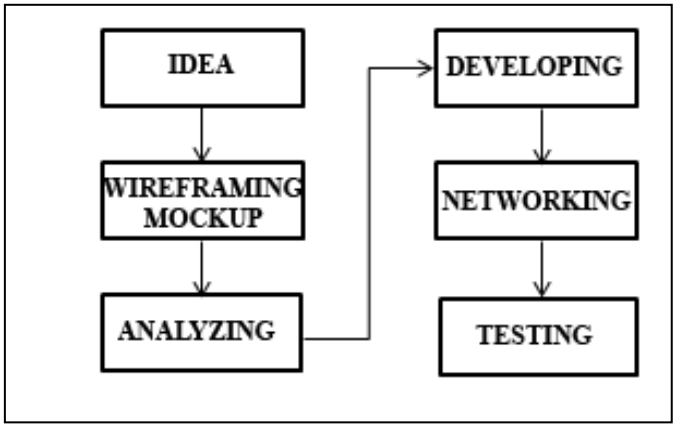



Figure 2. Steps for designing an application

To create a successful application, a developer has to be methodic, systematic and follow some important steps. We present our design flow in iOS platform, which is architecture independent and can be used from developers, in order to create their application from the beginning. Figure 2, shows the steps that we follow in order to create our application. First, we have an idea for an application. Then, we create a sketch of the application view. In the third step, we see how we can analyze the functions of our project. On steps four and five we begin developing the application and finally we testing it.

### *A. It all starts with an idea*

Before a developer initiates an application, it is suggested to perform a survey of similar applications, to investigate their drawbacks and to note down the best practices. We came to the conclusion that the most of the apps need social networks for playing multiplayer. Some multiplayer apps are exclusively developed for iPad, because of its large screen. So, we decided to create an application that does not require sharing the screen with other users, but allows every user to see the same information with the rest.

To take full advantage of the tools given to a developer, one must first describe as solid as possible the type of application. Our application belongs to the category of entertainment and educational applications - games [7] designed to test the knowledge of the users. For the development of a successful application in iOS a developer should know some basic things. These have to do with designing, proper memory management, responsiveness, energy consumption and security.

### *B. Wireframing the idea (Sketching)*

To start with, the designer needs to create on a paper a sketch of how he imagines the interface. This process is very important because it helps him to arrive at the desired result with less effort. Also by drawing on paper, it is easier for the designer to show his ideas to others so that he receives comments and reviews about them. Then, the final result may be completely different from the original sketch.

Wireframing [2] is a very important step in the designing and development of an application. It is a low-level outline (draft, framework, and blueprint) of someone's design, which helps them and their customers to illustrate the structure and the sensation that leaves their designing. In wireframes, a designer replicates data throughout the application with their real dimensions and sizes, but nevertheless it does not contain graphic elements, but only the position that they will take in the final designing on screen. Through the wireframe, the designer can make structural changes, changes to buttons, text, titles, and application components in a very short time, and thus they can easily achieve a clearer picture of their application.

For our application we originally created every single view on paper. Then we used software to digitize our design and make a more accurate picture of our application. Figure 3, saws the digital design (mockup) of our multiplayer game view controller.

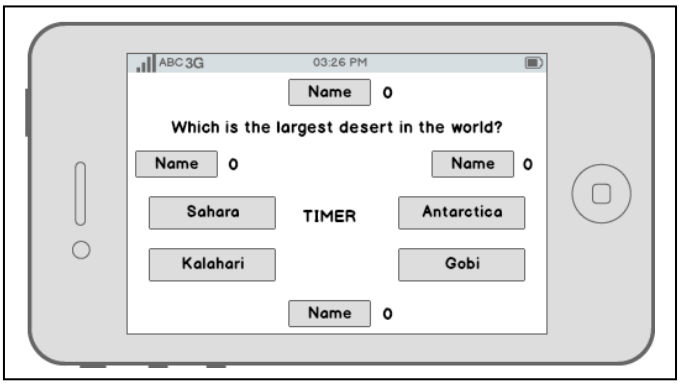



Figure 3. Digital wireframing of game view controller

### *C. Analyzing the modes of execution*

Before a developer begins to develop his application programmatically, it is important to clarify how it operates. This is usually analyzed by the user's actions in relation to the application. It would be even easier for the developer to

analyze his app's modes of execution, by creating diagrams with the actions of users.

