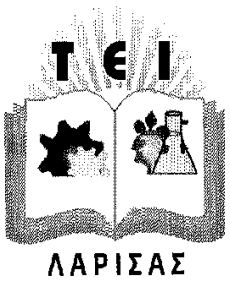


ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ ΛΑΡΙΣΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση εφαρμογής information aggregator για πληροφορίες σχετικές με το τμήμα ΤΠΤ σε πλατφόρμα Apple iOS.

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ

Λουμάνης Βασίλειος

ΕΠΙΒΛΕΠΩΝ: Χρήστος Σωμαράς

ΛΑΡΙΣΑ 2011

«Δηλώνω υπεύθυνα ότι το παρόν κείμενο αποτελεί προϊόν προσωπικής μελέτης και εργασίας και πως όλες οι πηγές που χρησιμοποιήθηκαν για τη συγγραφή της δηλώνονται σαφώς είτε στις παραπομπές είτε στη βιβλιογραφία. Γνωρίζω πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαι ενήμερος/η για* την *επέλευση των νομίμων συνεπειών»

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Τόπος, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή

2. Ονοματεπώνυμο, Υπογραφή

3. Ονοματεπώνυμο, Υπογραφή

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Στη σημερινή εποχή γινόμαστε μάρτυρες μιας τεράστιας ανάπτυξης στην πληροφορική και τις τηλεπικοινωνίες. Είναι η εποχή της επικοινωνίας και για αυτό οι άνθρωποι θέλουν όπου και αν βρίσκονται να έχουν πρόσβαση στις πληροφορίες που τους ενδιαφέρουν. Για αυτό τον λόγο υπάρχει όπως είναι αναμενόμενο μεγάλη ανάπτυξη και στις κινητές πλατφόρμες (Mobile Platforms). Συσκευές όπως κινητά τηλέφωνα, mp3 players, ταμπλέτες και άλλες φορητές συσκευές έχουν γίνει αναπόσπαστο κομμάτι της καθημερινότητας μας.

Για την δημιουργία της παρούσας πτυχιακής εργασίας χρειάστηκε αρκετή μελέτη αλλά και χρόνος που αφιερώθηκε στην έρευνα μέσω του διαδικτύου και διάφορων συγγραμμάτων. Πριν από την ενασχόλησή μου με την παρούσα πτυχιακή δεν υπήρχε προηγούμενη επαφή ούτε με το περιβάλλον εργασίας (Mac OS X Snow Leopard) ούτε με το λογισμικό ανάπτυξης εφαρμογών (Xcode) αλλά ούτε και με την γλώσσα προγραμματισμού (Objective-C). Όπως είναι φυσικό ο χρόνος που απαιτήθηκε για την εξοικείωση μου με τα παραπάνω ήταν αρκετός, και μετά από αυτό ουσιαστικά ξεκίνησε η εκπόνηση της πτυχιακής καθώς και της εφαρμογής.

Ο αναγνώστης του παρόντος συγγράμματος θα γνωρίσει τα βασικά είδη κινητών πλατφόρμων καθώς και τα νέα λειτουργικά συστήματα για αυτές. Θα μάθει τα είδη εφαρμογών που υπάρχουν στις σύγχρονες κινητές συσκευές καθώς και τις διαφορές ανάμεσα στον προγραμματισμό σε αυτές και σε αυτόν σε υπολογιστές Desktop. Στην συνέχεια θα γνωρίσει λεπτομερώς το λογισμικό ανάπτυξης εφαρμογών iOS SDK για το οποίο υπάρχει πλήρη παρουσίασή του καθώς και για την γλώσσα προγραμματισμού Objective-C. Τέλος στο μεγαλύτερο μέρος του συγγράμματος αυτού παρουσιάζεται η δημιουργία μιας εφαρμογής η οποία θα συλλέγει πληροφορίες από τους διαδικτυακούς τόπους του τμήματος ΤΠΤ (cs.teilar.gr και dionysos.teilar.gr) και θα τις παρουσιάζει μέσα από λειτουργικό template εμφάνισης που θα σχεδιαστεί και θα αναπτυχθεί ώστε να εμφανίζονται από κάποιο iDevice (Apple iPhone, iPod Touch και iPad).

ΚΕΦΑΛΑΙΟ 1ο: Κινητές πλατφόρμες

1.1 Τι ονομάζεται κινητή πλατφόρμα

Μια κινητή πλατφόρμα (επίσης γνωστή ως κινητή συσκευή, συσκευή χειρός, υπολογιστής τσέπης ή απλά κινητό) είναι ένας υπολογιστής μεγέθους τσέπης, που συνήθως αποτελείται από μια οθόνη με δυνατότητα αφής και/ή ένα μικροσκοπικό πληκτρολόγιο. Στην περίπτωση ενός προσωπικού ψηφιακού υπολογιστή (PDA) όλη η αλληλεπίδραση με τον χρήστη γίνεται μέσω μιας οθόνης αφής. Τα έξυπνα κινητά (smartphones) και τα PDAs είναι πολύ διαδεδομένα σε όσους έχουν ανάγκη την βοήθεια και την ευκολία μερικών πτυχών των κανονικών ηλεκτρονικών υπολογιστών, σε περιβάλλοντα που δεν θα ήταν δυνατό να χρησιμοποιηθούν ή να μεταφερθούν. Τέτοιες κινητές πλατφόρμες μπορούν να έχουν και μεγάλη επιχειρηματική αξία για τις επιχειρήσεις που τις χρησιμοποιούν καθώς προσφέρουν την δυνατότητα να σκανάρουν αντικείμενα όπως barcodes και έξυπνες κάρτες.

Ένας άλλος ορισμός μιας κινητής πλατφόρμας είναι ο ακόλουθος : ο γενικός όρος που αφορά μια ποικιλία συσκευών που επιτρέπει στον χρήστη να έχει πρόσβαση σε δεδομένα και πληροφορίες όπου και αν αυτός βρίσκεται. Σε αυτές τις συσκευές περιλαμβάνονται και τα κινητά τηλέφωνα και οι κινητές συσκευές.

1.2 Τα είδη κινητών πλατφόρμων

Όπως αντιλαμβανόμαστε και από τους παραπάνω ορισμούς, υπάρχουν αρκετές κατηγορίες κινητών πλατφόρμων. Ακολουθούν τα βασικά είδη τους.

1.2.1 Προσωπικός ψηφιακός βοηθός (PDA)

Ένας προσωπικός ψηφιακός βοηθός (PDA), επίσης γνωστός ως υπολογιστής παλάμης (Palmtop Computer) είναι μια κινητή πλατφόρμα που λειτουργεί σαν προσωπικός διαχειριστής πληροφοριών. Τα σημερινά PDA έχουν την δυνατότητα να συνδέονται στο διαδίκτυο και μέσω της οθόνης τους μπορούν να περιλαμβάνουν πρόγραμμα περιήγησης σε αυτό. Η σύνδεση τους επιτυγχάνεται μέσω ασύρματων δικτύων (Wi-Fi).

Εικόνα 1.2: Το PDA PalmTX

Όμως τα σημερινά μοντέλα έχουν και άλλες πολλές δυνατότητες όπως το ότι έχουν μικρόφωνο και ακουστικό αλλά και ηχείο και όλα αυτά επιτρέπουν την χρήση τους ως κινητά τηλέφωνα αλλά και φορητά ηχοσυστήματα. Επίσης τα περισσότερα χρησιμοποιούν οθόνη με τεχνολογία αφής. Το Palm TX

Ο όρος PDA χρησιμοποιήθηκε για πρώτη φορά τον Ιανουάριο του 1992 από τον πρόεδρο της εταιρίας Apple, John Sculley σε μια έκθεση ηλεκτρονικών συσκευών, την CES (Consumer Electronics Show) στο Las Vegas, και αναφερόταν στη συσκευή Apple Newton. Το Apple Newton ήταν βασισμένο πάνω σε επεξεργαστές ARM και υποστήριζε αναγνώριση γραφής με έναν ειδικό στυλό. Το 1996 η Nokia δημιούργησε το πρώτο κινητό τηλέφωνο με όλες τις λειτουργίες ενός PDA, το 9000 Communicator, το οποίο στη συνέχεια έγινε το πρώτο σε πωλήσεις παγκοσμίως. Με την είσοδο στην αγορά αυτής της συσκευής δημιουργήθηκε ουσιαστικά ο όρος PDA τηλέφωνο που στην εποχή μας ονομάζουμε Smartphone. Σήμερα όλα τα PDA είναι ουσιαστικά και smartphones.
Τα PDA χωρίς λειτουργίες τηλεφώνου περιορίζονται πια σε πολύ χαμηλές πωλήσεις και δημιουργούνται για να καλύψουν συγκεκριμένες ανάγκες κυρίως στο τομέα της βιομηχανίας.

1.2.2 Έξυπνα κινητά

Έξυπνο κινητό (Smartphone) ονομάζεται ένα κινητό τηλέφωνο που προσφέρει αρκετά περισσότερο ανεπτυγμένες πληροφοριακές δυνατότητες αλλά και συνδεσιμότητα από ότι ένα κοινό τηλέφωνο. Τα smartphones επιτρέπουν στο χρήστη να χρησιμοποιεί πολλές εφαρμογές ταυτόχρονα οι οποίες είναι φτιαγμένες έτσι ώστε να αξιοποιούν το υλικό (Hardware) της κάθε συσκευής. Τα smartphones χρησιμοποιούν ολοκληρωμένα λειτουργικά συστήματα τα οποία παρέχουν στους προγραμματιστές μια πλατφόρμα στην οποία μπορούν να αναπτύξουν τις εφαρμογές τους. Αποτέλεσμα όλων αυτών είναι να συνδυάζουν όλες της δυνατότητες ενός κινητού τηλεφώνου με κάμερα με αυτές ενός PDA.

Εικόνα 1.2: Το Smartphone HTC Desire

Σύμφωνα με έρευνες στις αρχές του 2011 τα smartphones γίνονται όλο και πιο ελκυστικά για τους καταναλωτές καθώς το 22% αυτών στο Ηνωμένο Βασίλειο ήδη έχει ένα, με αυτό το ποσοστό να αυξάνεται στο 32% στις ηλικίες 24-35 ετών. Η αύξηση στη ζήτηση για πιο ανεπτυγμένες κινητές συσκευές οι οποίες θα χρησιμοποιούν δυνατότερους επεξεργαστές, περισσότερη μνήμη, μεγαλύτερες οθόνες αλλά και πιο 'ελεύθερα' (Open Source) λειτουργικά

συστήματα έχει ως αποτέλεσμα τα smartphones να κατέχουν το μεγαλύτερο ποσοστό στην αγορά κινητών τηλεφώνων τα τελευταία χρόνια. Σύμφωνα με μια άλλη έρευνα στις Ηνωμένες Πολιτείες πάνω από 45 εκατομμύρια άνθρωποι χρησιμοποιούν smartphones από τα 234 εκατομμύρια συνδρομητών κινητής τηλεφωνίας. Αν και αυτό σημαίνει ότι μόνο το 20% των συνδρομητών, τον τελευταίο χρόνο παρατηρείται αύξηση των πωλήσεων των smartphones περίπου 74%.

1.2.3 Ταμπλέτες

Μια ταμπλέτα (Tablet PC) είναι ένας ηλεκτρονικός υπολογιστής μεγέθους συνήθως 6 έως 10 ίντσες η οποία έχει ως βασικά χαρακτηριστικά αυτά ενός κανονικού Η/Υ και συνήθως περιλαμβάνει ένα πλήρες λειτουργικό σύστημα. Ένα φορητό Tablet PC περιλαμβάνει μια οθόνη αφής ως κύρια μέθοδος εισόδου δεδομένων και είναι σχεδιασμένο έτσι ώστε να χρησιμοποιείται για προσωπική χρήση. Ο όρος Tablet PC έγινε γνωστός από μια παρουσίαση της Microsoft το 2001 όπου και χρησιμοποιήθηκε για πρώτη φορά.

Εικόνα 1.3: Η ταμπλέτα Asus T91MT

Αλλά αυτός ο όρος πια χρησιμοποιείται από μια ευρεία γκάμα μοντέλων και προϊόντων προσωπικών ηλεκτρονικών υπολογιστών αυτού του μεγέθους ακόμα και αν δεν χρησιμοποιούν το λειτουργικό σύστημα της Microsoft.

Οι ταμπλέτες χρησιμοποιούν εικονικά πληκτρολόγια και αναγνώριση γραφής έτσι ώστε να μπορούμε να εισάγουμε κείμενο, μέσω της οθόνης αφής. Όλες οι ταμπλέτες έχουν δυνατότητα ασύρματης σύνδεσης μέσω Wi-Fi, αλλά και ενσύρματα, στο διαδίκτυο. Το λογισμικό τους περιλαμβάνει εφαρμογές γραφείου, προγράμματα περιήγησης στο διαδίκτυο, παιχνίδια, αλλά από την στιγμή που τρέχουν ολοκληρωμένα λειτουργικά συστήματα, μπορούν ουσιαστικά να χρησιμοποιήσουν οποιοδήποτε πρόγραμμα υποστηρίζει το λειτουργικό τους. Βέβαια συνήθως η υπολογιστική δύναμη αυτών των συστημάτων είναι περιορισμένη, οπότε δεν είναι πάντα ικανά να αντεπεξέλθουν στους πόρους που χρειάζονται μερικές πιο απαιτητικές εφαρμογές με αποτέλεσμα να μην προσφέρουν την καλύτερη εμπειρία χρήσης. Σύμφωνα με έρευνες που πραγματοποιήθηκαν στις αρχές του 2011 οι ταμπλέτες βρίσκονται ακόμα στο στάδιο εισαγωγής τους στην αγορά αφού μόνο το 5% των Αμερικάνων έχει στην κατοχή του μια, εκ των οποίων το 3% είναι οι συσκευές iPad της Apple.

1.2.4 Apple iPhone, iPod Touch, iPad

Η σειρά προϊόντων της Apple αποτελείται από 3 προϊόντα που ως κύριο χαρακτηριστικό τους παρέχουν στο χρήστη σχεδόν όλες τις λειτουργίες που αφορούν το διαδίκτυο, και περιλαμβάνουν μια μεγάλη γκάμα εφαρμογών πολυμέσων όπως βίντεο, ήχος και φωτογραφία. Όμως ο λόγος που ξεχωρίζει στην αγορά είναι οι διάφορες καινοτομίες που λανσάρει, όπως το φιλικότερο προς τον χρήστη περιβάλλον αλλά και ένα πλήρες και ελαφρύ λειτουργικό σύστημα. Το περιβάλλον χρήστη είναι σχεδιασμένο έτσι ώστε να χρησιμοποιεί στο έπακρο 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.

Εικόνα 1.4: Η τελευταία έκδοση του iPhone, το iPhone 4

Για την εισαγωγή δεδομένων και κειμένου αντί για κουμπιά προτιμήθηκε ένα εικονικό πληκτρολόγιο το οποίο αν και στην αρχή προκάλεσε τον προβληματισμό, πια έχει γίνει το στάνταρτ στην αγορά.

Επίσης η σειρά των iDevices περιλαμβάνει 3 διαφορετικούς αισθητήρες. Ο πρώτος αισθητήρας χρησιμοποιείται για να απενεργοποιεί την οθόνη όταν ο χρήστης μιλάει στο τηλέφωνο, ο δεύτερος είναι αισθητήρας φωτός και ρυθμίζει την φωτεινότητα της οθόνης 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 απρόσκοπτη αλλαγή από κάθετη (portrait) σε οριζόντια (landscape) προβολή των δεδομένων στην οθόνη. Εφαρμογές από την ίδια την Apple αλλά και από άλλες εταιρίες λογισμικού, αναπτύσσουν λογισμικό για τις συσκευές το οποίο είναι διαθέσιμο μέσω ενός online καταστήματος, το γνωστό πια App Store, που λανσαρίστηκε στα μέσα του 2008 και τώρα πια περιλαμβάνει περισσότερες από 500.000 εφαρμογές ελεγμένες από την ίδια την Apple. Αυτές οι εφαρμογές οποιασδήποτε κατηγορίας.

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 μπορεί να χρησιμοποιηθεί πια ως βίντεο κάμερα, κινητό τηλέφωνο, να στείλει και να δεχτεί voicemail, ενώ περιλαμβάνει λειτουργίες φορητού Media player και υποστηρίζει λειτουργίες διαδικτύου όπως

πλοήγηση σε αυτό και Email μέσω του δικτύου κινητής τηλεφωνίας αλλά και μέσω Wi-Fi.

Ως ώρας έχουν υπάρξει τέσσερις γενιές μοντέλων iPhone που η κάθε μια έχει συνοδευτεί και από μια από τις τέσσερις βασικές εκδόσεις του iOS (γνωστό και ως iPhone OS). Το πρώτο iPhone ήταν ένα GSM κινητό τηλέφωνο το οποίο χρησιμοποιούσε χαρακτηριστικά, όπως το μέγεθος της οθόνης και την θέση των κουμπιών, που έμειναν σταθερά και σε όλα τα υπόλοιπα μοντέλα. Το iPhone 3G πρόσθεσε την δυνατότητα χρησιμοποίησης του δικτύου κινητής τηλεφωνίας 3G και δέκτη GPS. Το iPhone 3GS πρόσθεσε μια πυξίδα, πιο γρήγορο επεξεργαστή και κάμερα υψηλότερης ανάλυσης που περιλαμβάνει και εγγραφή βίντεο στα 480p. Το iPhone 4 πρόσθεσε μια κάμερα στο μπροστινό μέρος του κινητού για χρήση βιντεοκλήσης με προγράμματα σαν το Skype όπως και καλύτερο επεξεργαστή αλλά και οθόνη με υψηλότερη ανάλυση.

Το iPod Touch είναι ουσιαστικά ένα iPhone χωρίς τις δυνατότητες ενός κινητού τηλεφώνου. Τα βασικά χαρακτηριστικά και τον 2 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. Αν και οι εφαρμογές που χρησιμοποιεί το iPhone ως κινητό, όπως τα SMS και άλλες, περιλαμβάνονται στο λογισμικό, δεν μπορούν να χρησιμοποιηθούν και ως εκ τούτου δεν εμφανίζονται. Για κάθε νέα έκδοση του iPhone υπήρχε και μια νέα έκδοση του iPod Touch με τις ίδιες σχεδόν βελτιώσεις. Από την στιγμή που δεν συνδέεται με δίκτυα κινητής τηλεφωνίας, τα iPod Touch είναι πιο λεπτά, καταναλώνουν λιγότερη ενέργεια άρα και η μπαταρία τους κρατάει πολύ περισσότερο, και κοστίζουν λιγότερο από τα αντίστοιχα μοντέλα iPhone.

Τα iPad είναι η τελευταία προσθήκη, χρονικά, της σειράς των iDevices της Apple καθώς το πρώτο μοντέλο τους ανακοινώθηκε το χειμώνα του 2010. Παρόμοιο στη λειτουργία του με το μικρότερο iPhone ή iPod Touch, τρέχει μια τροποποιημένη έκδοση του ίδιου λειτουργικού συστήματος , με ένα επανασχεδιασμένο περιβάλλον για να εκμεταλλευτεί τη μεγαλύτερη οθόνη. Το iPad έχει μια αναδρομικά φωτισμένη οθόνη αφής πολλαπλών σημείων 9.7 ιντσών (25 εκατ.). Επειδή η κύρια λειτουργία του iPad είναι η ανάγνωση ηλεκτρονικών βιβλίων (e-books) από το νέο ηλεκτρονικό κατάστημα βιβλίων της Apple (ibookstore) το iPad έχει συγκριθεί με το Kindle της Amazon και το Barnes & Noble's Nook.

ΚΕΦΑΛΑΙΟ 2ο: Λειτουργικά συστήματα και Εφαρμογές για κινητές πλατφόρμες

2.1 Λειτουργικά συστήματα για κινητές πλατφόρμες

Ένα λειτουργικό σύστημα για κινητές πλατφόρμες (γνωστό και ως Mobile OS) είναι το λειτουργικό σύστημα που ελέγχει μια κινητή πλατφόρμα. Έχει τις ίδιες χαρακτηριστικές αρχές με ένα λειτουργικό σύστημα όπως τα Windows, τα Mac OS, τα Linux και άλλα, τα οποία ελέγχουν τους ηλεκτρονικούς υπολογιστές. Όμως αν και έχουν πολλά κοινά με τα προαναφερθέντα λειτουργικά, είναι κατά μια έννοια πιο ελαφρά, καθώς είναι φτιαγμένα ώστε να μπορούν να λειτουργούν με λιγότερους υπολογιστικούς πόρους ενώ έχουν να κάνουν περισσότερο με ασύρματες επικοινωνίες και τοπικά δίκτυα, με διαφορετικά αρχεία πολυμέσων και διαφορετικούς τρόπους εισαγωγής εντολών. Τυπικά παραδείγματα τέτοιων συσκευών που χρησιμοποιούν τέτοια λειτουργικά συστήματα είναι τα smartphones, τα pda, οι ταμπλέτες και γενικώς συσκευές που συνήθως ονομάζουμε έξυπνες συσκευές και άλλες κινητές πλατφόρμες.

2.2 Τα πρώτα λειτουργικά συστήματα

Τα λειτουργικά συστήματα που ακολουθούν θεωρούνται τα πρώτα χρονικά, για κινητές πλατφόρμες, όμως η ανάπτυξή τους έχει πλέον σταματήσει και νέα λειτουργικά συστήματα έχουν πάρει την θέση τους.

Palm OS

Ίσως το πρώτο λειτουργικό σύστημα, με όλα τα χαρακτηριστικά που ένα τέτοιο πρέπει να έχει, για κινητή πλατφόρμα, το Palm OS έκανε τα πρώτα βήματα του στην αγορά το 1996. Αρχικά αναπτύχθηκε από την Palm και χρησιμοποιήθηκε σε pda. Το Palm OS σχεδιάστηκε με βάση την ευκολία χρήσης μιας οθόνης αφής και βασίστηκε στην αλληλεπίδραση του χρήστη με αυτήν μέσω ενός γραφικού περιβάλλοντος. Προσέφερε μια σουίτα με βασικές εφαρμογές με σκοπό την διαχείριση προσωπικών πληροφοριών. Αργότερα με την πάροδο των χρόνων, νέες εκδόσεις του λειτουργικού συστήματος υποστήριξαν και smartphones.

Εικόνα 2.1: Το μοντέλο Palm m505 με την έκδοση Palm OS 4.0

Υπήρξαν διάφορες εκδόσεις μέχρι και το 2004, με κάθε μια από αυτές να προσθέτει όλο και περισσότερες λειτουργίες με σκοπό την προσαρμογή του λειτουργικού συστήματος στις νέες απαιτήσεις της αγοράς.

Windows Mobile

Τα Windows Mobile είναι ένα λειτουργικό σύστημα για κινητές πλατφόρμες που αναπτύχθηκε από την Microsoft και χρησιμοποιείται σε smartphones και pda. Στις μέρες μας έχουν αντικατασταθεί από τα Windows Phone 7. Η τελευταία έκδοση των Windows Mobile, τα Windows Mobile 6.5, είναι βασισμένα στα Windows CE και περιλαμβάνουν μια σουίτα με βασικές εφαρμογές που έχουν αναπτυχθεί από την Microsoft.

Εκτός όμως από εφαρμογές της Microsoft, τα Windows Mobile υποστηριζόντουσαν και από άλλες εταιρίες που ανέπτυξαν εφαρμογές. Ήταν σχεδιασμένα έτσι ώστε να μοιάζουν όσο το δυνατόν περισσότερο με την έκδοση για ηλεκτρονικούς υπολογιστές των Windows και από άποψη λειτουργιών αλλά και από αισθητικής πλευράς.

Εικόνα 2.2: Τα Windows Mobile στην τελευταία έκδοσή τους, την 6.5.3

Τις πρώτες μέρες που εμφανίστηκαν στην αγορά, όπου και χρησιμοποιήθηκαν σε pocket pc, οι περισσότερες συσκευές που τρέχαν Windows Mobile είχαν έναν μικρο στυλό (stylus) έτσι ώστε να είναι πιο εύκολη η χρησιμοποίηση της οθόνης αφής.
Η Microsoft το 2010 ανακοίνωσε ένα νέο λειτουργικό σύστημα για smartphones, τα Windows Phone 7, και παράλληλα σταμάτησε την υποστήριξη των κινητών τηλεφώνων που χρησιμοποιούν Windows Mobile.
Το μερίδιο που κατέχουν τα Windows Mobile στην αγορά τα τελευταία χρόνια γίνεται όλο και μικρότερο και σύμφωνα με τις τελευταίες έρευνες έχει φτάσει να είναι μόλις το πέμπτο πιο δημοφιλή κινητό λειτουργικό σύστημα με μόλις το 5% των χρηστών smartphones παγκοσμίως.

2.3 Τα λειτουργικά συστήματα νέας γενιάς

Αυτή είναι η νέα γενιά λειτουργικών συστημάτων. Τα λειτουργικά συστήματα για κινητές πλατφόρμες τα τελευταία χρόνια έχουν πραγματοποιήσει αλματώδη πρόοδο σε πολλούς τομείς όπως η ευκολία χρήσης και η υποστήριξη πολλών διαφορετικών εφαρμογών. Η αυξανόμενη

σημαντικότητα αυτών των συστημάτων στην παγκόσμια αγορά έχει καταστήσει ανταγωνιστικότατη την συγκεκριμένη κατηγορία Software, και έχει κινήσει το ενδιαφέρον ανάμεσα στους γίγαντες της ανάπτυξης λογισμικού όπως την Google, την Microsoft και την Apple, αλλά και στις εταιρίες που προηγούνται στην κατασκευή κινητών συσκευών όπως η Nokia, η Research in Motion και η Palm, με σκοπό την κατάκτηση του μεγαλύτερου μεριδίου στην αγορά.

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, μια γενιά που δίνει μεγάλη προσοχή στην ευκολία και εμπειρία χρήσης και βασίζεται στην αλληλεπίδραση του με την συσκευή μέσω της οθόνης αφής.
Τον Νοέμβριο του 2007 η Google ανακοίνωσε την δημιουργία του λειτουργικού συστήματος Android. Με το λανσάρισμα και του λειτουργικού συστήματος της Google, η αγορά των smartphones έχει αυξηθεί σημαντικά. Ενδεικτικά, τον Μάιο του 2010 τα smartphones που πουλήθηκαν έφτασαν το 17,3% της αγοράς κινητών τηλεφώνων παγκοσμίως. Αυτό οδήγησε του καταναλωτές στην γνωριμία τους με τα διάφορα λειτουργικά συστήματα και τους κατασκευαστές και τις εταιρίες τηλεπικοινωνιών στην διαφήμιση των πλεονεκτημάτων των λειτουργικών συστημάτων τους.

Σύμφωνα με πρόσφατες μελέτες παρατηρείται μια μεγάλη αύξηση στο μερίδιο του λειτουργικού Android που αυτή την στιγμή φαίνεται να κατέχει την πρώτη θέση στην αγορά των smartphones. Μέσα σε 3 χρόνια έχει καταφέρει να περάσει πρώτο, μπροστά και από το Symbian OS που κατείχε αυτή την θέση τα τελευταία χρόνια αλλά βλέπει το μερίδιο του στην αγορά να συρρικνώνεται συνεχώς στο ίδιο διάστημα. Το ακριβώς αντίθετο συμβαίνει όμως στο iOS, καθώς χρόνο με τον χρόνο αυξάνει με πιο αργούς βέβαια ρυθμούς το ποσοστό του στην αγορά από το Android. Αυτή την στιγμή φαίνεται να βρίσκεται στην τρίτη θέση, κάτι που είναι εντυπωσιακό από την στιγμή που χρησιμοποιείται μόνο από την σειρά των κινητών iPhone. Στην τέταρτη θέση, το Blackberry OS κρατάει ένα σημαντικό μερίδιο της αγοράς ενώ το νέο λειτουργικό Windows Phone 7 έχει μόλις μπει στην αγορά.

Πηγή	**Χρονιά**	**Symbian**	**Android**	**Blackberry OS**	**iOS**	**Microsoft**	**Άλλα OS**
Gartner	2011 Q1	27.4%	36.0%	12.9%	16.8%	3.6%	3.3%
Gartner	2010	37.6%	22.7%	16.0%	15.7%	4.2%	3.8%
Gartner	2009	46.9%	3.9%	19.9%	14.4%	8.7%	6.1%
Gartner	2008	52.4%	0.5%	16.6%	8.2%	11.8%	10.5%
Gartner	2007	63.5%	N/A	9.6%	2.7%	12.00%	12.10%

Αυτή η νέα γενιά λειτουργικών συστημάτων ρίχνει μεγάλο βάρος στην ανάπτυξη εφαρμογών από τρίτες εταιρίες αλλά και από ανεξάρτητους προγραμματιστές. Αυτός είναι και ο βασικός λόγος για τον οποίο όλα τα νέα λειτουργικά προμηθεύουν με τα δικά τους λογισμικά ανάπτυξης εφαρμογών οποιοδήποτε θέλει να ασχοληθεί και ενημερώνουν τους προγραμματιστές για τους τρόπους ανάπτυξης εφαρμογών σε αυτά.

Τα λειτουργικά συστήματα για κινητές πλατφόρμες είναι ακόμα στο αρχικό στάδιο της δημιουργίας τους και οποιαδήποτε πρόβλεψη για το μέλλον τους δεν είναι δυνατόν να γίνει με ακρίβεια. Όμως σύμφωνα με τα στοιχειά που υπάρχουν ως ώρας και με την τάση στην αγορά την συγκεκριμένη περίοδο, είναι δεδομένη η αύξηση του ποσοστού των πωλήσεων Smartphone συσκευών σε σε σχέση με τα απλά κινητά τηλέφωνα. Αυτή η τάση ενισχύεται και από αποφάσεις των μεγάλων εταιριών κατασκευής κινητών συσκευών, όπως αυτή της Nokia, που σύμφωνα με ανακοίνωσή της ξεκινά συνεργασία με την την Microsoft, κάτι που ουσιαστικά σταματάει και την περαιτέρω ανάπτυξη του Symbian OS, του πιο διαδεδομένου λειτουργικού συστήματος για απλά κινητά τηλέφωνα, μέχρι το τέλος του 2011, προτιμώντας την χρησιμοποίηση του νέας σειράς λειτουργικών συστημάτων Windows Phone.

