ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ

ΕΦΑΡΜΟΓΗ iOS – APPLICATION iOS

SOCIAL LOCATION BASED APP

ΣΟΥΜΠΛΗΣ ΑΝΤΩΝΙΟΣ – ΙΩΑΝΝΗΣ
ΤΖΑΝΑΚΗΣ ΑΡΝΑΟΥΤΑΚΗΣ ΛΕΑΝΔΡΟΣ

Επιβλέπων :

Ακρίτας Αλκιβιάδης, Καθηγητής Πανεπιστημίου Θεσσαλίας

Βόλος, Οκτώβριος 2015

Ευχαριστίες

Η παρούσα Διπλωματική Εργασία ξεκίνησε υπό την επίβλεψη του καθηγητή κ. Αλκιβιάδη Ακρίτα και ολοκληρώθηκε στο διάστημα 2014-2015. Γι αυτό το λόγο θα θέλαμε να τον ευχαριστήσουμε ιδιαίτερα που μας έδωσε την ευκαιρία και το κίνητρο να συγγράψουμε αυτή την εργασία για τον τομέα των εφαρμογών iOS που ήταν κάτι το οποίο είχαμε αρχικά ως επιχειρηματική ιδέα και στη συνέχεια υλοποιήσαμε ως διπλωματική . Επίσης θα θέλαμε να ευχαριστήσουμε τους γονείς μας ο καθένας ξεχωριστά , για την αμέριστη στήριξη τους σε οικονομικό , ηθικό και ψυχολογικό επίπεδο διότι χωρίς εκείνους δεν θα είχαμε φτάσει έως εδώ . Τέλος θα θέλαμε να ευχαριστήσουμε τον φίλο μας τον Δημήτρη για την πολύτιμη βοήθεια του πάνω σε θέματα της εφαρμογής και του κώδικα Objective C .

Περίληψη

Στη παρούσα διπλωματική εργασία παρουσιάζεται ο σχεδιασμός και η ανάπτυξη μιας εφαρμογής σε περιβάλλον iOS, η οποία έχει ως σκοπό το συντονισμό μιας συνάντησης μεταξύ μιας ομάδας ατόμων από άποψη οργάνωσης και εξοικονόμησης χρόνου – κόστους.

Η δημιουργία της εφαρμογής αυτής έγινε με την χρήση του λογισμικού Xcode, το οποίο είναι ένα ολοκληρωμένο περιβάλλον μέσα από το οποίο μπορούμε να γράψουμε και να εκτελέσουμε κώδικα. Είναι το ίδιο με αυτό που χρησιμοποιείται στο Mac OS X. Το iOS (γνωστό και ως iPhone OS) είναι το λειτουργικό σύστημα για κινητές πλατφόρμες της Apple.

Η εφαρμογή θα μπορεί να χρησιμοποιηθεί από όλους τους χρήστες συσκευών Apple με λειτουργικό I-OS. Πιο συγκεκριμένα, η εφαρμογή προσφέρει στο χρήστη τη δυνατότητα να επιλέξει, να προτείνει και να μοιραστεί μέσα από συγκεκριμένες δομές, τον τόπο, τον χρόνο και το σκοπό της συνάντησης με τα άτομα τα οποία επιθυμεί μέσα από τη λίστα επαφών της συσκευής του.

Abstract

This diploma dissertation concerns the design and development of an application in an environment iOS, which aims to coordinate a meeting between a group of people in terms of organization and time savings – cost.

The creation of this app was done using the Xcode software, which is an integrated environment through which we can write and execute code. It is the same as the one used in Mac OS X. The iOS (known as iPhone OS) is the operating system for mobile platforms of Apple.

The application can be used by all devices Apple users functional I-OS. More specifically, the application offers the user the ability to select, propose and share through specific structures,the place,the time and the purpose of the meeting with the people who wish through the list of device contacts.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 6: ΠΑΡΑΡΤΗΜΑ

A – Κώδικας Objective C

B - Βιβλιογραφία

Κεφάλαιο 1ο

Εισαγωγή

1.1 Στόχος και χρησιμότητα

Έπειτα από προσωπική εμπειρία και παρατήρηση διαπιστώσαμε εμφανές το πρόβλημα έλλειψης συντονισμού που παρατηρείται μεταξύ φίλων/ομάδων ατόμων για τον προγραμματισμό μια συνάντησης μεταξύ τους, κυρίως σε μεγάλα αστικά κέντρα. Αυτή η έλλειψη συντονισμού έχει να κάνει είτε με την αποδοχή-επικύρωση της πρότασης που έχει γίνει από κάποιο άτομο προς τους φίλους του που έχει ως συνέπεια την ασυνεννοησία μεταξύ τους και την σπατάλη χρόνου και κόστους τηλεφωνημάτων ,είτε και με την ασυνέπεια πολλές φορές ορισμένων ως προς την ώρα (και τοποθεσία-λόγω μπερδέματος) συνάντησης με αποτέλεσμα την άσκοπη σπατάλη σημαντικού χρόνου, εκνευρισμού καθυστέρησης κτλ των υπόλοιπων μελών της ομάδας. Με την εφαρμογή μας λοιπόν σκοπεύουμε στον εξορθολογισμό και επίλυση αυτού του προβλήματος που είναι καθοριστικής σημασίας για την καθημερινή προσωπική/επαγγελματική μας ζωή .

Στην ψηφιακή εποχή που ζούμε με την ραγδαία ανάπτυξη της τεχνολογίας ο προγραμματισμός συναντήσεων μεταξύ των ανθρώπων έχει επιτευχθεί με την χρήση κινητών τηλεφώνων, διαδικτυακών μέσων (skype , facebook , what'sapp etc..). Ωστόσο κανένα από τα προαναφερθέντα δεν έχει την δυνατότητα να συντονίσει μια ομάδα με αποτελεσματικότητα 100% χωρίς σπατάλη χρόνου –κόστους και απόλυτου συντονισμού σε πραγματικό χρόνο των μελών χωρίς να υπάρχουν «στησίματα» και ανακρίβειες χρόνου-τοποθεσίας. Έπειτα τα παραπάνω μέσα δεν έχουν τέτοιο απολύτως στοχευμένο χαρακτήρα εφαρμογής και λειτουργίας εξορθολογισμού του προβλήματος.

Η εφαρμογή μας λοιπόν στηρίζεται στην επιτυχία δημιουργίας μιας πρακτικής και πολλή απλής στα δεδομένα του χρήστη διεπαφής σε compact μορφή όσο το δυνατό πιο εύχρηστης και λειτουργικής που έχει τη δυνατότητα να επιλύσει τις παραπάνω αδυναμίες του «συστήματος» των συναντήσεων. Συγκεκριμένα ο χρήστης θα μπορεί να προγραμματίσει μια συνάντηση ενημερώνοντας όλα τα μέλη μεταξύ τους με πλήρη περιγραφή σκοπού/τόπου/χρόνου κτλ και να λάβει απαντήσεις αποδοχής-απόρριψης. Έπειτα, θα έχει σε λίστα όλα τα προγραμματισμένα ραντεβού του στα οποία θα του έρχεται ειδοποίηση αφύπνισης καθώς και θα γνωρίζει εκ των

προτέρων ποια άτομα είναι καλεσμένα και ποια θα παρευρεθούν στην εκάστοτε συνάντηση που δημιούργησε ή είναι καλεσμένος.

Στο μέλλον έχουμε σκοπό όλοι οι προσκεκλημένοι να έχουν τη δυνατότητα να ενημερώνονται για την τρέχουσα θέση –εναπομείναντα χρόνου άφιξης των μελών που επρόκειτο να παρευρεθούν στο σημείο συνάντησης ώστε να κάνουν την κατάλληλη διαχείριση του χρόνου τους προς αποφυγή «καθυστερήσεων» μεταξύ τους.

1.2 Πρακτικές δυσκολίες στην υλοποίηση

Η βασικότερη δυσκολία της υλοποίησης της εφαρμογής είναι η έλλειψη ευρείας γνώσης και εμπειρίας μας πάνω σε εφαρμογές της γλώσσας Objective C καθώς επίσης και της εξοικείωσης μας με το λειτουργικό του mac OS και του προγραμματιστικού περιβάλλοντος Xcode . Στη συνέχεια προέκυψαν δυσκολίες και ως προς την ενσωμάτωση και διασύνδεση μερικών λειτουργιών του προγράμματος για παράδειγμα την υλοποίηση της διασύνδεσης της βάσης του foursquare με τους χάρτες Google maps καθώς και της ενσωμάτωσης των επαφών του χρήστη από τον τηλεφωνικό κατάλογο της συσκευής του . Στα επόμενα κεφάλαια θα αναφερθούμε εκτενώς με περισσότερες λεπτομέρειες.

Εξίσου σημαντική δυσκολία είναι η ανάπτυξη της εφαρμογής σε συγκεκριμένη γλώσσα προγραμματισμού Objective C, και στην συγκεκριμένη πλατφόρμα που (τουλάχιστον στην Ελλάδα) δεν είναι τόσο διαδεδομένη. Τα «έξυπνα» τηλέφωνα (κοινώς smart phones) της κατασκευάστριας εταιρείας Apple, iPhone, υποστηρίζουν εφαρμογές που αναπτύσσονται σε Objective-C, και για την παρούσα εργασία η ανάπτυξη έγινε μέσω της πλατφόρμας Xcode, που είναι **αποκλειστικότητα** του λειτουργικού συστήματος της Apple, Mac OS.

Κεφάλαιο 2ο

Γενικές Γνώσεις

2.1 Η Γλώσσα Προγραμματισμού Objective-C

Ξεκινώντας την περιγραφή της γλώσσας προγραμματισμού που χρησιμοποιήθηκε, αρκεί να δούμε το όνομά της και να πάρουμε δύο πολύ σημαντικά δεδομένα:
- Objective
- C

Πρώτον, πρόκειται για μια αντικειμενοστραφή γλώσσα, όπως οι περισσότερες νέες γλώσσες υψηλού επιπέδου που έχουν αναπτυχθεί. Ως αντικειμενοστραφής, βασίζεται στην έννοια της κλάσης (class) και του στιγμιότυπου (instance).

Δεύτερον, πρόκειται για μια γλώσσα που δημιουργήθηκε ως επέκταση της γλώσσας προγραμματισμού C, μια από τις πιο βασικές γλώσσες προγραμματισμού. Περιμένουμε να δούμε λοιπόν πολλές ομοιότητες, συντακτικού κυρίως χαρακτήρα με την γλώσσα C, αλλά πολύ διαφορετικές δυνατότητες, καθώς η C είναι διαδικαστική γλώσσα.

2.1.1 Αντικειμενοστραφής και Διαδικαστικός Προγραμματισμός

Ο αντικειμενοστραφής προγραμματισμός (Object-Oriented Programming) διαφέρει από τον διαδικαστικό προγραμματισμό (Procedural Programming) κυρίως στον βαθμό της αφαιρετικότητας που προσφέρει.

Ο διαδικαστικός προγραμματισμός προσφέρει αφαιρετικότητα μέσω των συναρτήσεων, και συνδυάζει τις δυνατότητες τους για να εκτελέσει το κυρίως πρόγραμμα. Τα δεδομένα που απαιτούνται περνιούνται ως παράμετροι στις συναρτήσεις, ή υπάρχουν ως καθολικά δεδομένα. Όταν όμως τα κύρια προγράμματα μεγαλώνουν, όπως μεγαλώνουν κι οι απαιτήσεις, δεν είναι δυνατόν και συνήθως ούτε ασφαλές να βλέπουν όλοι όλα τα δεδομένα. Επίσης τα πράγματα δυσκολεύουν όταν η συμπεριφορά που επιθυμούμε δεν είναι μια απλή κλήση σε μια συνάρτηση, αλλά συμπεριφορά μιας συνάρτησης που διαφέρει ανάλογα με τα δεδομένα.

Προχωρώντας λοιπόν στον αντικειμενοστραφή προγραμματισμό, ορίζουμε την έννοια του αντικειμένου. Κάθε αντικείμενο έχει τα δικά του δεδομένα, τα οποία μπορεί να μοιράζεται με

άλλους, μπορεί και όχι, και την δική του συμπεριφορά που μπορεί να καθορίζεται από τον τύπο του αντικειμένου, από τα δεδομένα του και τα λοιπά.