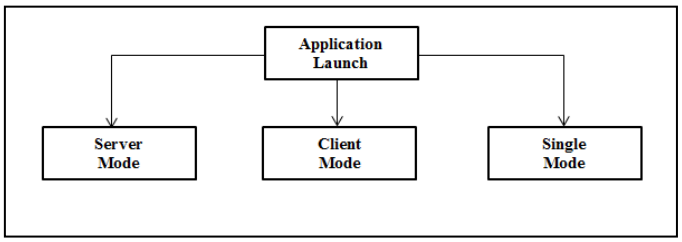



Figure 4. Flow diagram of user actions

Starting the quiz game, the user can choose the "behavior" of the device. Figure 4 shows that there are three options of execution. The first is about hosting an online-web game (server), and the second about a client in a network game. These two options use networking technologies (Bluetooth Wi-Fi). The third option is about the single use that does not require any networking technology.

After that, the choices the user has when connecting as an administrator are being analyzed. If he chooses to host a game, the device starts transmitting the beacon in order for the clients to discover his device. When networking technologies are disabled, the user is notified with an alert message to enable at least one of networking technologies. Then, he waits for other clients to connect, with the maximum number of three. Once the connection is completed, he chooses the time and the questions that he is going to play with the rest of the players.

As a client, the user sees the available servers and he connects to one of them. When all the clients are connected, the server sets the game, and the first question is appeared.

Finally, we have the function of single user. By selecting to play alone, the user selects the category and then sets the time and the questions. Networking technologies are not needed in this option.

## III. ANALYSING AND DESIGNING THE DATABASE

Usually, mobile applications have to store data. This can either be fulfilled by means of a file, or by a database, or in a web location. Each one of these techniques has its advantages and disadvantages. For storing the data, we have decided to use a database created by SQLite [3], because of the size that it occupies. Its use is ideal for devices where memory and process power are limited.

Before beginning coding, developer should begin with the creation of the external files. These files usually consist of graphic features such as button images, or a database. All external files will be added to his total project when he starts coding. In case of a database, in order to achieve a functional and working database, the developer should determine the architecture of his application and the roles of its users.

This method helped us in our application and we achieved to optimize our database. As a result, we created only two tables for the quiz game instead of six.

### *A. Architecture*

The developer has to determine the architecture of his application so that he can simplify his project and start implementing it step by step. Our system is based on two different categories of implementation. In the first one, we use the client-server model with bidirectional communication. That means that we communicate and share information between all devices on the creation of the game. In the second case, we only use the system in which a simple extraction of data to the user is being done.

In the first case, the system supports up to four users, one of them being the host of the game. For this function, a connection via wireless networks from the server is being created and starts sending information to the rest of the available clients. The entire game architecture is based on the message packets between server and clients.

In the second case, we have a different approach to the game. Networking technologies not executed and the application runs with the basic configurations. We have a single user who receives on-screen information about the game and acts on them.

### *B. Roles of the system*

In our application we analyzed all possible roles and operational flows, in order to construct the database. Due to lack of space, we will present only one flow of execution as an example for the administrator role. The user who chooses to use the application in communication protocols IEEE 802.11 (Wi-Fi) - Bluetooth, is defined as an administrator and becomes the local server that will host the remaining players.

Basic operational flow for administrator role

- In this use case, the administrator starts the host game by selecting the corresponding button.
- Then, he/she enters the name of his/her preference for the creation of the local server that will host the game.
- He/She waits for the other players who wish to connect to the administrator's network to do it.
- After the number of the available players is completed, he/she selects "Start".
- Then, the form for selecting the question categories is displayed.
- The administrator has the option to select one or more of them if he/she wishes.
- Finally, by pressing the button "Start", he/she also defines the desired duration time of the game, and the number of the questions on which he/she will compete with the rest connected users.

Alternative flow - Canceling creating the local server

Administrator has the option to cancel the impending start of the game that has been created whenever he/she wishes by clicking on the "X" at the bottom left of the game form.