Ακολουθεί μια παρουσίαση των βασικών λειτουργικών συστημάτων που κατέχουν το μεγαλύτερο μερίδιο αυτή την στιγμή καθώς και παρουσίαση των λογισμικών ανάπτυξης εφαρμογών που διαθέτουν

2.3.1 Blackberry OS (RIM)

Το Blackberry OS ως κινητό λειτουργικό σύστημα πρώτο- εμφανίστηκε το 2005. Για την ανάπτυξη του υπεύθυνη είναι η εταιρία Research in Motion και χρησιμοποιείται στα smartphones Blackberry.

Το λειτουργικό αυτό σύστημα δίνει την δυνατότητα χρήσης πολλαπλών εφαρμογών ταυτόχρονα και είναι ειδικά φτιαγμένο ώστε να υποστηρίζει συγκεκριμένες συσκευές εισόδου δεδομένων που χρησιμοποιεί η Research in Motion στα κινητά της τηλέφωνα όπως το trackwheel και το trackball.

Εικόνα 2.3: Το νέο Blackberry OS 7

Η πλατφόρμα Blackberry είναι αρκετά γνωστή για την υποστήριξή της σε εταιρικές εφαρμογές όπως Email και για αυτό το λόγο χρησιμοποιείται κυρίως σε εταιρικό επίπεδο. Αυτός ήταν εξάλλου και ο αρχικός στόχος του λειτουργικού.

Τον τελευταίο χρόνο έχει αυξήσει σε μεγάλο βαθμό την υποστήριξή του από τρίτες εταιρίες ανάπτυξης λογισμικού με αποτέλεσμα το ηλεκτρονικό κατάστημα πώλησης εφαρμογών του, το Blackberry App World να διαθέτει προς εγκατάσταση πάνω από 20.000 εφαρμογές.

Για την ανάπτυξη εφαρμογών στην συγκεκριμένη πλατφόρμα χρησιμοποιείται το IDE (intergrated development environment) Eclipse ενώ η γλώσσα προγραμματισμού είναι η Java.

2.3.2 Apple iOS

Το iOS (γνωστό και ως iPhone OS) είναι το λειτουργικό σύστημα για κινητές πλατφόρμες της Apple. Αν και αρχικά αναπτύχθηκε μόνο για το iPhone έχει από τότε επεκταθεί ώστε να υποστηρίζει και άλλες συσκευές της Apple όπως τα iPod Touch και iPad. Το συγκεκριμένο λειτουργικό σύστημα δεν υποστηρίζει άλλες συσκευές εκτός από αυτές της Apple. Ένα από τα μεγάλα πλεονεκτήματα του είναι το App Store το οποίο περιέχει περισσότερες από 500.000 εφαρμογές σύμφωνα με την τελευταία μέτρηση που έχει γίνει στα τέλη Μαΐου του 2011. Στο τελευταίο τετράμηνο του 2010 το iOS κατείχε το 16% της αγοράς των smartphones πίσω από το Google Android και το Nokia Symbian.

Εικόνα 2.4: Η κεντρική οθόνη του iOS 4.3.3 σε ένα iPhone 3GS

Το περιβάλλον χρήσης του είναι βασισμένο στην άμεση αλληλεπίδραση του χρήστη με την οθόνη αφής της συσκευής. Με αυτόν τον τρόπο ο χειρισμός γίνεται πολύ ευχάριστος, γρήγορος αλλά και απλός για τον χρήστη αφού μπορεί να αλληλεπιδρά με φυσικότητα με τα αντικείμενα που προβάλλονται στην οθόνη.
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 ζουμάρει σε μια φωτογραφία με το άνοιγμα των δυο δάκτυλών του ή μπορεί να αλλάζει φωτογραφίες με μια απλή κίνηση του δάκτυλού 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 πάνω στα Mac OS X οπότε από την φύση του είναι και βασισμένο στα UNIX. Το λειτουργικό σύστημα χρησιμοποιεί περίπου 500 MB μνήμης από τον αποθηκευτικό χώρο της συσκευής.

Το iOS, για την ανάπτυξη εφαρμογών στο περιβάλλον του, χρησιμοποιεί το λογισμικό ανάπτυξης εφαρμογών iOS SDK το οποίο αναπτύχθηκε από την Apple και δόθηκε στους

προγραμματιστές τον Φεβρουάριο του 2008. Τους δίνει την δυνατότητα να δημιουργήσουν εφαρμογές και να τις δοκιμάσουν σε ένα εξομοιωτή που ονομάζεται iPhone Simulator. Όμως για την εγκατάσταση μια εφαρμογής στη συσκευή, καθώς και για την πώληση της μέσω του App Store πρέπει ο χρήστης να είναι εγγεγραμμένος στο πρόγραμμα των προγραμματιστών iPhone που κοστίζει 99 ευρώ τον χρόνο. Ο δημιουργός μιας εφαρμογής μπορεί να την πουλήσει σε οποιαδήποτε τιμή πάνω από την μικρότερη επιτρεπτή τιμή (0.99 ευρώ) και να έχει κέρδος το 70% αυτής, με το υπόλοιπο 30% να αντιστοιχεί στο κέρδος της Apple. Εναλλακτικά, μπορεί να δίνει την εφαρμογή δωρεάν και να μην ζημιώνεται καθόλου από τα έξοδα κυκλοφορίας και διανομής, εκτός βεβαία από τα έξοδα εγγραφής.

Το iOS SDK χρησιμοποιεί τον ίδιο πρόγραμμα γραφής κώδικα που χρησιμοποιεί και το Mac OS X, το Xcode, και περιλαμβάνει και τον iPhone Simulator, ένα πρόγραμμα που μπορεί να χρησιμοποιηθεί για να εξομοιώσει το πως θα φαίνονταν οι εφαρμογές και το πως θα δούλευαν αν έτρεχαν στο iPhone, και όλα αυτά από υπολογιστή του προγραμματιστή. Το SDK της Apple έχει ως απαιτήσεις συστήματος για να χρησιμοποιηθεί, έναν Intel Mac με λειτουργικό σύστημα Mac OS X Leopard ή και νεότερο. Άλλα λειτουργικά όπως τα Windows αλλά και παλιότερες εκδόσεις Mac OS X δεν υποστηρίζονται.

Στο κεφάλαιο 3 γίνεται πλήρη παρουσίαση των περιεχομένων του iOS SDK, συμπεριλαμβανομένων και τον βασικών προγραμμάτων που χρησιμοποιούνται για την ανάπτυξη εφαρμογών σε iDevices.

2.3.3 Google Android

Το Android είναι λειτουργικό σύστημα για συσκευές κινητής τηλεφωνίας το οποίο τρέχει τον πυρήνα του λειτουργικού Linux. Αρχικά αναπτύχθηκε από την Google και αργότερα από την Open Handset Alliance. Επιτρέπει στους κατασκευαστές λογισμικού να συνθέτουν κώδικα με την χρήση της γλώσσας προγραμματισμού Java ελέγχοντας την συσκευή μέσω βιβλιοθηκών λογισμικού ανεπτυγμένων από την Google.

Τα Android αρχικά αναπτύχθηκαν από μια μικρή εταιρία λογισμικού η οποία εξαγοράστηκε από την Google.

Εικόνα 2.5: Ο προσομοιωτής Android στην αρχική οθόνη.

Η πρώτη παρουσίαση της πλατφόρμας Android έγινε τον Νοέμβριο του 2007, παράλληλα με την ανακοίνωση της ίδρυσης του οργανισμού Open Handset Alliance, μιας κοινοπραξίας 79 τηλεπικοινωνιακών εταιριών, εταιριών λογισμικού καθώς και κατασκευής hardware, οι οποίες είναι αφιερωμένες στην ανάπτυξη και εξέλιξη ανοιχτών προτύπων στις συσκευές κινητής τηλεφωνίας.

Η Google δημοσίευσε το μεγαλύτερο μέρος του κώδικα του Android υπό τους όρους της Apache License, μιας ελεύθερης άδειας λογισμικού. Μια μεγάλη κοινότητα προγραμματιστών ασχολείται με τον προγραμματισμό στο Android και με αυτό τον τρόπο αυξάνει τις δυνατότητες των συσκευών που το χρησιμοποιούν. Αυτή την στιγμή υπάρχουν πάνω από 200.000 εφαρμογές στο Android Market, το ηλεκτρονικό κατάστημα που έχει φτιάξει η Google, αν και υπάρχει και η δυνατότητα αγοράς εφαρμογών από τρίτες εταιρίες.

Από την στιγμή της εισόδου του στην αγορά το Android έχουν παρουσιάσει μια τεράστια αύξηση και στον αριθμό των συσκευών που το χρησιμοποιούν αλλά και του μεριδίου του στην αγορά, και αυτή την στιγμή θεωρείται το πιο διαδεδομένο λειτουργικό σύστημα για smartphones.

Για την ανάπτυξη εφαρμογών στο περιβάλλον του λειτουργικού χρησιμοποιείται το Android Software Development Kit το οποίο περιλαμβάνει ένα μεγάλο σετ από εργαλεία ανάπτυξης. Σε αυτό περιλαμβάνεται ένας debugger, βιβλιοθήκες, ένας εξομοιωτής, βιβλιογραφία, δείγματα κώδικα καθώς και σεμινάρια. Αυτή την στιγμή οι πλατφόρμες που υποστηρίζονται περιλαμβάνουν υπολογιστές που χρησιμοποιούν Linux (οποιαδήποτε μοντέρνα έκδοση), Mac OS X 10.4.9 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 και μέσω της γραμμής εντολών, η δημιουργία, κτίσιμο και debug εφαρμογών για Android αλλά και η δυνατότητα ελέγχου των συσκευών Android που έχουν συνδεθεί στον υπολογιστή. Με κάθε νέα έκδοση του λειτουργικού συστήματος δημιουργείται και μια νέα έκδοση του SDK, με κάθε νέα έκδοση να μην σταματάει την υποστήριξη για ανάπτυξη εφαρμογών για την προηγούμενη έκδοση του λειτουργικού.

Όμως υπάρχουν και άλλοι τρόποι δημιουργίας εφαρμογών για το Android όπως το Native Development Kit το οποίο μπορεί να συντάξει βιβλιοθήκες γραμμένες σε C και άλλες γλώσσες προγραμματισμού σε κώδικα που χρησιμοποιούν οι επεξεργαστές ARM. Μια από τις καινοτομίες της πλατφόρμας Android είναι η δημιουργία εφαρμογών με το App Invertor, ένα περιβάλλον ανάπτυξης προγραμμάτων το οποίο βασίζεται σε Web τεχνολογίες και προορίζεται για νέους προγραμματιστές. Είναι κάτι που δείχνει τα προτερήματα ενός λειτουργικού που έχει τόσο ανοικτή αρχιτεκτονική.

2.3.4 Symbian OS και Symbian Platform

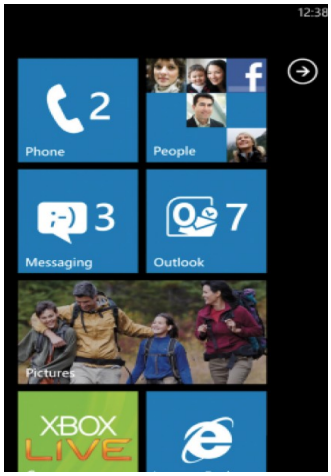
Το Symbian OS είναι λειτουργικό σύστημα για φορητές συσκευές, αποτελεί εξέλιξη του λειτουργικού συστήματος EPOC από την Psion. Το Symbian OS δημιουργήθηκε με τη γλώσσα προγραμματισμού C++ από τη Symbian Ltd. Πριν το 2009 το Symbian OS υποστήριζε διαφορετικά περιβάλλοντα χρήστη. Όμως με την δημιουργία του Symbian Platform, το ίδιο έτος, τα 3 βασικά περιβάλλοντα χρήστη ενώθηκαν σε ένα, το οποίο εξαγοράστηκε από την Nokia και στην συνεχεία μετατράπηκε σε λογισμικό ανοικτού κώδικα. Αν και οι συσκευές με λογισμικό Symbian εξακολουθούν να πωλούνται σε μεγάλους αριθμούς στην αγορά, τα τελευταία χρόνια το μερίδιο του λειτουργικού αυτού συστήματος στην αγορά μειώνεται.

Εικόνα 2.6: Η κεντρική οθόνη του Symbian OS 9 με το vHome

Για την ανάπτυξη εφαρμογών στο περιβάλλον του λειτουργικού υπάρχει το Symbian SDK το οποίο χρησιμοποιεί ως 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, ένα παλιότερο IDE που χρησιμοποιείται για ανάπτυξη εφαρμογών Symbian. Ένας εξομοιωτής χρησιμοποιείται, για τη δοκιμή των εφαρμογών, που τρέχει τον κώδικα απευθείας αντί να να προσομοιώνει την λειτουργία του κινητού τηλεφώνου.

2.3.5 Windows Phone 7

Τον Φεβρουάριο του 2010, η Microsoft ανακοίνωσε τον διάδοχο των Windows Mobile, την νέα γενιά λειτουργικών συστημάτων για κινητές πλατφόρμες, τα Windows Phone 7. Το νέο λειτουργικό σύστημα περιλαμβάνει ένα εντελώς νέο περιβάλλον χρήσης το οποίο έχει δημιουργηθεί με μια γλώσσα σχεδίασης της ίδιας της εταιρίας, που ονομάζεται Metro. Παρέχει πλήρη υποστήριξη των υπηρεσιών της Microsoft όπως το Windows Live, το Zune, το Xbox Live και το Bing, αλλά και υπηρεσιών τρίτων εταιριών όπως το Facebook και τα Google Accounts. Αν και αυτή την στιγμή το νέο λειτουργικό βρίσκεται στα πρώτα του βήματα στην αγορά, μελλοντικά, μετά την συμφωνία με την Nokia, όπου θα χρησιμοποιείται ως το βασικό λειτουργικό στα κινητά τηλέφωνα της, δείχνει να είναι ικανό να ανταγωνιστεί τα άλλα 2 μεγάλα



Εικόνα 2.7: Η κεντρική οθόνη των Windows Phone 7

λειτουργικά συστήματα, το Android και το iOS.

Για τον προγραμματισμό σε αυτή την πλατφόρμα, οι εφαρμογές πρέπει να βασίζονται ή στο XNA, ένα σετ εργαλείων της Microsoft με διαχειρίσιμο περιβάλλον ανάπτυξης εφαρμογών, ή σε μια συγκεκριμένη έκδοση του Silverlight που να υποστηρίζει τα Windows Phone 7. Για να υπάρχει η δυνατότητα σχεδίασης και δοκιμής εφαρμογών με το Visual Studio 2010, η Microsoft προσφέρει τα Windows Phone Developer Tools ως επέκταση. Αυτό το σετ εργαλείων υποστηρίζει υπολογιστές που χρησιμοποιούν Windows Vista SP2 ή νεότερα.

2.4 Εφαρμογές για κινητές πλατφόρμες

Τα νέα λειτουργικά συστήματα σίγουρα έχουν αλλάξει τον τρόπο με τον οποίο χρησιμοποιούνται τα κινητά συστήματα τα τελευταία χρονιά αλλά 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 είναι ένα μέρος της παγκόσμιας αγοράς κινητών συσκευών που μεγαλώνει και αναπτύσσεται ραγδαία. Αποτελούνται από λογισμικό που 'τρέχει' σε μια κινητή πλατφόρμα και εκτελεί συγκεκριμένες λειτουργίες για τον χρήστη του.
Αυτές οι 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, που περιλαμβάνει 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. Ανεξάρτητα με το πως οι εφαρμογές καταλήγουν στον χρήστη, οι εφαρμογές για κινητές πλατφόρμες είναι ήδη ένα μεγάλο και συνεχώς αυξανόμενο μέρος της αγοράς αυτής, και όπως είναι επόμενο, ο αριθμός των εταιριών

ανάπτυξης τέτοιων εφαρμογών αυξάνεται.

Από τεχνικής άποψης, μπορούμε να τις χωρίσουμε 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 Web/browser περιβάλλον όπως τα Webkit, Mozilla/Firefox, Opera Mini και RIM
- Άλλες πλατφόρμες και εικονικά συστήματα όπως τα Java/J2ME, BREW, Flash Lite και Silverlight

Από άποψη λειτουργιών μπορούμε να χωρίσουμε τις εφαρμογές για κινητές πλατφόρμες ως εξής:

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

Τα τελευταία χρόνια οι εφαρμογές για κινητές πλατφόρμες έχουν εξελιχθεί ως ένα σημείο που προσφέρουν στον χρήστη μια 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.

2.5 Προγραμματισμός σε κινητές πλατφόρμες.

Η ανάπτυξη εφαρμογών για κινητά είναι η διαδικασία με την οποία κατασκευάζονται εφαρμογές για χαμηλής κατανάλωσης φορητές συσκευές, όπως τα PDA και τα κινητά τηλέφωνα. Αυτές οι εφαρμογές είναι είτε προεγκατεστημένες στα τηλέφωνα κατά τη διάρκεια της κατασκευής, είτε μπορεί να τις κατεβάσει ο πελάτης από τα καταστήματα εφαρμογών και άλλες κινητές πλατφόρμες διανομής λογισμικού.

Η ανάπτυξη λογισμικού για κινητά είναι η διαδικασία της μετατροπής των υφιστάμενων λογισμικών που χρησιμοποιούνται από τους υπολογιστές σε λογισμικό το οποίο μπορεί να χρησιμοποιηθεί σε οποιαδήποτε κινητή συσκευή. Αναφέρεται επίσης στη δημιουργία του νέου λογισμικού. Ένα λογισμικό για κινητό μπορεί να αναπτυχθεί με τη χρήση διαφόρων πλατφορμών και γλωσσών προγραμματισμού με βάση τον τύπο της κινητής συσκευής. Διαφορετικές κινητές συσκευές χρησιμοποιούν διαφορετικά χαρακτηριστικά ως περιβάλλοντα χρήσης. Ως εκ τούτου, το λογισμικό τους και κινητές εφαρμογές θα πρέπει να αναπτυχθούν με διαφορετικές αρχιτεκτονικές λογισμικού που να χρησιμοποιούν τις δυνατότητες της κάθε μιας από αυτές. Η ανάπτυξη λογισμικού για κινητά είναι μια δύσκολη διαδικασία, επειδή οι χρήστες των εφαρμογών έχουν διαφορετικές προτιμήσεις. Ως εκ τούτου, οι προγραμματιστές κινητών εφαρμογών πρέπει να αναπτύξουν εφαρμογές που βασίζονται στη ζήτηση των χρηστών.

Ακολουθούν οι διαφορές και οι ιδιαιτερότητες ανάπτυξης εφαρμογών για κινητές πλατφόρμες, σε σχέση με την ανάπτυξη τους σε Desktop υπολογιστές.

Ίδια προγραμματιστικά Framework, διαφορετικά εργαλεία και υλικό

Τα τελευταία χρονιά, ο προγραμματισμός σε κινητές πλατφόρμες μεγαλώνει 10 φορές πιο γρήγορα από τον προγραμματισμό σε Desktop πλατφόρμες. Αν και η ανάπτυξη λογισμικού και στις 2 περιπτώσεις χρησιμοποιεί τα ίδια Framework όπως για παράδειγμα την C++ και το Web, τα περιβάλλον και τα εργαλεία διαφέρουν κατά πολύ. Όταν ένας προγραμματιστής σε Desktop πλατφόρμες μπαίνει στον προγραμματισμό σε κινητές πλατφόρμες, αμέσως αντιλαμβάνεται τις βασικές διαφορές ανάμεσα στα 2 αυτά περιβάλλοντα. Ένας απλός υπολογιστής σήμερα έχει το λιγότερο 1,5GHz επεξεργαστική ταχύτητα, 1GB μνήμη RAM και μερικά gigabytes σκληρό δίσκο. Από την άλλη πλευρά οι κινητές πλατφόρμες μπορούν να έχουν ακόμα και 100 MHz επεξεργαστική ταχύτητα και 300kb μνήμης. Αυτή είναι μια τεράστια διαφορά στους διαθέσιμους υπολογιστικούς πόρους ανάμεσα στα Desktop και τις κινητές πλατφόρμες. Η διαχείριση μνήμης

και η χρησιμοποίηση των πόρων πρέπει να γίνεται με προσοχή για την αποφυγή προβλημάτων.

Η κάλυψη των υπηρεσιών δικτύου δεν είναι αξιόπιστη

Μια προσωρινή έλλειψη των υπηρεσιών δικτύου μπορεί να γίνει μεγάλο πρόβλημα όταν προγραμματίζουμε σε κινητές πλατφόρμες. Όταν ο χρήσης βρίσκεται σε κίνηση και ταξιδεύει σε διάφορες περιοχές που υπάρχει προβληματική κάλυψη του δικτύου κινητής τηλεφωνίας η εφαρμογή πρέπει να μπορεί να χρησιμοποιηθεί έστω και με δεδομένα πριν την έλλειψη υπηρεσιών δικτύου, όπως και πρέπει να εκτελεστεί ξανά την στιγμή που εντοπιστεί ένα τέτοιο δίκτυο. Με άλλα λόγια ο κώδικας πρέπει να είναι προσαρμόσιμος σε τέτοιου είδους περιπτώσεις κάτι που δεν χρειάζεται να εφαρμόζεται σε προγράμματα για Desktop.

Τα πρότυπα χρησιμοποίησης είναι διαφορετικά

Οι διαφορές στον τρόπο χρησιμοποίησης ενός Desktop με μια κινητή πλατφόρμα πρέπει οπωσδήποτε να ληφθούν υπ' όψιν. Οι άνθρωποι χρησιμοποιούν τα κινητά τους στο γραφείο, κατά την διάρκεια του φαγητού, όταν παρακολουθούν τηλεόραση, όταν κάθονται με τους φίλους τους, ενώ συζητούν, ενώ ταξιδεύουν, μέσα σε πλήθος ανθρώπων, στο φως του ηλίου ή σε μέρος με πολύ φασαρία. Ο χρήστης μιας κινητής πλατφόρμας μπορεί να μην χρησιμοποιήσει ξανά την εφαρμογή για πολύ ώρα. Για όλους αυτούς τους λογούς μια εφαρμογή για μια κινητή συσκευή πρέπει να σχεδιάζεται έτσι ώστε να κάνει ένα πράγμα καλά και με τον όσο πιο δυνατόν απλό τρόπο χρήσης. Η αλληλεπίδραση του χρήστη με ένα Desktop και ένα κινητό σύστημα διαφέρει πολύ. Η οθόνη ενός κινητού τηλεφώνου είναι πολύ μικρότερη από ενός υπολογιστή. Επίσης οι τρόποι εισαγωγής δεδομένων διαφέρουν πολύ. Οι εφαρμογές για κινητά συστήματα χρησιμοποιούνται κυρίως με το ένα χέρι. Αυτοί οι παράγοντες πρέπει να λαμβάνονται υπ' όψιν κατά την σχεδίαση της εφαρμογής.

Η εξατομίκευση των εφαρμογών είναι σημαντική

Μια κινητή συσκευή είναι πάντα ανοικτή και συνδεδεμένη στο δίκτυο. Αυτές οι συσκευές προορίζονται για προσωπική χρήση και δεν μοιράζονται με άλλους χρήστες. Για αυτό μια εφαρμογή πρέπει να δίνει την δυνατότητα εύκολης προσαρμογής και εξατομίκευσης στον χρήστη. Κάποιοι χρήστες είναι πολύ επιλεκτικοί στα χρώματα και στον τρόπο που φαίνονται οι εφαρμογές που χρησιμοποιούν. Αυτό φαίνεται και από τα διάφορα μοντέλα κινητών που κατασκευάζονται και διανέμονται σε πολλά χρώματα.

ΚΕΦΑΛΑΙΟ 3ο: Προγραμματισμός στο iPhone

3.1 Το iOS SDK

Το iOS SDK είναι το προγραμματιστικό εργαλείο που προσφέρει η Apple για την ανάπτυξη εφαρμογών στις κινητές πλατφόρμες της, δηλαδή στα iPhone, iPod Touch και iPad.
Μια πρώτη γνωριμία με αυτό έχει γίνει στο υποκεφαλαιο 2.3.2. Ακολουθεί μια πιο λεπτομερή παρουσίαση των χαρακτηριστικών του καθώς και γνωριμία με τα βασικά προγραμματιστικά εργαλεία που περιέχονται σε αυτό.

3.1.1 Τα χαρακτηριστικά του iOS SDK

Το iOS χρησιμοποιεί ένα μέρος του πυρήνα XNU. Ο πυρήνας XNU αποτελεί την βάση του λειτουργικού συστήματος Mac OS X. Αρχικά αναπτύχθηκε από την εταιρία NeXT η οποία εξαγοράστηκε από την Apple και συνέχισε την ανάπτυξή του.
Το iOS χρησιμοποιεί και το ίδιο λογισμικό για την ανάπτυξη εφαρμογών για iDevices που χρησιμοποιείται στο Mac OS X, το οποίο ονομάζεται Xcode.
Το SDK χωρίζεται στα παρακάτω μέρη.

Το Cocoa Touch

Το Cocoa Touch είναι ένα API για τον σχεδιασμό σε iDevices και είναι γραμμένο στην γλώσσα προγραμματισμού Objective-C. Το Cocoa Touch χειρίζεται διάφορα μέρη των εφαρμογών όπως:

- Χειρίζεται τα γεγονότα από την χρήση πολλαπλών σημείων αφής στην οθόνη του κινητού. Για παράδειγμα με αυτόν τον τρόπο γίνεται εφικτός προγραμματισμός διαφορετικών συμπεριφορών της εφαρμογής ανάλογα με το πόσα δάκτυλα ακούμπησε ο χρήστης στην οθόνη ή αν έκανε κίνηση προς τα πάνω ή προς τα κάτω
- Τα γεγονότα στην αλλαγή κλίσης της συσκευής (Accelerometer).
- Την υποστήριξη της κάμερας από εφαρμογές
- Ελέγχει την ιεραρχία των διάφορων προβόλων (views). Κάθε φορά που πρέπει για κάποιο λόγο να αλλάξει η προβολή στοιχείων στην οθόνη και κάθε φορά που θέλουμε να προβάλλουμε κάτι διαφορετικό σε αυτήν

- Την μετατροπή των δεδομένων ανάλογα με την τοποθεσία που βρίσκεται ο χρήστης ή ανάλογα την χώρα, αλλάζει η προβολή κάποιων δεδομένων όπως η ημερομηνία κλπ.

Τα Media

Η διαχείριση των πολυμέσων γίνεται από διάφορα API τα οποία περιλαμβάνονται στο Media κομμάτι του iOS SDK. Αυτά είναι:

- Το OpenAL (open audio library) το οποίο είναι ένα λογισμικό που χρησιμοποιείται σε πολλές διαφορετικές πλατφόρμες. Είναι σχεδιασμένο έτσι ώστε να αποδίδει ποιοτικό πολυκάναλο τρισδιάστατο ήχο.
- Ελέγχει την μείξη και εγγραφή ήχου
- Υποστηρίζει την αναπαραγωγή βίντεο
- Υποστηρίζει την προβολή διαφορετικών format εικόνας
- Το Quartz, δηλαδή το γραφικό περιβάλλον της Apple το οποίο υποστηρίζει κατ την σχεδίαση δισδιάστατων γραφικών αλλά και την δημιουργία κώδικα για την υλοποίηση τους από την κάρτα γραφικών
- Το Core Animation το οποίο είναι ένα API που παράγει κινούμενα περιβάλλοντα χρήσης
- Το OpenGL ES που αποτελεί ένα μέρος του OpenGL 3D API και έχει σχεδιαστεί για κινητές πλατφόρμες. Χρησιμοποιείται για την παραγωγή δισδιάστατων και τρισδιάστατων γραφικών και αναπτύσσεται από την Khronos Group.

Τα Core Services

Τα Core Services περιέχουν API που έχουν να κάνουν με την διαχείριση του δικτύου αλλά και των δεδομένων. Αυτά είναι τα ακόλουθα:

- Οι λειτουργίες Δικτύου (Networking). Ελέγχει όλες τις λειτουργίες δικτύου από την σύνδεση με αυτό μέχρι την αποστολή και λήψη δεδομένων
- Η ενσωματωμένη βάση δεδομένων SQLite
- Το Core Location που ελέγχει όλες τις λειτουργίες εύρεσης τοποθεσίας μέσω του ενσωματωμένου GPS και των κεραιών κινητής τηλεφωνίας
- Την εκτέλεση διάφορων Threats
- Το Core Motion

Τον πυρήνα OS X

Τον πυρήνα OS X που αποτελείται από τα ακόλουθα:

- Το TCP/IP για την διασύνδεση των εφαρμογών με το διαδίκτυο
- Τα Sockets
- Την διαχείριση ενέργειας
- Το σύστημα αρχείων
- Την ασφάλεια των δεδομένων

Αυτά τα 4 μέρη αποτελούν και τα στρώματα του λειτουργικού συστήματος iOS.

3.1.2 Τα εργαλεία προγραμματισμού του iOS SDK

Τα εργαλεία που περιλαμβάνονται στο iOS SDK είναι αρκετά, χρήσιμα και εύχρηστα. Υπάρχουν εφαρμογές οι οποίες μπορούν να χρησιμοποιηθούν για την δημιουργία οποιασδήποτε εφαρμογής. Εξάλλου σύμφωνα με την ίδια την Apple είναι τα ίδια με αυτά που έχει η ίδια χρησιμοποιήσει για την δημιουργία όλων των εφαρμογών της.

Σε αυτό το υποκεφαλαιο θα παρουσιαστούν λεπτομερώς τα 3 βασικά εργαλεία ανάπτυξης εφαρμογών, το Xcode, ο Interface Builder καθώς και ο iPhone Simulator, δηλαδή αυτά που θα χρησιμοποιήσουμε και για την εφαρμογή που θα δημιουργήσουμε στο επόμενο κεφάλαιο.

Το Xcode

Στο Xcode βρίσκεται ουσιαστικά όλος ο κώδικας της εφαρμογής μας. Εδώ είναι το μέρος που δημιουργούμε τις κλάσεις του προγράμματος μας, δηλώνουμε τις μεταβλητές, και γενικότερα προγραμματίζουμε όλη την συμπεριφορά της εφαρμογής μας. Για να ξεκινήσουμε την δημιουργία μιας εφαρμογής πρέπει πρώτα να ξεκινήσουμε τον Xcode.