Έτσι μια φαινομενικά απλή πράξη, όπως η πρόσθεση, μπορεί να έχει εντελώς διαφορετική εκτέλεση ανάλογα με το αντικείμενο που την εκτελεί:

- Ένας ακέραιος θα μας έδινε την γνωστή μαθηματική πρόσθεση

- Η πρόσθεση όμως δύο συμβολοσειρών είναι κάτι εντελώς διαφορετικό, κι όμως υπακούει στην ίδια εντολή (+)

2.1.2 Η Objective C ως επέκταση της C

Η Objective C είναι ένα υπερσύνολο της C με την αυστηρή έννοια. Είναι δυνατόν να συνθέσουμε (compile) οποιοδήποτε πρόγραμμα γραμμένο σε C σε έναν οποιονδήποτε συνθέτη γλώσσας Objective C (compiler). Έτσι προφανώς μπορούμε να συμπεριλάβουμε κώδικα C μέσα ένα πρόγραμμα γραμμένο σε Objective C, χωρίς καμία απολύτως επιπλοκή. Μια τέτοια επιλογή θα είχε λογική για εξοικονόμηση πόρων του συστήματος, όπως μνήμη, χρόνο και υπολογιστική ισχύ. Ως γλώσσα αρκετά υψηλού επιπέδου, η Objective C είναι αρκετά πιο «ακριβή» σε ισχύ από μια γλώσσα όπως η C.

Ως επέκταση της C, όλο το συντακτικό που αφορά μη αντικειμενοστραφείς λειτουργίες είναι αυτό της C, ενώ για το υπόλοιπο κομμάτι, η Objective C ακολουθεί το συντατικό όπως ακολουθεί το στυλ των μηνυμάτων Smalltalk.

Ένα αντικείμενο μπορεί να κάνει ακριβώς τρία πράγματα:

1. Να έχει μια εσωτερική κατάσταση (συνήθως και με αναφορές σε άλλα αντικείμενα)
2. Να λαμβάνει μηνύματα από τον εαυτό του ή από κάποιο άλλο αντικείμενο
3. Καθώς επεξεργάζεται ένα μήνυμα, μπορεί να στείλει μηνύματα στον εαυτό του ή σε άλλο αντικείμενο

Έτσι μια «κλήση» σε μια μέθοδο, ή γενικότερα όπως αναφέρεται, η αποστολή ενός μηνύματος, ακολουθεί το παρακάτω συντακτικό:

```
[obj method:argument];
```

Όπου obj το αντικείμενο, method η μέθοδός του που καλούμε και argument η πρώτη από τις παραμέτρους που απαιτεί η μέθοδος.

Μια λίγο πιο γενική μορφή είναι η:

```
[obj method:argument1 moreMethodText:argument2...
methodFinalText:argumentN];
```

Η χρήση των τετράγωνων παρενθέσεων (`square brackets – []`) λοιπόν σημαίνει την κλήση σε μια μέθοδο, την αποστολή μηνύματος σε ένα αντικείμενο, και επίσης όπως κάθε εντολή

της C, τελειώνει με το ελληνικό ερωτηματικό (`semicolon – ;`).

2.1.3. Δομή αρχείων της γλώσσας Objective C

Ας δούμε τη δομή των αρχείων ενός προγράμματος γραμμένου σε Objective C.
Γνωστά και από την C, έχουμε τα αρχεία κεφαλίδων (header files), με δηλώσεις ονομάτων συναρτήσεων ή και μεταβλητών, και τα αρχεία κώδικα (code files), που παρέχουν την υλοποίηση.

Στην Objective C, τα header files, έχουν κατάληξη .h, και περιέχουν την «διεπαφή» μιας κλάσης (interface). Δηλώνονται όσες μεταβλητές χρησιμοποιούνται από τις περισσότερες μεθόδους (όπως θα δούμε παρακάτω, η Objective C είναι δυναμική γλώσσα και επιτρέπει την υλοποίηση μεταβλητών και δέσμευση μνήμης κατά τη διάρκεια εκτέλεσης (run time), όχι μόνο κατά τη διάρκεια σύνθεσης (compilation time)) και οι μορφές των μεθόδων της κλάσης.

```
@interface classname : superclassname {
// instance variables
}
+ classMethod1;
+ (return_type)classMethod2;
+ (return_type)classMethod3:(param1_type)param1_varName;
- (return_type)instanceMethod1:(param1_type)param1_varName :
(param2_type)param2_varName;
- (return_type)instanceMethod2WithParameter :
(param1_type)param1_varName
andOtherParameter:(param2_type)param2_varName;
@end
```

Όπου με άνω κάτω τελεία (colon) μετά το όνομα της κλάσης (classname) δηλώνουμε την κληρονομικότητα (inheritance), δίνοντας το όνομα της άλλης κλάσης από την οποία λάμβανει τα κληρονομικά χαρακτηριστικά, γνωστή και ως υπερ-κλάση (superclass). Αν δεν έχουμε κάποια κλάση που επιθυμούμε να κάνουμε subclass, τότε πάντα πρέπει να κάνουμε subclass την NSObject, η οποία είναι η βασική κλάση (root class).

Οι μέθοδοι με το σύμβολο συν (+) αριστερά αναγνωρίζονται ως μέθοδοι κλάσης (class methods), ενώ αυτές με το σύμβολο πλην (-) είναι μέθοδοι στιγμιοτύπου (instance methods). Ο τύπος του αντικειμένου που επιστρέφουν αναγράφεται αμέσως μετά, μέσα σε παρενθέσεις, και μπορεί να είναι void (κοινώς να μην επιστρέφει κάτι), κάποιος πρωτογενής τύπος της C (int, float, κλπ), δείκτης σε ένα αντικείμενο συγκεκριμένου τύπου, από αυτά που ορίζουν οι βιβλιοθήκες της Objective C, ή τέλος, δείκτης σε αντικείμενο του οποίου τον τύπο έχει ορίσει ο προγραμματιστής.

Τα αρχεία κώδικα, περιέχουν την υλοποίηση των μεθόδων που αναφέρονται στα αρχεία κεφαλίδων. Αρχικά σημαίνοντας μηνύματα (messages), τα αρχεία αυτά έχουν την κατάληξη .m, και έχουν την ακόλουθη μορφή:

```
@implementation classname
+ (return_type)classMethod {
// implementation
}
- (return_type)instanceMethod {
```

```
// implementation
}
@end
```

Δημιουργία στιγμιοτύπων (αντικειμένων)

Αφού μια κλάση έχει γραφεί σε κώδικα, για να πραγματοποιηθεί χρειάζονται δύο βήματα:

- Το πρώτο δεσμεύει χώρο στην μνήμη για ένα μη αρχικοποιημένο στιγμιότυπο της κλάσης

- Το δεύτερο το αρχικοποιεί

Μόνο αν γίνουν και τα δύο βήματα μπορεί να χρησιμοποιηθεί το αντικείμενο. Συνήθως αυτά τα δύο βήματα γίνονται σε μια γραμμή, για να είναι σίγουρο ότι ένα αντικείμενο έχει δεσμευτεί και αρχικοποιηθεί, καθώς επίσης επειδή το ενδιάμεσο αντικείμενο δεν είναι κάτι το οποίο θα χρησιμοποιηθεί, και μερικές φορές δεν συμπίπτει καν με το αντικείμενο που επιστρέφει η αρχικοποίηση.

```
MyObject *o = [[MyObject alloc] init];
```

Ο παραπάνω τρόπος είναι ο δεδομένος για όλες τις κλάσεις. Πέρα από αυτόν μπορούν να γραφούν διαφορετικοί προσαρμοσμένοι τρόποι (στο κομμάτι της αρχικοποίησης, όχι στο κομμάτι της δέσμευσης της μνήμης). Για παράδειγμα:

```
MyObject *o = [[MyObject alloc] initWithString:myString];
```

Στην πρώτη περίπτωση, που δεν έχουμε διαφορετική αρχικοποίηση, μπορεί να χρησιμοποιηθεί και η μέθοδος new:

```
MyObject *o = [MyObject new];
```

Το μήνυμα alloc δεσμεύει αρκετό χώρο για όλες τις μεταβλητές που δηλώνει το στιγμιότυπο, δίνει σε όλες την τιμή μηδέν και μετατρέπει τη μνήμη σε ένα στιγμιότυπο της κλάσης.

Το μήνυμα init αρχικοποιεί το στιγμιότυπο αφού δημιουργηθεί. Η υλοποίηση του μηνύματος έχει την ακόλουθη μορφή:

```
- (id)init {

     self = [super init];

     if (self) {
     // perform initialization of object here
     }
    return self;
 }
```

Το id είναι ένας τύπος αντικειμένου που σημαίνει «δείκτης σε οτίδηποτε αντικείμενο». Μετατρέπεται σε οποιοδήποτε αντικείμενο κατά την ώρα εκτέλεσης (ξανά ένα από τα στοιχεία που καθιστούν την Objective-C δυναμική γλώσσα προγραμματισμού).

Γίνεται πρώτα μια κλήση στην υπερκλάση και αναθέτει το αποτέλεσμα στο ίδιο το αντικείμενο (self). Αν είναι ένα έγκυρο αντικείμενο γίνονται οι αρχικοποιήσεις. Ένα μη έγκυρο αντικείμενο (ή καλύτερα ένας μη έγκυρος δείκτης σε αντικείμενο), έχει την τιμή nil (γνωστό ως Null σε άλλες γλώσσες). Έτσι σε περίπτωση σφάλματος επιστρέφεται η τιμή nil.

Πρωτόκολλα

Ένα επίσης νέο στοιχείο της Objective-C είναι τα πρωτόκολλα (protocols). Επειδή η Objective-C δεν υποστηρίζει πολλαπλή κληρονομικότητα (σε άμεση υλοποίηση), την υποστηρίζει έμμεσα μέσω των πρωτοκόλλων.

Ξεχωρίζουμε τα πρωτόκολλα σε επίσημα και ανεπίσημα (formal & informal). Τα επίσημα πρωτόκολλα ορίζουν μεθόδους που η κλάση που το εφαρμόζει υποχρεούται να υλοποιήσει, ενώ τα ανεπίσημα ορίζουν προαιρετικές μεθόδους. Για να δηλώσει μια κλάση ότι υποστηρίζει ένα πρωτόκολλο, το αναφέρει στο Interface της, ως εξής:

```
@interface SomeClass : SomeSuperClass <Locking>
@end
```

ενώ το ίδιο το πρωτοκόλλο ορίζεται ως εξής:

```
@protocol Locking
- (void)lock;
- (void)unlock;
@end
```

Η κατευθυντική εντολή #import

Αντίστοιχα με το #include της C, η Objective C έχει την ντιρεκτίβα #import, που συμπεριλαμβάνει τα αρχεία που την ακολουθούν στον κώδικα κατά τη διάρκεια της σύνθεσης (compilation). Το πλεονέκτημά της σε σχέση με την #include, είναι ότι συμπεριλαμβάνει τα αρχεία μόνο μια φορά, καθιστώντας αχρείαστη την προφύλαξη σε περιπτώσεις διπλής εισαγωγής.

Ένα αρχείο κώδικα (.m) συμπεριλαμβάνει πάντα το αρχείο κεφαλίδας (header file) για την διεπαφή του (interface), καθώς και οποιαδήποτε πρωτόκολλα στα οποία συμμορφώνεται, ή άλλα αρχεία κεφαλίδων άλλων κλάσεων των οποίων στιγμιότυπα χρησιμοποιεί.

Τύποι Δεδομένων

Όπως έχει ήδη αναφερθεί πιο πάνω, η Objective C υποστηρίζει τους γνωστούς πρωτογενείς τύπους δεδομένων της C, όπως int, float, double, char και το id, το οποίο όπως είπαμε είναι ένας δείκτης σε αντικείμενο οποιασδήποτε κλάσης, το οποίο μπορεί να αλλάξει δυναμικά τύπο στην εκτέλεση του προγράμματος (ή καλύτερα, απλά δεν είναι γνωστός ο τύπος του κατά την σύνθεση του κώδικα).

Υπάρχει επίσης ο ψευδο-τύπος BOOL για τις δυαδικές μεταβλητές (boolean) οι οποίες ουσιαστικά εκφυλίζονται από YES/NO σε 1/0.

Πέρα από αυτούς τους τύπους, ανάλογα με το framework που συμπεριλαμβάνει η εφαρμογή μας, υπάρχουν έτοιμες κλάσεις των οποίων μπορούμε να δημιουργήσουμε αντικείμενα.