### C. Database operation

In our application, the database was created using SQLite by the terminal. It is suggested for a developer to use the terminal in order to begin familiarizing with writing code. After its creation, he can use the SQLite plugin for Firefox to add faster entries into tables.

In our quiz game the database is read only. This means that the three users that we defined above have not the ability to add or remove questions through the application. The only way to add data in the database is online, or through a new refreshed version of the entire application.

When the application starts, the database is initially set to be renewed after choosing the category. That means that it writes in the field «Used» the value 'NO' to all the questions that are saved in the database. This is the proper use of the database in the process of the game. Choosing the category of questions they want, either the server or the single user start the game.

The database starts exporting the category questions the user selected randomly and with selection criteria the fact that the field «Used» has the value 'NO'. For each question that is being displayed, the change of the variable of the field «Used» with the value 'YES' to this question is automatically being done. This way we can avoid the appearance of a double question.

Finally, so that there is no abuse-intemperance of the device memory, we have defined that the database closes each time it performs a question.

## IV. CREATING THE PROJECT

All new mobile operating systems are coming with their own software application for development, in the case of anyone wants to occupy and develop apps for each operating system. So, if a developer decides to create an application, he has all the tools he needs to achieve that.

We chose the iOS because the experience is much more enjoyable since we can focus on the product and innovative ideas. We refused developing for Android because we needed to consider about the many different screen sizes of devices. Objective C resembles to C++ but in reality it has a lot different syntax.

When programming for iOS we wrote our code in Xcode and used its simulation for testing. XCode gives to developer the ability to choose between different options for his project. We suggest that the developers choose an “Empty Application” project on xCode, because with this option they can customize View Controllers for their applications.

### A. Analysing game view controller

All view controllers in our application where created from “Empty view controllers” by adding only the elements we needed. By starting our application, the user joins the menu. Menu consists of the following three buttons: “Host Game”, “Join Game”, and “Single Player”.

When the user selects “Single Player”, the category controller is appeared. This controller consists of one table view which is filled with values of category table. Additionally, two buttons are available, one for starting the game, and the other for exiting to the main menu [5]. By choosing a category, the user has the ability to know the total number of its questions. For that purpose, an alert message that displays the necessary information was created.

Figure 5 represents the game view controller, which consists of seven labels. One of them is for the display of the time, one for the score of the correct answers, and the other five for the question and its’ possible answers. Also, there are four more buttons that cover the labels of the answers and will be used for the method that will check the right answer.


Figure 5. View of the game when loading the answers

### B. Analysing the game code

In the process of programming, developers should use techniques to minimize the effort in building and deploying their application. The most important is to learn about errors, how to handle them, and how to fix bugs of their application. Many times, we had to deal with problems in our application. Bellow we analyze the game code and some techniques we used.

Starting the game, the first thing to do is to clear the screen so that the notification message of time input and number of questions are displayed. To achieve this in every controller, there is a method - (void)WillAppear:(Bool) animated, stating the methods that will be activated at the start of the controller. Here we created a method to achieve the screen clearing and disabling the use of the buttons in case the user accidentally touches the screen.

After clearing the screen, if we wish to start the game we must set the time and questions. To complete the notification message we create two variables of "Int" type. After the user has set the data on the time and questions by pressing the "OK" button on the notification message, the method of time display, timer start up, and questions are triggered. At the same time we get the values and we place them in their own labels. In order for this button to activate, certain conditions must be fulfilled.

For the game time, the user needs to enter more than two digits, while the number of questions should not exceed the listed questions. To do that, we need a variable of "NSInteger", which we will import from the previous view controller (category view controller).

When a question is presented to the user, they must choose a possible answer. It would not be right to derive the answers in the order they are written to the database. So, we create a condition that randomly shuffles the answers "arc4random" and it saves them temporarily in a table of "NSMutableArray" type.

When the temporary table is filled with the four scrambled numbers of responses, we display the answers shuffled to the user, setting the text in every label.

In order to ensure the proper operation of the answer method, we set tags on each label, which receives the correct answer with numbers "999" and "0" for the wrong. We also equate the labels to the corresponding buttons to implement the method of the correct answer.