Το πρώτο πράγμα που βλέπουμε όταν ανοίγουμε τον Xcode είναι ένα παράθυρο που μας καλωσορίζει στο πρόγραμμα. Από εδώ μπορούμε να διαλέξουμε αν θέλουμε να ξεκινήσουμε μια νέα εφαρμογή, εάν θέλουμε να ανοίξουμε κάποια πρόσφατη εφαρμογή που έχουμε ξεκινήσει ή αν θέλουμε βοήθεια στην εκμάθηση του προγράμματος, η οποία είναι και πολύ χρήσιμη για όσους δεν έχουν ξαναχρησιμοποιήσει τον Xcode. Εάν επιλέξουμε τη δημιουργία μιας νέας εφαρμογής τότε θα μας ζητηθεί να επιλέξουμε το πρότυπο που θα χρησιμοποιήσουμε και αργότερα θα πρέπει να την

ονομάσουμε.

Τα πρότυπα είναι στην ουσία έτοιμες κλάσεις που μας δίνει απευθείας το Xcode ανάλογα με το τι εφαρμογή θέλουμε να δημιουργήσουμε. Υπάρχουν 6 διαφορετικά πρότυπα για προγραμματισμό στο iPhone:

- Το πρότυπο πλοήγησης (Navigation-based) το οποίο μας δίνει τις βασικές κλάσεις και τον κώδικα για να μας διευκολύνει στην δημιουργία μιας εφαρμογής που σκοπό θα έχει την πλοήγηση από τη μια προβολή δεχόμενων στην επόμενη
- Το πρότυπο OpenGL ES που μας δίνει ένα αρχικό στάδιο μιας εφαρμογής που χρησιμοποιεί το OpenGL στην αρχική προβολή της.
- Το πρότυπο με μπάρα επιλόγων (Tab Bar) έτσι ώστε να δημιουργήσουμε μια εφαρμογή που να εναλλάσσεται σε πολλές προβολές μεσώ μιας μπάρας επιλογής στο κάτω μέρος της εφαρμογής
- Το πρότυπο μιας
- Το πρότυπο μιας προβολής που δημιουργεί μια κλάση και με μια κεντρική προβολή
- Το πρότυπο μιας εφαρμογής παραθύρου που απλά δίνει τα πιο βασικά αρχεία για την δημιουργία μιας οποιασδήποτε εφαρμογής

Αφού επιλέξουμε κάποιο πρότυπο και ονομάσουμε την εφαρμογή μας το Xcode θα δημιουργήσει τις κατάλληλες κλάσεις και θα προσθέσει τα αρχεία που χρειάζονται καθώς και τα Framework έτσι ώστε να τρέχει σωστά.

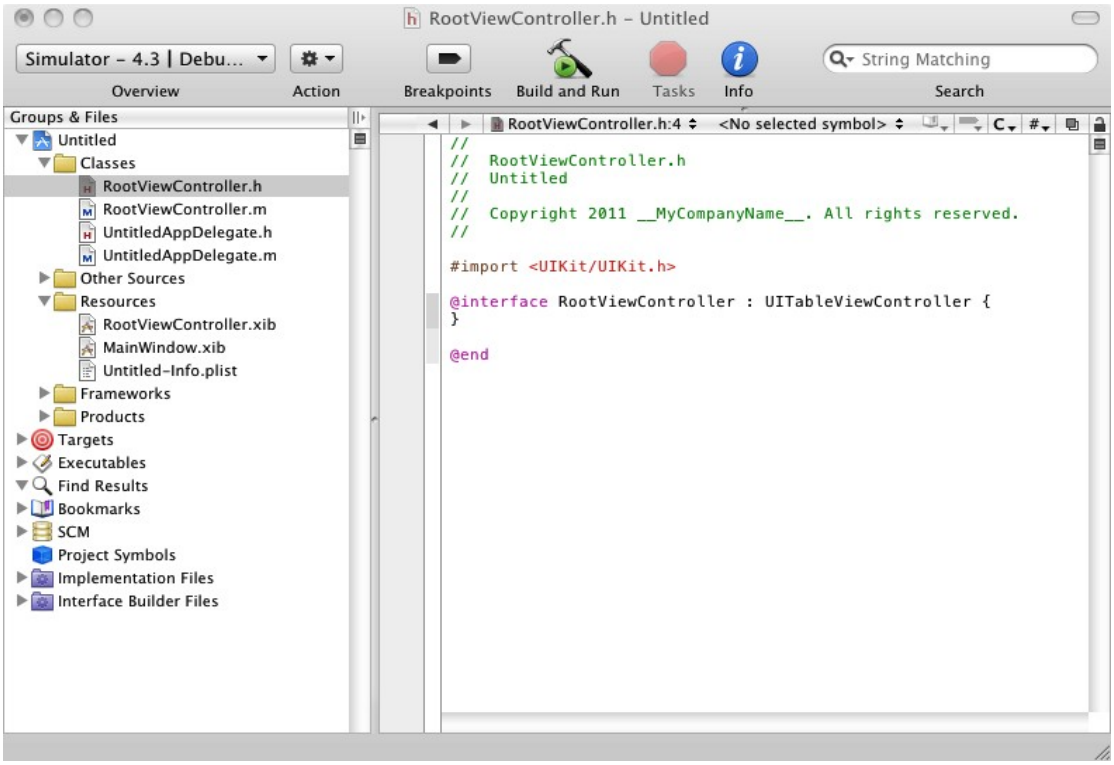
Στο αριστερό μέρος του παραθύρου μπορούμε να διακρίνουμε το υπό-παράθυρο με τίτλο Groups & Files. Εδώ μπορούμε να δούμε τις κλάσεις, πατώντας στο μικρο βελάκι δίπλα από την λέξη Classes, που χρησιμοποιεί η εφαρμογή μας, και αν επιλέξουμε κάποια από αυτές τότε μπορούμε να προσθέσουμε κώδικα στο δεξιό μέρος του παραθύρου μας οπού και προβάλει τα περιεχόμενα της. Ένας άλλος πολύ σημαντικός φάκελος, που μπορούμε να δούμε τα περιεχόμενα του με τον ίδιο τρόπο, είναι ο φάκελος Resources, που περιέχει την προβολή της εφαρμογής μας. Είναι το αρχείο με τύπο .xib. Πρόκειται για αρχείο που ανοίγει προς επεξεργασία μέσω του δεύτερου προγράμματος που θα παρουσιάσουμε στη συνέχεια, του Interface Builder. Όμως στο φάκελο αυτό υπάρχει και ένα άλλο σημαντικό αρχείο, το -info.plist, το οποίο περιέχει μερικές χρήσιμες ρυθμίσεις για την εφαρμογή μας όπως ποιο θα είναι το όνομα της εφαρμογής όταν εγκαθίσταται στη συσκευή αλλά και ποιο θα είναι το εικονίδιο της.

Στη περίπτωση που θέλουμε να δημιουργήσουμε μια νέα κλάση, τότε θα πρέπει να πατήσουμε δεξί κλικ πάνω στον φάκελο που θέλουμε να την προσθέσουμε (συνήθως στον φάκελο

Classes) και να επιλέξουμε Add και στη συνέχεια New File. Θα εμφανιστεί ένα νέο παράθυρο που μας δίνει διάφορα πρότυπα ανάλογα με το τι ακριβώς θέλουμε να κάνει η νέα κλάση μας.

Στο πάνω αριστερό μέρος του παραθύρου βλέπουμε ένα αναδυόμενο μενού στο οποίο αναγράφεται η λέξη Simulator. Μέσω αυτού του μενού μπορούμε να διαλέξουμε που θέλουμε να τρέξουμε το πρόγραμμα μας, ανάμεσα σε συσκευή που είναι συνδεδεμένη με τον υπολογιστή μας ή στο iPhone Simulator. Με το κουμπί, στη μέση και πάνω της εφαρμογής, που ονομάζεται Build and Run μπορούμε να δοκιμάσουμε την εφαρμογή μας. Εάν επιλέξαμε τον εξομοιωτή, τότε θα τον δούμε να προσπαθεί να τρέξει την εφαρμογή μας.

Το Xcode έχει αρκετά εργαλεία που μπορούν να μας βοηθήσουν στο να γράψουμε τον κώδικά μας. Το μενού Help μπορεί να μας δώσει χρήσιμες πληροφορίες για τις μεθόδους που μπορούμε να χρησιμοποιήσουμε ενώ υπάρχει και η επιλογή να πατήσουμε το κουμπί Command και να κάνουμε διπλό κλικ σε οποιαδήποτε λέξη κλειδί ή μέθοδο και να δούμε την δήλωσή της στις βιβλιοθήκες της Apple. Επίσης εμφανίζει με κόκκινη γραμματοσειρά τις λέξεις κλειδιά αλλά και τις μεθόδους και με ανοικτό μπλε τις μεταβλητές, ενώ ανάλογα με το τι ξεκινούμε να γράφουμε, μας προσφέρει αυτόματη συμπλήρωση λέξεων σε πολλές περιπτώσεις.



Εικόνα 3.1 : Το περιβάλλον χρήσης του Xcode

Ο Interface Builder

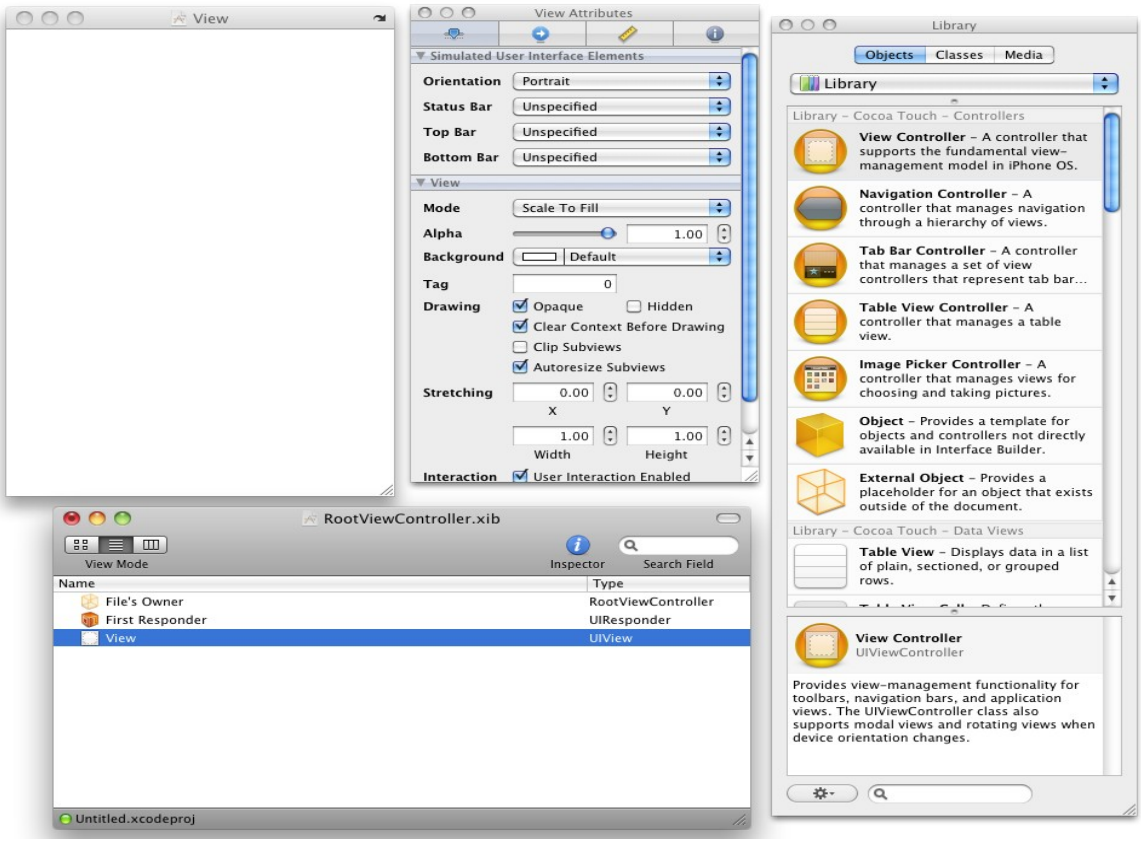
Για να ξεκινήσουμε τον Interface Builder, πρέπει να κάνουμε διπλό κλικ στο αρχείο με τύπο .xib που συνήθως βρίσκεται στον φάκελο Resources του υπό-παραθύρου Groups & Files του Xcode. Ο Interface Builder είναι ένα πρόγραμμα που ασχολείται με την εμφάνιση της εφαρμογής μας στον χρήστη. Μέσω αυτού μπορούμε να τοποθετήσουμε κουμπιά, τίτλους, μονοδιάστατους πίνακες, εικόνες γενικότερα οτιδήποτε υπάρχει ως περιβάλλον χρήσης στο iOS. Στην συνέχεια πρέπει να τα συνδέσουμε με κάποια μέθοδο που έχουμε γράψει στον Xcode έτσι ώστε να δέχονται τα δεδομένα που στη συνέχεια θα προβάλλουν στην οθόνη με τον τρόπο που εμείς τα τοποθετήσαμε.

Ο Interface Builder αποτελείται από τέσσερα διαφορετικά παράθυρα. Το πιο βασικό είναι το κεντρικό παράθυρο, το οποίο περιέχει το όνομα του αρχείου που έχουμε ανοίξει. Σε αυτό το παράθυρο μπορούμε να δούμε τα αντικείμενα που περιέχει η προβολή μας. Το αντικείμενο File's Owner περιέχει όλες τις μεθόδους που έχουμε περάσει στην κλάση με την οποία συνδέεται η προβολή μας και μέσω αυτού μπορούμε να συνδέσουμε τα αντικείμενα που χρησιμοποιούμε στο παράθυρο προβολής με τις αντίστοιχες μεθόδους. Το επόμενο παράθυρο περιέχει την προβολή μας. Εδώ μπορούμε να προσθέσουμε τα αντικείμενα και να τα τοποθετήσουμε στη θέση που θέλουμε. Αυτά τα αντικείμενα μπορούμε να τα επιλέξουμε από το τρίτο παράθυρο που ονομάζεται Library. Εκεί υπάρχουν όλα τα αντικείμενα που μπορούμε να χρησιμοποιήσουμε ομαδοποιημένα με διάφορους τρόπους. Μπορούμε να τα τοποθετήσουμε στο παράθυρο προβολής κάνοντας τα drag and drop. Το τελευταίο παράθυρο είναι ο Inspector. Μέσω αυτού μπορούμε να δούμε τις ιδιότητες των αντικειμένων τις οποίες μπορούμε να αλλάξουμε. Χωρίζεται σε τέσσερα μέρη: το Attributes με το οποίο μπορούμε να αλλάξουμε ιδιότητες που είναι διαφορετικές ανάλογα το αντικείμενο που έχουμε επιλέξει, το Connections που μας δίνει τις συνδέσεις του αντικείμενου με τις μεθόδους, το Size που ελέγχει το μέγεθος του παράθυρου και την τοποθέτηση του στην προβολή και τέλος το Identity όπου μπορούμε να αλλάξουμε την κλάση από την οποία το αντικείμενο κληρονομεί τις ιδιότητές του.

Ο iPhone Simulator

Οποιαδήποτε στιγμή θέλουμε να δοκιμάσουμε την εφαρμογή μας, μπορούμε να πατήσουμε το κουμπί Build and Run και να προσωμοιώσουμ την λειτουργία της εφαρμογής μας στον Simulator. Από εκεί μπορούμε χρησιμοποιήσουμε το ποντίκι μας όπως θα χρησιμοποιούσαμε τα δάκτυλά μας στην συσκευή μας. Πατώντας το Option μπορούμε να προσημειώσουμε τις κινήσεις

pitch in και out των δακτύλων μας όταν θέλουμε να κάνουμε zoom-in και zoom-out ενώ πατώντας το Command και τα κουμπιά με τα βελάκια μπορούμε να αλλάξουμε την γωνία της συσκευής κατά 90 μοίρες ώστε να δούμε πως θα φαίνεται η εφαρμογή μας σε οριζόντια (landscape) προβολή.



Εικόνα 3.2 : Το περιβάλλον χρήσης του Interface Builder

3.2 Η γλώσσα προγραμματισμού Objective-C

Η Objective-C είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού. Είναι ένα παρακλάδι της C γλώσσας, παρέχοντας κατασκευαστές που μας επιτρέπουν να ορίζουμε κλάσεις και αντικείμενα.

Ένα από τα βασικά χαρακτηριστικά της είναι ότι είναι φτιαγμένη έτσι ώστε να είναι προτιμότερη η χρήση, πολλών μεν αλλά μικρότερων σε περιεχόμενο, κλάσεων. Από την στιγμή που, κάποιος που έχει γνώσεις σε αντικειμενοστραφής γλώσσες, αντιληφθεί αυτόν τον τρόπο λειτουργίας της τότε όλα θα του φαίνονται αρκετά οικεία. Όμως υπάρχουν και κάποιες διαφορές στις οποίες θα αναφερθούμε σε αυτό το υποκεφάλαιο. Μια από τις πιο σημαντικές, ειδικά αν προέρχεται κάποιος από προγραμματισμό Java, είναι το πως η objective-C χειρίζεται την την διαχείριση μνήμης.

3.2.1 Δήλωση και ορισμός κλάσεων

Όπως ισχύει και σε σχεδόν όλες τις αντικειμενοστραφείς γλώσσες προγραμματισμού, στην Objective-C οι κλάσεις παρέχουν τα θεμέλια που επιτρέπουν την λήψη δεδομένων και την χρησιμοποίηση μεθόδων για την επεξεργασία τους. Τα αντικείμενα είναι συγκεκριμένες καταστάσεις μιας κλάσης και περιέχουν τα δικά τους δεδομένα ανά περίπτωση αλλά και τους δείκτες (pointers) οι οποίοι δείχνουν στις μεθόδους που περιλαμβάνονται σε κάθε κλάση. Οι κλάσεις χωρίζονται σε 2 κατηγορίες: το Interface (διεπαφή ή προβολή δεδομένων) και το implementation (εφαρμογή). Το Interface περιέχει τη δήλωση της κλάσης και συνήθως βρίσκεται σε ένα αρχείο που έχει κατάληξη .h. Το implementation περιέχει τον κώδικα που καθορίζει μια κλάση και συνήθως βρίσκεται σε ένα αρχείο που έχει κατάληξη .m. Ας τα δούμε πιο αναλυτικά.

Δήλωση μιας κλάσης

Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια κλάση η οποία θα δέχεται μια εντολή από τον χρήστη και θα τυπώνει στην οθόνη “Hello World”. Θα πρέπει κατ' αρχήν να δηλώσουμε αυτήν την κλάση. Η δήλωση της ξεκινάει με την λέξη κλειδί @interface ακολουθούμενη από το όνομα της κλάσης που δηλώνουμε και στη συνέχεια με το σύμβολο “:” το οποίο ακολουθείται από το όνομα της κλάσης από την οποία κληρονομεί:

```
@interface HelloWorldViewController : UIViewController
```

Μια κλάση στην Objective-C δεν μπορεί να κληρονομήσει από πολλές κλάσεις αλλά η κλάση από την οποία κληρονομεί μπορεί με την σειρά της να κληρονομεί από μια άλλη. Στη δική μας περίπτωση η κλάση μας HelloWorldViewController κληρονομεί από την UIViewController η οποία και αυτή με την σειρά της κληρονομεί από την UIResponder, η οποία τελικά κληρονομεί από την NSObject, που είναι η κλάση ρίζα στις περισσότερες ιεραρχίες κλάσεων στην Objective-C.
Μετά από αυτή την πρώτη γραμμή οι δηλώσεις των μεταβλητών βρίσκονται μέσα σε αγκύλες. Μετά ακολουθούν οι δηλώσεις των μεθόδων και των properties που σχετίζονται με την κλάση. Τα properties είναι ένας εύκολος τρόπος να δημιουργούμε τα set και τα get για κάθε μεταβλητή.
Τέλος κλείνουμε την δήλωση της κλάσης μας με την λέξη κλειδί @end:

```
#import <UIKit/UIKit.h>

@interface HelloWorldViewController : UIViewController {
```

```
    UIButton *button;
    UILabel *label;
}
@property (nonatomic, retain) IBOutlet UILabel *label;
-(IBAction)sayHello:(id) sender;

@end
```

Καθορίζοντας το περιεχόμενο μιας κλάσης στο αρχείο Implementation

Συνεχίζοντας με το ίδιο παράδειγμα, πρέπει πια να υλοποιήσουμε με κώδικα ότι έχουμε δηλώσει στο αρχείο .h. Η εφαρμογή του κώδικά μας ξεκινάει με την λέξη κλειδί @implementation και τελειώνει με την λέξη κλειδί @end:

```
@implementation HelloWorldViewController
...
@end
```

Στην αρχή πρέπει να χρησιμοποιήσουμε την λέξη κλειδί @synthesize έτσι ώστε να δημιουργηθούν για μας τα set και get των properties και στην συνέχεια να εφαρμόσουμε τις μεθόδους που δηλώσαμε παραπάνω:

```
#import "HelloWorldViewController.h"

@implementation HelloWorldViewController

@synthesize label;

-(IBAction) sayHello:(id) sender {

    label.text = @"Hello World";
}

- (void)didReceiveMemoryWarning {
```

```
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {
}

 (void)dealloc {
    [label release];
    [button release];
    [super dealloc];
}

@end
```

Τώρα που ρίξαμε μια γρήγορη μάτια στη δομή μιας δήλωσης και μιας εφαρμογής μιας κλάσης, ας δούμε με περισσότερες λεπτομέρειες τα κάθε μέρη της ξεχωριστά

Δήλωση αντικειμένων

Όταν οι ίδιες οι μεταβλητές είναι αντικείμενα, παραδείγματος χάριν όταν η κλάση HelloWorldViewController δηλώνει μια UIButton και μια UILabel μεταβλητή, πρέπει πάντα να χρησιμοποιείται ένας τύπος δείκτη. Όμως με την Objective-C συμβαίνει κάτι ενδιαφέρον: υποστηρίζει και σαφείς δηλώσεις και ασαφείς δηλώσεις αντικείμενων. Ακολουθεί μια σαφή δήλωση:

```
UIButton *button;
```

Εδώ δηλώνουμε το είδος του αντικείμενου, δηλώνοντας ξεκάθαρα τι είναι.

Παρακάτω όμως κάνουμε μια ασαφή δήλωση, δηλώνοντας ότι το αντικείμενο είναι ένα id:

```
id button
```

Ο τύπος id είναι ένας γενικός C τύπος τον οποίο η Objective-C χρησιμοποιεί για να δείξει ένα αυθαίρετο αντικείμενο. Είναι ένας γενικός τύπος που αντιπροσωπεύει οποιοδήποτε αντικείμενο

ανεξαρτήτως κλάσης και μπορεί να χρησιμοποιηθεί ως τρόπος αποθήκευσης τόσο μιας κλάσης όσο και μιας αναφοράς προς ένα αντικείμενο. Όλα τα αντικείμενα οπότε είναι τύπου id. Αυτό μπορεί να αποδειχτεί πολύ χρήσιμο καθώς αν ας πούμε θέλαμε να φτιάξουμε μια γενική κλάση που θα περιείχε μια συνδεδεμένη λίστα, ο τύπος κάθε στοιχείου της θα μπορούσε να είναι τύπου id και έτσι θα μπορούσαμε να αποθηκεύσουμε σε αυτήν οποιοδήποτε τύπου αντικείμενα.

Properties

Η δήλωση των properties χρησιμοποιώντας την λέξη κλειδί @property είναι ένας εύχρηστος τρόπος να αποφύγουμε την δήλωση και συνήθως και την εφαρμογή των μεθόδων προσπέλασης για τις μεταβλητές-μελή μιας κλάσης. Μπορούμε να θεωρήσουμε την δήλωση ενός property ως αντίστοιχη της δήλωσης μεθόδων Set και Get. Μπορούμε επίσης να δηλώσουμε πως οι αυτοματοποιημένες αυτές μέθοδοι θα συμπεριφέρονται, δηλώνοντας συγκεκριμένες ιδιότητες. Στη κλάση HelloWorldViewController δηλώσαμε το property ως (nonatomic, retain):

@property (nonatomic, retain) IBOutlet UILabel *label;

Μπορούμε επίσης να δηλώσουμε και τα 2 properties ως IBOutlet. Αν και στην πραγματικότητα δεν είναι μέρος των ιδιοτήτων για μια δήλωση property, το IBOutlet δηλώνει ότι αυτό το αντικείμενο είναι ένα στοιχείο το οποίο μπορούμε να το συνδέσουμε με άλλα μέσω του Interface Builder.

Synthesize

Όταν δηλώνουμε ένα @property στο αρχείο .h, πρέπει επίσης να χρησιμοποιούμε το @synthesize στο αρχείο .m, εκτός αν θέλουμε να υλοποιήσουμε εμείς τα set και get, όπως κάναμε και για το property label της κλάσης HelloWorldViewController:

@synthesize label;
Με αυτό τον τρόπο ο compiler δημιουργεί τις μεθόδους προσπέλασης σύμφωνα με τις ιδιότητες που δηλώσαμε στο property και έτσι μειώνει σημαντικά τον κώδικα που πρέπει να γράψουμε εμείς.

Η σύνταξη με την χρήση τελείας

Όταν δηλώνουμε μια μεταβλητή-μέλος ως property και στην συνεχεία κάνουμε synthesize, μπορούμε να χρησιμοποιήσουμε κάποιες ευκολίες που η Objective-C προσφέρει και οι οποίες ονομάζονται σύνταξη με χρήση τελείας (the dot syntax) αντί να χρησιμοποιήσουμε τις μεθόδους άμεσα.
Για παράδειγμα μας επιτρέπει να γράψουμε το ακόλουθο:

```
label.text = @”hello World”
```

αντί για αυτό (προσέξτε ότι η Objective-C έγραψε με κεφαλαίο το t της λέξης text όταν δημιούργησε την μέθοδο):

```
[label setText:@”Hello World”];
```

Ο πρώτος τρόπος γραφής είναι σίγουρα πιο απλός και ευκολοδιάβαστος.

Δηλώνοντας Μεθόδους

Δηλώσαμε μια μέθοδο στην κλάση HelloWorldViewController την οποία ονομάσαμε sayHello:

```
#import <UIKit/UIKit.h>

@interface HelloWorldViewController : UIViewController {
    UILabel     *label;
    UIButton *button;
}

@property (nonatomic, retain) IBOutlet UILabel *label;
-(IBAction)sayHello:(id) sender;

@end
```

Το μείον (-) στην αρχή της δήλωσης της μεθοδου δείχνει τον τύπο της, που σε αυτή την περίπτωση είναι μια κατάσταση. Ένα συν (+) ορίζει μια κλάση-μέθοδο όπως παρακάτω:

```
+(void)aMethod:(id) anObject;
```

Η μέθοδος sayHello: παίρνει ένα αντικείμενο id ως είσοδο και σημειώνεται ως IBAction για το Interface Builder. Όταν περάσει από τον compiler το IBAction αντικαταστάτρια από ένα void όπως επίσης το ίδιο συμβαίνει στο IBOutlet αφού και τα 2 αυτά αντικείμενα χρησιμοποιούνται για να δείχνουν στο Interface Builder ότι μπορούν να συνδεθούν. Αυτή η μέθοδος δέχεται ως είσοδο ένα αντικείμενο id αφού έχουμε ως σκοπό να εκτελείται όταν κάνει κάτι ο χρήστης, και θέλουμε να μην περιοριζόμαστε ως προς το τι ακριβώς γεγονός θα είναι αυτό. Στο περιβάλλον χρήστη θέλουμε ένα κουμπί που όταν ο χρήστης το πατήσει να εκτελεστεί η μέθοδος οπότε στη προκειμένη περίπτωση το id θα είναι ένα UIButton που ο χρήστης χρησιμοποίησε ώστε να τρέξει η μέθοδος. Μπορούμε να ανακτήσουμε το αντικείμενο UIButton χρησιμοποιώντας το αντικείμενο sender σε ένα UIButton:

```
UIButton theButton = (UIButton *) sender;
```

Είναι συνηθισμένη πρακτική στην Objective-C να καλούμε τέτοια αντικείμενα sender. Εάν δεν είμαστε σίγουροι το τι τύπος είναι πραγματικά το αντικείμενο που έχει τύπο id τότε μπορούμε να μάθουμε χρησιμοποιώντας την μέθοδο isKindOfClass:

```
if([thisObject isKindOfClass:[anotherObject class]]) { ... }
```

Καλώντας Μεθόδους

Εάν θέλουμε να καλέσουμε μια μέθοδο μέσω ενός αντικείμενου, μπορούμε να το κάνουμε αυτό στέλνοντας ένα μήνυμα σε αυτό το αντικείμενο. Το μήνυμα πρέπει να περιέχει το όνομα της μεθοδου όπως επίσης και διάφορες παραμέτρους της. Τα μηνύματα πρέπει να κλείνονται από παρενθέσεις τύπου []. Το αντικείμενο που δέχεται το μήνυμα βρίσκεται στα αριστερά και οι παράμετροι στα δεξιά, με την παράμετρο να ακολουθείται από άνω κάτω τελείες. Εάν η μέθοδος δέχεται περισσότερα από ένα δεδομένα, είναι υποχρεωτικό να χρησιμοποιηθούν και αυτά, έτσι οι επόμενες παράμετροι ακολουθούνται και αυτοί με άνω κάτω τελείες. Αυτό επιτρέπει πολλαπλές μεθόδους με το ίδιο όνομα και τύπους δεδομένων να δηλώνονται.

```
[anObject someMethod];
[anObject someMethod: anotherObject];
[anObject someMethod: anotherObject withAnotherArgument: yetAnotherObject];
```

Το όνομα της μεθόδου είναι η αλληλουχία του ονόματος της μεθοδου και οποιονδήποτε

ονομαζόμενων δεδομένων. Έτσι στον κώδικα που προηγήθηκε έχουμε 2 μεθόδους με ονόματα someMethod: και someMethod:withAnotherArgument:. Αυτό ίσως να φαίνεται λίγο περίεργο σε κάποιους που έρχονται με γνώσεις από άλλες γλώσσες που συνήθως έχουν διαφορετικούς τρόπους ονομασίας, αλλά σε γενικές γραμμές οι ονομασίες στην Objective-C είναι αρκετά πιο επεξηγηματικές ως το τι κάνουν σε σχέση με άλλες γλώσσες. Οι ονομασίες των μεθόδων μπορούν σχεδόν να διαβαστούν σαν κανονικές προτάσεις.