Για παράδειγμα, για το Foundation Framework (NSFoundation), έχουμε εξ αρχής πρόσβαση σε χρήσιμες κλάσεις όπως την NSString, NSArray, NSNumber και τα λοιπά. Καθώς το string (αλφαριθμητικό) είναι από τους πιο συχνά και ευρέως χρησιμοποιούμενους τύπους δεδομένων, υπάρχει η δυνατότητα να κατασκευαστεί στιγμιότυπο, χωρίς την διαδικασία που αναφέραμε πιο πάνω.

Απλά γράφοντας:

@"Οποιοδήποτε κείμενο εδώ"

έχουμε δημιουργήσει ένα στιγμιότυπο της κλάσης NSString (το οποίο όμως δεν γίνεται deallocate ποτέ, το σύστημα το βλέπει και το αποθηκεύει ως δεδομένο και καταλήγει στο κομμάτι δεδομένων της εφαρμογής).

Μέθοδοι Πρόσβασης (Accessor Methods)

Στην Objective C τα δεδομένα και οι μεταβλητές μιας κλάσης είναι προσβάσιμα μόνο από την ίδια. Αν όμως θέλουμε να περάσουμε ή να χρησιμοποιήσουμε την τιμή μιας μεταβλητής σε κάποια άλλη κλάση, πρέπει να έχουμε τρόπο να αναφερθούμε σε αυτήν.
Μπορούμε να γράψουμε δικές μας συναρτήσεις που θα υλοποιούν αυτό το κομμάτι. Αυτές αναφέρονται γενικά ως **μέθοδοι τοποθέτησης και ανάκτησης (setter & getter methods).**

Η σύμβαση που ακολουθείται είναι η εξής:

```
variable = [object varName];
```

Και

```
[object setVarName: value];
```

με προσοχή πάντα το πρώτο γράμμα στο όνομα της μεταβλητής μετά το set να είναι κεφαλαίο.

Στην δήλωση των μεθόδων αυτών έχουμε προνοήσει να έχουμε δηλώσει σωστά τους τύπους, δηλαδή:

```
//Δήλωση μεταβλητών int age;
//Δήλωση μεθόδων - (int) age; - (void) setAge: (int) newage;
```

Ακριβώς την ίδια λειτουργικότητα προσφέρουμε και σε οποιοδήποτε αντικείμενο αν χρησιμοποιήσουμε τα **@property** και **@synthesize.**

```
//Δήλωση μεταβλητών int age; NSString *name;
```

```
//Δήλωση μεθόδων - (int) age;

- (void) setAge: (int) newage;

- (NSString *) name;

- (void) setName: (NSString *) newname;
```

Οι τελευταίες γραμμές μπορούν να συμπτιχθούν σε δύο, αν χρησιμοποιήσουμε τα properties:

```
@property int age; @property (copy) NSString *name;
```

Μέσω των properties μπορούμε να συμπιέσουμε τον κώδικα, αλλά μας δίνει επίσης την δυνατότητα της διαχείρισης μνήμης (εκεί αναφέρεται το (copy) στην μεταβλητή name), και σε περίπτωση που δεν θέλουμε να υπάρχει setter, υπάρχει η δυνατότητα δημιουργίας read-only property με την προσθήκη του (readonly) μετά το @property.
Αυτά για το κομμάτι δηλώσεων, που είναι στο .h αρχείο, δηλαδή στο interface της κλάσης. Στο .m αρχείο που θα γράφαμε την υλοποίηση της μεθόδου γράφουμε απλά:

```
@synthesize age;
@synthesize name;
```

και δεν θα υπήρχε ανάγκη να συμπληρώσουμε κάτι παραπάνω. Υπάρχει επίσης η δυνατότητα, η μεταβλητή για την οποία κάνουμε το synthesize, να έχει διαφορετικό όνομα από την μεταβλήτη του στιγμιοτύπου. Για παράδειγμα:

```
@interface ClassName :NSObject { int numberOfYearsOld; }
 @property int age;
@end

@implementation ClassName
 @synthesize age = numberOfYearsOld;
@end
```

Έτσι καλώντας την `[obj setAge: someIntValue]`, αυτό που θα έπαιρνε την τιμή someIntValue θα ήταν η μεταβλητή στιγμιότυπου numberOfYearsOld.

Σύνταξη με τελεία (Dot syntax)

Μετά από την έκδοση της Objective C 2.0, υποστηρίζεται και η σύνταξη με την τελεία για τις κλήσεις μεθόδων.

Κοινώς η εντολή:

```
self.height = 50;
```

Είναι ισοδύναμη με την εντολή:

```
[self setHeight:50];
```

Και αξίζει να σημειωθεί ότι η σύνταξη με την τελεία πάντα θα καλεί τις μεθόδους τοποθέτησης και

ανάκτησης (setter & getter). Αυτό έχει σημασία γιατί δεν γίνεται απλή ανάθεση τιμών, αλλά ακολουθείται η διαδικασία διαχείρησης μνήμης που έχει δηλωθεί στην ιδιότητα (property).

2.2. Διαχείριση Μνήμης

Από τα σημαντικότερα θέματα που προκύπτουν όταν προγραμματίζουμε για φορητές συσκευές, είναι αυτό της διαχείρισης μνήμης, καθώς οι πόροι είναι αρκετά περιορισμένοι σε σχέση με έναν σταθερό ηλεκτρονικό υπολογιστή. Συνεπώς είναι αναγκαία η σωστή διαχείριση μνήμης ώστε να αποφύγουμε τυχόν απρόβλεπτες αποτυχίες λογισμικού, όπως τον ξαφνικό τερματισμό της εφαρμογής.

Η Objective C ως δυναμική γλώσσα προγραμματισμού, επιτρέπει την δέσμευση και αποδέσμευση μνήμης για αντικείμενα κατά την διάρκεια της εκτέλεσης και χωρίς να είναι γνωστό το μέγεθος τους εκ των προτέρων, δηλαδή κατά την διάρκεια της σύνθεσης (complile).

Κατ' αντιστοιχία με την C, η οποία υποστήριζε τις εντολές malloc και free, για δέσμευση και απελευθέρωση χώρου στη μνήμη, η Objective C υποστηρίζει τις alloc και dealloc.

2.2.1 Εντολές διαχείρισης μνήμης (Alloc, Retain, Copy και Release)

Έχουμε ήδη αναφέρει την διαδικασία δημιουργίας ενός στιγμιοτύπου, και ότι είναι απαραίτητο ένα αντικείμενο να δεσμεύεται και να αρχικοποιείται στην ίδια γραμμή. Αυτό διότι σε κάποιες σπάνιες περιπτώσεις, η init που κάνει την αρχικοποίηση επιστρέφει (δείκτη πάντα, σε) διαφορετικό αντικείμενο από αυτό που έκανε την κλήση της. Επίσης, καθώς ένα αντικείμενο ουσιαστικά δεν είναι χρήσιμο αναρχικοποίητο, δεν έχει νόημα να κρατάμε δείκτη σε αυτό.

Έτσι έχουμε ήδη κάνει το πρώτο κομμάτι, αυτό της δέσμευσης της μνήμης. Μένει μόνο όταν τελειώσουμε με την εργασία που χρειαζόμαστε το αντικείμενο, να αποδεσμεύσουμε αυτήν τη μνήμη. Πρέπει να λοιπόν να κληθεί η dealloc στο αντικείμενο, και έχει επέλθει ισορροπία.

Αν δεν το κάνουμε, και απλά «ξεχάσουμε» τον δείκτη που έχουμε, έχουμε τις λεγόμενες διαρροές μνήμης (memory leaks), οι οποίες αν ξεφύγουν σε πλήθος, μπορεί να αναγκάσουν το σύστημα να τερματίσει την εφαρμογή μας.

Επίσης η εφαρμογή μπορεί να τερματίσει απρόσμενα και λανθασμένα, αν υπάρξει κλήση σε ένα δείκτη που δείχνει σε μνήμη που έχει αποδεσμευτεί (bad memory access), ή αν προσπαθήσουμε να αποδεσμεύσουμε μνήμη που έχει ήδη αποδεσμευτεί (βασικά επειδή πρόκειται για την προηγούμενη περίπτωση, στέλνουμε ένα μήνυμα σε μη έγκυρη περιοχή μνήμης).

Συνήθως βέβαια δεν καλείται απ' ευθείας μες στον κώδικα η εντολή dealloc. Αυτό που χρησιμοποιείται, είναι ένας αριθμός που δείχνει πότε ένα αντικείμενο πρέπει να καταστραφεί. Αυτό είναι το retain count.

Κάθε αντικείμενο, όταν δημιουργείται από εντολές alloc ή copy, ξεκινά με retain count ίσο με 1. Μπορούμε να αυξήσουμε αυτόν τον αριθμό κατά ένα καλώντας την εντολή retain, και να τον μειώσουμε κατά ένα καλώντας την εντολή release. Με το που φτάσει το retain count στο 0, το αντικείμενο καταστρέφεται, καλείται η dealloc, και δεν υπάρχει κανένας απολύτως τρόπος να το ανακτήσουμε.

Και γιατί να θέλουμε να αυξήσουμε αυτόν τον αριθμό?

Ας σκεφτούμε την παρακάτω περίπτωση:

Έστω ένας αριθμός, στιγμιότυπο της κλάσης NSNumber για να μιλάμε για αντικείμενο, και δύο ξεχωριστές μέθοδοι, η μια θέλει να τον προσθέσει με κάτι και η άλλη να τον εκτυπώσει 50 φορές. Αν η πρώτη μέθοδος με το που τελειώσει αποδεσμεύσει τη μνήμη του αριθμού, η δεύτερη θα συναντήσει κακή μνήμη. Θα μπορούσαμε να βάλουμε την δεύτερη μέθοδο να κάνει την αποδέσμευση. Αν όμως αντίθετα για κάποιο λόγο τερματίσει η δεύτερη μέθοδος πρώτη (όχι απαραίτητα λόγο φόρτου εργασίας, αλλά για παράδειγμα λόγω χρονοπρογραμματισμού διεργασιών του συστήματος), θα έχουμε το ίδιο πρόβλημα.

Η λύση:

Όταν η πρώτη μέθοδος πάρει το αντικείμενο, το κάνει retain.

Συνεπώς το retain count αυξάνεται κατά ένα.

Αντίστοιχα και η δεύτερη.

Όταν τελειώσουν, η κάθε μία κάνει από ένα release στο αντικείμενο.

Συνεπώς το retain count είναι στο ίδιο νούμερο που ήταν πριν αρχίσει η διαδικασία, και πλέον ουσιαστικά είναι ευθύνη του δημιουργού του να το αποδεσμεύσει. Αν ήταν αυτή η πρώτη διαδικασία που το δημιούργησε (ξεκίνησε δηλαδή το retain count από 0 και έγινε 1 με ένα alloc), το αντικείμενο απελευθερώνεται με το που γίνει το δεύτερο release (αφού το count έχει επανέλθει στο 0).

Τέλος να αναφέρουμε ότι η dealloc καλείται σε μια και μόνο περίπτωση ευθέως. Αφού έχουμε κάνει release όλα τα αντικείμενα για τα οποία είμαστε υπεύθυνοι, καλούμε την dealloc στην υπερκλάση της κλάσης μας, ως εξής:

```
[var1 release];
[var2 release];
 …
[super dealloc];
```

Όπου το super υποδεικνύει πέρασμα μηνύματος σε υπερκλάση. Με άλλα λόγια, αφού έχουμε «ξεκαθαρίσει» όση μνήμη είμαστε εμείς υπεύθυνοι, περνάμε τον έλεγχο προς τα πάνω για να «καθαριστούν» και τα υπόλοιπα. Μην ξεχνάμε ότι υπερκλάση όλων, κλάση-ρίζα είναι το NSObject, και όλες οι κλήσεις εν τέλει θα καταλήξουν εκεί. Αυτή η διαδικασία είναι πολύ εμφανής και στην ανάπτυξη εφαρμογών, καθώς τα View Controllers όπως θα δούμε, καθορίζουν την δική τους dealloc, όπου εμείς πρέπει να γράψουμε τι διαχειρίσεις γίνονται, και είναι ευθύνη του συστήματος να την καλέσει.

