116 Boulevard d'Avroy

4000 Liège

Belgium

Documentation Iphone

Année: 2016-2017

Sommaire:

I) Architecture and Infrastructure:

1) Requirements :

a) IOS development with Xamarin requires:

- A Mac running OS X El Capitan (10.11) or above.

- Latest version of Xcode and iOS SDK installed from the App Store .

b) Xamarin.IOS works with any of the following setups:

- Latest version of Xamarin Studio on a Mac that fits the above specifications.

- Latest version of Visual Studio Professional or higher on Windows 7 or above, paired with a Mac build host that fits the above specifications.

2) Components in project:

a) References : Contains the assemblies required to build and run the application. Expand the directory to see references to .NET assemblies such as System , System.Core, andSystem.Xml , as well as a reference to Xamarin's Xamarin.iOS assembly.

b) Components : The Components directory houses ready-made features from the Xamarin Components store , a public marketplace for Xamarin code. I have a created an asset catalogue named “Media” where we can add some images or icons for the application.

c) Resources: The Resources folder stores the launch page of application and a storyboard page where you can create user interface with the designer iOS integrated into Visual Studio.

d) Main.cs: This contains the main entry point of the application. To start the application, the name of the main application class, the AppDelegate, is passed in.

e) AppDelegate.cs: This file contains the main application class and is responsible for creating the Window, building the user interface, and listening to events from the operating system.

f) Main.storyboard: The Storyboard contains the visual design of the application’s user interface. Storyboard files open in a graphical editor called the iOS Designer.

g) Controller: The Controller folder contains all of View Controllers. The View Controller is responsible for handling interactions between the user and the View.

h) ViewController.designer.cs: The designer.cs is an auto-generated file that serves as the glue between controls in the View and their code representations in the View Controller. More information go to Introduction to the iOS Designer.

i) Info.plist: The Info.plist is where application properties such as the application name, icons, launch images, and more are set. I have set a key “Location When Use Usage Description” and his value string for location’s user request.

j) Entitlements.plist: The entitlements property list lets us specify application capabilities (also called App Store Technologies) such as iCloud, PassKit, and more. More information on the Entitlements.plist can be found in the Working with Property Lists guide.

k) ViewModels: This folder contains all of Views models like TableView, CollectionView, MapView, where I can customize my View and bound it to my View controller.

3) Application Fundamental:

a) Before application load a user interface, the application need to define an entry point by the page Main.cs and then it have to define a class to handle application-wide events and interact with the operating system:

Main.cs :

The main entry point of an iOS application is the Main.cs file. It contains a static Main method that creates a new Xamarin.iOS application instance and passes the name of the Application Delegate class that will handle OS events.

Application Delegate:

The *ApplicationDelegate* class handles system events and manages the application *Window*. The Window is a single instance of the `UIWindow` class that serves as a container for the user interface. By default, an application gets only one Window onto which to load its content, and the Window is attached to a *Screen* (single `UIScreen` instance) that provides the bounding rectangle matching the dimensions of the physical device screen.

User Interface:

The user interface of an iOS app is like a storefront, the application typically gets one Window, but it can fill the Window up with as many objects at it needs, and the objects and arrangements can be changed depending on what the app wants to display by using the ios designer, a visual tool for building user interfaces. The objects in screen, that the user sees, are called Views. Application have a Content View Hierarchies, each with its own View Controller, and the application places Views in the Window to create a different Content View Hierarchy based on the screen that the user is on.

Storyboard:

This file contains visual designs of application's screens. The representation of an application's screen in a Storyboard is called a Scene. Each Scene represents a View Controller and the stack of Views that it manages (Content View Hierarchy).

View:

The View is an instance of the `UIView` class that defines an area of the screen and provides interfaces for working with the content in that area. In the application, there is a root View Controller named "View Controller" defines by Navigation Controller.

SubView:

The objects like Button, Textfield, Label are called SubViews. They can be added to View for manage the interaction between View and User. Each SubView is identified by a name property which you can reference to View to determine his action.