Οι μέθοδοι μπορούν να επιστρέψουν το αποτέλεσμα τους όπως φαίνεται παρακάτω:

output = [anObject someMethodWithOutput: anotherObject];

Και μπορούν να χρησιμοποιούνται φωλιασμένες:

output = [anObject someMethodWithOutput: [anotherObject someOtherMethod]];

Καλώντας Μεθόδους με nil

Στην Objective-C το αντικείμενο nil είναι σε λειτουργία ισάξιο με τον δείκτη NULL που υπάρχει σε άλλες C γλώσσες. Όμως αντίθετα από ότι σε περισσότερες από αυτές, είναι δυνατόν να καλέσουμε μεθόδους με nil χωρίς να καταρρεύσει η εφαρμογή μας. Εάν καλέσουμε μια μέθοδο (αν και όπως είπαμε στην objective-C στη πραγματικότητα περνάμε ένα μήνυμα) με αντικείμενο το τύπο nil, τότε θα μας επιστρέψει ένα nil.

3.2.2 Διαχείριση μνήμης

Ο τρόπος με τον οποίο η Objective-C στο iPhone διαχειρίζεται την μνήμη μάλλον θα είναι κάτι που δεν είναι ιδιαίτερα γνώριμο αν προερχόμαστε από μια γλώσσα σαν την Java. Στο iPhone είμαστε περιορισμένοι να χρησιμοποιούμε κάτι που ονομάζεται αναφορά μέτρησης (Reference Counting). Αυτό δεν είναι τόσο περίπλοκο όσο ίσος φαίνεται, και αν ακολουθούμε μερικούς απλούς κανόνες, τότε θα μπορούμε να διαχειριζόμαστε την μνήμη που διαθέτουμε (allocate).

Δημιουργώντας Αντικείμενα

Μπορούμε να δημιουργήσουμε ένα αντικείμενο με 2 τρόπους. Όπως φαίνεται και στον

κώδικα που ακολουθεί, μπορούμε να διαθέσουμε μνήμη απευθείας με την λέξη alloc και να την αρχικοποιήσουν με την λέξη init ή την κατάλληλη initWith μέθοδο (π.χ. Ένα αντικείμενο NSString χρησιμοποιεί την μέθοδο initWithString):

```
NSString *string = [[NSString alloc] init];
NSString *string = [[NSString alloc] initWithString:@"This is a string"];
```

Εναλλακτικά μπορούμε να χρησιμοποιήσουμε μια μέθοδο-κατασκευαστή. Για παράδειγμα η κλάση NSString έχει μια μέθοδο stringWithString η οποία επιστρέφει ένα αντικείμενο NSString:

```
NSString *string = [NSString stringWithString:@"This is a string"];
```

Με τους 2 πρώτους τρόπους είμαστε υποχρεωμένοι να απελευθερώσουμε την μνήμη που διαθέσαμε με την λέξη alloc. Εάν δημιουργήσουμε ένα αντικείμενο με το alloc, πρέπει να τον απελευθερώσουμε (release) μετά. Αντιθέτως με τον τρίτο τρόπο το αντικείμενο θα απελευθερώσει την μνήμη που χρησιμοποιεί αυτόματα (autorelease). Δεν πρέπει ποτέ να απελευθερώνουμε μόνοι μας ένα τέτοιο αντικείμενο καθώς κάτι τέτοιο θα προκαλέσει κατάρρευση της εφαρμογής μας. Ένα τέτοιο αντικείμενο, τις περισσότερες φορές, θα απελευθερωθεί στο τέλος της χρήσης του εκτός εάν έχουμε χρησιμοποιήσει την λέξη retain (διατήρηση).

Αυτόματη απελευθέρωση μνήμης

Αυτόματη απελευθέρωση μνήμης (autorelease pool) ονομάζουμε την διαδικασία κατά την οποία στέλνεται ένα μήνυμα απελευθέρωσης μνήμης σε ένα αντικείμενο, αφού αυτό πρώτα έχει παύσει να χρησιμοποιείται από την μέθοδο από την οποία δημιουργήθηκε. Κάθε αντικείμενο που δημιουργείται με αυτόν τον τρόπο αποθηκεύεται μαζί με άλλα τέτοιου είδους αντικείμενα σε ένα συγκεκριμένο μέρος (pool) και με το πέρας της χρησιμοποίησης όλων αυτών των στοιχείων, όλα μαζί απελευθερώνονται. Όλες οι εφαρμογές iPhone είναι υποχρεωμένες να έχουν μια autorelease pool και για αυτό ο Xcode δημιουργεί για μας μια μέσα στο αρχείο main.m:

```
int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
```

}

Η autorelease pool δημιουργείται πριν μπει στην κεντρικό βρόγχο και στη συνέχεια απελευθερώνει την μνήμη αφού έχει βγει από τον βρόγχο. Μια επιπλέον εσωτερική autorelease pool δημιουργείται στην αρχή κάθε κύκλου γεγονότων και απελευθερώνεται στο τέλος του.

Η ανάγκη και η ύπαρξη της αυτόματης απελευθέρωσης μνήμης έχει περισσότερο νόημα μόλις αντιληφθούμε τον στόχο της, ο οποίος είναι να μεταφέρει τον έλεγχο του κύκλου ζωής ενός αντικείμενου από την ιδιοκτησία ενός αντικειμένου σε ένα άλλο χωρίς να απελευθερώνει αμέσως το αντικείμενο.

Η διάθεση, διατήρηση, αντιγραφή και απελευθέρωση μνήμης

Αν και η αυτόματη απελευθέρωση μνήμης είναι χρήσιμη, πρέπει να είμαστε προσεχτικοί κατά την χρησιμοποίηση της καθώς μπορεί με αυτόν τον τρόπο να επεκτείνουμε χωρίς λόγο τον χρόνο που ένα αντικείμενο έχει δεσμεύσει μνήμη και κατά συνέπεια να μεγαλώνει η απαίτηση μνήμης της εφαρμογής μας. Μερικές φορές είναι λογικό να την χρησιμοποιούμε. Όμως, ειδικά προγραμματιστές χωρίς μεγάλη εμπειρία στην Objective-C, συχνά χρησιμοποιούν σε υπερβολικό βαθμό αυτή την τεχνική

Στην πραγματικότητα η ίδια η Apple στους οδηγούς χρήσης της, επισήμως αποθαρρύνει την χρήση autorelease αντικείμενων στον προγραμματισμό στο iPhone, καθώς στο iPhone OS όταν μια εφαρμογή εκτελείται σε ένα τέτοιο σύστημα που διαθέτει περιορισμένη μνήμη, η χρήση τέτοιων αντικειμένων δεν πρέπει να γίνεται σε μεθόδους ή μέρη κώδικα οπού η εφαρμογή δημιουργεί πολλά αντικείμενα. Αντί για αυτό, πρέπει να απελευθερώνουμε τα αντικείμενα όσο μπορούμε πιο σύντομα.

Όταν θέλουμε να διαχειριστούμε την μνήμη μόνοι μας, χρησιμοποιώντας τον κύκλο διάθεση, διατήρηση και απελευθέρωση (alloc, retain και release cycle, δείτε εικόνα) δεν πρέπει να απελευθερώνουμε αντικείμενα που δεν έχουμε εμείς δημιουργήσει. Πρέπει επίσης να είμαστε σίγουροι ότι οι φόρες που έχουμε διατηρήσει ένα αντικείμενο είναι ίσες με τις φορές που το απελευθερώσαμε. Αντικείμενα που έχουμε εμείς δημιουργήσει είναι αυτά που έχουμε χρησιμοποιήσει την λέξη alloc για να τους διαθέσουμε μνήμη ή την λέξη retain έτσι ώστε να τα διατηρήσουμε. Αντιθέτως αντικείμενα που έχουν δημιουργηθεί με την βοήθεια μεθόδων-κατασκευαστών όπως ο stringWithString, δεν τα έχουμε δημιουργήσει εμείς οπότε και δεν πρέπει να τα απελευθερώσουμε εμείς.

Όταν απελευθερώνουμε ένα αντικείμενο έχουμε την επιλογή να στείλουμε είτε ένα μήνυμα χρησιμοποιώντας την λέξη release είτε την λέξη autorelease:

```
[anObject release];
[anObject autorelease];
```

Όταν στείλουμε ένα μήνυμα με την λέξη release, τότε απελευθερώνουμε αμέσως την μνήμη που το αντικείμενο χρησιμοποιεί, εφόσον με αυτό το μήνυμα η αναφορά μέτρησης (Reference Count) του γίνεται μηδέν, ενώ άμα στείλουμε ένα μήνυμα με την λέξη autorelease τότε το αυτό το αντικείμενο προστίθεται στην autorelease pool. Έτσι το αντικείμενο απελευθερώνεται όταν απελευθερωθούν όλα τα αντικείμενα που την απαρτίζουν, κάτι που συνήθως συμβαίνει στο τέλος της κάθε κλάσης.
Εάν το αντικείμενο είναι εκπρόσωπος (delegate) ενός άλλου αντικειμένου τότε, πριν απελευθερώσουμε το αρχικό αντικείμενο ,πρέπει να το θέσουμε ίσο με nil.

Η μέθοδος dealloc

Η μέθοδος dealloc καλείται όταν ένα αντικείμενο απελευθερώνεται. Δεν πρέπει ποτέ να καλούμε αυτήν την μέθοδο άμεσα αλλά αντί αυτού να στέλνουμε ένα μήνυμα με την λέξη release στο αντικείμενο καθώς σε διαφορετική περίπτωση το αντικείμενο μπορεί να περιέχει αναφορές σε άλλα αντικείμενα τα οποία δεν πρέπει να απελευθερωθούν (deallocated).
Όπως κάναμε και με την κλάση HelloWorldViewController, πρέπει πάντα να προσθέτουμε στη μέθοδο dealloc τα αντικείμενα που εμείς δημιουργήσαμε:

```
- (void)dealloc {
[label release];
[button release]
[super dealloc];
}
```

Σε αυτήν την μέθοδο απελευθερώνουμε το αντικείμενο label καθώς και το αντικείμενο button. Στη συνέχεια καλούμε την μέθοδο dealloc της υπέρ-κλάσης. Επιτρέπεται επίσης να στείλουμε ένα μήνυμα με την λέξη release σε ένα nil αντικείμενο.

Απαντώντας σε ειδοποιήσεις μνήμης

Ο κώδικάς μας πρέπει να μπορεί να απαντάει σε ειδοποιήσεις μνήμης. Ας δούμε ξανά λίγο το παράδειγμα μας στην εφαρμογή της κλάσης HelloWorldViewController. Περιέχει την μέθοδο didReceiveMemoryWarning:

```
- (void)didReceiveMemoryWarning {
[super didReceiveMemoryWarning];
}
```

Εδώ είναι το κατάλληλο σημείο να απελευθερώσουμε μεγάλα μέρη μνήμης, όπως για παράδειγμα φωτογραφίες ή ότι μπορεί να έχουμε αποθηκεύσει από τον διαδίκτυο. Εάν αγνοήσουμε μια ειδοποίηση μνήμης τότε η εφαρμογή μας μπορεί να καταρρεύσει. Το iPhone δεν διαθέτει κάποιο είδος εικονικής μνήμης οπότε όποτε η συσκευή δεν έχει χώρο στη μνήμη της τότε δεν υπάρχει απλά άλλη μνήμη που να μπορεί να διαθέσει. Είναι δυνατόν, αλλά και καλός τρόπος να δοκιμάσουμε την εφαρμογή μας, να εξομοιώσουμε μια ειδοποίηση μνήμης στον iPhone Simulator, το οποίο μπορούμε να το κάνουμε επιλέγοντας Hardware και μετά Simulate Memory

3.2.3 Θεμελιώδη σχεδιαστικά πρότυπα

Όταν γράφουμε κώδικα τις περισσότερες φορές χρησιμοποιούμε κάποια πρότυπα αν και μερικές φορές μπορεί να το κάνουμε ασυναίσθητα. Ένα σχεδιαστικό πρότυπο είναι απλά μια εύκολα επαναχρησιμοποιήσιμα λύση, ένα σχέδιο, που σκοπό έχει να προσεγγίσει ένα σύνηθες πρόβλημα.

Ένα τέτοιο πρότυπο δεν είναι έτοιμος κώδικας, αλλά η περιγραφή του πως πρέπει να μοντελοποιήσουμε την εφαρμογή μας σε σχέση με τις κλάσεις που χρησιμοποιούμε και πως αυτές πρέπει να κατασκευαστούν αλλά και τις συνδέσεις και τις σχέσεις που πρέπει να έχουν μεταξύ τους.

Το πλαίσιο Cocoa Touch που υπάρχει στα κατώτερα στρώματα των εφαρμογών μας στο iPhone είναι βασισμένο σε ένα από τα πιο παλιά σχεδιαστικά πρότυπα, το πρότυπο Model-View-Controller (MVC) , το οποίο υπάρχει από το 1970. Το πρότυπο MVC χρησιμοποιείται για να διαχωρίζει την προγραμματιστική λογική από το περιβάλλον χρήστη (User Interface, UI) και είναι ένας ευρέως διαδεδομένος τρόπος με τον οποίο μπορούμε να κατασκευάσουμε εφαρμογές για το iPhone. Από την στιγμή που χρησιμοποιείται τόσο πολύ από τα πλαίσια (Framework) που μας

προμηθεύει η Apple, και στα οποία περιλαμβάνεται και το UIKit Framework, είναι έτσι και αλλιώς αρκετά δύσκολο να γράψουμε μια εφαρμογή χωρίς να χρησιμοποιήσουμε αυτό το σχεδιαστικό πρότυπο στην υλοποίηση του κώδικά μας. Αν και μπορούμε να γράψουμε ένα πρόγραμμα χωρίς να ακολουθήσουμε το πρότυπο αυτό, είναι πολύ δύσκολο να πολεμήσουμε τα Framework που υλοποιούν τον κώδικά μας, και για αυτό θα πρέπει αντί αυτού να δουλεύουμε μαζί τους.

Το πρότυπο Model-View-Controller

Το πρότυπο MVC χωρίζει την εφαρμογή μας σε τρία λειτουργικά μέρη.

Το Model (μοντέλο):

Το model διαχειρίζεται την κατάσταση της εφαρμογής μας (και τα σχετιζόμενα δεδομένα) και συνήθως δουλεύει αδιάκοπα. Είναι εντελώς ανεξάρτητη με το UI ή την παρουσίαση της κατάστασης της εφαρμογής στον χρήστη.

Το View (προβολή):

Το view ουσιαστικά είναι το τι ο χρήστης βλέπει, και το τι το μοντέλο δείχνει στο χρήστη. Επιτρέπει στον χρήστη να αλληλεπιδρά με το μοντέλο και απαντά ή δημιουργεί γεγονότα. Σε εφαρμογές στο iPhone το view κατασκευάζεται συνήθως στο Interface Builder και όχι προγραμματιστικά.

Το Controller (ρυθμιστής):

Ο controller συντονίζει τις αλλαγές στο view σε σχέση με το model, όταν ο χρήστης αλληλεπιδρά με το view αυτός κάνει αλλαγές στο model και ανάποδα. Εδώ είναι συνήθως το μέρος οπού υπάρχει η μεγαλύτερη λογική του προγραμματικός μας.
Στο παράδειγμά μας, την εφαρμογή HelloWorld, θα έπρεπε να δημιουργήσουμε το view με το Interface Builder και η κλάση HelloWorldViewController στην ουσία είναι αυτή που θα διαχειριζόταν το view. Αυτή η εφαρμογή ήταν πολύ απλή και για αυτό το model της συμπεριλήφθηκε στη ίδια κλάση. Αν θέλαμε όμως να ακολουθήσουμε αυστηρά το σχεδιαστικό πρότυπο MVC τότε θα έπρεπε να είχαμε δημιουργήσει μια κλάση που θα περιελάμβανε την μέθοδο sayHello:.

Views και View Controllers

Έχουμε ως ώρας αναφερθεί και στα views αλλά και στα view controllers, και αν και συνηθίζεται να δημιουργούμε τα views στον Interface Builder, και μετά να τα διαχειριζόμαστε από τον δικό μας view controller, αυτός δεν είναι ο μόνος τρόπος να δημιουργήσουμε ένα view. Μπορούμε να δημιουργήσουμε ένα προγραμματιστικά, και στην πραγματικότητα στις πρώτες μέρες του προγραμματισμού στο iPhone ήμασταν υποχρεωμένοι να δουλεύουμε με αυτόν τον τρόπο.

Όμως ο Interface Builder έκανε τα πράγματα αρκετά πιο εύκολα για μας και είναι ο πιο απλός και εύχρηστος τρόπος να δημιουργήσουμε ένα view. Όταν χρησιμοποιούμε τον Interface Builder στη ουσία δημιουργούμε ένα αρχείο με κώδικα XML. Με αυτό τον τρόπο η δημιουργία αντικειμένων και η δήλωση των σχέσεων μεταξύ τους μας ελευθερώνει από μεγάλα μέρη κώδικα που αλλιώς θα χρειαζόταν να γράψουμε μονοί μας για να διαχειριστούμε το view.

Τα Delegates και το πρότυπο DataSource

Τα delegates (εκπρόσωποι) είναι πρωτόκολλα που επιτρέπουν σε ένα αντικείμενο να δρα εκ μέρους ενός άλλου αντικειμένου. Για να δεχτούμε μια ειδοποίηση ενός γεγονότος στο οποίο πρέπει να απαντήσουμε, η κλάση delegate πρέπει να υλοποιεί μια μέθοδο notification η οποία να έχει δηλωθεί ως πρωτόκολλο delegate το οποίο να συνδέεται με αυτό το γεγονός. Το πρωτόκολλο μπορεί, και συνήθως το κάνει, να προσδιορίσει έναν αριθμό μεθόδων που η κλάση delegate πρέπει να περιλαμβάνει.

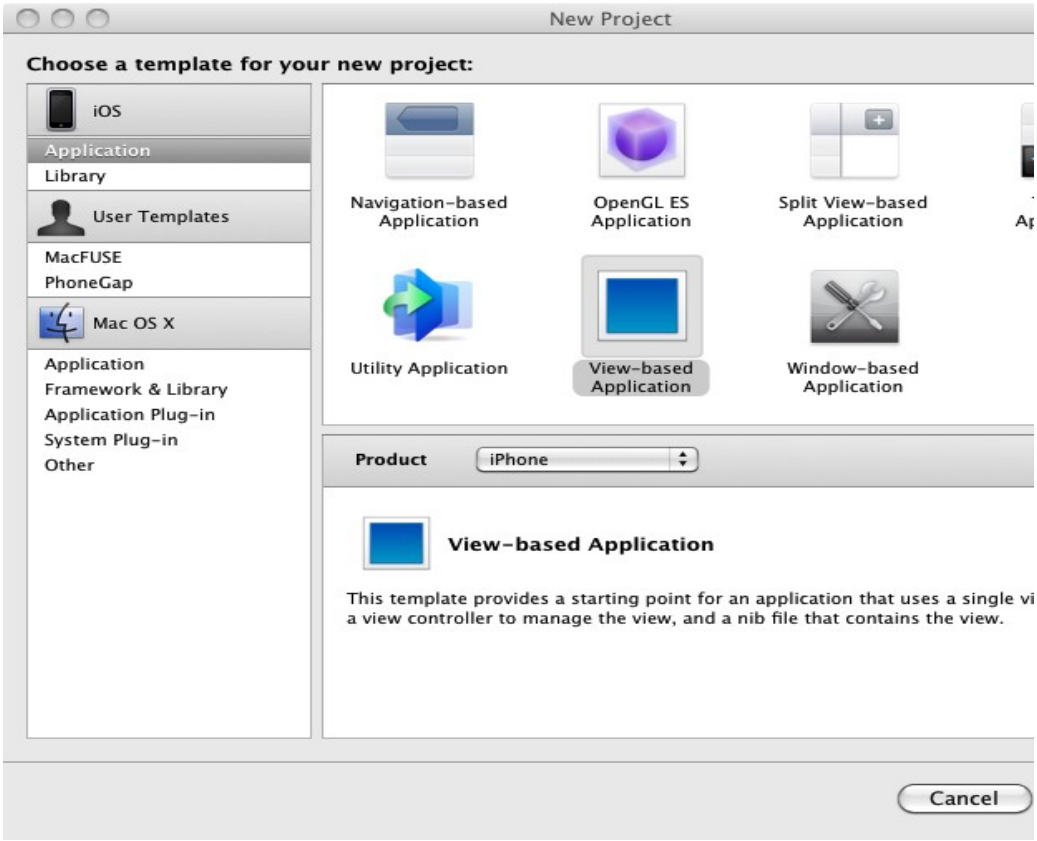
Συμπεράσματα

Σε αυτό το υποκεφάλαιο καλύψαμε ένα μεγάλο μέρος των βασικών λειτουργιών της γλώσσας και το πρότυπο MVC είναι ένα πολύ βασικό στοιχείο ώστε να καταλάβουμε πως μπορούμε να χρησιμοποιήσουμε την Objective-C, καθώς και τα δυνατά της σημεία αλλά και τις αδυναμίες της σε σχέση με άλλες γλώσσες προγραμματισμού.

ΚΕΦΑΛΑΙΟ 4ο : Δημιουργία της εφαρμογής - Κορμός της εφαρμογής

4.1 Δημιουργία ενός νέου project στο Xcode και εισαγωγή ενός table view

Στην αρχή θα πρέπει να ανοίξουμε το Xcode πατώντας το εικονίδιο του. Επιλέγουμε το "Create a new Xcode project" έτσι ώστε να ξεκινήσουμε ένα πρόγραμμα από το μηδέν. Από το νέο παράθυρο που θα ανοίξει επιλέγουμε το "View-based Application" έτσι ώστε το Xcode να μας ετοιμάσει τα βασικά αρχεία για ένα πρόγραμμα που σκοπό έχει να δείχνει πληροφορίες στον χρήστη. Όταν μας ζητηθεί ονομάζουμε το project CSTeilar και το αποθηκεύουμε οπουδήποτε θέλουμε.

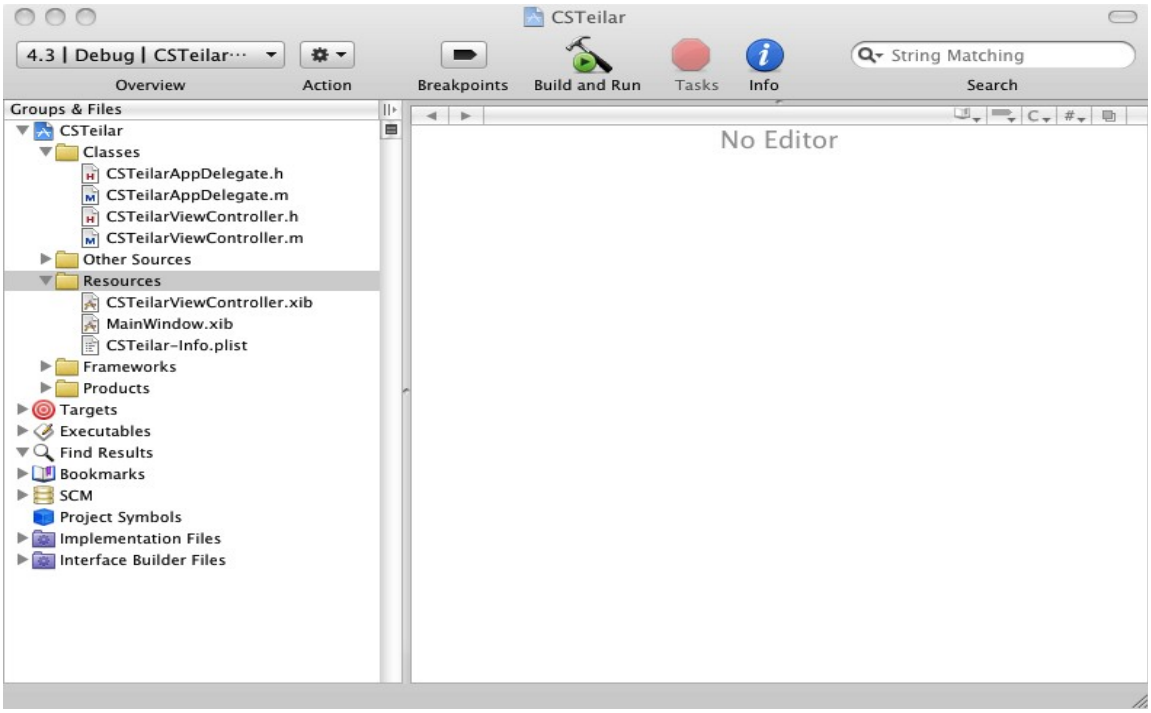


Εικόνα 4.1 : Το μενού επιλογών για την δημιουργία νέας εφαρμογής

Αυτή τη στιγμή έχουμε ήδη ένα πρόγραμμα το οποίο δεν έχει σφάλματα και μπορεί να τρέξει. Ελέγχουμε στο πάνω αριστερό μέρος του παραθύρου αν στο αναδυόμενο μενού είναι επιλεγμένος ο iPhone Simulator και στη συνεχεία μπορούμε να πατήσουμε το Build and Run έτσι ώστε να πάρουμε μια ιδέα για το πως λειτουργεί ο εξομοιωτής του Xcode.

Αφού όλα λειτουργούν όπως πρέπει μπορούμε να ξεκινήσουμε το πρόγραμμά μας φτιάχνοντας το βασικό περιβάλλον χρήστη (User Interface). Ανοίγουμε με διπλό κλικ το

CSTeilarViewController.xib το οποίο και ανοίγει στο Interface Builder. Μετά κάνουμε διπλό κλικ στο εικονίδιο “View” στο παράθυρο CSTeilarViewController.xib ώστε να το φέρουμε μπροστά και επιλέγουμε το παράθυρο “Library”. Σε αυτό υπάρχουν πολλά στοιχεία τα οποία μπορούμε να χρησιμοποιήσουμε για να φτιάξουμε το περιβάλλον χρήστη που θέλουμε. Στη προκειμένη περίπτωση θέλουμε ένα UITableView οπότε μπορούμε να το βρούμε κάνοντας αναζήτηση στο κάτω μέρος του παράθυρου ένα “table view”. Εναλλακτικά μπορούμε να το βρούμε κάτω από την κατηγορία “Cocoa Touch” και στη συνεχεία “Data Views”. Στην συνέχεια το τοποθετούμε με drag and drop στο παράθυρο “View” και το προσαρμόζουμε στο κέντρο του. Όπως μπορείτε να διαπιστώσετε το πρόγραμμα βγάζει από μόνο του τις κατάλληλες γραμμές έτσι ώστε να τοποθετήσουμε το “table view” όσο το δυνατόν πιο σωστά στο “View”.



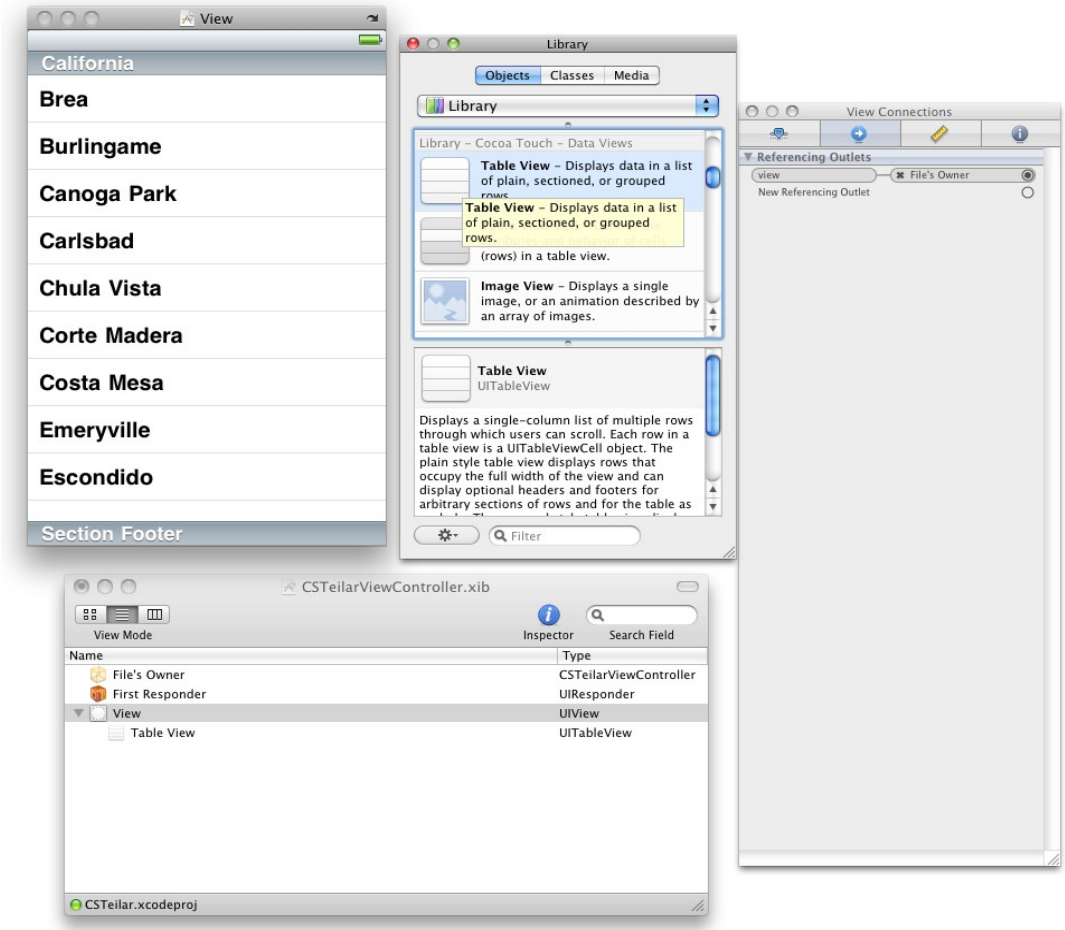
Εικόνα 4.2 : Η νέα εφαρμογή CSTeilar που δημιουργήθηκε από τον Xcode

Πρέπει να σιγουρευτούμε ότι τοποθετήσαμε το “table view” ως “subview” του βασικού “View” πατώντας σε ένα από τα πάνω δεξιά κουμπάκια του CSTeilarViewController.xib παράθυρου έτσι ώστε να δούμε τα αρχεία του σε λίστα. Θα πρέπει όλα να φαίνονται πια όπως στην εικόνα 4.3.Εδώ θα πρέπει να τονίσουμε ότι κάθε φορά που μεταφερόμαστε από το Xcode στο Interface Builder αλλά και ανάποδα θα πρέπει να αποθηκεύουμε τις αλλαγές μας έτσι ώστε να θέτονται σε ισχύ.