2.2.2 Setter μέθοδοι

Έχοντας μια πρώτη άποψη για την διαχείριση της μνήμης, μπορούμε να δούμε λίγο καλύτερα τι κάνει μια setter μέθοδος, όταν μιλάμε για αντικείμενα (δεν μιλάμε για τους πρωτογενείς τύπους της C, καθώς σε αυτούς δεν αναφερόμαστε συνήθως με δείκτες, οπότε οι περιπτώσεις που περιγράφονται παρακάτω συνήθως δεν εφαρμόζονται):

Υπάρχουν 3 διαφορετικοί τρόποι που θα μπορούσαμε να κάνουμε μια τοποθέτηση «τιμής» (ουσιαστικά γίνεται αντιστοίχιση δεικτών - αντικειμένων).

```
- (void)setName:(NSString *)newName {
     name = newName;
}
```

Αυτός ο τρόπος, παίρνει τον δείκτη της τοπικής μεταβλητής name, και τον βάζει να δείχνει όπου δείχνει ο newName. Το κακό με αυτόν τον τρόπο είναι ότι δεν είμαστε ιδιοκτήτες αυτού του αντικειμένου. Ανά πάσα στιγμή, όποιοσδήποτε έχει ένα δείκτη στο αντικείμενό μας, μπορεί να το αποδεσμεύσει και να μείνουμε με ένα δείκτη σε κακή μνήμη.

```
- (void)setName:(NSString *)newName {
     if (name != newName) {
          [name release];
          name = [newName retain];
     }
}
```

Αυτός ο τρόπος είναι καλύτερος από την άποψη του ότι, ότι και να γίνει, εμείς θα έχουμε πάντα έναν έγκυρο δείκτη στο αντικείμενο, αφού του αυξήσαμε το retain count κατά 1. Εφόσον δεν χρειαζόμαστε το παλιό αντικείμενο, το κάνουμε release, για να έρθει το retain count σε ισορροπία. Ο έλεγχος στην αρχή έχει το εξής νόημα. Αν για κάποιο λόγο περαστεί ως παράμετρος το ίδιο το αντικείμενο, αν το κάνουμε release και είμαστε οι μόνοι που έχουμε δείκτη σ' αυτό, η αμέσως επόμενη εντολή θα καταλήξει σε πρόσβαση σε κακή μνήμη. Με αυτό τον έλεγχο προλαβαίνουμε αυτή την περίπτωση.

```
- (void)setName:(NSString *)newName{
      if (name != newName) {
          [name release];
          name = [newName copy];
      }
}
```

Αυτός ο τρόπος είναι ίδιος με τον δεύτερο, με τη μόνη διαφορά ότι αντί να αυξάνουμε το retain count, δημιουργούμε ένα νέο αντικείμενο, αντίγραφο του αρχικού και βάζουμε τον δείκτη να δείχνει εκεί. Το retain count μετά από το copy είναι 1.

Αυτοί ακριβώς οι 3 τρόποι είναι αυτοί που υποστηρίζονται από την @property, με μορφή:

```
@property   (assign)   ClassName   *varName
@property   (retain)   ClassName   *varName
@property (copy) ClassName *varName
```

Συχνά παρεμβάλεται και το πρόθεμα non-atomic, ως εξής:

```
@property (nonatomic, copy) ClassName *varName
```

Με την έννοια ότι οι μέθοδοι setter και getter δεν είναι ατομικές. Αν παραλείπεται, θα θεωρούνται ατομικές.

Ατομικότητα μιας μεθόδου ορίζεται ως η ιδιότητα ότι δεν μπορεί να τμηθεί. Με άλλα λόγια, αυτή η μέθοδος είτε εκτελείται είτε όχι. Δεν υπάρχει η περίπτωση να διακοπεί στην μέση μιας ανάθεσης. Η ατομικότητα παίζει πολύ σημαντικό ρόλο στον συγχρονισμό διεργασιών και νημάτων, αλλά κάνει το σύστημα πιο «δυσκίνητο», καθώς όταν εκτελείται μια τέτοια διεργασία, δεν επιτρέπεται να εκτελεστεί παράλληλα κάποια άλλη η οποία πιθανόν να χρειαστεί την μεταβλητή (πιθανόν και όχι). Γενικότερα χρησιμοποιείται αρκετά αυτό το πρόθεμα, καθώς οι περισσότερες εφαρμογές είναι ενός νήματος (single threaded) και δεν απαιτούνται πολύπλοκες διαδικασίες συγχρονισμού.

2.2.3 Μελλοντική αποδέσμευση μνήμης (Autorelease και Autorelease pool)

Ως τώρα έχουμε δει πως μπορούμε να δημιουργήσουμε ένα αντικείμενο και να κρατήσουμε ένα δείκτη σε αυτό, και πως αν πάρουμε ένα αντικείμενο, να κρατήσουμε ένα δείκτη σε αυτό και να είμαστε σίγουροι πως όσο τον κρατάμε, το αντικείμενο δεν πρόκειται να αποδεσμευτεί.

Αν όμως θέλουμε να δημιουργήσουμε ένα αντικείμενο με τον σκοπό του να το περάσουμε ή να το δώσουμε σε κάποιον άλλο, πρέπει να δώσουμε σε αυτόν τον άλλον να καταλάβει και πως πρέπει να το διαχειριστεί. Το αν θα πρέπει να το αποδεσμεύσει η άλλη κλάση ή αυτή που το έδωσε, πρέπει να φαίνεται στο όνομα της μεθόδου, και αυτό είναι μια σύμβαση που πρέπει να ακολουθείται, προκειμένου να μην υπάρχουν μπερδέματα και κακή διαχείρηση μνήμης.

Αν το όνομα της μεθόδου περιέχει κάποια από τις λέξεις alloc, copy, ή new, τότε ο παραλήπτης πρέπει να κάνει το release. Διαφορετικά πρέπει να το κάνει η δημιουργός μέθοδος.

Πως όμως θα ξέρει η μέθοδος που το δημιούργησε, πότε πρέπει να το αποδεσμεύσει?

Για παράδειγμα:

```
- (NSString *)fullName
      {
       NSString *result;
      result = [[NSString alloc] initWithFormat:@"%@ %@",
      firstName, lastName];
      [result release];

      return result;
}
```

Με αυτή την μέθοδο, έχουμε ήδη αποδεσμεύσει το αντικείμενο, πριν καν το επιστρέψουμε σε αυτόν που κάλεσε τη μέθοδο.

Η λύση:

```
- (NSString *)fullName {
      NSString *result;
      result = [[NSString alloc] initWithFormat:@"%@ %@",
      firstName, lastName];
      [result autorelease];

      return result;
}
```

Με την λέξη-κλειδί autorelease, μαρκάρουμε το αντικείμενο για να γίνει release σε κάποια μελλοντική στιγμή, και όχι αμέσως. Έτσι όποιος κάνει την κλήση μπορεί ελεύθερα να το κάνει retain και να κρατήσει ένα δείκτη στο αντικείμενό του. Αντίστοιχα από την μεριά του δημιουργού, ξέρει ότι έχει εξισορροπήσει τις κλήσεις του σε retain και release, αφού το αντικείμενο θα γίνει release αργότερα.

Για να δούμε πότε θα γίνει αυτό, στρεφόμαστε στην ιδέα του autorelease pool. Ένα τέτοιο pool δημιουργείται σε κομβικά θα λέγαμε σημεία, και κάθε αντικείμενο που δημιουργείται και μαρκάρεται ως autoreleased μπαίνει εκεί μέσα. Όταν τελειώσει η όποια διεργασία, το pool γίνεται release, και αυτό αυτόματα καλεί το release σε όλα τα αντικείμενα που είχε. Είναι ένας πολύ κομψός και βολικός τρόπος διαχείρησης

μνήμης, και προσθέτει πολύ μεγάλη ευελιξία στον κώδικα. Μπορούμε επίσης να έχουμε εμφωλευμένα pools, χωρίς κανένα πρόβλημα απολύτως.

Επίσης το πρώτο πράγμα που κάνει η εφαρμογή που φτιάχνουμε, είναι να δημιουργήσει ένα τέτοιο pool, και το τελευταίο που κάνει είναι το κάνει release. Αυτό φαίνεται ξεκάθαρα στο main.m αρχείο που δημιουργείται για μας αυτόματα, και περιέχει απλά τον παρακάτω κώδικα:

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

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

Αυτή είναι η main συνάρτηση που καλείται πριν καν ξεκινήσουν οι διαδικασίες αρχικοποίησης για την εφαρμογή μας, και όλη η εφαρμογή τρέχει από την UIApplicationMain().

Επίσης μπορούμε να ορίσουμε εμείς δικό μας autorelease pool σε σημείο που ξέρουμε ότι θα δεσμεύσουμε αρκετή μνήμη, την οποία δεν θα χρειαζόμαστε μετά. Δημιουργούμε όσα αντικείμενα χρειαζόμαστε, φροντίζουμε να τα μαρκάρουμε ως autoreleased, και στο τέλος απλά αδειάζουμε το pool, κάνοντάς το release.

Είναι πολύ σημαντικό αυτό να εφαρμόζεται και σε περίπτωση που δημιουργούμε ξεχωριστά νήματα, καθώς κάθε νήμα πρέπει να βλέπει την δική του μνήμη και να την διαχειρίζεται κατάλληλα (πέρα φυσικά από όποια επικοινωνία μπορεί να υπάρχει μεταξύ των νημάτων).

2.3 Προγραμματισμός για iOS με χρήση του Xcode

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η πλατφόρμα του Xcode, που δίνει πολλά βασικά στοιχεία για να ξεκινήσει καποιος να δημιουργήσει μια εφαρμογή.

2.3.1 Model, View, Controller (MVC)

Η τεχνική που θα χρησιμοποιήσουμε για την δημιουργία της εφαρμογής ακολουθεί το μοντέλο ανάπτυξης MVC(Model, View, Controller), κοινώς κατηγοριοποιεί την εφαρμογή σε 3 είδη :

• **Model.** Το μοντέλο είναι υπεύθυνο για τα δεδομένα που παρέχονται στην εφαρμογή σε οποιοδήποτε σημείο της. Αυτό θα πρέπει να είναι συμπαγές και να μην διατηρεί καμία εξάρτηση από τον τρόπο παρουσίασης των. Ένα καλά δομημένο μοντέλο

μπορεί εμφανιστεί χωρίς προβλήματα μέσα από διαφορετικές διεπαφές. Το μοντέλο συνηθίζεται να είναι κάποια βάση δεδομένων (SQLite ή CoreData).

• **View.** Η εικόνα δηλαδή αυτό που βλέπει ο χρήστης είναι υπεύθυνη για την προβολή και μόνο των δεδομένων και των αλλαγών στον χρήστη. Δεν θα πρέπει να αποθηκεύει και να εξαρτάται από τα δεδομένα που εμφανίζει σε οποιοδήποτε κομμάτι της. Ο χρήστης θα πρέπει να είναι ικανός να αλληλεπιδράσει με τα στοιχεία της και κατ' επέκταση τα δεδομένα αλλά όχι άμεσα. Ακόμη, ιδανικό θα ήταν να μπορεί να επαναχρησιμοποιηθεί και να προσαρμόζεται στις ανάγκες του χρήστη.

• **Controller.** Ο ελεγκτής είναι υπεύθυνος για όλες τις ενδιάμεσες λειτουργίες μεταξύ του μοντέλου και της εικόνας. Αναλαμβάνει την ενημέρωση της εικόνας για τις όποιες αλλαγές στο μοντέλο δεδομένων αλλά και το ανάποδο, δηλαδή ενημέρωση των δεδομένων βάσει των όποιων αλλαγών έχει πραγματοποιήσει ο χρήστης διαμέσου της εικόνας. Πρακτικά είναι το κομμάτι με το οποίο συνήθως ασχολούμαστε περισσότερο καθώς αυτός περιέχει τον κώδικα της λογικής που θέλουμε να υλοποιήσουμε. Θα λέγαμε ότι είναι η καρδιά της εφαρμογής μας.

Συνεπώς το κομμάτι του κώδικα που γράφουμε που καθορίζει την εφαρμογή μας είναι αυτό του ελεγκτή. Συγκεκριμένα το περιβάλλον του Xcode διευκολύνει την εφαρμογή αυτού του μοντέλου ανάπτυξης, διαχωρίζοντας αυτά τα τρία κομμάτια.

Πέρα από το βασικό περιβάλλον που γράφουμε τον κώδικα, υπάρχει ένα ξεχωριστό κομμάτι, το Interface Builder, που απευθύνεται στην δημιουργία των views. Είναι μια γραφική διεπαφή που μας επιτρέπει να προσθέτουμε στοιχεία όπως κουμπιά, διακόπτες και άλλα, κάνοντας drag & drop, χωρίς να γράψουμε ούτε μια γραμμή

κώδικα. Σε εμάς μένει μόνο να κάνουμε τη διασύνδεση των στοιχείων του view με τις πράξεις στον ελεγκτή. Τα αρχεία που δημιουργούνται ονομάζονται nib files και έχουν την κατάληξη .xib .