We create a method -(IBAction) which we connect with all the buttons. Depending on the user's choice, if the answer is correct, the method that increases the points for every correct answer and the method of the next question are running in sequence. If the answer is wrong, the method of the next question is executed.

Finally, to arouse the interest of the user, the moment he/she is choosing an answer, at the same time a hidden second timer of ten seconds with a countdown is "triggered". With this timer we created a method on the total user points. We have set the maximum number of points at one hundred. The time of ten seconds starts counting the moment that all the answers are displayed. This means that the faster the user answers the more points he/she gets.

## V. CREATING NETWORKING PART

In our interconnected world, it is very important for an application to exploit the communication abilities of modern mobile phones, such as Wi-Fi, Bluetooth or near field communication (nfc). Each of these technologies has its advantages and disadvantages. For example if you create an RSS Feeder you have to use Wi-Fi. In that case neither Bluetooth nor NFC are functional. Depending on the category your application belongs, it is recommended to use the most suitable networking technologies.

In our application, we selected Wi-Fi and Bluetooth because of their diversity. In the first case, Wi-Fi technology gives us the ability to connect with other users in a local network, but also with the tethering technology, which allows the use of a mobile phone as an intermediate (access point) so that an internet access is provided to another connected device, either via cable or wirelessly. Bluetooth on the other hand, is ideal for use when the users are less than ten meters far from each other and on places where Wi-Fi is not available.

### *A. Networking architecture*

For creating online-network games, developers require to know that they actually have a choice between two architectures, which are: client - server and peer users (peer-to-peer) [6]. For the implementation of the game, the architecture peer-to-peer was chosen. The user who hosts the game is the server, while all the other players are the customers.

In the client-server model, the server is responsible for everything and determines the state of application. The clients send update messages to the server all the time and the server informs all the clients respectively. The clients do not have the capability to communicate with each other.

In the peer-to-peer model, all participants are equal and they all do the same job, but we must ensure that each participant sees the same things as the rest, since there is no central administrator.

### *B. Client-server state machines*

When developing networking applications, it is handy for a developer to create state machines. State machines give us the opportunity to understand the logic of our application. Bellow we present a sample of our quiz application state machines.

Starting from the client, it is good to create a diagram which describes the possible states of the objects in the class. This makes the usage of the objects simpler.

Overall, we create four states. If the client chooses to join a game he enters the state "Idle", in which he remains idle. Then, he moves to the "Searching for Servers" state. When the user decides to connect to a specific server, the client goes into "Connecting to Server" state, where he tries to connect to the server and finally moves to state "Connected" once the connection is successfully established. If at any time during the last two states the server drops the connection, the client moves back to “Idle” state.

The state machine of the server in relation to the client has only three situations. By choosing to become a server, a user gets into the state "Idle" in which he waits without doing anything. Then, he automatically moves to state "Accepting connections". When the user presses the "Start" button to start the game between users, he is transferred to the "Ignoring new connections" state. This means that no new clients are allowed to connect.

### *C. Identifying error and disconnection reasons*

In networking applications, the connection between devices is often sensible. It is suggested that the developer identifies networking errors and disconnections and demonstrate them to the users of his application [4]. In our application in specific, we produced an alert message that warns users about the following errors:

*a) Quit reason: No network.*

*b) Quit reason: Conection dropped*.

c) *Quit reason: User quit*.

*d) Quit reason: Server Quit.*

### D. *Packet structure*

For all messages the developer has to follow a specific format. Each package must be at least ten bytes. Different types of packages have the same header but different payload. Figure 6 shows the structure of our packages.

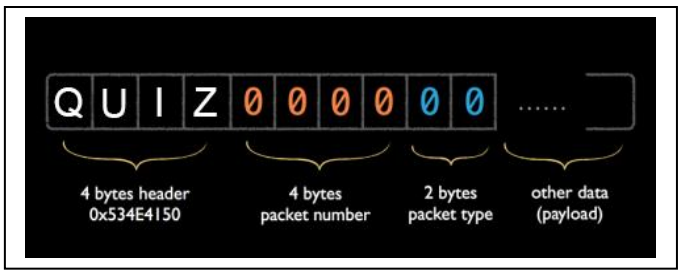



Figure 6. Quiz application data structure

The first four bytes, from the header with the name QUIZ verify that the packages are ours. The next four bytes are used to recognize when a packet arrives out of order, and consist of a 32-bit integer. The last two bytes represent the packet type. Due to the fact that we have many kinds of messages that are sent between the server and clients, this 16-bit integer determines what type of packet it is.