Γυρίζουμε πίσω στο Xcode για να προσθέσουμε τις απαραίτητες διασυνδέσεις στο κώδικα έτσι ώστε να μπορεί να επικοινωνεί το UITableView με αυτόν. Ανοίγουμε το αρχείο CSTeilarViewController.h και προσθέτουμε μια μεταβλητή UITableView στη δήλωση @interface,

κατόπιν ορίζουμε αυτό με την ιδιότητα property και IBOutlet. Όταν ένα αντικείμενο ορίζεται ως IBOutlet τότε μας δίνει την δυνατότητα να μπορούμε να το συνδέσουμε με άλλες κλάσεις και μεθόδους μέσω του Interface Builder.

Πρέπει επίσης να δηλώσουμε ότι το CSTeilarViewController.h εφαρμόζει και το UITableViewDataSource και τα πρωτόκολλα UITableViewDelegate. Αυτό σημαίνει ότι και παρέχει τα στοιχεία για να γεμίσει το table view και χειρίζεται τα γεγονότα που παράγονται από την αλληλεπίδραση του χρήστη με την αυτό.



Εικόνα 4.3 : Ο Interface Builder με το table view τοποθετημένο ως subview

Αφού κάνουμε όλα αυτά το CSTeilarViewController.h πρέπει να είναι ως εξής:

```
#import <UIKit/UIKit.h>

@interface CSTeilarViewController : UIViewController <UITableViewDataSource,
UITableViewDelegate> {
    UITableView *tableView;
```

```
}
@property (nonatomic, retain) IBOutlet UITableView *tableView;
@end
```

Σε οποιαδήποτε λέξη-κλειδί του κώδικα μπορούμε πατώντας το Command (⌘) και διπλό κλικ να δούμε την δήλωσή της, δηλαδή τον κώδικα που η ίδια η Apple έχει γράψει και τρέχει κάθε φορά που την χρησιμοποιούμε. Επίσης μπορούμε να δούμε με αυτόν τον τρόπο ποιες μεθόδους πρέπει υποχρεωτικά να χρησιμοποιήσουμε και ποιες είναι συμπληρωματικές. Στη συγκεκριμένη περίπτωση βλέπουμε ότι δεν χρειάζεται να χρησιμοποιήσουμε υποχρεωτικά καμία μέθοδο. Όμως αν θέλουμε να έχουμε κάποια λειτουργική αξία πρέπει οπωσδήποτε να χρησιμοποιήσουμε τις μεθόδους tableView:cellForRowAtIndexPath: and tableView:numberOfRowsInSection:.

Η πρώτη από αυτές τις μεθόδους επιστρέφει ένα κελί, το οποίο ανήκει στο UITableView Cell αντικείμενο. Το table view θα ρωτάει τη πηγή των δεδομένων (data source delegate) κάθε φορά που ένα κελί χρειάζεται να εμφανιστεί στην οθόνη. Η δεύτερη μέθοδος επιστρέφει έναν ακέραιο αριθμό (NSInteger) το οποίο δείχνει πόσα μέρη υπάρχουν στο table view. Το table view μπορεί να χωριστεί σε μέρη και ένας τίτλος να προστίθεται στη κορυφή του καθενός.

Τώρα πρέπει να προσθέσουμε τον κώδικα εφαρμογής αυτών των δύο υποχρεωτικών μεθόδων στη κλάση CSTeilarViewController (CSTeilarViewController.m) Από τη στιγμή που γράψουμε τον κώδικα και τον τρέξουμε θα δούμε και την μέθοδο tableView:didSelectRow AtIndexPath:. Αυτή η μέθοδος καλείται κάθε φορά που διαλέγουμε ένα αντικείμενο στην οθόνη και επιστρέφει την τιμή αυτού του αντικειμένου ώστε να χρησιμοποιηθεί για περαιτέρω επεξεργασία.

Παρακάτω βλέπουμε τα περιεχόμενα του CSTeilarViewController.m.

```
#import "CSTeilarViewController.h"
@implementation CSTeilarViewController
@synthesize tableView;

- (void)didReceiveMemoryWarning {

   [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {

}
- (void)dealloc {
      [tableView release];
```

```
    [super dealloc];
}

#pragma mark Instance Methods

- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell = [tv dequeueReusableCellWithIdentifier:@"cell"];
    if( nil == cell ) {
      cell = [[[UITableViewCell alloc]
                    initWithStyle:UITableViewStyleGrouped reuseIdentifier:@"cell"]
autorelease];
    }
    return cell;
}

- (NSInteger)tableView:(UITableView *)tv numberOfRowsInSection:(NSInteger)section {
    // Our table view will consist of only 3 cells
    return 3;
}
```

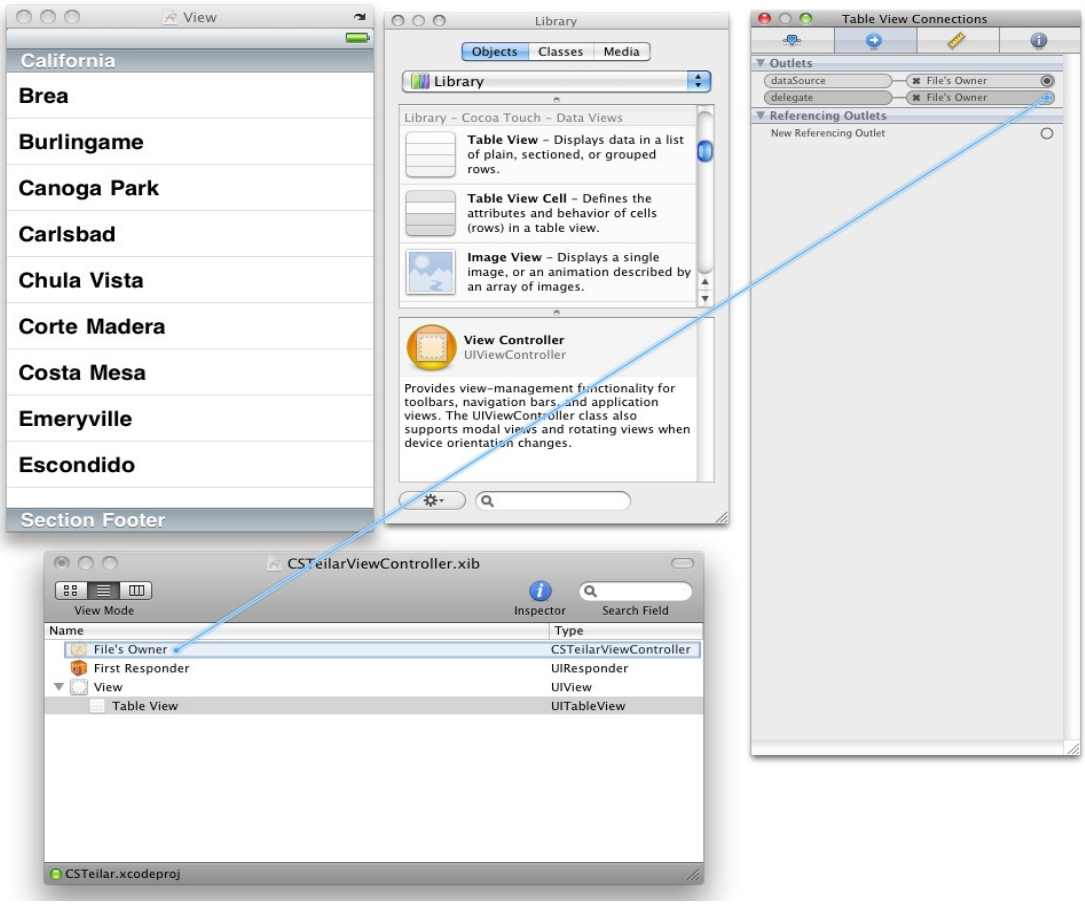
Χρησιμοποιούμε το @synthesize έτσι ώστε να φτιαχτούν τα setters και τα getters της μεταβλητής. Το #pragma mark υπάρχει απλά για να μας βοηθάει να ξεχωρίζουμε μέρη του κώδικά μας. Στο τέλος του κώδικα υλοποιούμε τις 2 βασικές μεθόδους του UITableView. Στη πρώτη φτιάχνουμε ένα αντικείμενο UITableViewCell που το ονομάζουμε cell. Στη συνέχεια το βγάζουμε από μια λίστα η οποία έχει δημιουργηθεί για εμάς από την μέθοδο και στην περίπτωση που το περιεχόμενο του αντικείμενου δεν είναι ίσο με nil τότε το αρχικοποιούμε και το επιστρέφουμε στο table view. Όσον αφορά την δεύτερη μέθοδο απλά για τώρα επιστρέφουμε τον ακέραιο αριθμό 3 που δηλώνει ότι το table view θα έχει 3 κελιά.

Τώρα πρέπει να επιστρέψουμε στο Interface Builder και να συνδέσουμε το κώδικα που μόλις προσθέσαμε με τα αντικείμενα. Ανοίγουμε το αρχείο CSTeilarViewController.xib και όταν ανοίγει μέσω του παραπάνω προγράμματος επιλέγουμε στο βασικό παράθυρο το κατάλληλο κουμπί έτσι ώστε τα αρχεία του να φαίνονται ως λίστα και μετά ανοίγουμε τη λίστα του view για να αποκαλυφθεί το table view.

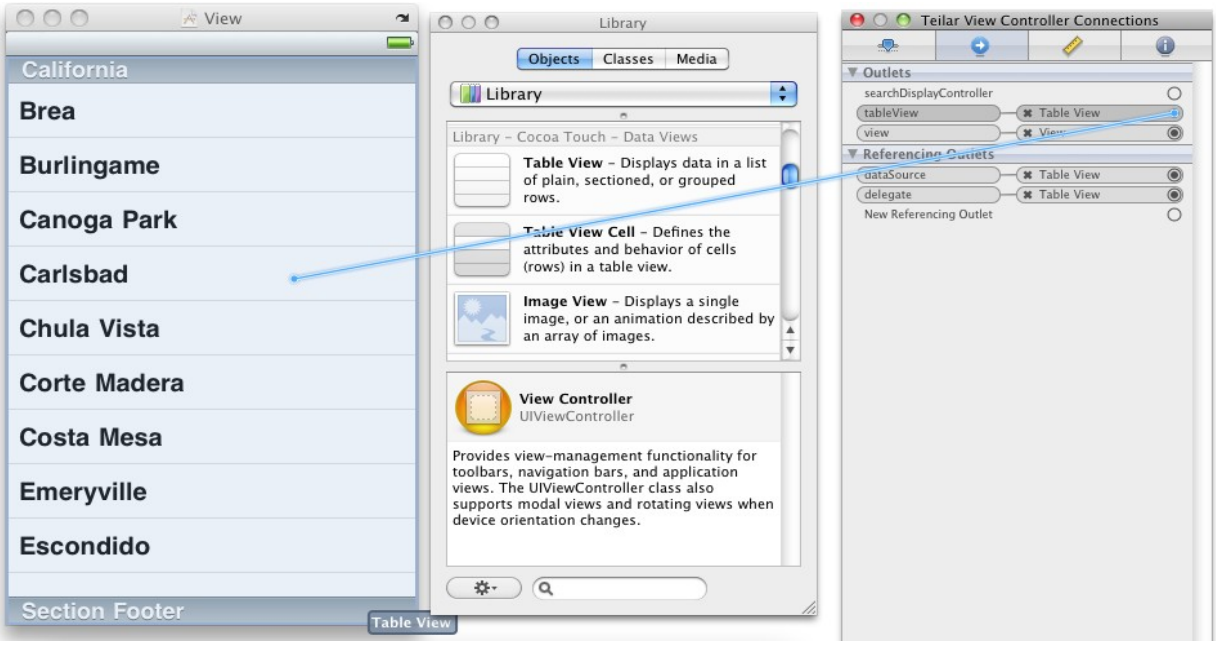
Στη συνέχεια επιλέγουμε το εικονίδιο table view και θέτουμε τον Inspector έτσι ώστε να δείχνει τις συνδέσεις που μπορούμε να πραγματοποιήσουμε(Connections Inspector) Αυτό αποκαλύπτει τις συνδέσεις με το data source και το delegate. Συνδέουμε και τα 2 με το File's Owner στο κεντρικό παράθυρο όπως φαίνεται στην εικόνα 4.4.

Τώρα επιλέγουμε το File's Owner. Στο τμήμα των outlets του Connections Inspector βλέπουμε το αντικείμενο tableView το οποίο ορίσαμε ως IBOutlet στο αρχείο

CSTeilarViewController.h. Το συνδέουμε στο UITableView όπως φαίνεται στην εικόνα 4.5.



Εικόνα 4.4 : Συνδέοντας το table view με το File's Owner



Εικόνα 4.5 : Συνδέοντας το table view με την προβολή

Φτάσαμε σε ένα σημείο στο οποίο μπορούμε να τρέξουμε τον κώδικα έτσι ώστε να είμαστε σίγουροι ότι όλα δουλεύουν σωστά. Αποθηκεύουμε τις αλλαγές στο Interface Builder και επιστρέφουμε στο Xcode. Ο κώδικας αυτή τη στιγμή πρέπει να τρέχει χωρίς προβλήματα αν και αυτή τη στιγμή δεν πρέπει να κάνει και πολλά. Οπότε επιλέγουμε το Build and Run έτσι ώστε να ξεκινήσει η εφαρμογή στον εξομοιωτή. Η εικόνα 4.6 δείχνει τι πρέπει να βλέπουμε.

Εικόνα 4.6 : Το κενό table view

Τώρα που έχουμε το βασικό κώδικα έτσι ώστε να έχουμε ένα table view μπορούμε να γυρίσουμε στην εφαρμογή του CSTeilarViewController (CSTeilarViewController.m) και να δούμε με λεπτομέρεια την μέθοδο tableView:cellForRowAtIndexPath: όπου δημιουργούμε και επιστρέφουμε κελιά table view. Για λόγους απόδοσης το UITableView μπορεί να επαναχρησιμοποιεί τα κελιά έτσι ώστε να ενισχύσει την ταχύτητα με την οποία εμφανίζονται τα κελιά στην οθόνη (scroll), μειώνοντας την ανάγκη για την διανομή μνήμης κατά τη διάρκεια αυτής της διαδικασίας. Όμως για να αξιοποιήσουμε αυτή την δυνατότητα πρέπει να δώσουμε ένα συγκεκριμένο επαναχρησιμοποιήσιμο προσδιοριστικό string. Το UITableView το χρησιμοποιεί για να βρει ήδη υπάρχοντα κελιά με τον ίδιο προσδιορισμό χρησιμοποιώντας τη μέθοδο dequeueReusableCellWithIdentifier:. Αν δεν βρει ένα κελί με τον ίδιο προσδιορισμό που να μην χρησιμοποιείται, τότε δημιουργεί ένα αλλά αν βρει (ίσος το αντικείμενο να βρίσκεται σε σημείο που να μην είναι ορατό στην οθόνη) τότε το επαναχρησιμοποιεί:

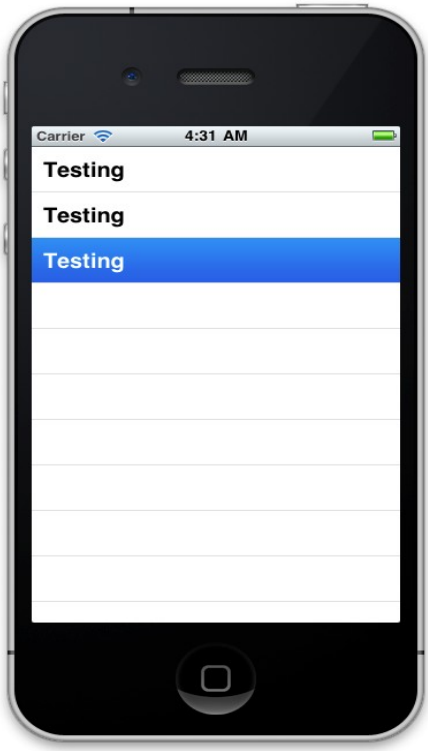
```
- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell = [tv dequeueReusableCellWithIdentifier:@"cell"];
    if( nil == cell ) {
      cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero reuseIdentifier:@"cell"] autorelease];
    }
    return cell;
}
```

Ως ώρας το πρόγραμμά μας δεν είναι και τόσο χρήσιμο οπότε μπορούμε να προσθέσουμε περιεχόμενο και χρησιμότητα. Για να το κάνουμε αυτό πρέπει να χρησιμοποιήσουμε τη μέθοδο tableView:didSelectRowAtIndexPath: του CSTeilarViewController.m. Όπως φαίνεται και από την ονομασία της αυτή η μέθοδος καλείται κάθε φορά που ο χρήστης επιλέγει ένα κελί του table view. Επειδή τα κελιά μας είναι άδεια αυτή τη στιγμή, πρέπει να προσθέσουμε και κείμενο σε κάθε κελί πριν το επιστρέψουμε από αυτή τη μέθοδο.

```
#pragma mark UITableViewDataSource Methods

- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell = [tv dequeueReusableCellWithIdentifier:@"cell"];
    if( nil == cell ) {
      cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero reuseIdentifier:@"cell"] autorelease];
    }
    cell.textLabel.text = @"Testing";
    return cell;
}

- (NSInteger)tableView:(UITableView *)tv numberOfRowsInSection:(NSInteger)section {
    // Our table view will consist of only 3 cells
```

```
    return 3;
}

#pragma mark UITableViewDelegate Methods

- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tv deselectRowAtIndexPath:indexPath animated:YES];
}
@end
```

Προσθέτουμε το Testing ως περιεχόμενο στην ετικέτα του κάθε κελιού που επιστρέφεται από τη μέθοδο tableView:cellForRowAtIndexPath:. Στο τέλος του κώδικα εφαρμόζουμε τη μέθοδο tableView:didSelectRowAtIndexPath: κάθε φορά που ο χρήστης σταματάει να επιλέγει κάποιο αντικείμενο τότε το κελί χάνει το χρώμα του (fades out) και σταματάει να φαίνεται ως επιλεγμένο. Χωρίς αυτό, το κελί θα παραμένει επιλεγμένο επ' αόριστον. Μπορούμε να δούμε τις αλλαγές στην εικόνα 4.7

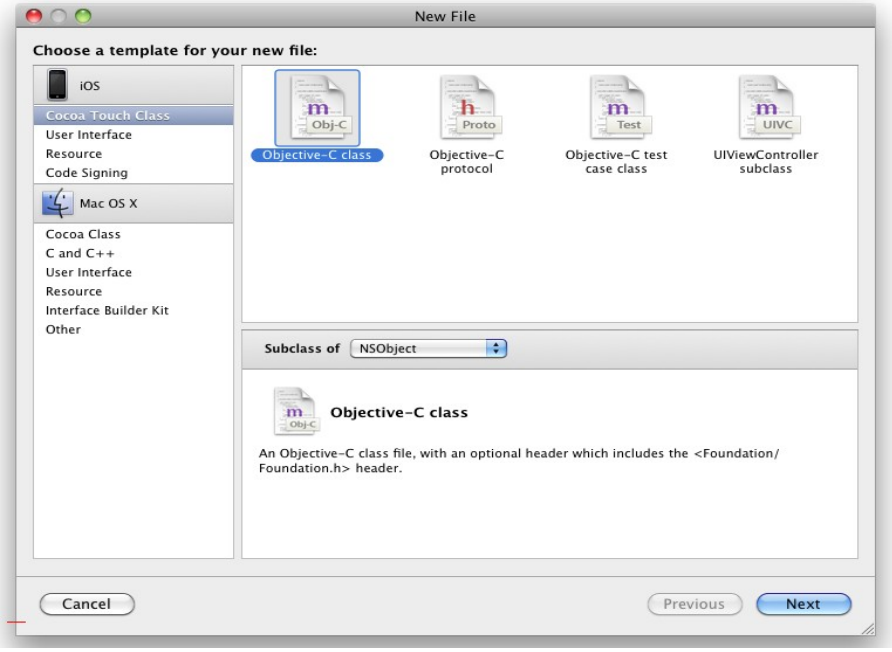


Εικόνα 4.7: Ο κώδικας που προσθέσαμε ο οποίος τρέχει στο iPhone Simulator

4.2 Κατασκευάζοντας το Model.

Ως ώρας έχουμε ένα UITableView το οποίο δουλεύει. Έχουμε δηλαδή υλοποιήσει και το view άλλα και τον controller του μοντέλου MVC (model, viewer, controller). Τώρα θα εφαρμόσουμε το model μέρος του. Αυτό θα πρέπει να είναι ξεχωριστό από το view και τον controller αφού θέλουμε τα αντικείμενα τα οποία αποθηκεύουμε να είναι όσο το δυνατόν πιο ανεξαρτητοποιημένα από τον τρόπο που τα δείχνουμε στην οθόνη. Με αυτό τον τρόπο θα επεκτείνουμε την επαναχρησιμοποίηση του κώδικά μας και για τις 2 κλάσεις δηλαδή και για την κλάση που χειρίζεται το User Interface και για την κλάση που αποθηκεύει τα δεδομένα, επιτρέποντας μας να αλλάξουμε μέρη της εφαρμογής μας χωρίς να επηρεάζεται από αυτό μεγάλο μέρος του κώδικά μας.

Κάνοντας δεξί κλικ στο φάκελο Classes στο υπο-παράθυρο Groups & Files και επιλέγοντας Add και στη συνέχεια New File βλέπουμε αυτό που φαίνεται στην εικόνα 4.8.



Εικόνα 4.8: Δημιουργώντας μια νέα κλάση

Σιγουρευόμαστε ότι η κλάση Cocoa Touch είναι η επιλεγμένη στην αριστερή πλευρά του παράθυρου και επιλέγουμε την Objective-C class προσέχοντας να είναι υποκλάση της NSObject και επιλέγουμε Next. Στη συνέχεια πρέπει να δώσουμε ένα όνομα στα αρχεία της νέας κλάσης. Γράφουμε MainMenu.m και πατάμε Finish. Το Xcode θα δημιουργήσει ένα ζευγάρι αρχείων, το MainMenu.m και το MainMenu.h τα οποία περιέχουν την δήλωση και την υλοποίηση της νέας κλάσης, και θα τα τοποθετήσει στον φάκελο Classes. Εάν δούμε αυτά τα αρχεία θα διαπιστώσουμε ότι από την στιγμή που το μόνο που διαλέξαμε είναι η κλάση να είναι υποκλάση του NSObject, το

Xcode δεν δημιούργησε πολύ κώδικα. Δεν ήξερε για ποιο λόγο φτιάξαμε το αντικείμενο οπότε θα πρέπει εμείς να τον γράψουμε.

Ανοίγουμε το αρχείο MainMenu.h και προσθέτουμε τις μεταβλητές που θα κρατάνε το όνομα του κελιού καθώς και άλλη μια που θα κρατάει μια πληροφορία σημαντική για το κάθε κελί:

```
#import <Foundation/Foundation.h>

@interface MainMenu : NSObject {
    NSString *menuItemName;
    NSString  *menuItemLink;
    }
@property (nonatomic, retain) NSString *menuItemName;
@property (nonatomic, retain) NSString *menuItemLink;

@end
```

Δηλώνουμε και τις 2 μεταβλητές ως NSString. Στη συνέχεια ανοίγουμε το MainMenu.m στο οποίο πρέπει να υλοποιήσουμε τις παραπάνω δηλώσεις. Πρέπει όμως να προσθέσουμε και την μέθοδο dealloc: για να απελευθερώσουμε την μνήμη όταν η κλάση δεν χρησιμοποιείται πια. Ακολουθεί ο κώδικας του MainMenu.m:

```
#import "MainMenu.h"

@implementation MainMenu

@synthesize menuItemName;
@synthesize menuItemLink;

-(void) dealloc {
    [menuItemName release];
    [menuItemLink release];
    [super dealloc];
}
@end
```

Επειδή χρησιμοποιήσαμε properties, οι μέθοδοι set και get δημιουργήθηκαν για μας αυτόματα. Οπότε είμαστε έτοιμοι για τώρα. Όπως είναι φανερό, αυτή είναι μια μικρή κλάση η οποία κρατάει λίγα δεδομένα αλλά μπορούμε να δούμε πόσο χρήσιμα μπορούν να είναι τα properties για μεγαλύτερες και πιο περίπλοκες κλάσεις.

Ας γυρίσουμε τώρα στη κλάση CSTeilarAppDelegate και να περάσουμε στο κώδικα μερικά αντικείμενα που θα αναπαραστήσουν τα κελιά στο βασικό μενού του προγραμματικός. Επιλέγουμε αυτήν την διαδικασία και περνάμε τα δεδομένα στον κώδικα καθώς είναι τόσο μικρός ο όγκος των δεδομένων που δεν υπάρχει λόγος να χρησιμοποιήσουμε κάτι πιο περίπλοκο και απαιτητικό. Έτσι θα πρέπει να φαίνεται το CSTeilarAppDelegate.m μετά την προσθήκη του νέου κώδικα:

```
#import "CSTeilarAppDelegate.h"
#import "CSTeilarViewController.h"
#import "MainMenu.h"

@implementation CSTeilarAppDelegate

@synthesize window;
@synthesize viewController;
@synthesize menuItems;

#pragma mark -
#pragma mark Application lifecycle

(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    MainMenu *anakinosis = [[MainMenu alloc] init];
        anakinosis.menuItemName = @"Ανακοινώσεις Τμήματος";
        anakinosis.menuItemLink = @"http://www.cs.teilar.gr/rss";

    MainMenu *anakinosisKathigiton = [[MainMenu alloc] init];
        anakinosisKathigiton.menuItemName = @"Ανακοινώσεις Καθηγητών";
        anakinosisKathigiton.menuItemLink = @"http://www.teilar.gr/rss_ekp_news_xml.php?
tid=2";
```

```
    MainMenu *vathmoi = [[MainMenu alloc] init];
        vathmoi.menuItemName = @"Βαθμολογίες Μαθημάτων";
        vathmoi.menuItemLink = @"http://dionysos.teilar.gr/unistudent/stud_CResults.asp?
studPg=1&mnuid=mnu3&";

    MainMenu *dilosis = [[MainMenu alloc] init];
        dilosis.menuItemName = @"Δηλώσεις Εξαμήνων";
        dilosis.menuItemLink = @"http://dionysos.teilar.gr/unistudent/stud_vClasses.asp?
studPg=1&mnuid=diloseis;showDil&";

        self.menuItems = [[NSMutableArray alloc] initWithObjects:anakinosis,
anakinosisKathigiton , vathmoi, dilosis, nil];
        [anakinosis release];
        [anakinosisKathigiton release];
        [vathmoi release];
        [dilosis release];

    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [viewController release];
    [window release];
    [menuItems release];
    [super dealloc];
}
@end
```

Πρώτα περιλαμβάνουμε το αρχείο MainMenu.h. Μετά κάνουμε synthesize την μεταβλητή mainItems αφού την έχουμε δηλώσει ως property. Συνεχίζουμε δηλώνοντας και γεμίζοντας 4 διαφορετικές περιπτώσεις αντικειμένων της κλάσης MainMenu. Κάθε ένα το αρχικοποιούμε, και μετά δίνουμε στις μεταβλητές που περιέχει, τιμές. Μετά αρχικοποιούμε μια μεταβλητή

MSMutableArray και περνάμε σε αυτήν τα παραπάνω αντικείμενα. Το αντικείμενο nil που περνάμε στη μέθοδο initWithObjects: είναι σημαντικό καθώς πρέπει να είμαστε σίγουροι ότι σε αυτή την λίστα αντικειμένων, τα οποία χωρίζονται με κόμμα, το τελευταίο αντικείμενο είναι το nil αλλιώς ο κώδικάς θα δείχνει σε μη αρχικοποιημένη μνήμη και θα οδηγεί σε σφάλμα (exception). Αμέσως μετά πρέπει να απελευθερώσουμε τα 4 αντικείμενα από την μνήμη καθώς με την χρήση του self.menuItems ανεβάσαμε τον αριθμό κράτησής τους στη μνήμη (retain count). Έτσι το retain count επανέρχεται στον αριθμό 1, οπότε όταν το αντικείμενο self.menuItems απελευθερωθεί από την μνήμη θα απελευθερωθούν και τα υπόλοιπα αντικείμενα. Τέλος, απελευθερώνουμε στη μνήμη και το αντικείμενο που δημιουργήσαμε, και είμαστε υποχρεωμένοι, έτσι ώστε να μην πιάνει χώρο στη μνήμη μέχρι τον τερματισμό του προγράμματος και να μην έχουμε διαρροή μνήμης (memory leak).