Όσο για το μοντέλο, υποστηρίζονται βάσεις δεδομένων SQLite (μια μορφή για σχετικά πιο μικρές βάσεις δεδομένων), αν και η βασική διεπαφή με μια βάση δεδομένων γίνεται μέσω CoreData.

2.3.2 Πρότυπα αρχεία

Από το περιβάλλον του XCode έχουμε αρκετούς τρόπους να ξεκινήσουμε να χτίζουμε την εφαρμογή μας, δίνοντάς μας διαφορετικά πρότυπα αρχεία για να ξεκινήσουμε.

1) Window-based application
2) View-based application
3) Navigation-based application
4) TableView-based application
5) Και άλλα

Όλες οι περιπτώσεις δημιουργούν το appDelegate για την εφαρμογή μας, και ένα .xib αρχείο με την γραφική εικόνα της εφαρμογής μας.

Στην περίπτωση 1) το αρχείο αυτό είναι ένα κενό παράθυρο, στην 2) δημιουργείται αυτό το παράθυρο και ταυτόχρονα ένα δεύτερο .xib αρχείο για το view, και συνδέει τα δύο. Στις περιπτώσεις 3) και 4) συμπληρώνεται στο view και το αντίστοιχο στοιχείο που αναφέρει το όνομα, ήτοι ένα Navigation view ή ένα TableView. Στην περίπτωση 4), μας προτρέπει να δημιουργήσουμε και την αντίστοιχη βάση δεδομένων CoreData για τα περιεχόμενα του πίνακα.

Δεν είναι υποχρεωτικό να επιλέξουμε κάτι από αυτά, υπάρχει και η επιλογή για κενή εφαρμογή, αλλά μας διευκολύνει διότι προσθέτει πάρα πολύ κώδικα που ούτως ή άλλως θα κληθούμε να γράψουμε.

Κεφάλαιο 3ο

Η εφαρμογή

3.1 Αρχεία στο Xcode

Η εφαρμογή ξεκίνησε ως ένα view-based application, με το ένα βασικό view και τον controller του που προσφέρει το XCode από μόνο του και την κλάση για το ApplicationDelegate. Πιο συγκεκριμένα αυτή η κλάση φορτώνει τα αρχικοποιημένα δεδομένα τα οποία έχουμε ορίσει εμείς εκεί ,ένα TabBar και μια default View την οποία έχουμε αναθέσει να είναι η meetingList. Σε αυτά προστέθηκαν τελικά:

- 5 επιπλέον .m αρχεία στο φάκελο views με τα αντίστοιχα header files (σύνολο 10 αρχεία)

- 2 .m αρχεία με κώδικα με τα header files (σύνολο 4 αρχεία) στο φάκελο model

- 10 .m αρχεία με τα header files τους μαζί με τα 3 . xib αρχεία που παρήχθησαν μέσω του Interface Builder (σύνολο 23) στο φάκελο controllers

- 3 .m αρχεία μαζί με τα header files τους (σύνολο 6 αρχεία) στο φάκελο Foursquare

- το αρχείο main.m που βρίσκεται σε ξεχωριστό φάκελο supporting files.

3.2 View Controllers της Εφαρμογής

3.2.1 Δημιουργία λίστας από meetings

Το αρχικό View Controller της εφαρμογής που ευθύνεται για τη δημιουργία της λίστας συναντήσεων είναι το MAMeetingListVC το οποίο αναλύεται παρακάτω:

MAMeetingListVC

Ουσιαστικά αποτελεί ένα TableView της εφαρμογής με τη μορφή ενός πίνακα με τις ετικέτες των συναντήσεων του χρήστη και είναι το αρχικό Screen που βλέπει ο χρήστης με το που ξεκινήσει η εφαρμογή. Στο αρχείο αυτό υλοποιούνται όλες οι συναρτήσεις που θα χρησιμοποιηθούν για την κατασκευή του πίνακα.

```
 - (id)init
{
    self = [super init];
    if(self) {
        self.title = @"Meetings"; // Label Meetings
        self.tabBarItem.image = [UIImage
imageNamed:@"879-mountains"];  // image mountains
    }
    return self;
}
```

Το background color δημιουργείται με τον παρακάτω κώδικα :

```
- (void)loadView
{
 self.tableView = [[UITableView alloc]
initWithFrame:CGRectZero style:UITableViewStyleGrouped];
 self.tableView.backgroundColor = [UIColor
colorWithRed:0.40 green:0.71 blue:0.9 alpha:1.0];
 self.tableView.dataSource = self;
 self.tableView.delegate = self;

 self.view = self.tableView;
}
```

Με την παρακάτω εντολή γίνονται load από τη μνήμη (το μοντέλο των meetings) τα meetings :

```
 self.dataSource = [[Meetings getMeetingsFromArchive]
allMeetings];
```

και στη συνέχεια φορτώνει ξανά τον πίνακα απο την αρχή:

```
[self.tableView reloadData];
```

Επίσης όταν ο χρήστης βλέπει τον πίνακα και εμφανιστεί ένα καινούριο meeting τότε έρχεται ειδοποίηση(notification):

```
[[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(newMeetingAdded)

name:@"newMeetingAdded"

object:nil];
}
```

Στην συνέχεια υλοποιούνται συναρτήσεις που έχουν να κάνουν καθαρά με την κατασκευή του πίνακα όπως για παράδειγμα , τις διαστάσεις και τον αριθμό των κελιών του πίνακα κ.τ.λ.

Στο τέλος του αρχείου υλοποιούνται μέθοδοι UITableViewDelegate που είναι υπεύθυνες για τη διαχείριση του πίνακα δηλαδή όταν ο χρήστης πατήσει σε κάποιο button.

MAMeetingMapVC

Στο αρχείο αυτό υλοποιούνται οι συναρτήσεις που είναι υπεύθυνες για τον χάρτη. Πιο συγκεκριμένα, αυτός ο View Controller ελέγχει και "κανονίζει" το τι θα συμβεί όταν πατήσουμε πάνω στο χάρτη και πώς αυτό θα εμφανιστεί στην οθόνη μας.

Αρχικά δημιουργείται το view του χάρτη (μέσω Google Maps συναρτήσεων) :

```
self.meetingMapView.myLocationEnabled = YES;// να
εμφανίζεται η τοποθεσία

self.meetingMapView.settings.compassButton = YES;// να
εμφανίζεται κουμπί πυξίδας

self.meetingMapView.settings.myLocationButton = YES;// να
εμφανίζεται κουμπί τοποθεσίας

self.meetingMapView.settings.zoomGestures = YES;// να
επιτρέπεται το zoom in, zoom out

[self.meetingMapView setMinZoom:5 maxZoom:18];// όρια zoom
```

Με τον παρακάτω κώδικα κεντρικοποιείται η εικόνα του χάρτη στην οθόνη μας καθώς και το αν θα καλύπτει το status bar:

```
-(BOOL)prefersStatusBarHidden
    {
        return NO;                          //Επιλέγουμε αν το status
                                              bar Θα εμφανίζεται η όχι
    }


    _meetingMapView.padding =
    UIEdgeInsetsMake(self.topLayoutGuide.length + 100,
                                            0,

    self.bottomLayoutGuide.length +100,
                                             0);
```

Στη συνέχεια υλοποιείται συνάρτηση για τη διαχείριση του marker(δείκτης τοποθεσίας στο χάρτη), δηλαδή όταν ο χρήστης "πατήσει" επάνω του να εμφανίζονται οι πληροφορίες για την συγκεκριμένη τοποθεσία.

```
- (void)mapView:(GMSMapView *)mapView
didTapInfoWindowOfMarker:(GMSMarker *)marker
```

MAMeetingDetailVC

Στο αρχείο αυτό υλοποιείται η εμφάνιση του meeting, δηλαδή οι λεπτομέρειες για το συγκεκριμένο meeting όπως είναι ο χάρτης με το σημείο συνάντησης, ημερομηνία και ώρα της συνάντησης , τα άτομα που θα παρευρεθούν καθώς και κάποια περιγραφή(notes) που θα έχει δώσει ο δημιουργός-χρήστης.

Υλοποιούνται συναρτήσεις για την κατασκευή του view όπως για παράδειγμα, για το χρώμα, τα buttons, τα labels καθώς και τις διαστάσεις τους, καθώς επίσης επιτυγχάνεται και η ταξινόμηση τους στο χώρο του view.

Δεν χρησιμοποιήθηκε ο interface builder για την κατασκευή των subviews καθώς και των παραπάνω.

Στη συνέχεια επιτυγχάνεται η διαχείριση και εμφάνιση του χάρτη(GoogleMaps), με τον ίδιο τρόπο όπως και στο αρχείο MAMeetingMapVC.m

Στον παρακάτω κώδικα δεσμεύουμε τον απαιτούμενο χώρο για την εμφάνιση των frames, και των buttons :

```
 self.functionBar.frame = CGRectMake(0,

CGRectGetMaxY(self.meetingMapView.frame),

CGRectGetWidth(self.view.frame),
                                        44);


    self.notesView.frame = CGRectMake(10,

CGRectGetMaxY(self.functionBar.frame) + 10,

CGRectGetWidth(self.view.frame) - 10,
                                      140);
    self.meetingDate.frame = CGRectMake(10,

CGRectGetMaxY(self.notesView.frame) + 10,

CGRectGetWidth(self.view.frame) - 10,
```

```
                                        40);

    self.meetingMembersCount.frame = CGRectMake(10,

CGRectGetMaxY(self.meetingDate.frame) + 10,

CGRectGetWidth(self.scrollView.frame) - 10,
                                                40);
    self.viewMeetingMembersButton.frame =
CGRectMake(CGRectGetMidX(self.view.frame)-50,

CGRectGetMaxY(self.meetingMembersCount.frame) + 10,
                                                     100,
                                                     50);
    [self.viewMeetingMembersButton addTarget:self
```

και στη συνέχεια υλοποιούνται συναρτήσεις που αφορούν τη λειτουργία των buttons :

```
- (void)editButtonTapped:(id)sender

- (void)linkButtonTapped:(id)sender

- (void)viewMembers

- (void) mapView:(GMSMapView *) mapView
didLongPressAtCoordinate:(CLLocationCoordinate2D) coordinate
```

MAEditMeetingNote

Είναι το view που εμφανίζεται όταν ο χρήστη "πατάει" το button edit note. Πρόκειται για έναν απλό ViewController που δίνει τη δυνατότητα στο χρήστη για επεξεργασία ενός κειμένου, πιο συγκεκριμένα κάποια περιγραφή για την συγκεκριμένη συνάντηση.