## VI. TESTING

The application was initially created for the single user scenario. Then, it was based on the technologies of Bluetooth-Wi-Fi for multi-user scenarios. Decisive role in the functioning of the application played the SQLite, which was used to create the application database. Additionally, with the simple graphic elements that composed the application, we achieved a lightweight version.

For the implementation of the application, a laptop computer with operating system Mac OS X was used. The application was placed on the iPhone 3GS and iPhone 5S devices and was tested extensively. The result was considered satisfactory, based on the different users that tested the application. Trying the application practically arose various errors that have been corrected along the way.

### A. *Usage performance*

By running the famous "Who wants to be a millionaire" application, we came to the conclusion that the RAM usage was approximately 55Mb. Our application with three players connected, consumes approximately 19Mb of RAM, which is less than 1/3 of the above mentioned application which was tested in single player mode.

A mobile developer should be careful about the resource usage, like memory. For this reason he should minimize all unnecessary RAM usage, by ignoring non-related packets. For example, while playing multiplayer, in case a player gives a wrong answer, there is no need for the server to send back verification for the wrong answer. Instead, the only sent packet is a packet for informing the next player that it's his turn.

## VII. SYNOPSIS AND CONCLUSIONS

Programming for the iOS operating system is not only an interesting experience, but also a promising source of income, especially for young developers. It may be a rough and climbly road, but in the end rewards are awaiting for the systematic, careful and endured developer. Developing for the mobile platform is a slippery track, and for that reason we present our design flow and experience to better help all enthusiastic mobile developers and to motivate indecisive developers. We presented that a careful step by step approach alleviates most of the problems, and guides towards the finalization of a nature project, useful to the community.

### A. *Future work*

The main future goal is to provide the application to the public via the App Store. For this process, we must be enrolled in Apple's iOS developers program, with the cost of $99 per year. This allows every device holder with iOS operating system to download the application, test and rate it.

It is also important to enrich the database with more questions, so that the user always discovers something new. Then, out of the users' reviews we will be able to achieve better results with an update of the application, such as more user-friendly environment.

Below are some of the most important extensions of the application:

- A Greek version of the application and a translation thereof in the major foreign languages, so that additional users would be able to use it.
- Addition of different sounds for the correct or the wrong answer, and of different colors on the buttons of responses.
- Addition of a connection to the most popular social networks and high scores among the friends of the user.
- Hospitality and database support through internet, so that every user refreshes the database before the start of the game.

## VIII. REFERENCES

[1] «VisionMobile,» [Digital]. Available: http://www.visionmobile.com/blog/2013/04/which-apps-make-more-money/,2013

[2] «GreekTuts,» [Digital]. Available: http://www.greektuts.net/wireframe-before-design/, 2013.

[3] «SQLite,» [Digital]. Available: http://www.sqlite.org/docs.html, 2014.

[4] Peter Bakhirev, PJ Cabrera, Ian Marsh, Scott Penberthy, Ben Britten Smith and Eric Wing, "Beginning iPhone Games DEvelopment", Apress, 2010.

[5] Erica Saudan, "The iPhone Developer's Cookbook - Building Applications with the iPhone 3.0 SDK", 2nd Edition – Addison Wesley 2010.

[6] «iOSTutorials,» [Digital]. Available: http://www.raywenderlich.com/tutorials, 2013.

[7] Ελένη Ρώσσιου, Σπύρος Παπαδάκης "Μαθαίνουμε Παίζοντας: Το Εκπαιδευτικό Παιχνίδι στη Διδασκαλία της Πληροφορικής στο Γυμνάσιο", Σύρος 2007.

# Βιβλιογραφία


[1] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Mac_os. [Πρόσβαση 24 Αυγούστου 2013].