4.3 Συνδέοντας τον Controller με το Model

Αφού φτιάξαμε το model τώρα πρέπει να επιστρέψουμε στη κλάση CSTeilarViewController και να δημιουργήσουμε μια γέφυρα ανάμεσα στο view controller και στο model. Για να το κάνουμε αυτό πρέπει να κάνουμε μόνο μια αλλαγή στο CSTeilarViewController.h. Προσθέτουμε έναν δείκτη σε ένα αντικείμενο NSMutableArray το οποίο θα το γεμίσουμε στη μέθοδο viewDidLoad:.

```
@interface CSTeilarViewController : UIViewController <UITableViewDataSource,
UITableViewDelegate> {

    UITableView *tableView;
    NSMutableArray *menuItems;

}
```

Οι αλλαγές που πρέπει να κάνουμε στο CSTeilarViewController.m είναι λιγάκι πιο εκτενείς. Ο κώδικας πρέπει να περιλαμβάνει και το MainMenu.h και το CSTeilarAppDelegate.h αφού θα χρησιμοποιήσουμε στοιχεία και από τις 2 κλάσεις:

```
#import "CSTeilarViewController.h"
#import "CSTeilarAppDelegate.h"
```

```
#import "MainMenu.h"
```

Όπως αναφέραμε και παραπάνω θα πρέπει να υλοποιήσουμε την μέθοδο viewDidLoad:. Η συγκεκριμένη μέθοδος καλείται μετά την φόρτωση του προγράμματος στην μνήμη και είναι συνήθως η μέθοδος που χρησιμοποιούμε για να ετοιμάσουμε τα αντικείμενα που θα προβάλει η κλάση όταν κληθεί. Βλέπουμε ότι ο Xcode έχει ήδη δημιουργήσει για μας αυτή την μέθοδο αλλά την έχει σαν σχόλιο στον κώδικα έτσι ώστε να μην περνάει από τον compiler. Αντικαθιστούμε τον κώδικα αυτόν και σιγουρευόμαστε ότι έχουμε διαγράψει τα /* και */:

```
- (void)viewDidLoad {
   [super viewDidLoad];
      CSTeilarAppDelegate *delegate = (CSTeilarAppDelegate *)[[UIApplication
sharedApplication] delegate];
      menuItems = delegate.menuItems;
}
```

Εδώ αποκτούμε μια αναφορά στο delegate της εφαρμογής μας χρησιμοποιώντας τη μέθοδο [[UIApplication sharedApplication] delegate]. Αυτή η μέθοδος επιστρέφει ένα οποιουδήποτε τύπου αντικείμενο (id object), το οποίο έπρεπε να το δηλώσουμε ως CSTeilarAppDelegate αντικείμενο πριν το αναθέσουμε. Μετά χρησιμοποιούμε έναν δείκτη στα αντικείμενα menuItems που έχουν δηλωθεί στο delegate του προγράμματος. Αφού ο κώδικας μας δηλώνει μια νέα μεταβλητή πρέπει και να την απελευθερώσουμε μέσα στη μέθοδο dealloc:

```
- (void)dealloc {
      [tableView release];
      [menuItems release];
   [super dealloc];
}
```

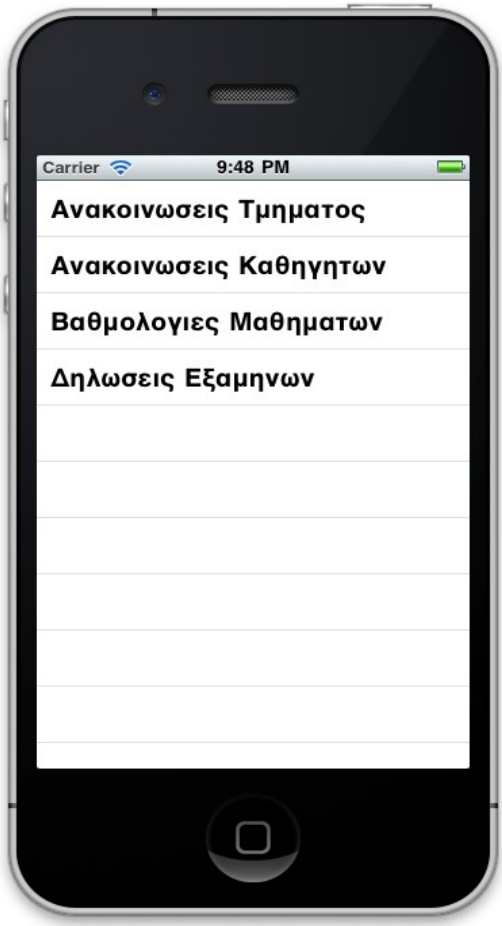
Τέλος θα πρέπει να χρησιμοποιήσουμε το model για να περάσουμε τα αντικείμενα στο table view. Ο αριθμός των σειρών στο table view πρέπει πια να καθορίζεται από τον αριθμό των menuItems στο NSMutableArray αντί να επιστρέφεται ο αριθμός “3” όλη την ώρα. Πρέπει να αλλάξουμε το περιεχόμενο της μεθοδου tableView: numberOfRowsInSection: ώστε να αντικατοπτρίζει τα παραπάνω:

```
- (NSInteger)tableView:(UITableView *)tv numberOfRowsInSection:(NSInteger)section {

    return [menuItems count];
}
```

Και πρέπει να αλλάξουμε τη μέθοδο tableView:cellForRowAtIndexPath: έτσι ώστε να παίρνει το σωστό όνομα του κάθε menuItem. Για να το καταφέρουμε αυτό πρέπει να προσθέσουμε τα παρακάτω στο κώδικά μας. Με αυτόν τον τρόπο το πρόγραμμα καταλαβαίνει πια σειρά του table προσπαθούμε να γεμίσουμε και ψάχνει το κατάλληλο στοιχείο από το menuItems:

```
- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell = [tv dequeueReusableCellWithIdentifier:@"cell"];
    if( nil == cell ) {
      cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero reuseIdentifier:@"cell"] autorelease];
    }
    MainMenu *thisMainMenu = [menuItems objectAtIndex:indexPath.row];
    cell.textLabel.text = thisMainMenu.menuItemName;
    return cell;
}
```

Φτάσαμε σε ένα σημείο στο οποίο έχουμε ένα πρόγραμμα που δουλεύει και κάνει κάτι χρήσιμο πια. Όμως ενώ το table view τώρα δείχνει το model, ακόμα δεν μπορούμε να χρησιμοποιήσουμε τα δεδομένα που έχουμε περάσει στα αντικείμενα menuItem. Όταν πατάμε σε ένα αντικείμενο του μενού θέλουμε το πρόγραμμα να μας δίνει περισσότερες πληροφορίες για αυτό, και για να το κάνουμε αυτό θα πρέπει να τροποποιήσουμε την μέθοδο tableView:didSelectRowAtIndexPath:. Αλλά για τώρα μπορούμε να πατήσουμε στο κουμπί Build and Run του Xcode ώστε να δούμε τι έχουμε ως ώρας φτιάξει μέσω του iPhone Simulator (εικόνα 4.9)



Εικόνα 4.9: Γεμίζοντας το table view με αντικείμενα μέσω του Model

4.4 Προσθέτοντας μια μπάρα καθοδήγησης στο πρόγραμμα

Τώρα πια ήρθε η ώρα να τοποθετήσουμε μια μπάρα καθοδήγησης (navigation controller) στο πρόγραμμά μας. Αυτό σημαίνει ότι θα πρέπει να προσθέσουμε έναν UINavigationController. Σε πολλές εφαρμογές που έρχονται εγκατεστημένες στο iPhone μπορείτε να δείτε αυτόν τον τρόπο διεπαφής με τον χρήστη. Είναι ένας από τους πιο διαδεδομένους τρόπους σχεδίασης ενός προγράμματος. Όταν ο χρήστης επιλέγει ένα στοιχείο από ένα table view, αυτό το στοιχείο κάνει όλο το μενού να ολισθήσει προς τα αριστερά και ένα καινούριο στοιχείο τοποθετείται στην οθόνη. Στην συνέχεια μπορούμε να επιστρέψουμε στο αρχικό μενού επιλέγοντας το κουμπί “back”.

Το πρώτο πράγμα που πρέπει να κάνουμε είναι να προσθέσουμε ένα IBOutlet σε ένα UINavigationController στο delegate του προγράμματός μας (CSTeilarAppDelegate.h):

```
#import <UIKit/UIKit.h>
```

```
@class CSTeilarViewController;

@interface CSTeilarAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    CSTeilarViewController *viewController;
        NSMutableArray *menuItems;
        UINavigationController *navController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet CSTeilarViewController *viewController;
@property (nonatomic, retain) NSMutableArray *menuItems;
@property (nonatomic, retain) IBOutlet UINavigationController *navController;

@end
```

Πρέπει επίσης να κάνουμε μερικές αλλαγές στην υλοποίηση του delegate του προγράμματος (CSTeilarAppDelegate.m). Προσθέτουμε μια νέα γραμμή στον κώδικα κάτω από τα @synthesize των υπόλοιπων μεταβλητών:

```
@synthesize window;
@synthesize viewController;
@synthesize menuItems;
@synthesize navController;
```

Τώρα πρέπει να αντικαταστήσουμε το μέρος του κώδικα που προσθέτει την βασική προβολή της κλάσης CSTeilarViewController ως υποπροβολή του κεντρικού παραθύρου (τη βασική προβολή δηλαδή του προγράμματος). Για αυτό πρέπει να διαγράψουμε την επόμενη γραμμή που βρίσκεται στο applicationDidFinishLaunching:

```
self.window.rootViewController = self.viewController;
```

Στη συνέχεια τοποθετούμε τον κώδικα που βρίσκεται παρακάτω. Αυτό που κάνουμε είναι να προσθέτουμε το CSTeilarViewController στο σωρό των προβολών (view controllers) της

μεταβλητής NavController, κάνοντας ως προβολή του δεύτερου την τρέχουσα προβολή του πρώτου. Στη συνέχεια τοποθετούμε την τρέχουσα προβολή του NavController ως υποπροβολή του κεντρικού παραθύρου. Μετά από αυτές τις αλλαγές οι τελευταίες γραμμές κώδικα της μεθοδου applicationDidFinishLaunching: πρέπει να είναι ως εξής:

```
    navController.viewControllers = [NSArray arrayWithObject:viewController];
        [window addSubview:navController.view];

    [self.window makeKeyAndVisible];
        return YES;
}
```

Καθώς η τρέχουσα προβολή του NavController αλλάζει, αυτόματα θα αλλάζει και η υποπροβολή (subview) του κεντρικού παραθύρου και κατά συνέπεια και το τι βλέπει ο χρήστης στην οθόνη. Αλλά ας συνεχίσουμε έτσι ώστε να δούμε όλο τον τρόπο λειτουργίας του NavController και έτσι θα μπορέσουμε να καταλάβουμε καλύτερα πως χειρίζεται και τον σωρό των προβολών που έχει στη διάθεσή του.

Αφού αποθηκεύσουμε τις αλλαγές μας στο Xcode, ανοίγουμε το αρχείο MainWindow.xib μέσω του Interface Builder και κάνουμε drag and drop έναν navigation controller (UINavigationController) στο κεντρικό παράθυρο με τίτλο MainWindow ή MainWindow.xib. Ο navigation controller βρίσκεται στο παράθυρο Library κάτω από τα Cocoa Touch και Controllers.

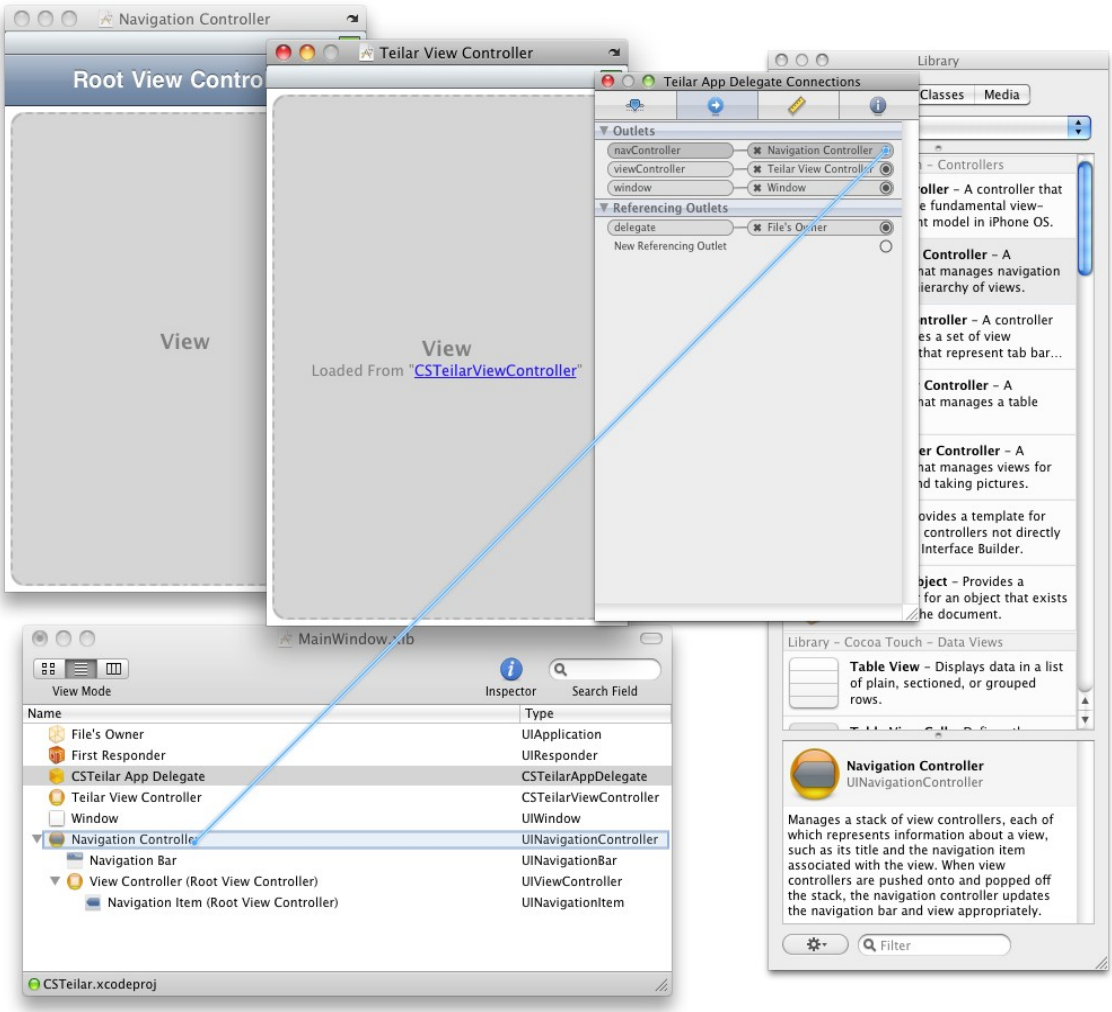
Μετά από αυτό θα πρέπει να βλέπουμε κάτι σαν αυτό που υπάρχει στην εικόνα 4.10. Προσέχουμε ότι μια μπάρα καθοδήγησης (navigation bar) υπάρχει στο πάνω μέρος του παραθύρου Navigation Controller. Αφού προσθέσαμε το UINavigationController στο MainWindow, θα πρέπει να επιλέξουμε το εικονίδιο του αρχείου CSTeilar App Delegate στο ίδιο κεντρικό παράθυρο και να επιλέξουμε το κουμπί Connections στο παράθυρο Inspector. Συνδέουμε το outlet navController με το UINavigationController όπως φαίνεται στην εικόνα 4.10.

Μετά από αυτό το βήμα, αποθηκεύουμε τις αλλαγές που κάναμε στο Interface Builder και επιστρέφουμε στο Xcode. Ανοίγουμε το αρχείο CSTeilarViewController.m και προσθέτουμε την γραμμή κώδικα που ακολουθεί στο πάνω μέρος της μεθοδου viewDidLoad:

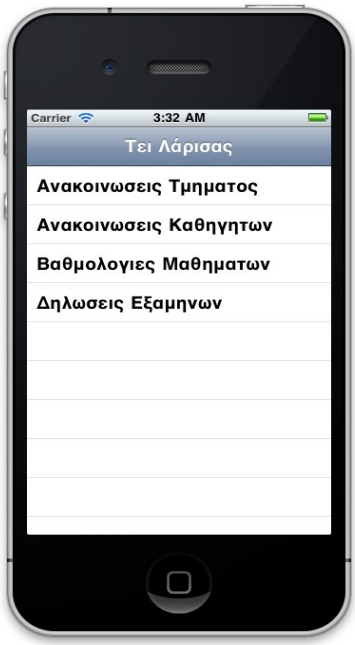
```
self.title = @"Τει Λάρισας";
```

Φτάσαμε σε ένα ακόμα καλό σημείο να δοκιμάσουμε τον κώδικά μας, οπότε πατάμε το κουμπί

Build and Run. Αν όλα έχουν πάει καλά θα πρέπει να βλέπουμε το αποτέλεσμα της εικόνας 4.11.



Εικόνα 4.10 : Η προβολή των αντικειμένων στο Interface Builder καθώς και η σύνδεση του Navigation Controller με την μεταβλητή της κλάσης



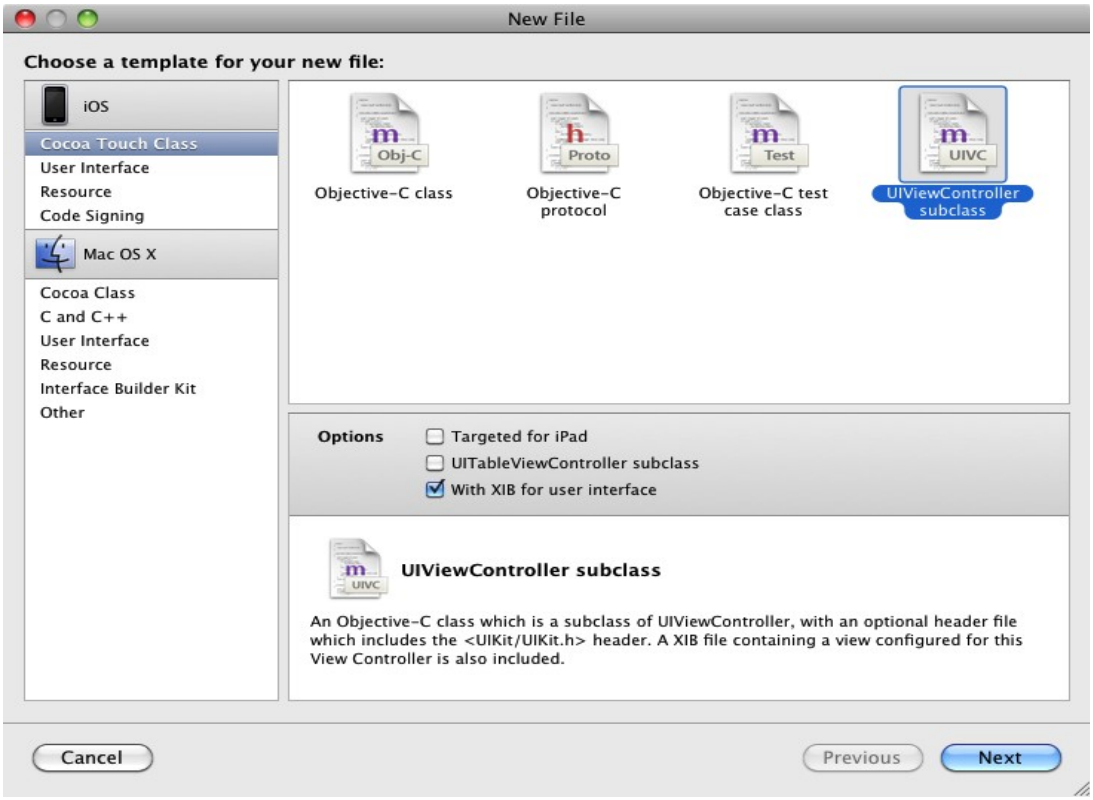
Εικόνα 4.11: Το table view με την νέα Navigation Bar

4.5 Προσθέτοντας μια προβολή για κάθε αντικείμενο του πίνακα

Μπορεί να έχουμε μια ωραία μπάρα καθοδήγησης αλλά δεν έχουμε κανένα τρόπο να μεταφερθούμε μπρος και πίσω ακόμα. Για να προσθέσουμε αυτή την λειτουργία πρέπει να δημιουργήσουμε ένα καινούριο view controller που να κατευθύνει τον χρήστη εκεί που επιλέγει κάθε φορά.

Κάνουμε δεξί κλικ στο φάκελο Classes και στη συνέχεια επιλέγουμε το Groups & Files και μετά Add και New File. Επιλέγουμε μια υποκλάση του UIViewController και ελέγχουμε ώστε να είναι επιλεγμένο το τετραγωνάκι έτσι ώστε να ζητήσουμε από τον Xcode να δημιουργήσει το αντίστοιχο .xib αρχείο, όπως βλέπουμε στην εικόνα 014. Όταν μας ζητηθεί να ονομάσουμε την νέα κλάση, δίνουμε το όνομα AnakinosisController.m αφού αυτή θα είναι η κλάση που θα δείχνει τις ανακοινώσεις με RSS feed. Θα διαπιστώσουμε ότι δημιουργήθηκαν 3 αρχεία: AnakinosisController.h, AnakinosisController.m και AnakinosisController.xib. Για ευκολία καλό είναι να μεταφέρουμε το .xib αρχείο στο φάκελο Resources έτσι ώστε να είναι στο ίδιο σημείο με τα άλλα .xib αρχεία του προγράμματος.

Αυτά τα αρχεία είναι όμως μόνο το view και το view controller των ανακοινώσεων των καθηγητών. Εμείς θέλουμε ένα διαφορετικό view και επίσης ένα διαφορετικό view controller τον



Εικόνα 4.12 : Δημιουργία των 2 κλάσεων και των αρχείων .xib

οποίο θα ανοίγουμε όταν επιλέγουμε να δούμε τις βαθμολογίες μας και την δήλωση μας από το site του dionysos.teilar.gr. Για αυτό θα πρέπει να δημιουργήσουμε άλλη μια κλάση ώστε να χειριζόμαστε αυτά τα γεγονότα. Με τον ίδιο τρόπο όπως δημιουργήσαμε την προηγούμενη κλάση, θα φτιάξουμε μια ακόμα, αυτή τη φορά με το όνομα DionisosController. Έτσι θα έχουμε 3 αρχεία: το DionisosController.h, το DionisosController.m και το DionisosController.xib.

Αυτή την στιγμή το UI (AnakinosisController.xib και DionysosController.xib) είναι απλά ένα άδειο view. Θα ασχοληθούμε όμως λίγο αργότερα με αυτό καθώς πρώτα πρέπει να προσθέσουμε κώδικα στη μέθοδο tableView:didSelectRowAtIndexPath: στο αρχείο CSTeilarViewController.m ώστε να ανοίγει, είτε την μια είτε την άλλη, προβολή ανάλογα με το ποιο στοιχείο του πίνακα επιλέγεται από το table view:

```
- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    MainMenu *thisMainMenu = [menuItems objectAtIndex:indexPath.row];
    CSTeilarAppDelegate *delegate = (CSTeilarAppDelegate *)[[UIApplication
sharedApplication] delegate];

    if (thisMainMenu.menuItemName == @"Βαθμολογίες Μαθημάτων"||
thisMainMenu.menuItemName == @"Δηλώσεις Εξαμήνων") {

      DionisosController *dionisos = [[DionisosController alloc] init];
      [delegate.navController pushViewController:dionisos animated:YES];
      [dionisos release];
    }
    else {

    AnakinosisController *anakinosis = [[AnakinosisController alloc] init];
    [delegate.navController pushViewController:anakinosis animated:YES];
    [anakinosis release];
    }
    [tv deselectRowAtIndexPath:indexPath animated:YES];
}
```

Στην αρχή δημιουργούμε ένα αντικείμενο MainMenu που το ονομάζουμε thisMainMenu στο

οποίο αποθηκεύουμε όλα τα στοιχεία του table. Στην συνέχεια φτιάχνουμε μια αναφορά στο delegate της εφαρμογής μας. Χρησιμοποιούμε μια if για να διαχωρίσουμε μέσω του ονόματος την επιλογή του αντικείμενου που θα κάνει ο χρήστης στο table view έτσι ώστε να προωθήσουμε το σωστό view. Στο τέλος ανάλογα με το τι επέλεξε ο χρήστης τρέχει ή ο κώδικας που προωθεί το AnakinosisController.xib ή το DionysosController.xib. Η προώθηση γίνεται δημιουργώντας ένα αντικείμενο view και αρχικοποιώντας το και στην συνέχεια χρησιμοποιούμε το navController μέσω του delegate και επιλέγουμε αυτή η κίνηση να γίνει animated:YES. Δεν πρέπει να ξεχάσουμε να κάνουμε release το view που μόλις χρησιμοποιήσαμε ώστε να απελευθερώσουμε την μνήμη από ότι δεν χρειαζόμαστε πια. Για να έχουμε τελειώσει με την σύνδεση των views με τον κορμό της εφαρμογής μας, πρέπει πρώτα να προσθέσουμε την δήλωση καθενός από τις νέες κλάσεις στο CSTeilarViewController.m:

```
#import "AnakinosisController.h"
#import "DionisosController.h"
```

Αυτό είναι ένα ακόμα καλό σημείο να ελέγξουμε αν όλα έχουν πάει καλά, οπότε μπορούμε να πατήσουμε το Build and Run. Αυτό που θα πρέπει να βλέπουμε είναι ότι πατώντας οποιοδήποτε στοιχείο στο table view, το view απομακρύνεται προς τα αριστερά με ωραίο τρόπο και αποκαλύπτει ένα νέο view το οποίο όμως για την ώρα είναι κενό αφού δεν έχουμε προσθέσει κάτι ακόμα. Πάνω αριστερά βλέπουμε το κουμπί το οποίο γράφει Τει Λάρισας που μπορούμε να χρησιμοποιήσουμε για να επιστρέψουμε στο αρχικό μενού.

ΚΕΦΑΛΑΙΟ 5ο : Δημιουργία της εφαρμογής – Διασύνδεση με τους δικτυακούς τόπους

5.1 Σύνδεση με τον ιστότοπο dionisos.teilar.gr

Από εδώ και πέρα πρέπει να προσθέσουμε στις κλάσεις AnakinosisController και DionisosController κώδικα για να τις γεμίσουμε με τα δεδομένα που θέλουμε να έχουν και στη συνέχεια να προσθέσουμε αυτά τα αντικείμενα στο view μας μέσω του Interface Builder.

Μπορεί αυτή την στιγμή να ανοίγουμε το σωστό view controller αλλά δεν ξέρουμε ποιο από τα στοιχεία πατιέται κάθε φορά. Για παράδειγμα όταν ο χρήστης θέλει να δει τις βαθμολογίες του ή την δήλωση των μαθημάτων τότε θα ανοίξουμε τον DionisosController. Αλλά δεν ξέρουμε ακόμα πιο από τα 2 έχει επιλέξει. Για αυτό τον λόγο θα προσθέσουμε στις δηλώσεις και των 2 views μια μεταβλητή που μας ενδιαφέρει να περαστεί σε αυτά, και η οποία είναι το menuItemLink.
Οπότε θα προσθέσουμε στα αρχεία AnakinosisController.h και DionisosController.h τα εξής:

```
@interface ...  : UIViewController {
    NSString *itemLink;
}
@property (retain, nonatomic) NSString *itemLink;

@end
```

Και θα τα κάνουμε synthesize στα αντίστοιχα αρχεία .m:

```
@synthesize itemLink;
```

Τώρα κάθε φορά που πατάμε σε ένα αντικείμενο του μενού όχι μόνο πάμε στο σωστό view αλλά περνάμε και το αντίστοιχο link του αντικείμενου.