Μόλις τελειώσει η επεξεργασία, το κείμενο αποθηκεύεται με τον παρακάτω κώδικα :

```
NSArray *allMeetings = [[Meetings getMeetingsFromArchive]
allMeetings];

NSMutableArray *allMetingsMutable = [[NSMutableArray alloc]
initWithArray:allMeetings];

[allMeetings enumerateObjectsUsingBlock:^(id obj, NSUInteger
idx, BOOL *stop) {
        if([obj meetingId] == self.meeting.meetingId) {
```

```
            self.meeting.notes = self.notesTextView.text;
            [allMetingsMutable replaceObjectAtIndex:idx
withObject:self.meeting];
```

3.2.2 Δημιουργία ενός νέου meeting

MACreateMeetingVC

Το αρχείο αυτό έχει να κάνει με τον view-controller “create a meeting” δηλαδή για την δημιουργία μιας συνάντησης. Στα headers files δηλώνουμε όλες τις λειτουργικότητες που θα χρησιμοποιήσουμε παρακάτω . Πιο συγκεκριμένα για το foursquare , το venue search , για το ημερολογιο κτλ. Αρχικά δηλώνουμε τα στοιχεία που θα εμφανίζονται μέσα στον view-controller πχ ετικέτα του meeting , ετικέτα με τα notes , όνομα τοποθεσίας , λίστα καλεσμένων , δομή ημερολογίου με τα αντίστοιχα background colors και borders .

```
   self.nameText = [[UITextField alloc] init];
    self.nameText.borderStyle =
UITextBorderStyleRoundedRect;
    self.nameText.contentVerticalAlignment =
UIControlContentVerticalAlignmentCenter;
    self.nameText.placeholder = @"Type a name...";
    self.meetingNotesLabel = [[UILabel alloc] init];
    self.notesText = [[UITextField alloc] init];
    self.notesText.borderStyle =
UITextBorderStyleRoundedRect;
    self.notesText.contentVerticalAlignment =
UIControlContentVerticalAlignmentCenter;
    self.notesText.placeholder = @"Type a name...";
    self.meetingNameLabel.backgroundColor = [UIColor
clearColor];
    self.meetingNameLabel.textColor = [UIColor
whiteColor];
    self.meetingDateLabel.backgroundColor = [UIColor
clearColor];
    self.meetingDateLabel.textColor = [UIColor
whiteColor];
    self.meetingMembersLabel.backgroundColor = [UIColor
clearColor];
    self.meetingMembersLabel.textColor = [UIColor
whiteColor];
    self.meetingPlaceLabel.backgroundColor = [UIColor
clearColor];
    self.meetingPlaceLabel.textColor = [UIColor
whiteColor];

    self.pickDateButton = [UIButton
buttonWithType:UIButtonTypeRoundedRect];
```

```
    [self.pickDateButton setTitle:@"Select a date"
forState:UIControlStateNormal];
    self.pickDateButton.tintColor = [UIColor
purpleColor];
    [self.contentSubview addSubview:self.pickDateButton];

    self.pickLocationButton = [UIButton
buttonWithType:UIButtonTypeRoundedRect];
    [self.pickLocationButton setTitle:@"Select a place"
forState:UIControlStateNormal];
    self.pickLocationButton.tintColor = [UIColor
brownColor];
```

Έπειτα υλοποιούμε και τα subviews τους ακριβώς παρακάτω :

```
self.contentSubview.frame = CGRectMake(0,
                                       0,

CGRectGetWidth(self.view.frame),

CGRectGetHeight(self.view.frame));

    self.meetingNameLabel.frame = CGRectMake(50,
                                             30,
                                             200,

44);//self.meetingNameLabel.font.pointSize + 5);
    self.nameText.frame = CGRectMake(50, 80, 280, 30);

    self.meetingNotesLabel.frame = CGRectMake(50, 120,
350, 44);
    self.notesText.frame = CGRectMake(50, 150, 280, 30);
    self.meetingDateLabel.frame = CGRectMake(50, 200,
250, 30);
    self.pickDateButton.frame = CGRectMake(50, 210, 150,
50);
    self.meetingMembersLabel.frame = CGRectMake(50, 250,
250, 30);
    self.pickMembersButton.frame = CGRectMake(50, 280,
150, 50);
    self.meetingPlaceLabel.frame = CGRectMake(50, 340,
250, 30);
    self.meetingPlaceText.frame = CGRectMake(50, 370,
250, 30);
    self.pickLocationButton.frame = CGRectMake(50, 400,
150, 50);

    self.createMeetingButton.frame = CGRectMake(50, 450,
80, 50);
```

```
    self.contentSubview.contentSize =
CGSizeMake(CGRectGetWidth(self.contentSubview.frame),
CGRectGetMaxY(self.meetingNameLabel.frame));
```

Και στο τέλος υλοποιούνται οι συναρτήσεις των views δηλαδή τι θα γίνει αν ο χρήστης επιλέξει καθεμία από τις παραπάνω λειτουργίες .

```
-(void)pickDate
-(void)pickMembers
-(void)pickLocation
-(void)saveMeeting
```

MAContactPickerVC

Το αρχείο αυτό έχει να κάνει με την διαχείριση των επαφών του χρήστη. Πιο συγκεκριμένα πώς θα εμφανίζονται και πως θα αποθηκεύονται οι επαφές (υπό μορφή πίνακα) από τον τηλεφωνικό κατάλογο του κινητού στην εφαρμογή μας .

Οπότε και αυτό το αρχείο περιέχει ένα TableView και τις αντίστοιχες συναρτήσεις όπως και τα υπόλοιπα . Αρχικά μετράει με την μεταβλητή contactsData.count τον αριθμό των επαφών του χρήστη για να προσθέσει γραμμές από τις ετικέτες ονομάτων αλλιώς αν δεν έχει επαφές καθόλου δεν προσθέτει καμία.

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
    if (_contactsData){
        return _contactsData.count;
    }
    Else{
        return 0;}
}
```

Έπειτα υλοποιούμε μια σειρά από συναρτήσεις για παράδειγμα την showAddressBook η οποία διαχειρίζεται το πώς εμφανίζεται το AddressBook δηλαδή ο κατάλογος των επαφών .

```
- (void)showAddressBook
{
    _addressBookController =
[[ABPeoplePickerNavigationController alloc] init];
    [_addressBookController
setPeoplePickerDelegate:self];
    [self presentViewController:_addressBookController
animated:YES completion:nil];
}
```

Παρακάτω στη συνάρτηση peoplePickerNavigationController που υλοποιούμε το ευρετήριο των επαφών από τις τιμές των τηλεφωνικών αριθμών , προσθέτουμε και τα emails των χρηστών που τα έχουν συμπεριλάβει .

```
emailsRef = ABRecordCopyValue(person,
```

```
     kABPersonEmailProperty);
         for (int i=0; i<ABMultiValueGetCount(emailsRef); i++)
         {
             CFStringRef currentEmailLabel =
     ABMultiValueCopyLabelAtIndex(emailsRef, i);
             CFStringRef currentEmailValue =
     ABMultiValueCopyValueAtIndex(emailsRef, i);

             if (CFStringCompare(currentEmailLabel,
     kABHomeLabel, 0) == kCFCompareEqualTo)
             {
            [contactInfoDictionary setObject:(__bridge NSString
*)currentEmailValue forKey:@"homeEmail"];
        }

        if (CFStringCompare(currentEmailLabel, kABWorkLabel, 0)
== kCFCompareEqualTo)
        {
            [contactInfoDictionary setObject:(__bridge NSString
*)currentEmailValue forKey:@"workEmail"];
        }

        CFRelease(currentEmailLabel);
        CFRelease(currentEmailValue);
    }
    CFRelease(emailsRef);
```

MAContactDetailVC

Στο αρχείο αυτό υλοποιείται η λειτουργία του τι ακριβώς γίνεται εάν πατήσει ο χρήστης πάνω σε κάποια από τις επαφές του. Η επαφή αποτελείται από μία ετικέτα-όνομα , κάποια εικόνα (image) και κάποιες πληροφορίες τις οποίες αυτές τι παίρνουμε μέσα από το AddressBook .
Στη συνάρτηση populateContactData επιλέγουμε τη μορφοποίηση με τη οποία θα εμφανίζεται το όνομα , η εικόνα και οι πληροφορίες του εκάστοτε χρήστη .

```
-(void)populateContactData{
    NSString *contactFullName = [NSString stringWithFormat:@"%@
%@", [_dictContactDetails objectForKey:@"firstName"],
[_dictContactDetails objectForKey:@"lastName"]];

    [_lblContactName setText:contactFullName];
    // Set the contact image.
    if ([_dictContactDetails objectForKey:@"image"] != nil)
    {
        [_imgContactImage setImage:[UIImage
```

```
imageWithData:[_dictContactDetails objectForKey:@"image"]]];
    }
    [_tblContactDetails reloadData];
}
```

Στους παρακάτω μεθόδους υλοποιούμε κάποιες περιπτώσεις για να δείξουμε σε ποιο κομμάτι – section πληροφορίας της επαφής βρίσκεται ο χρήστης κατά την περιήγηση του στο ευρετήριο . Για παράδειγμα λειτουργία για ανάκτηση πληροφορίας σχετικά με τον αριθμό κινητού τηλεφώνου , οικιακό τηλεφωνικό αριθμό , email εργασίας και σπιτιού καθώς και εμφάνιση διεύθυνσης της επαφής .

```
switch (indexPath.section)
                case 0:
                    cellText = [_dictContactDetails
objectForKey:@"mobileNumber"];
                    detailText = @"Mobile Number";
                         break;
                     case 1:
                    cellText = [_dictContactDetails
objectForKey:@"homeNumber"];
                    detailText = @"Home Number";
                    break;
            }
            break;

        case 1:
            switch (indexPath.row)
        {
                case 0:
                    cellText = [_dictContactDetails
objectForKey:@"homeEmail"];
                    detailText = @"Home E-mail";
                    break;
                case 1:
                    cellText = [_dictContactDetails
objectForKey:@"workEmail"];
                    detailText = @"Work E-mail";
                    break;
            }
            break;

        case 2:
            switch (indexPath.row)
        {
                case 0:
                    cellText = [_dictContactDetails
objectForKey:@"address"];
                    detailText = @"Street Address";
                    break;
```

```
                case 1:
                    cellText = [_dictContactDetails
objectForKey:@"zipCode"];
                    detailText = @"ZIP Code";
                    break;
                case 2:
                    cellText = [_dictContactDetails
objectForKey:@"city"];
                    detailText = @"City";
                    break;
            }
```

MADatePickerVC

Το αρχείο αυτό έχει να κάνει με την διαχείριση της εμφάνισης του ημερολογίου . Κατ'αντιστοιχία όπως είναι για τις επαφές υλοποιούμε και τη δομή του ημερολογίου. Σε αυτό το κομμάτι του κώδικα χρησιμοποιήσαμε τη τεχνική του εργαλείου Interface Builder την οποία διαθέτει το Xcode για την άμεση υλοποίηση κάποιων δομών για σχεδίαση γραφικού περιβάλλοντος . Απλώς αλλάξαμε την μορφοποίηση για την εμφάνιση της ημερομηνίας και που αποθηκεύεται δηλαδή το ότι αναθέτουμε στο meeting πιο συγκεκριμένα στη μεταβλητή date την τιμή datevalue δηλ. την ημερομηνία διεξαγωγής του meeting .

```
- (void)datePickerChanged:(UIDatePicker *)datePicker
{
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc]
init];
    [dateFormatter setDateFormat:@"dd-MM-yyyy HH:mm"];
    NSString *dateValue = [dateFormatter
stringFromDate:datePicker.date];
    self.selectedDate.text = dateValue;
    self.meeting.date = dateValue; //save the date value
}
```

3.2.3 Δημιουργία Προφίλ του Χρήστη

MAProfileVC

Στο αρχείο αυτό έγινε μία προσπάθεια να υλοποιηθεί το profile section ενός χρήστη ο οποίος είναι εγγεγραμένος στην εφαρμογή για περεταίρω μελλοντική χρήση και ανάπτυξη της υπάρχουσας εφαρμογής σε πραγματικό επέπδο . Στην προκειμένη φάση πήραμε ένα κείμενο από ένα tutorial της Objective C και το προσθέσαμε στην κενή σελίδα του profile σχετικά με πληροφορίες για την γλώσσα .

```
NSAttributedString *headerText = [[NSAttributedString alloc]
initWithString:@"PhotoNotes" attributes:@{NSShadowAttributeName:
textShadow}];
    self.headerLabel.attributedText = headerText;

    self.authorLabel.text = @"Created by";
```

self.descriptionTextView.text = @"PhotoNotes is a simple app created for Code School's Core iOS 7 course. If you install this app on your phone then you will be able to take photos and add some notes about them. For example, you might take a picture of a tree and leave a note about why you found that tree worthy of capturing.";

self.csInfoTextView.text = @"Code School teaches web and app development technologies in the comfort of your browser with video lessons, coding challenges, and screencasts. With the release of

iOS 7, we're now expanding our code challenges to the environment that most iOS developers work in - Xcode. By completing challenges in Xcode and still earning points and badges on codeschool.com, we're able to bring you the best of both worlds so you can start building apps quickly. We strive to help you learn by doing.";

```
[[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(contentSizeChanged:)

name:@"UIContentSizeCategoryDidChangeNotification"

object:nil];
```

3.3 Τα Models της Εφαρμογής

Meeting

Το meeting που περιέχει τις αναλυτικές λεπτομέρειες μίας συνάντησης που εισάγει ο χρήστης . Για παράδειγμα , το όνομα της συνάντησης , τις επιπλέον πληροφορίες , την ημερομηνία , τα μέλη, το σημείο συνάντησης κτλ .

(create meeting)

```
@interface Meeting : NSObject <NSCoding>
@property (assign, nonatomic) NSInteger meetingId;
@property (strong, nonatomic) NSString *name;
@property (strong, nonatomic) NSString *notes;
@property (strong, nonatomic) NSDate *date;
@property (strong, nonatomic) NSArray *meetingMembers;
@property (strong, nonatomic) CLLocation *meetingPoint;
@property (strong, nonatomic) NSString *venueName;
@end
```

Με τον coder που περιέχει από μόνο του το ios κάνουμε την αποθήκευση-store των παραπάνω στοιχείων με αυτόν εδώ τον τρόπο :

```
[aCoder encodeInteger:self.meetingId forKey:@"photoId"];
    [aCoder encodeObject:self.name forKey:@"name"];
    [aCoder encodeObject:self.date forKey:@"date"];
    [aCoder encodeObject:self.notes forKey:@"notes"];
    [aCoder encodeObject:self.meetingMembers forKey:@"invitees"];
    [aCoder encodeObject:self.meetingPoint forKey:@"location"];
    [aCoder encodeObject:self.venueName forKey:@"venueName"];
```

Και αντίστοιχα τα κάνουμε loading με τον decoder του ios με τον παρακάτω τρόπο :

```
self.meetingId = [aDecoder decodeIntegerForKey:@"photoId"];
        self.name = [aDecoder decodeObjectForKey:@"name"];
        self.date = [aDecoder decodeObjectForKey:@"date"];
        self.notes = [aDecoder decodeObjectForKey:@"notes"];
        self.meetingMembers = [aDecoder
decodeObjectForKey:@"invitees"];
        self.meetingPoint = [aDecoder
decodeObjectForKey:@"location"];
        self.venueName = [aDecoder
decodeObjectForKey:@"venueName"];
```

Meetings

Το αρχείο με τα αποθηκευμένα δεδομένα της λίστας τα οποία παίρνει ένα tableView και τα εμφανίζει στο screenhome . Αρχικά αποθηκεύουμε το meeting στη λίστα με το αντίστοιχο coder και έπειτα το loading ενός νέου meeting με τον αντίστοιχο decoder .