[2] «Wikipedia,» [Ηλεκτρονικό]. Available: http://el.wikipedia.org/wiki/Bluetooth. [Πρόσβαση 7 Σεπτεμβρίου 2013].

[3] «Wikipedia,» [Ηλεκτρονικό]. Available: http://el.wikipedia.org/wiki/Wifi. [Πρόσβαση 7 Σεπτεμβρίου 2013].

[4] «Βικιπαιδεία,» [Ηλεκτρονικό]. Available: http://el.wikipedia.org/wiki/VoIP. [Πρόσβαση 7 Σεπτεμβρίου 2013].

[5] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Personal_digital_assistant. [Πρόσβαση 13 Αυγούστου 2013].

[6] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Smartphone. [Πρόσβαση 12 Αυγούστου 2013].

[7] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Tablet_computer. [Πρόσβαση 12 Αυγούστου 2013].

[8] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Mobile_operating_system. [Πρόσβαση 12 Αυγούστου 2013].

[9] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Palm_os. [Πρόσβαση 12 Αυγούστου 2013].

[10] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Windows_mobile. [Πρόσβαση 12 Αυγούστου 2013].

[11] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Android_%28operating_system%29. [Πρόσβαση 13 Αυγούστου 2013].

[12] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/BlackBerry_OS. [Πρόσβαση 13 Αυγούστου 2013].

[13] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Symbian. [Πρόσβαση 13 Αυγούστου 2013].

[14] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Windows_Phone. [Πρόσβαση 13 Αυγούστου 2013].

[15] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/IOS. [Πρόσβαση 13 Αυγούστου 2013].

[16] «Wikipedia,» [Ηλεκτρονικό]. Available:

http://en.wikipedia.org/wiki/Mobile_app. [Πρόσβαση 14 Αυγούστου 2013].

[17] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Apple_Inc. [Πρόσβαση 24 Αυγούστου 2013].

[18] «iOS Developer Library,» [Ηλεκτρονικό]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898-CH1-SW1. [Πρόσβαση 24 Αυγούστου 2013].

[19] «Sqlite,» [Ηλεκτρονικό]. Available: http://www.sqlite.org/about.html http://en.wikipedia.org/wiki/SQLite#Development. [Πρόσβαση 4 Σεπτεμβρίου 2013].

[20] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Objective-C. [Πρόσβαση 26 Αυγούστου 2013].

[21] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Automatic_Reference_Counting. [Πρόσβαση 26 Αυγούστου 2013].

[22] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/NEXTSTEP. [Πρόσβαση 26 Αυγούστου 2013].

[23] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Xcode. [Πρόσβαση 27 Αυγούστου 2013].

[24] «Developer,» Apple, [Ηλεκτρονικό]. Available: https://developer.apple.com/technologies/tools/features.html. [Πρόσβαση 27 Αυγούστου 2013].

[25] «GreekTuts,» [Ηλεκτρονικό]. Available: http://www.greektuts.net/wireframe-before-design/. [Πρόσβαση 29 Αυγούστου 2013].

[26] «Balsamiq Mockups,» [Ηλεκτρονικό]. Available: http://balsamiq.com/products/mockups/. [Πρόσβαση 31 Αυγούστου 2013].

[27] «Wikipedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Atomicity,_consistency,_isolation,_durability. [Πρόσβαση 4 Σεπτεμβρίου 2013].

[28] «Wikipedia,» [Ηλεκτρονικό]. Available: http://el.wikipedia.org/wiki/Endianness. [Πρόσβαση 11 Δεκεμβρίου 2013].

[29] B. D.-C. Adamson, iPhone SDK Development, The Pragmatic Bookshelf, 2009.

[30] P. B.-P. C.-I. Marsh, Beginning iPhone Games Development, Apress, 2010.

[31] «Developers Help,» [Ηλεκτρονικό]. Available: http://stackoverflow.com/. [Πρόσβαση 12 Μαρτίου 2013].

[32] «iOS Tutorials,» [Ηλεκτρονικό]. Available: http://www.raywenderlich.com/tutorials. [Πρόσβαση 16 Φεβρουαρίου 2013].