Σειρά έχει να δηλώσουμε τα υπόλοιπα αντικείμενα που θα χρησιμοποιήσουμε στο κάθε view. Ας ξεκινήσουμε με το DionisosController το οποίο είναι αρκετά πιο απλό στη δημιουργία του. Εδώ θέλουμε ένα subview του site βαθμολογιών και της δήλωσης μας. Θα πρέπει να παίρνουμε τα link τους και να τα ανοίγουμε στο πρόγραμμά μας. Για αυτό το λόγο υπάρχει το αντικείμενο UIWebView με το οποίο μπορούμε να προβάλουμε ιστοσελίδες. Θα προσθέσουμε οπότε στο αρχείο DionisosController.h τα ακόλουθα:

```
@interface DionisosController : UIViewController {
        NSString *itemLink;
        IBOutlet UIWebView *webView;
}

@property (nonatomic, retain) UIWebView *webView;
@property (nonatomic, retain) NSString *itemLink;

@end
```

Στην συνέχεια προσθέτουμε τα ακόλουθα στο DionisosController.m:

```
@synthesize webView;
```

Αλλά αυτές οι μεταβλητές υπάρχουν για να αποθηκεύσουν τα links τα οποία βρίσκονται στο delegate της εφαρμογής μας. Πρέπει με κάποιο τρόπο να μεταφέρουμε το περιεχόμενό τους, και ο πιο απλός τρόπος είναι να το στείλουμε στις παραπάνω κλάσεις την τη στιγμή που τις καλούμε. Για αυτο πρέπει να προσθέσουμε 2 προτάσεις στη μέθοδο tableView:didSelectRowAtIndexPath: στο αρχείο CSTeilarViewController.m:

```
if (thisMainMenu.menuItemName == @"Βαθμολογίες Μαθημάτων"||
thisMainMenu.menuItemName == @"Δηλώσεις Εξαμήνων") {

          DionisosController *dionisos = [[DionisosController alloc] init];
          dionisos.itemLink = thisMainMenu.menuItemLink;
          [delegate.navController pushViewController:dionisos animated:YES];
          [dionisos release];
        }
        else {

        AnakinosisController *anakinosis = [[AnakinosisController alloc] init];
        anakinosis.itemLink = thisMainMenu.menuItemLink;
        [delegate.navController pushViewController:anakinosis animated:YES];
        [anakinosis release];
}
```

Τώρα πρέπει να γράψουμε τον κώδικά μας. Από την στιγμή που θέλουμε να τρέχει με το που ανοίξει το view, μπορούμε να χρησιμοποιήσουμε την μέθοδο viewDidLoad. Σβήνουμε τα σημεία που την κάνουν να θεωρείται σχόλιο, και προσθέτουμε τα εξής:

```
- (void)viewDidLoad {
```

```
    [super viewDidLoad];
    NSURL *url = [[NSURL alloc] initWithString:itemLink];
    NSString* result = [NSString stringWithContentsOfURL:url
encoding:NSWindowsCP1253StringEncoding error:nil];

    [webView loadHTMLString:result  baseURL:url];
    [url release];
}
```

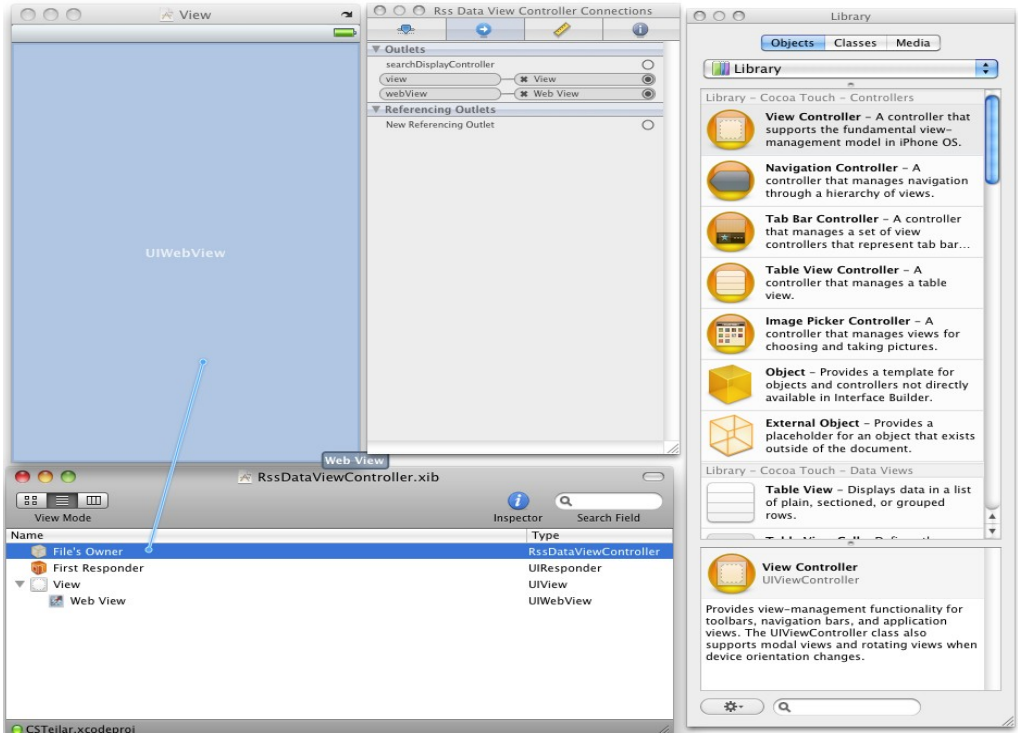
Εδώ περνάμε ως στοιχείο NSURL την διεύθυνση του αντικειμένου και στη συνέχεια παίρνουμε το αποτέλεσμα που επιστρέφει αυτή η διεύθυνση περνώντας το από το συγκεκριμένο encoder γιατί αλλιώς δεν θα φαίνεται σωστά η σελίδα μας. Τέλος το φορτώνουμε στο webView μας και απελευθερώνουμε το url. Πρέπει βεβαία να απελευθερώσουμε όποια μεταβλητή έχουμε χρησιμοποιήσει στη μέθοδο dealloc:

```
- (void)dealloc {
    [itemLink release];
    [webView release];
    [super dealloc];
}
```

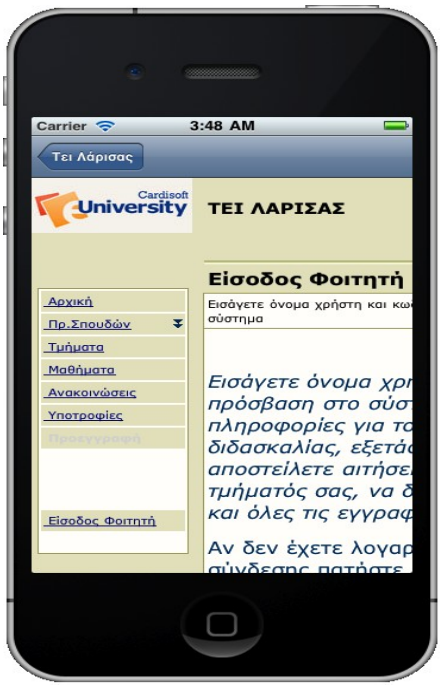
Τώρα αυτό που μένει είναι να συνδέσουμε το webView με το view μας. Κάνουμε διπλό κλικ στο αρχείο DionisosController.xib και ανοίγουμε έτσι το Interface Builder. Προσθέτουμε στο παράθυρο view ένα web view από το παράθυρο Library έτσι ώστε να πιάνει όλο το view μας.

Στη συνέχεια στο βασικό παράθυρο κάνουμε δεξί κλικ στο File's Owner και κρατώντας το πατημένο το αφήνουμε πάνω στο web view μας και επιλέγουμε από το αναδυόμενο μενού το webView που έχουμε δηλώσει πιο πριν στο κώδικά μας. Με την βοήθεια του Inspector μπορούμε να ελέγξουμε ότι το view είναι συνδεδεμένο με το File's Owner ώστε να είμαστε σίγουροι ότι το συγκεκριμένο view προωθείται όταν καλείται η κλάση μας.

Τώρα μπορούμε να πατήσουμε Build and Run ώστε να δούμε τα αποτελέσματα. Όταν διαλέγουμε κάποιες από τις ανακοινώσεις δεν υπάρχει διαφορά αφού δεν ασχοληθήκαμε καθόλου με αυτή την κλάση. Όταν όμως πατάμε στις βαθμολογίες βλέπουμε ότι όντως ανοίγει το dionysos.teilar.gr ώστε να συνδεθούμε, και αν το κάνουμε αυτό θα δούμε τις βαθμολογίες μας ή τις δηλώσεις μας, ανάλογα με το τι επιλέξαμε στην αρχή, οπότε και τα σωστά links περνάνε στο webView μας. Βέβαια ο τρόπος που το site φαίνεται δεν είναι και ο ιδανικότερος, αλλά αυτό θα το αφήσουμε για αργότερα.



Εικόνα 5.1 : Η σύνδεση του web view με την μεταβλητή της κλάσης μας



Εικόνα 5.2 : Η διασύνδεση με τον ιστότοπο dionysos.teilar.gr

5.2 Προβολή των RSS feeds

Τώρα πρέπει να περάσουμε στο άλλο view controller μας, το AnakinosisController. Εδώ θα πρέπει να φτιάξουμε ένα νέο table view το οποίο θα περιέχει τις ανακοινώσεις που θα έχουμε πάρει από το RSS feed του site cs.teilar.gr. Για να το κάνουμε αυτό θα πρέπει να δημιουργήσουμε μια νέα κλάση που μπορούμε να την ονομάσουμε Parser, η οποία θα παίρνει ως δεδομένα το link του αντικείμενου μας, που κάλεσε την κλάση AnakinosisController, και θα επιστρέφει σε αυτό όλα τα δεδομένα που αποθήκευσε από το RSS feed όπως τον τίτλο, και το περιεχόμενο, την ημερομηνία και τον αριθμό των αντικειμένων έτσι ώστε να φτιαχτεί το κατάλληλο table view.

Ας ξεκινήσουμε από την δήλωσή του, δηλαδή από το AnakinosisController.h:

```
@interface AnakinosisController : UIViewController {

        NSString *itemLink;
        UITableView *tableView;
        NSArray *items;

}
@property (nonatomic, retain) NSString *itemLink;
@property (nonatomic, retain) IBOutlet UITableView *tableView;

- (void)receivedItems:(NSArray *)theItems;
- (void)loadData;

@end
```

Εδώ προσθέσαμε το table view μας και ένα αντικείμενο NSArray για να αποθηκεύσουμε τα δεδομένα που θα μας επιστρέψει η νέα κλάση. Στο τέλος δημιουργούμε 2 μεθόδους που θα επικοινωνούν με την νέα κλάση. Η πρώτη θα επιστρέψει στο AnakinosisController τα δεδομένα που συνέλεξε και η δεύτερη θα ζητήσει από την νέα κλάση Parser να ξεκινήσει την διαδικασία συλλογής δεδομένων.

Ακολουθεί η υλοποίηση αυτής της δήλωσης που θα είναι στο AnakinosisController.m:

```
#import "AnakinosisController.h"

@implementation AnakinosisController

@synthesize itemLink;
@synthesize tableView;
- (void)viewDidLoad {
   [super viewDidLoad];
       [self loadData];
```

```
}

- (void)didReceiveMemoryWarning {

    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {
    [super viewDidUnload];
}

- (void)dealloc {
        [tableView release];
        [items release];
    [super dealloc];
}

- (void)loadData {
        if (items == nil) {
          Parser *rssParser = [[Parser alloc] init];

          [rssParser release];

        } else {
          [self.tableView reloadData];
        }

}

- (void)receivedItems:(NSArray *) theItems {
        items = theItems;
        [self.tableView reloadData];
}

#pragma mark UITableViewDataSource Methods


- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath {


    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
       cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:CellIdentifier] autorelease];
    }

        // Configure the cell.
```

```
        return cell;
}

- (NSInteger)tableView:(UITableView *)tv numberOfRowsInSection:(NSInteger)section {

}
#pragma mark Table view delegate


- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   [tv deselectRowAtIndexPath:indexPath animated:YES];
}
@end
```

Στην αρχή κάνουμε synthesize το tableView. Μετά εκτελούμε την μέθοδο loadData αμέσως μόλις η κλάση μας τελειώσει την φόρτωση της προβολής της και λίγο πιο κάτω απελευθερώνουμε τις μεταβλητές που δεν θα χρειαζόμαστε πια όταν σταματήσουμε να χρησιμοποιούμε την κλάση μας. Στην συνέχεια υλοποιούμε τις μεθόδους loadData και receivedItems:. Στην πρώτη ελέγχουμε αν την έχουμε χρησιμοποιήσει ήδη και αν όχι τότε δημιουργούμε ένα αντικείμενο της κλάσης Parser, την οποία θα δημιουργήσουμε σε λίγο, και την αρχικοποιούμε. Σε αυτό το σημείο πρέπει να περάσουμε στην Parser το link του RSS feed που θέλουμε. Για να το κάνουμε όμως αυτό χρειαζόμαστε μια άλλη μέθοδο την οποία θα δημιουργήσουμε στη νέα κλάση μας, οπότε αφήνουμε αυτό το βήμα για τώρα. Αλλά πρέπει να απελευθερώσουμε την μνήμη του αντικειμένου που δημιουργήσαμε. Στη δεύτερη αποθηκεύουμε τα αντικείμενα που συνέλεξε η κλάση Parser και ζητάμε από το table view να ανανεώσει τα περιεχόμενά του με τα νέα αντικείμενα. Οι υπόλοιπες μέθοδοι είναι αυτές που έχουμε χρησιμοποιήσει και στην κλάση CSTeilarViewController. Στην συνέχεια θα προσθέσουμε όλο τον απαραίτητο κώδικα.

5.3 Δημιουργία της κλάσης συλλογής πληροφοριών από RSS feed

Τώρα πρέπει να δημιουργήσουμε την νέα κλάση Parser για να δούμε πως αυτή θα δουλεύει. Ξεκινάμε κάνοντας δεξί κλικ στο φάκελο Classes και επιλέγουμε Add και New File. Αυτή την φορά θα επιλέξουμε ως πρότυπο το Objective-C Class καθώς η νέα κλάση δεν θα χρειαστεί αρχείο .xib αφού αυτό που θα κάνει είναι να δέχεται ένα link και να επιστρέφει τα δεδομένα που θέλουμε, και δεν θα προβάλει κάτι στην οθόνη. Οπότε τώρα έχουμε 2 νέα αρχεία. Το Parser.h και το Parser.m. Από την στιγμή που αυτή η νέα κλάση είναι εξ' ολοκλήρου δικιά μας δημιουργία, θα πρέπει να δημιουργήσουμε και την δικιά της μέθοδο delegate για να μπορούν οι άλλες κλάσεις να την

καλούν. Επίσης για διευκόλυνση μας θα χρησιμοποιήσουμε μια ήδη δημιουργημένη κλάση από την Apple, την NSXMLParser, που σκοπό έχει όπως μπορούμε να καταλάβουμε και από το όνομά της να συλλέγει πληροφορίες που εμείς ορίζουμε από XML αρχεία.

5.3.1 Η δήλωσή της

Ας δούμε την δήλωσή μας στο Parser.h:

```
#import <Foundation/Foundation.h>

@protocol ParserDelegate <NSObject>
- (void)receivedItems:(NSArray *)theItems;
@end
@interface Parser : NSObject <NSXMLParserDelegate> {

	id _delegate;

	NSMutableData *responseData;
	NSMutableArray *items;
	NSMutableDictionary *item;
	NSString *currentElement;
	NSMutableString * currentTitle, * currentDate, * currentSummary, * currentLink;
}

@property (nonatomic, retain) NSMutableData *responseData;
@property (nonatomic, retain) NSMutableArray *items;
@property (nonatomic, retain) NSMutableString *currentTitle;
@property (nonatomic, retain) NSMutableString *currentDate;
@property (nonatomic, retain) NSMutableString *currentSummary;
@property (nonatomic, retain) NSMutableString *currentLink;


- (void)parseRssFeed:(NSString *)url withDelegate:(id)aDelegate;

- (id)delegate;
- (void)setDelegate:(id)new_delegate;

@end
```

Στην αρχή δηλώνουμε την μέθοδο delegate για να μπορούμε να την καλούμε από άλλες κλάσεις. Μετά ξεκινάμε την δήλωση των μεταβλητών που θα χρειαστούμε. Μια μεταβλητή delegate που θα παίρνουμε από την κλάση που την κάλεσε, την responseData όπου θα αποθηκεύεται το αποτέλεσμα της κλήσης του διαδικτυακού τόπου το οποίο θα είναι ένα RSS feed.

Μια μεταβλητή NSMutableArray που ονομάσαμε items θα αποθηκεύει το κάθε ένα feed που με την σειρά του θα αποτελείται από μια μεταβλητή NSMutableDictionary στην οποία θα αποθηκεύονται τα στοιχεία του. Ακολουθούν οι μεταβλητές που θα αποθηκεύουν κάθε φορά τα διάφορα χαρακτηριστικά κάθε στοιχείου του RSS feed. Χρησιμοποιούμε το @property για να μπορούμε να αλλάζουμε αυτές τις μεταβλητές. Τέλος δηλώνουμε τις μεθόδους που θα χρησιμοποιήσουμε. Η πρώτη είναι η μέθοδος που θα καλέσουμε από το AnakinosisController και θα περάσουμε το link του RSS feed ενώ οι άλλες 2 υπάρχουν για να μας διευκολύνουν στην διαχείριση των delegates. Εδώ πρέπει να σημειωθεί ότι οι ονομασίες των μεταβλητών αλλά και ο τρόπος που είναι φτιαγμένη η κλάση Parser είναι ένας γενικός τρόπος για να μπορούμε να παίρνουμε τα δεδομένα ενός RSS feed και αυτό το κάνουμε έτσι ώστε να είναι δυνατή η χρησιμοποίηση της σε άλλες ανάλογες περιπτώσεις, και για να ισχύσει κάτι τέτοιο πρέπει η κλάση να είναι όσο το δυνατόν πιο ανεξάρτητη. Η επαναχρησιμοποίηση τέτοιων κλάσεων έχει μεγάλη σημασία στον προγραμματισμό, και το μοντέλο MCV στο οποίο βασίζεται ο προγραμματισμός στο iPhone υποστηρίζει αυτή την προσέγγιση.
Ας δούμε πως υλοποιούνται αυτά στο αρχείο Parser.h.

5.3.2 Υλοποίηση Διασύνδεσης

Για να είναι πιο εύκολο να παρακολουθήσουμε το τι κάνει κάθε μέρος του κώδικα, θα δούμε το πρώτο μέρος της κλάσης που ασχολείται με το synthesize των μεταβλητών και με την πραγματοποίηση σύνδεσης μέσω του link με τον ιστότοπο:

```
#import "Parser.h"

@implementation Parser

@synthesize items, responseData;
@synthesize currentTitle;
@synthesize currentDate;
@synthesize currentSummary;
@synthesize currentLink;

- (void)parseRssFeed:(NSString *)url withDelegate:(id)aDelegate {
    [self setDelegate:aDelegate];

    responseData = [[NSMutableData data] retain];
    NSURL *baseURL = [[NSURL URLWithString:url] retain];
    NSURLRequest *request = [NSURLRequest requestWithURL:baseURL];
```

```
        [[[NSURLConnection alloc] initWithRequest:request delegate:self] autorelease];
        [baseURL release];
}

- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse
*)response
{
    [responseData setLength:0];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
{
    [responseData appendData:data];
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error
{
    NSString * errorString = [NSString stringWithFormat:@"Unable to download xml data (Error
code %i )", [error code]];
        UIAlertView * errorAlert = [[UIAlertView alloc] initWithTitle:@"Error loading content"
message:errorString delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [errorAlert show];
        [errorAlert release];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
        self.items = [[NSMutableArray alloc] init];
        NSXMLParser *rssParser = [[NSXMLParser alloc] initWithData:responseData];
        [rssParser setDelegate:self];
        [rssParser parse];
}
```

Στην αρχή κάνουμε synthesize τις μεταβλητές μας. Στην συνέχεια καλούμε την μέθοδο parseRssFeed:withDelegate ώστε να αποθηκεύσουμε το delegate της κλάσης που κάλεσε αυτή τη μέθοδο. Στην ίδια μέθοδο αρχικοποιούμε το responceData και δημιουργούμε μια μεταβλητή NSURL όπου αποθηκεύουμε το link. Με την βοήθεια της μεθοδου NSURLRequest πραγματοποιούμε μια σύνδεση τον ιστότοπο και στην συνέχεια καλούμε την κλάση NSURLRConnection, η οποία με τις μεθόδους της μας επιτρέπει να ελέγξουμε την κατάσταση της σύνδεσής μας με τον ιστότοπο, και απελευθερώνουμε την μεταβλητή baseURL. Μετά υλοποιούμε 4 μεθόδους της κλάσης NSURLConnection. Η πρώτη είναι η connection:didReceiveResponce στην οποία θέτουμε το μήκος του NSArray responseData ίσο με μηδέν για να είμαστε σίγουροι ότι είναι άδειο από στοιχεία. Η δεύτερη είναι η connection:didReceiveData όπου περνάμε τα δεδομένα που επέστρεψε η μέθοδος στο

responseData. Η τρίτη είναι η connection:didFailWithError μέσω της οποίας μπορούμε να ειδοποιήσουμε τον χρήστη για πιθανόν πρόβλημα στην σύνδεση. Στην προκείμενη δημιουργούμε ένα UIAlertView με τα περιεχόμενα του λάθους που μας επέστρεψε η μεταβλητή. Αυτό έχει ως αποτέλεσμα ένα παράθυρο λάθους να εμφανιστεί στην οθόνη του χρήστη. Η τελευταία μέθοδος είναι η connectionDidFinishLoading που εκτελείται όταν η σύνδεση είναι επιτυχής. Εδώ αρχικοποιούμε τη μεταβλητή items και δημιουργούμε ένα αντικείμενο της κλάσης NSXMLParser το rssParser το οποίο αρχικοποιούμε με τα δεδομένα του responseData και στην συνέχεια τοποθετούμε ως μέθοδο delegate το delegate της κλάσης μας. Τέλος περνώντας στο rssParser το μήνυμα parse ξεκινάμε την διαδικασία της κλάσης NSXMParser.

5.3.3 Υλοποίηση συλλογής δεδομένων

Τώρα ακολουθεί η υλοποίηση των μεθόδων της NSXMLParser:

```
#pragma mark rssParser methods
- (void)parserDidStartDocument:(NSXMLParser *)parser {
}

- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName namespaceURI:
(NSString *)namespaceURI qualifiedName:(NSString *)qName attributes:(NSDictionary
*)attributeDict{
        currentElement = [elementName copy];

    if ([elementName isEqualToString:@"item"]) {
        item = [[NSMutableDictionary alloc] init];
        self.currentTitle = [[NSMutableString alloc] init];
        self.currentDate = [[NSMutableString alloc] init];
        self.currentSummary = [[NSMutableString alloc] init];
        self.currentLink = [[NSMutableString alloc] init];
    }
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName namespaceURI:
(NSString *)namespaceURI qualifiedName:(NSString *)qName{

    if ([elementName isEqualToString:@"item"]) {
        [item setObject:self.currentTitle forKey:@"title"];
        [item setObject:self.currentLink forKey:@"link"];
        [item setObject:self.currentSummary forKey:@"summary"];
        [item setObject:self.currentDate forKey:@"pubDate"];
        [items addObject:item];
        }
```

```
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{
   if ([currentElement isEqualToString:@"title"]) {
      [self.currentTitle appendString:string];
   } else if ([currentElement isEqualToString:@"link"]) {
      [self.currentLink appendString:string];
   } else if ([currentElement isEqualToString:@"description"]) {
      [self.currentSummary appendString:string];
   } else if ([currentElement isEqualToString:@"pubDate"]) {
         [self.currentDate appendString:string];
         NSCharacterSet* charsToTrim = [NSCharacterSet
characterSetWithCharactersInString:@" \n"];
         [self.currentDate setString: [self.currentDate stringByTrimmingCharactersInSet:
charsToTrim]];
      }
}

- (void)parserDidEndDocument:(NSXMLParser *)parser {
      if ([_delegate respondsToSelector:@selector(receivedItems:)])
      [_delegate receivedItems:items];
       else
      {
      [NSException raise:NSInternalInconsistencyException
      format:@"Delegate doesn't respond to receivedItems:"];
      }
 }
```

Εδώ υλοποιούμε 5 μεθόδους της NSXMLParser. Στην πρώτη δεν χρειάζεται να κάνουμε κάτι άλλα υπάρχει έτσι ώστε ο κώδικάς μας να μπορεί να επαναχρησιμοποιηθεί σε άλλες αντίστοιχες περιπτώσεις, όπου και θα μπορούσε να χρειαστεί. Η δεύτερη η οποία ονομάζεται parser:didStartElement εκτελείται κάθε φορά που βρίσκουμε ένα tag αντικειμένου στο rssParser. Εάν το αντικείμενο έχει όνομα “item” τότε πρέπει να αποθηκεύσουμε τα στοιχεία του, οπότε και αρχικοποιούμε τα δεδομένα των μεταβλητών μας και δημιουργούμε ένα αντικείμενο NSMutableDictionary στο οποίο θα αποθηκευτούν. Η τρίτη ονομάζεται parser:didEndElement και εκτελείται μόλις βρεθεί tag που δηλώνει το τέλος περιγραφής ενός αντικειμένου οπότε σε αυτή την περίπτωση και εφόσον έχουμε να κάνουμε με ένα αντικείμενο με όνομα “item” γίνεται η αποθήκευση των χαρακτηριστικών του στη μεταβλητή NSMutableDictionary που δημιουργήσαμε προηγουμένως. Ακολουθεί κώδικας για την σωστή επεξεργασία της ημερομηνίας που, ανάλογα με το πιο τρόπο παρουσίασης έχει σε κάθε feed, θα τρέχει έτσι ώστε πάντα να παρουσιάζεται το με τον ίδιο τρόπο από την εφαρμογή μας. Όλα αυτά τα αντικείμενα τα περνάμε στο NSMutableDictionary, το οποίο και αυτό με την σειρά του το αποθηκεύουμε στο NSMutableArray. Η τέταρτη μέθοδο ονομάζεται parser:foundCharacters και είναι αυτή που αποθηκεύει στο κάθε στοιχείο, του κάθε

αντικειμένου, το περιεχόμενο του. Τελευταία μέθοδος είναι η parser:didEndDocument η οποία καλείται μόλις τελειώσει το περιεχόμενο της μεταβλητής που επεξεργαζόμαστε. Εδώ ελέγχουμε αν έχει δημιουργηθεί σωστά η μέθοδος delegate και περνάμε τα περιεχόμενα της μεταβλητής items που περιέχει και όλα τα στοιχεία και τα χαρακτηριστικά τους από όλη αυτή την διαδικασία.

Το τελευταίο κομμάτι του κώδικα του αρχείου Parser.m είναι το εξής:

```
#pragma mark Delegate methods

- (id)delegate {
	return _delegate;
}

- (void)setDelegate:(id)new_delegate {
	_delegate = new_delegate;
}

- (void)dealloc {

	[item release];
	[items release];
	[responseData release];
	[super dealloc];
}
@end
```

Εδώ απλά υλοποιούμε τις μεθόδους delegate που χρησιμοποιήσαμε παραπάνω για την δική μας διευκόλυνση. Όπως σε κάθε κλάση, απελευθερώνουμε τις μεταβλητές που δημιουργήσαμε στη μέθοδο dealloc.

5.4 Σύνδεση της κλάσης συλλογής δεδομένων με τις κλάσεις προβολής τους

Τώρα έχουμε να κάνουμε μερικά πράγματα ώστε να χρησιμοποιεί η εφαρμογή μας την νέα κλάση. Πρέπει πρώτα να συνδέσουμε την κλάση μας με την κλάση που θα την καλεί και έτσι και με τον κορμό της εφαρμογής μας και θα χρειαστεί να δημιουργήσουμε άλλη μια κλάση ώστε κάθε φορά που ο χρήστης επιλέγει ένα αντικείμενο RSS feed να μεταφερόμαστε σε μια νέα προβολή με το περιεχόμενό του.

Ας δούμε τι χρειάζεται για να συνδέσουμε την κλάση AnakinosisController με την Parser. Στο αρχείο AnakinosisController.m και στη μέθοδο loadData: είναι το σημείο στο οποίο πρέπει να καλέσουμε την μέθοδο parseRssFeed:withDelegate οπότε ο κώδικας της μεθοδου θα είναι ο εξής:

```
- (void)loadData {
        if (items == nil) {
          Parser *rssParser = [[Parser alloc] init];
          [rssParser parseRssFeed:itemLink withDelegate:self];
          [rssParser release];

        } else {
          [self.tableView reloadData];
        }
}
```

Τώρα πρέπει να μεταφερθούμε στις μεθόδους UITableViewDataSource ώστε να ορίσουμε το περιεχόμενο του table view μας σύμφωνα με τα feeds που επιστρέφονται από την κλάση Parser. Οπότε το περιεχόμενο της μεθοδου tableView:cellForRowAtIndexPath διαμορφώνεται ως εξής:

```
- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath
*)indexPath {
        static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tv dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
    }
        // Configure the cell.
        cell.textLabel.text = [[items objectAtIndex:indexPath.row] objectForKey:@"title"];
        cell.detailTextLabel.text = [[items objectAtIndex:indexPath.row] objectForKey:@"pubDate"];
        return cell;
}
```

Ορίζουμε ως τίτλο του κάθε κελιού του table view τον τίτλο του κάθε feed και επιλέγουμε τον τρόπο παρουσίασης της ημερομηνίας σε κάθε κελί.
Στην μέθοδο tableView:numberOfRowsInSection: επιστρέφουμε τον αριθμό των feeds ώστε να δημιουργηθεί ο ίδιος αριθμός κελιών:

```
- (NSInteger)tableView:(UITableView *)tv numberOfRowsInSection:(NSInteger)section {
    return [items count];
}
```

Τέλος ακολουθεί η μέθοδος tableView:didSelectRowAtIndexPath: στην οποία πρέπει να ορίσουμε τι πρέπει να συμβεί αν ο χρήστης επιλέξει κάποιο από τα feeds που βρίσκονται στο table view. Εδώ θα πρέπει να χρησιμοποιήσουμε το όνομα μιας νέας κλάσης που θα προβάλει τα

περιεχόμενα των feeds:

```
- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
        NSDictionary *theItem = [items objectAtIndex:indexPath.row];
        CSTeilarAppDelegate *delegate = (CSTeilarAppDelegate *)[[UIApplication
sharedApplication] delegate];
        RssDataViewController *navController = [[RssDataViewController alloc]
initWithItem:theItem];
        [delegate.navController pushViewController:navController animated:YES];
        [navController release];
        [tv deselectRowAtIndexPath:indexPath animated:YES];
}
```

Όπως είδαμε στην κλάση Parser, τα δεδομένα κάθε feed αποθηκεύονται σε μεταβλητή τύπου NSDictionary οπότε για να πάρουμε αυτά τα δεδομένα και να τα δώσουμε στη νέα κλάση που θα τα προβάλλει δημιουργούμε μια μεταβλητή theItem. Επίσης δημιουργούμε και μια μεταβλητή delegate για να ζητήσουμε από το navController να ωθήσει την νέα προβολή μπροστά. Η κλάση που πρέπει να δημιουργήσουμε είναι η RssDataViewController.

Τώρα πρέπει με τον γνωστό πια τρόπο να δημιουργήσουμε μια ακόμα κλάση με πρότυπο το UIViewController Subclass. Έτσι θα έχουμε 3 νέα αρχεία: το RssDataViewController.h, το RssDataViewController.m και το RssDataViewController.xib. Αυτό που θέλουμε να κάνει η νέα μας κλάση είναι να παρουσιάζει τα δεδομένα κάθε φορά που επιλέγει ο χρήστης ένα RSS feed από το table view. Ας δούμε την δήλωση στο αρχείο RssDataViewController:

```
@interface RssDataViewController : UIViewController {

        NSDictionary *item;
        NSString *itemTitle;
        NSString *itemDate;
        NSString *itemSummary;
        IBOutlet UIWebView *webView;

}

@property (nonatomic, retain) NSDictionary *item;
@property (nonatomic, retain) NSString *itemTitle;
@property (nonatomic, retain) NSString *itemDate;
@property (nonatomic, retain) NSString *itemSummary;
@property (nonatomic, retain) IBOutlet UIWebView *webView;

- (id)initWithItem:(NSDictionary *)theItem;

@end
```

Δηλώνουμε μια μεταβλητή NSDictionary για την αποθήκευση των στοιχείων του feed και άλλες τρεις μεταβλητές για την αποθήκευση των επιμέρους χαρακτηριστικών του όπως ο τίτλος, η ημερομηνία και η περιγραφή του, ενώ δηλώνουμε και μια μέθοδο που σκοπό έχει την μεταφορά των χαρακτηριστικών του feed από την κλάση AnakinosisController σε αυτή που δημιουργούμε τώρα.