```
[aCoder encodeObject:self.allMeetings forKey:@"allMeetings"];
self.allMeetings = [aDecoder decodeObjectForKey:@"allMeetings"];

Στη συνέχεια ορίζουμε και το μονοπάτι-path το οποίο θα
αποθηκεύεται αυτή η λίστα .
 (NSString *)pathToArchive
 {
 NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *docsDir = [paths objectAtIndex:0];
    return [docsDir
stringByAppendingPathComponent:@"meetings.model"];
}
```

3.4 Τα Views της Εφαρμογής

MAMeetingCell

Το Xcode μέσα από έτοιμες συναρτήσεις δημιουργεί κάποια πρωτότυπα κελιά για έναν πίνακα. Στη συγκεκριμένη περίπτωση για να δημιουργήσουμε τον πίνακα του Meeting έπρεπε να δημιουργήσουμε και να τροποποιήσουμε ένα είδος κελιού που θα ταίριαζε στην περίπτωση αυτή. Το κελί αυτό αποτελείται από ένα CellName καθώς και από κάποια Notes (CellNotesView).

Στη συνέχεια υλοποιούμε συναρτήσεις που έχουν να κάνουν με το μέγεθος-έκταση του κελιού, τη γραμματοσειρά (font).

```
 self.cellName.font = [UIFont fontWithName:@"HelveticaNeue-Bold"
size:16.0f];
```

```
 self.cellName.frame =
CGRectMake(5,//CGRectGetMaxX(self.cellImage.frame) + 10,
                                        17,
                                        160,
                                        16);

    self.cellNotesView.frame =
CGRectMake(CGRectGetWidth(self.frame) - 60,
                                           20,
                                           40,
                                           40);
```

MAMeetingMembersListCell

Όπως και στο προηγούμενο αρχείο δημιουργούμε κελιά τα οποία θα περιέχουν τα Members, δηλαδή τα ονόματα από τη βάση των επαφών της συσκευής.

MAFoursquareCell

Παρόμοια, στο αρχείο αυτό γίνεται δημιουργία κελιών για την εμφάνιση των venues(διευθύνσεων) που επιστρέφει το Foursquare.

MADirections

Το Google maps όταν του ζητάμε να μας επιστρέψει πληροφορίες σχετικά με directions μιας τοποθεσίας (get http request), μας επιστρέφει δεδομένα σε μορφή dictionary και θα πρέπει εμείς να επιλέξουμε το πως θα εμφανίσουμε τα δεδομένα αυτά.Στην προκειμένη περίπτωση η εμφανιση θα γίνει με τη μορφή πίνακα (δημιουργούμε ενα TableView).

DirectionsCell

Στο αρχείο αυτό γίνεται η δημιουργία των κελιών του παραπάνω πίνακα.

3.5 Foursquare Integration

BZFoursquare

Το αρχείο αυτό είναι υπεύθυνο για τη δημιουργία των βασικών-basics συναρτήσεων για την ενσωμάτωση του Foursquare δηλαδή για την εγκαθίδρυση της βάσης του foursquare και του venue search στην εφαρμογή μας η οποία ολοκληρώνεται με τη δημιουργία της FoursquareRequest . Για παράδειγμα η παρακάτω συνάρτηση ξεκινάει να αρχικοποιήσει μια URL επικοινωνία με τη βάση δεδομένων

```
- (id)initWithClientID:(NSString *)clientID callbackURL:(NSString
*)callbackURL {
    NSParameterAssert(clientID != nil && callbackURL != nil);
```

```
    self = [super init];
    if (self) {
        self.clientID = myClientID;
        self.callbackURL = callbackURL;
        self.version = kMinSupportedVersion;
    }
    return self;
```

BZFoursquareRequest

Το αρχείο αυτό είναι μία URL request ενέργεια δηλαδή κατά βάση είναι μία get HTTP λειτουργία η οποία αυτό που κάνει είναι να μεταφέρει με τη μορφή ενός dictionary τη ζήτηση η οποία απαιτείται κάθε φορά . Η ζήτηση αποτελείται από το όνομα του venue που επιθυμεί ο χρήστης να αναζητήσει καθώς και από κάποιες άλλους παραμέτρους για παράδειγμα το κατά πόσο κοντά (near) ή μακριά (far) στο χρήστη πραγματοποιείται η αναζήτηση της εν λόγω τοποθεσίας κτλ . Όλα αυτά τα χαρακτηριστικά συνδυάζονται σε μία HTTP Request και στέλνονται στη βάση δεδομένων του foursquare και αυτό με τη σειρά του στέλνει πίσω σε μορφή dictionary πάλι τα αποτελέσματα της αναζήτησης .
Στην παρακάτω συνάρτηση αρχικά ορίζουμε τη μέθοδο που θα χρησιμοποιήσουμε ως μέθοδο GET

```
if ([_HTTPMethod isEqualToString:@"GET"]) {
        request = [self requestForGETMethod];
    }
```

Στη συνέχεια ορίζουμε τις παραμέτρους της GET μεθόδου για παράδειγμα το όνομα της τοποθεσίας venue το οποίο θα αποτελεί την παράμετρο και το key την επιστρεφόμενη τιμή πχ. Café entra

```
for (NSString *key in _parameters) {
        NSString *value = _parameters[key];
        if (![value isKindOfClass:[NSString class]]) {
            if ([value isKindOfClass:[NSNumber class]]) {
                value = [value description];
            } else {
                continue;
            }
        }
```

Και στο τέλος πραγματοποιούμε ένα URL Request δηλαδή στέλνεις μία αίτηση για το εν λόγω url – ανάκτηση διεύθυνσης.

```
Return
[NSURLRequestrequestWithURL:URLcachePolicy:NSURLRequestReloadIgnoringL
ocalCacheData timeoutInterval:kTimeoutInterval];
```

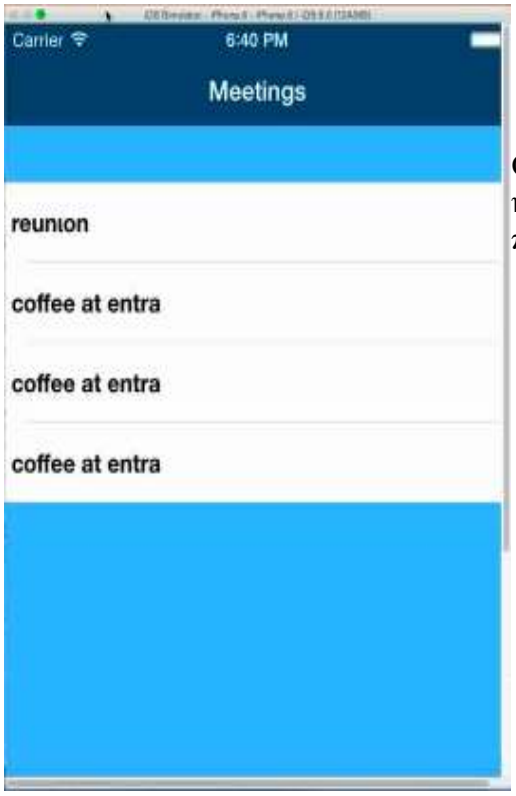
FSVenueSearchVC

Το αρχείο αυτό έχει να κάνει με τη διαχείριση της προβολής των αποτελεσμάτων της αναζήτησης – searching . Το venue search επιστρέφει μία απάντηση την οποία απάντηση αυτή θα πρέπει να διαχειριστούμε την προβολή της , δηλ πως θα εμφανιστεί στο χρήστη της εφαρμογής . Πιο συγκεκριμένα είναι ένας πίνακας , ένα tableView , όπως και η λίστα από τα meeting, που προβάλει τα αποτελέσματα της αναζήτησης του χρήστη .

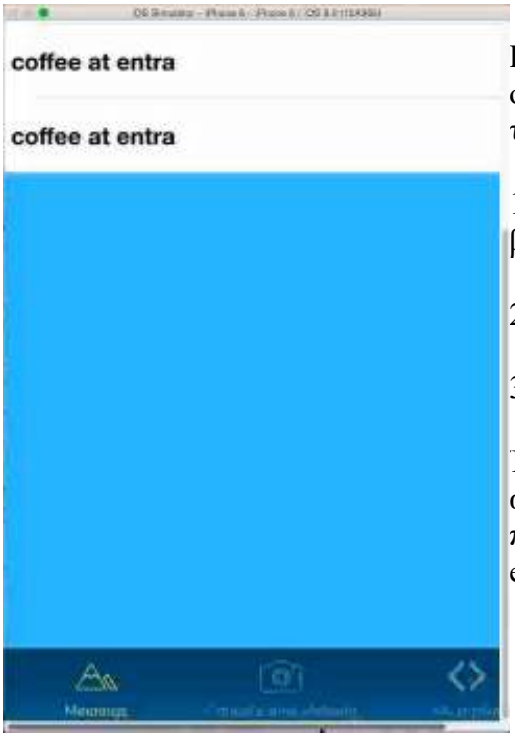
```
- (void)loadView
{
    self.tableView = [[UITableView alloc] initWithFrame:CGRectZero
style:UITableViewStyleGrouped];
    self.tableView.backgroundColor = [UIColor colorWithRed:0.40
green:0.71 blue:0.9 alpha:1.0];
    self.tableView.delegate = self;

    self.view = self.tableView;
    self.functionBar = [[UIToolbar alloc]
initWithFrame:CGRectMake(0,  0, 60, 44)];

    self.searchField = [[UITextField alloc]
initWithFrame:CGRectMake(0, 0, 250,30)];
    self.searchField.borderStyle = UITextBorderStyleRoundedRect;
    self.searchField.contentVerticalAlignment =
UIControlContentVerticalAlignmentCenter;
    self.searchField.placeholder = @"Search Foursquare";
    self.searchField.delegate = self;
    UIBarButtonItem *textFieldItem = [[UIBarButtonItem alloc]
initWithCustomView:self.searchField];

    self.customButton = [UIButton
buttonWithType:UIButtonTypeRoundedRect];
    [self.customButton setTitle:@"Search"
forState:UIControlStateNormal];
    self.customButton.tintColor = [UIColor redColor];
    [self.customButton addTarget:self
                          action:@selector(searchButtonTapped)
    UIBarButtonItem *searchButton = [[UIBarButtonItem alloc]
initWithCustomView:self.customButton];

    UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace
target:nil action:nil];

    [self.functionBar setItems:@[flexibleSpace, textFieldItem,
flexibleSpace, searchButton, flexibleSpace]];
    self.tableView.tableHeaderView = self.functionBar;
```

Κεφάλαιο 4ο

SIMULATION ΤΗΣ ΕΦΑΡΜΟΓΗΣ



Ο χρήστης μόλις ανοίξει την εφαρμογή, του παρουσιάζεται η λίστα με τα Meetings από τα οποία είναι προσκεκλημένος ή έχει δημιουργήσει ο ίδιος.



Κάνοντας scrolling στο home page παρατηρούμε το tap bar όπου ο χρήστης έχει τη δυνατότητα να επιλέξει ανάμεσα σε τρεις ενέργειες:

1. Να εμφανίσει τη λίστα με τα meetings(εκεί όπου βρίσκεται)

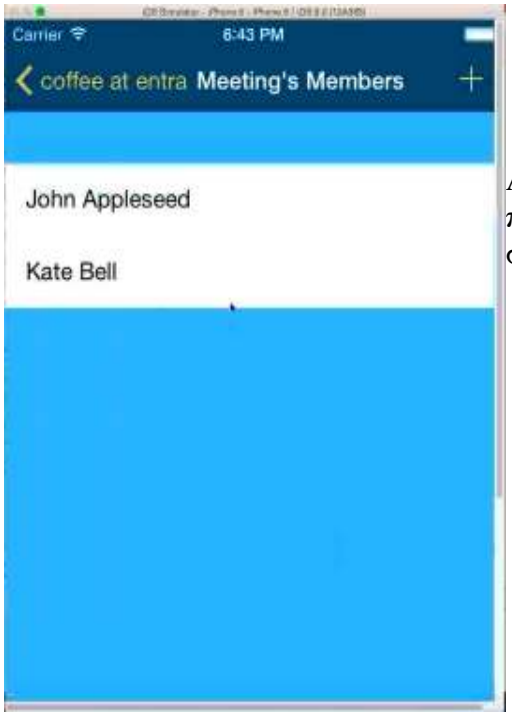
2. Να δημιουργήσει ένα καινούριο meetings

3. Να εμφανίσει το προφίλ του

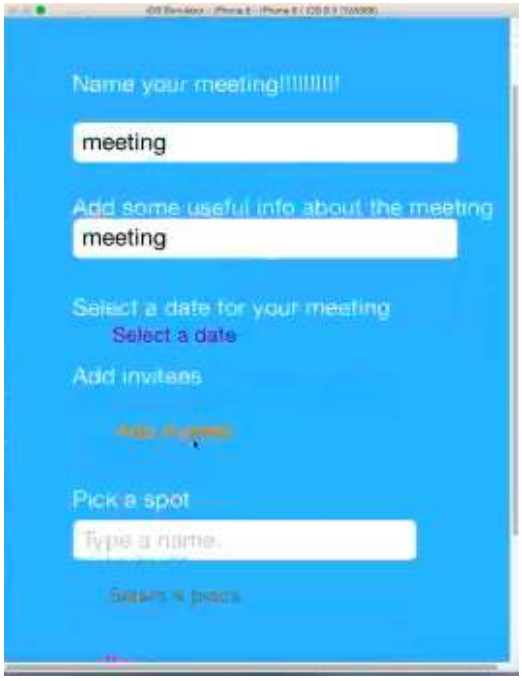
Έπειτα ο χρήστης έχει δυνατότητα να "πατήσει" σε κάποιο από τα υπάρχοντα meeting και να πλοηγηθεί στις περεταίρω πληροφορίες της συνάντησης.(π.χ. coffee at entra)

Στο σημείο αυτό ο χρήστης βλέπει τον δείκτη τοποθεσίας του σημείου συνάντησης επάνω στο χάρτη, καθώς και κάποια περιγραφή. Επίσης εμφανίζεται η ημερομηνία και ώρα της συνάντησης καθώς και οι καλεσμένοι.

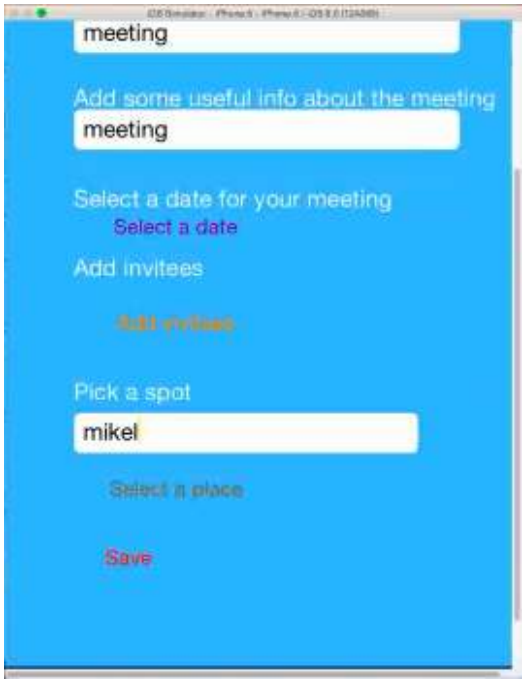
Ο χρήστης πατώντας παρατεταμένα πάνω στο χάρτη έχει τη δυνατότητα να κάνει zoom-in και zoom-out καθώς και να δει την ετικέτα marker(δείκτης τοποθεσίας). Σε μελλοντικό ανάπτυξη της εφαρμογής ο χρήστης πατώντας το κουμπί Direction θα λαμβάνει οδηγίες πλοήγησης προς το σημείο συνάντησης.



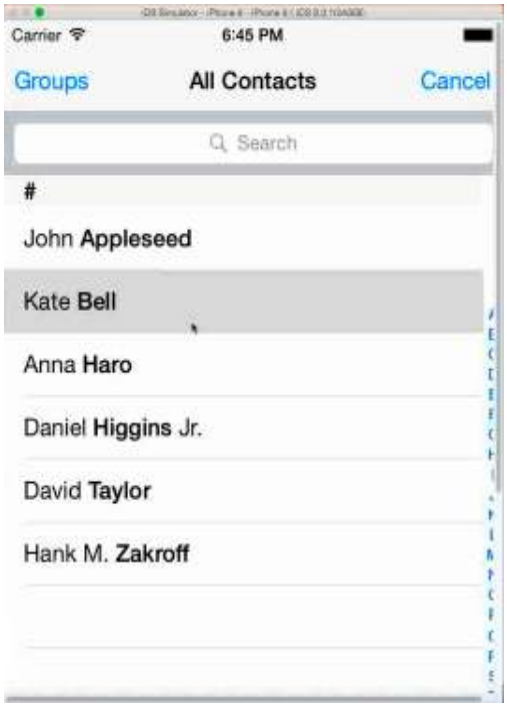
Αν ο χρήστης πατήσει πάνω στο κουμπί View invitees θα πλοηγηθεί στη λίστα από τα μέλη της ομάδας που συμμετέχουν στο meeting.



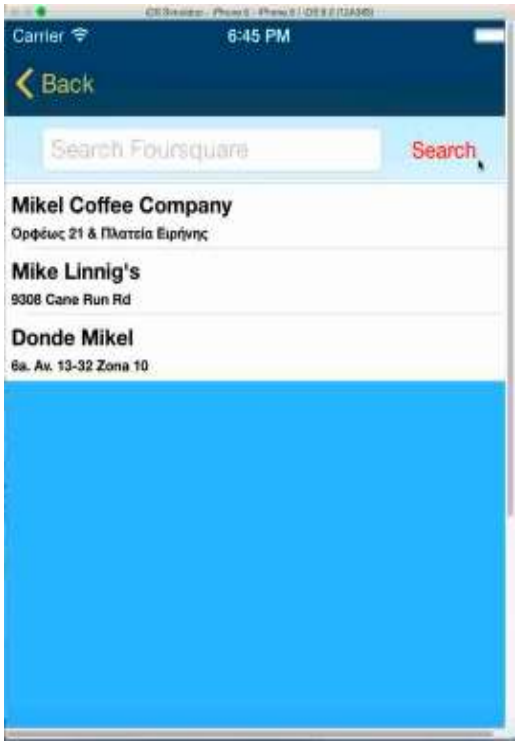
Επιλέγοντας από το tap bar “create a new meeting”.



Πατώντας το κουμπί “select a date” , ο χρήστης επιλέγει την ημερομηνία και ώρα της συνάντησης.



Πατώντας το κουμπί “add invitees”, ο χρήστης επιλέγει τα άτομα τα οποία επιθυμεί από τη λίστα επαφών του.



Πατώντας το κουμπί “pick a spot”, ο χρήστης καλείται να επιλέξει το σημείο συνάντησης που επιθυμεί.

Κεφάλαιο 5ο

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΒΕΛΤΙΩΣΕΙΣ

Ολοκληρώνουμε την διπλωματική εργασία, έχοντας μια εφαρμογή που δουλεύει όπως θέλαμε, χωρίς καθόλου λειτουργικά σφάλματα, και όσο το δυνατόν πιο εύχρηστη στον χρήστη στον οποίο απευθύνεται. Ωστόσο στο μέλλον θα μπορούσαμε να προσθέσουμε παραπάνω λειτουργίες ως προς την ανάκτηση directions από το google map ως προς το σημείο συνάντησης καθώς και διάφορα γραφικά εικονίδια και επιλογές που θα κάνουν πιο κατανοητή και φιλική την εφαρμογή μας στον χρήστη. Το πιο σημαντικό όμως είναι η απεικόνιση του υπολειπόμενου χρόνου άφιξης του κάθε μέλους της ομάδας με σκοπό την επίτευξη του τέλειου συντονισμού μεταξύ αυτών.

Από τη σκοπιά ενός Business Model

Το επιχειρηματικό μας μοντέλο στηρίζεται στη δωρεάν διανομή της εφαρμογής μας στο apple store και google play και σχεδόν μηδενική ροη εσόδων (reveue streams). Ωστόσο στόχος μας και στρατηγικός μας άξονας είναι η εφαρμογή μας να πάρει διαστάσεις κοινωνικής επιδημίας-social contagion σε μεγάλο εύρος χρηστών ανά τον κόσμο με αποτέλεσμα την εκτόξευση της υφιστάμενης αξίας της στην κορυφή των κοινωνικών μέσων τεχνολογίας. Το αρχικό κόστος επένδυσης και οι πόροι που απαιτούνται για το MVP θα καλυφθούν από τα ιδρυτικά μέλη και στη συνέχεια θα κατέβουμε για funding με τα μέσα που διαθέτουμε και σε crowdfunding για την περεταίρω ανάπτυξη του προιόντος και την ηγετική θέση του που θα επιδιώξουμε στην αγορά.
Η εφαρμογή μας «meetapp" θα είναι διαθέσιμη δωρεάν σε όλους τους χρήστες κινητών smartphones χωρίς κριτήρια και διακρίσεις . Θα είναι δωρεάν σε όλο το κύκλο ζωής της ως προιόν και δεν θα αλλάξει η τιμολογιακή της πολιτική σε οποιοδήποτε στάδιο αναβάθμισης ή πρόσθεσης παραπάνω λειτουργιών που θα την κάνουν πιο χρήσιμη και πρακτική στην καθημερινή μας ζωή . Σκοπός αποτελεί η συνεχής ανάπτυξη της σε παγκόσμιο εύρος με βελτιώσεις και πρόσθεση λειτουργιών που θα την κάνουν πιο άμεση και εύχρηστη με αποτέλεσμα να γίνει πιο αποδοτική ως προς την λειτουργία της και να καλύπτει τις ανάγκες των χρηστών της.
Σε πρωταρχικό στάδιο οι κινήσεις προώθησης και προβολής μας θα αποτελέσουν τα social media σελίδα και διαφημιστική δαπάνη στο facebook , twitter, Instagram etc .. , email –marketing , καθώς και διαφημιστικό trailer-promo video σε κοινωνικά μέσα προβολής. Επειδή ως ομάδα πιστεύουμε και έχουμε ως κύρια αρχή μας ότι το καλύτερο marketing είναι η **δια στόματος** διαφήμιση μεταξύ των ομάδων χρηστών και των **opinion leaders** conversation θα αναπτύξουμε με την πάροδο του χρόνου ολοένα και πιο αποτελεσματικά τις παραπάνω τεχνικές .

Κεφάλαιο 6ο

ΠΑΡΑΡΤΗΜΑ

Α. ΚΩΔΙΚΑΣ OBJECTIVE C

Β. ΒΙΒΛΙΟΓΡΑΦΙΑ

https://developer.foursquare.com

https://stackoverflow.com/questions/16167176/plot-venues-from-foursquare-on-google-maps?rq=1

http://en.wikipedia.org/wiki/Objective-C http://en.wikipedia.org/wiki/Object-

oriented_programming http://en.wikipedia.org/wiki/Smalltalk

http://www.raywenderlich.com/934/core-data-on-ios-5-tutorial-getting-started

https://developer.apple.com/library/mac/navigation/

http://www.merriampark.com/ldobjc.htm

http://stackoverflow.com

https://developers.google.com/maps/documentation/javascript/v2/reference?csw=1#GPolyli