Ας δούμε και την υλοποίησή της:

```
#import "RssDataViewController.h"

@implementation RssDataViewController

@synthesize item, itemTitle, itemDate, itemSummary;
@synthesize webView;

- (id)initWithItem:(NSDictionary *)theItem {
        self.item = theItem;

        return self;
}

- (void)viewDidLoad {
   [super viewDidLoad];

        self.itemTitle = [item objectForKey:@"title"];
        self.itemDate = [item objectForKey:@"pubDate"];
        self.itemSummary = [item objectForKey:@"summary"];

        NSString *result = [NSString stringWithFormat:@"<B>%@</B> <br><br>%@   <br><br>
%@", self.itemTitle, self.itemDate, self.itemSummary];

        [self.webView loadHTMLString:result baseURL:nil];
        [webView release];
}

- (void)didReceiveMemoryWarning {

   [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {

}

- (void)dealloc {
        [item release];
        [itemTitle release];
```

```
        [itemDate release];
        [itemSummary release];
    [super dealloc];
}
@end
```

Αφού κάνουμε synthesize τις μεταβλητές μας, υλοποιούμε την μέθοδο μας απλά αποθηκεύοντας τα δεδομένα στην μεταβλητή που έχουμε δηλώσει σε αυτή την κλάση και γράφουμε τον υπόλοιπο κώδικά μας στην μέθοδο viewDidLoad ώστε να εκτελεστεί μόλις φορτωθεί η νέα προβολή. Αυτή την φορά θα αποθηκεύσουμε όλα τα χαρακτηριστικά από τα feed σε μια μεταβλητή NSString με όνομα result και στην συνέχεια θα την φορτώσουμε στο webView μας για να παρουσιαστεί στον χρήστη.

Μπορεί να φτιάξαμε όλες τις κλάσεις που χρειαζόμαστε αλλά για να μπορεί η μια να καλεί τις μεθόδους της άλλης πρέπει να περιλαμβάνονται στην αρχή της υλοποίησης. Για αυτό θα προσθέσουμε στην κορυφή του αρχείου AnakinosisController.m τα εξής:

```
#import "CSTeilarAppDelegate.h"
#import "Parser.h"
#import "RssDataViewController.h"
```

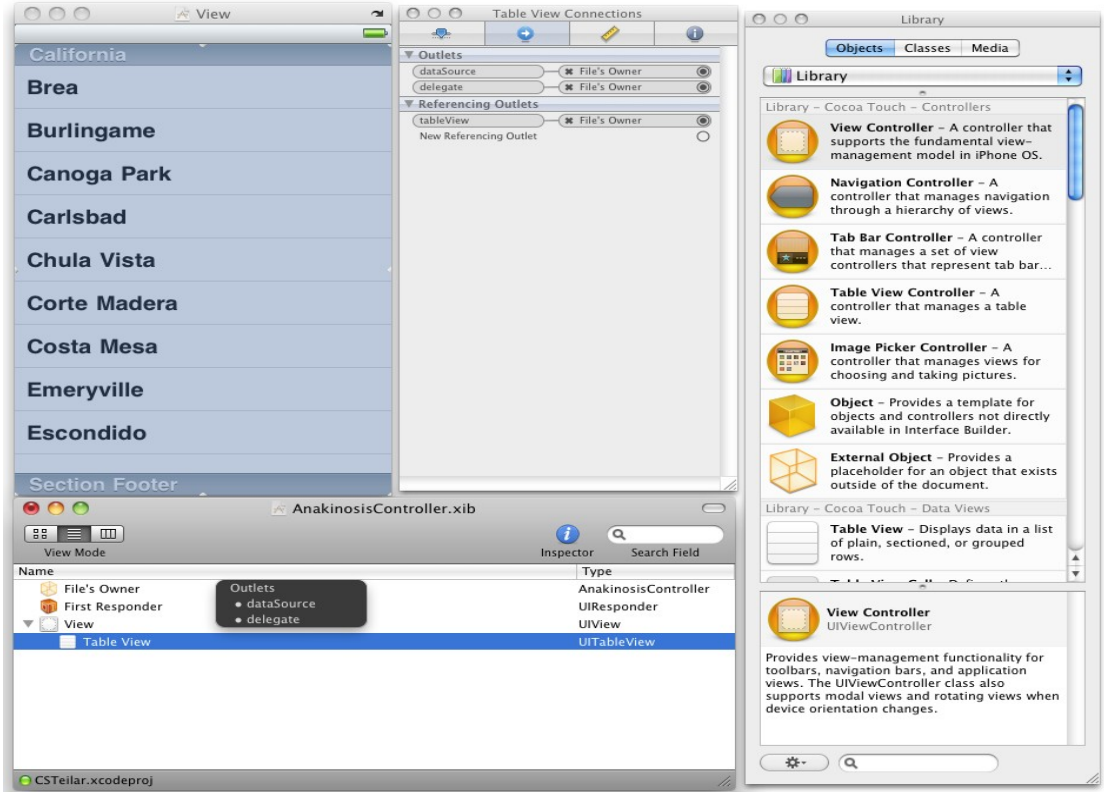
Τώρα είμαστε έτοιμοι να συνδέσουμε όλα αυτά τα στοιχεία με το περιβάλλον χρήστη μέσω του Interface Builder. Ας ξεκινήσουμε ανοίγοντας το αρχείο AnakinosisController.xib. Στο περιβάλλον χρήσης του προγράμματος, παίρνουμε ένα table view από το παράθυρο Library και με drag and drop και το αφήνουμε στο παράθυρο view, προσαρμόζοντας το σε όλο το παράθυρο καθώς όλη η προβολή αυτής της κλάσης θα είναι ένα table view. Στη συνέχεια πρέπει να ενώσουμε το table view με tableView του κώδικά μας.

Για να το καταφέρουμε αυτό πρέπει να πατήσουμε με δεξί κλικ στο αρχείο File's Owner στο παράθυρο AnakinosisController.xib και να το κρατήσουμε σέρνοντας το στο table view του παράθυρου view. Αφήνοντάς το θα επιλέξουμε από το μενού που εμφανίζεται την μεταβλητή tableView. Με τον ίδιο τρόπο θα ενώσουμε το αρχείο Table View του παραθύρου AnakinosisController.xib με το File's Owner και θα επιλέξουμε και τις 2 επιλογές που μας δίνει το μενού που εμφανίζεται.

Αποθηκεύουμε τις αλλαγές και ανοίγουμε το επόμενο αρχείο .xib, το RssDataViewController.xib το οποίο θα ανοίγει κάθε φορά που επιλέγεται ένα αντικείμενο από την προηγούμενη κλάση και θα προβάλει τις λεπτομέρειες του feed.

Κάνουμε διπλό κλικ στο αρχείο RssDataViewController.xib το οποίο ανοίγει με το Interface Builder. Εδώ πρέπει να χρησιμοποιήσουμε ένα web view οπότε παίρνουμε ένα από το παράθυρο Library και το σύρουμε στο παράθυρο View τοποθετώντας το έτσι ώστε να πιάνει όλη την προβολή μας. Τώρα συνδέουμε το αρχείο File's Owner όπως και στην προηγούμενη κλάση με δεξί κλικ και επιλέγουμε από το μενού που εμφανίζεται τη μεταβλητή webView που έχουμε δημιουργήσει στην κλάση αυτή. Αποθηκεύουμε και κλείνουμε το πρόγραμμα.

Εφόσον όλες αυτές οι πολλές αλλαγές που έχουμε κάνει έχουν γίνει σωστά έχουμε πια ένα πλήρως λειτουργικό πρόγραμμα. Τρέχοντας το μπορούμε να δούμε όλες τις λειτουργίες του. Πατώντας το Build and Run μπορούμε να δούμε στο iPhone Simulator όλες τις λειτουργίες.



Εικόνα 4.15 : Οι συνδέσεις του AnakinosisController με τις μεθόδους delegate της εφαρμογής μας

ΚΕΦΑΛΑΙΟ 6ο : Δημιουργία της εφαρμογής – Βελτιώσεις στην προβολή και συμπεράσματα

6.1 Βελτιώσεις στην προβολή της εφαρμογής

Όλα προγραμματιστικά να δουλεύουν σωστά αλλά ο τρόπος προβολής τους δεν είναι πολύ ωραίος και εύχρηστος. Όμως ένα από τα πιο χρήσιμα γνωρίσματα μιας εφαρμογής που έχει δημιουργηθεί με το μοντέλο MVC είναι ότι όλα τα χαρακτηριστικά είναι τόσο σωστά χωρισμένα που μπορούμε να αλλάξουμε την προβολή της εφαρμογής μας με απλές και μικρές αλλαγές είτε στον κώδικα είτε στο Interface Builder.

6.1.1 Αρχική οθόνη και ένδειξη προώθησης επιλογής

Ένα από τα βασικά πράγματα που πρέπει να έχει μια εφαρμογή είναι να δίνει άμεσα στον χρήστη τα δεδομένα που χρειάζεται ώστε να ξέρει αμέσως πως να την χρησιμοποιήσει. Στην αρχική οθόνη της εφαρμογής δεν έχουμε καμία ένδειξη ότι αυτές οι επιλογές θα μας προωθήσουν σε ένα νέο στοιχείο αλλά και από αισθητικής άποψης φαίνεται κάπως μονότονη λόγω του λευκού χρώματος.

Για να βελτιώσουμε την εικόνα της εφαρμογής μας πάνω σε αυτούς τους τομείς πρέπει να προσθέσουμε μια γραμμή κώδικα στο αρχείο CSTeilarViewController.m, στην μέθοδο tableView:cellForRowAtIndexPath: και στο σημείο που ορίζουμε την ρύθμιση του κελιού:

```
// Configure the cell.
MainMenu *thisMainMenu = [menuItems objectAtIndex:indexPath.row];
cell.textLabel.text = thisMainMenu.menuItemName;
cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
return cell;
```

Το ίδιο πρέπει να κάνουμε και στο άλλο table view της εφαρμογής μας το οποίο βρίσκεται στο αρχείο AnakinosisController.m. Οπότε θα μεταφερθούμε στη μέθοδο tableView:cellForRowAtIndexPath: και θα αλλάξουμε τον κώδικα ώστε να έχουμε το εξής αποτέλεσμα:

```
// Configure the cell.

cell.textLabel.text = [[items objectAtIndex:indexPath.row] objectForKey:@"title"];
cell.detailTextLabel.text = [[items objectAtIndex:indexPath.row] objectForKey:@"pubDate"];
cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
 return cell;
```

Από ότι βλέπουμε στην ουσία στέλνουμε ένα μήνυμα στη μέθοδο του κελιού που ονομάζεται accessoryType.

Στην συνέχεια θέλουμε να κάνουμε το αρχικό μενού να φαίνεται πιο ωραίο και διαδραστικό. Μια απλή λύση είναι να ανοίξουμε το αρχείο που περιλαμβάνει την προβολή του αρχικού μενού μας, δηλαδή το CSTeilarViewController.xib μέσω του Interface Builder. Εκεί θα επιλέξουμε το table view στο παράθυρο View και στη συνέχεια θα μεταφερθούμε στο παράθυρο Inspector και στην καρτέλα attributes θα επιλέξουμε από το αναδυόμενο παράθυρο ρύθμισης Style την επιλογή Grouped.

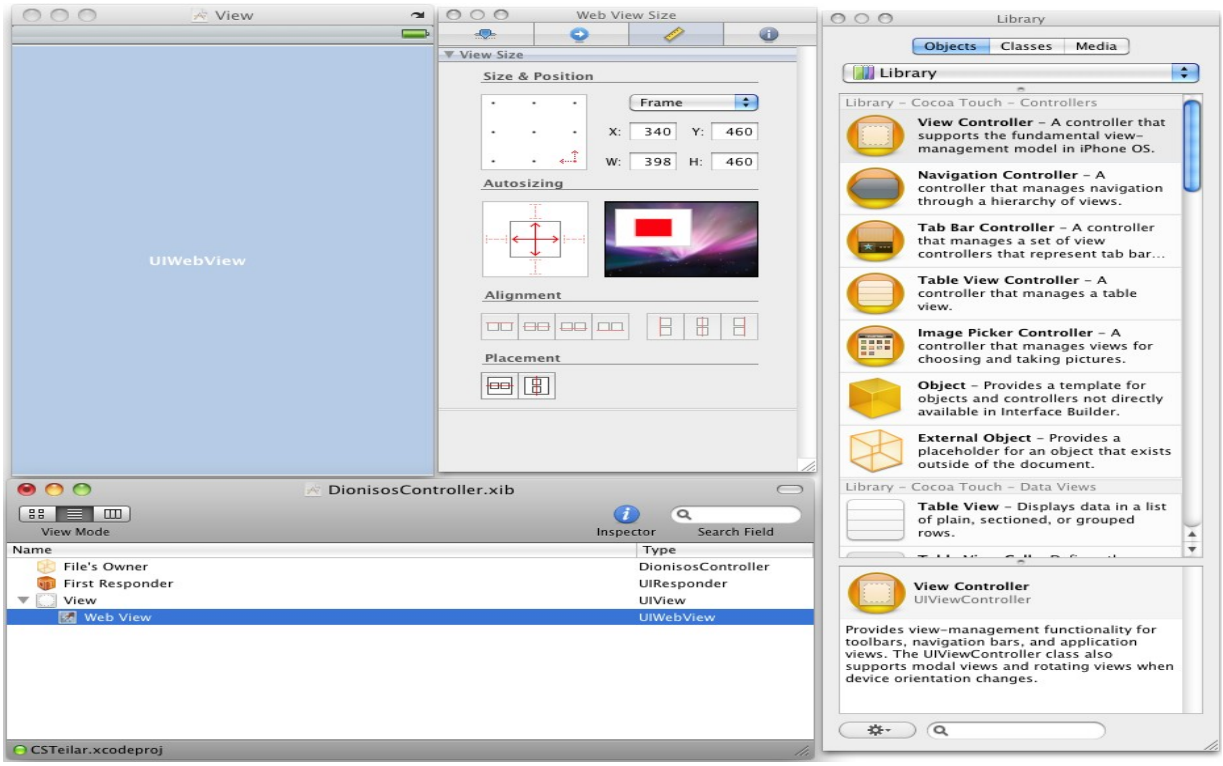
6.1.2 Η προβολή του ιστότοπου dionysos.teilar.gr

Ένα άλλο στοιχείο της εφαρμογής που μπορούμε με απλές ρυθμίσεις να διορθώσουμε είναι το πως θα φαίνεται πιο εύκολα και πιο ωραία η ιστοσελίδα dionysos.teilar.gr. Ανοίγοντας το αρχείο DionisosController.xib με τον Interface Builder επιλέγουμε το web view στο παράθυρο View και μετά στο παράθυρο Inspector διαλέγουμε την καρτέλα attributes και δίπλα από την ρυθμίσεις Scaling τσεκάρουμε το κουτάκι με την ένδειξη Scales Page To Fit. Στην συνέχεια επιλέγουμε την καρτέλα Size και ρυθμίζουμε τα αντίστοιχα κουτάκια με τους αριθμούς που ακολουθούν: x:340 , y:460 , w:398 , h: 460. Με αυτο τον τρόπο ρυθμίζουμε το πως θα φαίνεται το Web View στην προβολή μας. Τέλος ρυθμίζουμε και τις επιλογές Autoresizing της ίδιας καρτέλας όπως φαίνεται και στην παρακάτω εικόνα (εικόνα 027). Αυτό το κάνουμε για να ρυθμίσουμε την συμπεριφορά του web view όταν η συσκευή μας αλλάζει τον τρόπο προβολής της από κάθετη σε οριζόντια. Πολλές εφαρμογές επωφελούνται όταν χρησιμοποιείται η οριζόντια προβολή των δεδομένων τους έτσι ώστε να εκμεταλλεύονται όλο το πλάτος της οθόνης. Για να μπορεί η εφαρμογή να κάνει χρήση αυτής της λειτουργίας πρέπει να χρησιμοποιήσουμε μια μέθοδο στην υλοποίηση της συγκεκριμένης κλάσης. Οπότε στο αρχείο DionisosController.m προσθέτουμε τον παρακάτω κώδικα:

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
   // Return YES for supported orientations.
```

```
    return YES;
}
```

Έτσι η εφαρμογή μας όταν βρίσκεται στην συγκεκριμένη προβολή επιτρέπει την αλλαγή προβολής ανάλογα με το πως ο χρήστης κρατάει την συσκευή του



Εικόνα 6.1 : Οι αλλαγές στο .xib αρχείο της κλάσης DionisosController

Το ίδιο μπορούμε να κάνουμε και για όλες τις άλλες προβολές, δηλαδή στην προβολή των RSS feeds καθώς μερικά από αυτά περιέχουν συνημμένα αρχεία που μπορούν να διαβαστούν πολύ πιο εύκολα όταν η συσκευή τα προβάλλει οριζοντίως, αλλά και στις 2 προβολές των table views και έτσι να επιτρέπουμε την οριζόντια προβολή της εφαρμογής καθ' όλη την διάρκεια της χρησιμοποίησης της. Οπότε θα τοποθετήσουμε την ίδια ακριβώς μέθοδο και στα αρχεία RssDataViewController.m, AnakinosisController.m και CSTeilarViewController.m.

6.1.3 Τίτλοι στην μπάρα πλοήγησης

Ένα στοιχείο που θα κάνει πιο εύκολη την πλοήγηση της εφαρμογής είναι μέσω των τίτλων στην μπάρα πλοήγησης. Έτσι ο χρήστης θα ξέρει κάθε φορά σε ποια προβολή βρίσκεται και τι ακριβώς υπάρχει στο προηγούμενο μενού. Για να το καταφέρουμε αυτο θα πρέπει να προσθέσουμε μια γραμμή κώδικα σε κάθε αρχείο που υλοποιεί μια κλάση για προβολή των δεδομένων.

Εάν προσέξουμε στην εφαρμογή μας θα δούμε ότι στην αρχική οθόνη υπάρχει ήδη όνομα πάνω στην μπάρα πλοήγησης καθώς το έχουμε ήδη γράψει στο αρχείο CSTeilarViewController.m και στην μέθοδο viewDidLoad:

```
self.title = @"Τει Λάρισας";
```

Η μεταβλητή στην συγκεκριμένη περίπτωση, από την στιγμή που χρησιμοποιούμε την μεταβλητή navController, αναφέρεται στο τίτλο της για αυτο και τοποθετείται πάνω στην μπάρα πλοήγησης.
Με τον ίδιο τρόπο θα προσθέσουμε τίτλους και στις άλλες κλάσεις προβολής.
Επιλέγουμε το αρχείο AnakinosisController.m και στη μέθοδο viewDidLoad: προσθέτουμε:

```
self.title = @"Ανακοινώσεις";
```

Επιλέγουμε το αρχείο RssDataViewController.m και στην μέθοδο viewDidLoad: προσθέτουμε:

```
self.title = @"Ανακοίνωση";
```

Τέλος επιλέγουμε το αρχείο DionisosController και στην μέθοδο viewDidLoad: προσθέτουμε:

```
self.title = @"Διόνυσος";
```

Έτσι ο χρήστης θα είναι πάντα ενήμερος για το που βρίσκεται αλλά και τι βρίσκεται στο προηγούμενο μενού καθώς η μπάρα πλοήγησης αυτόματα αντικαταστεί το κουμπί back στα αριστερά της με το όνομα της προηγούμενης προβολής από την οποία ο χρήστης εισήλθε.

6.1.4 Ένδειξη φόρτωσης δεδομένων

Σε πολλές περιπτώσεις ο χρήστης μπορεί να επιλέξει να δει τις ανακοινώσεις του τμήματος αλλά λόγω μη πρόσβασης του στο internet ή πολύ αργής σύνδεσης να χρειαστεί να περιμένει αρκετά δευτερόλεπτα μέχρι να δει τα αποτελέσματα της επιλογής του στην οθόνη. Όμως σε αυτά τα δευτερόλεπτα ο χρήστης μπορεί να αναρωτιέται αν η εφαρμογή συνεχίζει να ανταποκρίνεται και να φορτώνει τα δεδομένα ή απλά έχει σταματήσει να λειτουργεί όπως θα έπρεπε. Για αυτό τον λόγο όταν φορτώνουμε δεδομένα από το διαδίκτυο είναι αναγκαία η ύπαρξη μιας ένδειξης που να ενημερώνει τον χρήστη ότι η εφαρμογή συνεχίζει να λειτουργεί σωστά. Όποτε πρέπει να προσθέσουμε στην κλάση που διαχειρίζεται την προβολή των αποτελεσμάτων του RSS feed έναν

indicator.
Για να προσθέσουμε μια νέα μεταβλητή πρέπει πρώτα να την δηλώσουμε οπότε ανοίγουμε το αρχείο AnakinosisController.h και προσθέτουμε τα εξής:

```
UIActivityIndicatorView *activityIndicator;
```

και έξω από το @interface:

```
@property (retain, nonatomic) IBOutlet UIActivityIndicatorView *activityIndicator;
```

Ανοίγουμε το αρχείο AnakinosisController.m και κάνουμε synthesize την νέα μεταβλητή μας:

```
@synthesize activityIndicator;
```

Θέλουμε η ένδειξη να εμφανίζεται στην οθόνη του χρήστη όταν καλούμε την μέθοδο loadData οπότε τοποθετούμε σε αυτήν, πριν ακριβώς καλέσουμε την κλάση Parser, την εξής σειρά κώδικα:

```
[activityIndicator startAnimating];
```

Και θέλουμε να εξαφανίζεται όταν η κλάση μας δέχεται τα δεδομένα του RSS parser οπότε τοποθετούμε την παρακάτω γραμμή κώδικα στην μέθοδο receivedItems:

```
[activityIndicator stopAnimating];
```

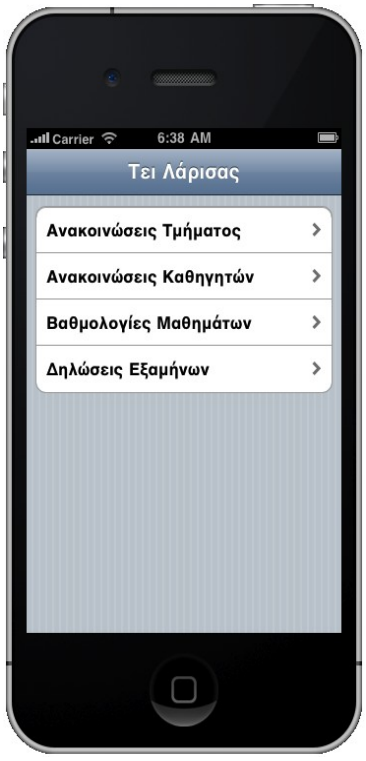
Τέλος πρέπει να τοποθετήσουμε έναν indicator στη προβολή της κλάσης οπότε ανοίγουμε το αρχείο AnakinosisController.xib και στον Interface Builder τοποθετούμε ένα αντικείμενο Activity indicator από το παράθυρο Library στο παράθυρο View και το αφήνουμε στο κέντρο του. Στην συνέχεια επιλέγουμε στο παράθυρο Inspector και στην καρτέλα attributes ως Style την επιλογή Large White και κάτω από αυτο τσεκάρουμε το κουτάκι που ονομάζεται Hide When Stopped έτσι ώστε να εξαφανίζεται όταν σταματάει να κινείται.

6.1.5 Εικονίδιο και όνομα εφαρμογής

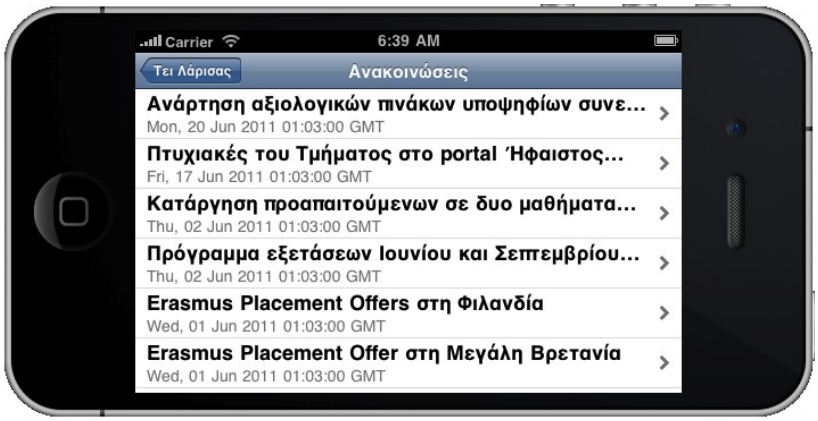
Το εικονίδιο και το όνομα της εφαρμογής είναι το πρώτο πράγμα που παρατηρεί ο χρήστης από την εφαρμογή μας. Είναι αυτο που θα φαίνεται στο κεντρικό μενού της συσκευής μας οπότε θα

πρέπει να είναι κάτι που να ταιριάζει με την λειτουργία της αλλά και να 'δένει' με το υπόλοιπο περιβάλλον χρήσης. Ένα εικονίδιο για να μπορεί να χρησιμοποιηθεί από το Xcode ως εικονίδιο εφαρμογής πρέπει να είναι 57x57 pixels. Για να το διαβάζει η εφαρμογή μας πρέπει να το προσθέσουμε στον φάκελο Resources του Bundle μας. Στη συνέχεια επιλέγουμε το αρχείο CSTeilar-Info.plist στο οποίο μπορούμε να κάνουμε κάποιες τέτοιου είδους ρυθμίσεις στην εφαρμογή μας. Στην λίστα Key βλέπουμε ένα στοιχείο που ονομάζεται Icon File και στο οποίο θα δώσουμε ως τιμή το όνομα του εικονιδίου μας. Επίσης αν πατήσουμε το κουμπί + στα δεξιά του πίνακα, μπορούμε να προσθέσουμε στοιχεία και να κάνουμε και άλλες ρυθμίσεις, όπως για παράδειγμα να επιλέξουμε το στοιχείο Status bar style και να του δώσουμε τιμή Opaque black style. Έτσι η μπάρα που μας δείχνει την ώρα και την μπαταρία της συσκευής μας θα έχει μαύρο φόντο όταν ο χρήστης μπαίνει στην εφαρμογή μας. Με τον ίδιο τρόπο μπορούμε να δώσουμε διαφορετικό όνομα από το όνομα του project στην εφαρμογή μας. Απλά αλλάζοντας την τιμή του στοιχείου Bundle display name και δίνοντάς του μια νέα.

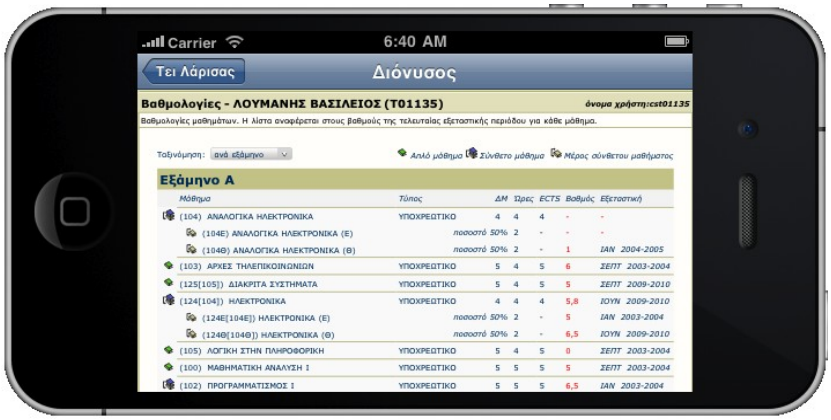
Τώρα μπορούμε να τρέξουμε την εφαρμογή μας για να δούμε πως αυτές οι αλλαγές επηρέασαν την εμφάνιση της εφαρμογής μας.



Εικόνα 6.2 : Η αρχική οθόνη της εφαρμογής



Εικόνα 6.3 : Τα RSS feeds των ανακοινώσεων του τμήματος



Εικόνα 6.4 : Οι βαθμολογίες στο Διόνυσος

6.2 Συμπεράσματα

Μέσα από αυτή την πτυχιακή γνώρισα τις κινητές πλατφόρμες, τα λειτουργικά τους συστήματα καθώς και τα προγραμματιστικά τους εργαλεία. Επίσης, προγραμματίζοντας για μια πλατφόρμα την οποία χρησιμοποιώ ως χρήστης σε καθημερινή βάση, την iOS, μπόρεσα να διαπιστώσω καλύτερα τις ιδιαιτερότητες προγραμματισμού σε κινητές πλατφόρμες σε σχέση με τον προγραμματισμό σε Desktop περιβάλλοντα.

Μετά από αρκετούς μήνες ενασχόλησης μου με τον προγραμματισμό στο Apple iOS νιώθω ότι είμαι αρκετά εξοικειωμένος με την γλώσσα προγραμματισμού Objective-C αλλά και με τα εργαλεία ανάπτυξης εφαρμογών της Apple και συγκεκριμένα το Xcode.

Κατά την διάρκεια της υλοποίησης της εφαρμογής βρέθηκα αρκετές φορές σε προβλήματα που δεν περίμενα ότι θα συναντήσω στην ανάπτυξή της. Όμως η διαδικασία της εύρεσης λύσης στα όποια προβλήματα είναι πολύ χρήσιμη στον προγραμματισμό. Ήταν ένα δυνατό τεστ για την δημιουργία μιας εφαρμογής που πραγματοποιεί αρκετά διαφορετικές λειτουργίες από οποιοδήποτε project είχα υλοποιήσει στα εργαστήρια της σχολής μας και σίγουρα μια πιο ολοκληρωμένη εμπειρία μου με τον προγραμματισμό

Είμαι πολύ χαρούμενος που μεσώ της πτυχιακής, μου δόθηκε η ευκαιρία να ασχοληθώ με τον προγραμματισμό σε κινητές πλατφόρμες καθώς ήταν κάτι που πάντα ήθελα να κάνω. Είναι ο τομέας με τον οποίο εδώ και χρόνια έχω αποφασίσει να ασχοληθώ όσο το δυνατόν περισσότερο και να διευρύνω τις γνώσεις μου, και μέσω της παρούσας πτυχιακής νιώθω ότι αυτο συνέβη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Mobile Applications: Architecture, Design, and Development Prentice Hall
- Learning iPhone Programming, Alasdair Allan
- The iPhone Developer's Cookbook, Erica Sadun
- Beginning iPhone Development, Dave Mark, Jeff LaMarche
- Apple iOS documentation
- iPhone Application Development Lectures, Stanford University
- iOS Dev Center, http://developer.apple.com
- Stack Overflow Dev Community, www.stackoverflow.com
- iPhone Dev SDK, www.iphonedevsdk.com/
- www.wikipedia.org/