ViewController:

This file is in charge to loading and unloading View from the Window. I call the method **ViewDidLoad** , once the first time the View Controller loads its Content View Hierarchy into memory. All of codes, placed after this method will be executed when the load of view is done. It's a file in which I implement behavior of a SubView responding to user's interaction.

ViewModels:

This file contains View model like tableView, collectionView, MapView which will be bound to the View of the ViewController.

4) Pattern MVC (Model, View, Controller):

I used the pattern MVC to separate my View, ViewController, Data of my application. MVC is architecture for applications with a Graphical User Interface (GUI). It assigns objects in the application one of three roles:

-Model: data from my Model stores in SqLite, accessing by Repository.

-View: user interface created by ViewModels and Designer Ios in storyboard.

-Controller: code in file View Controller to check out all of request from user.

II) Actor and Functional Requirement:

1) Actor's Application:

-Agent.

2) Functional Requirement:

-Display calendar's Agent.

-Display new list of visits.

-Display all of visits on map.

-Accept/Refusal a visit.

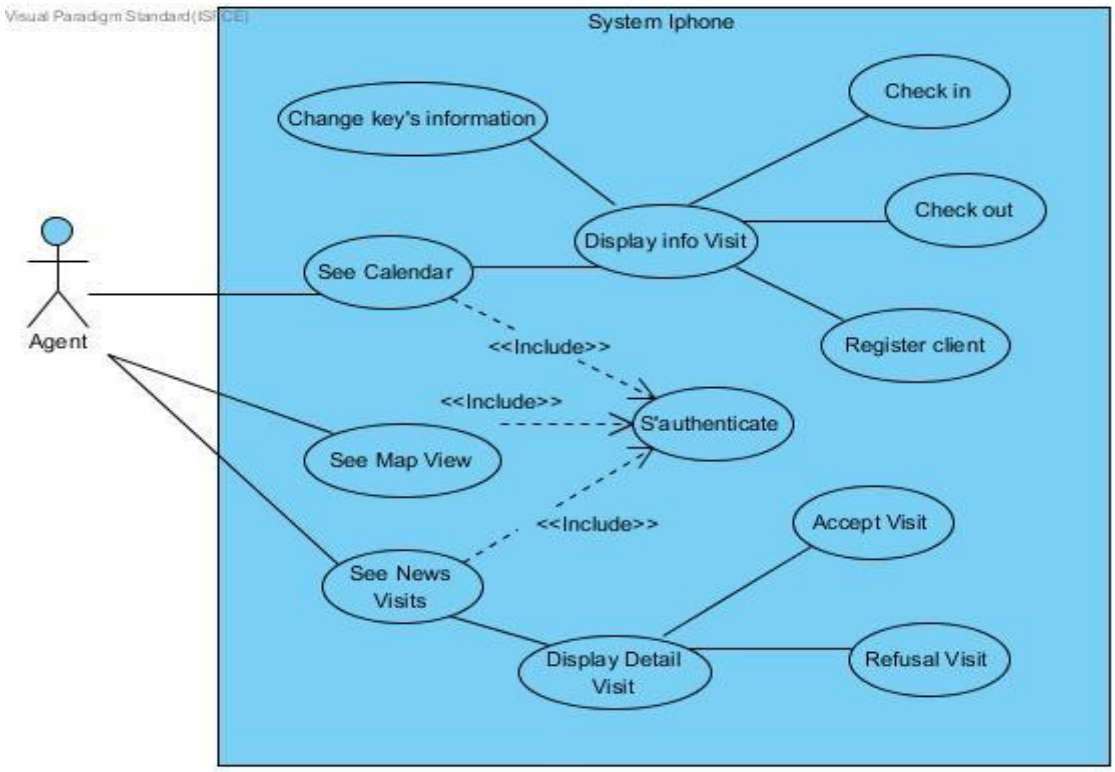
-Change key's information.

-Check in and check out for a visit.

-Form to register a client.

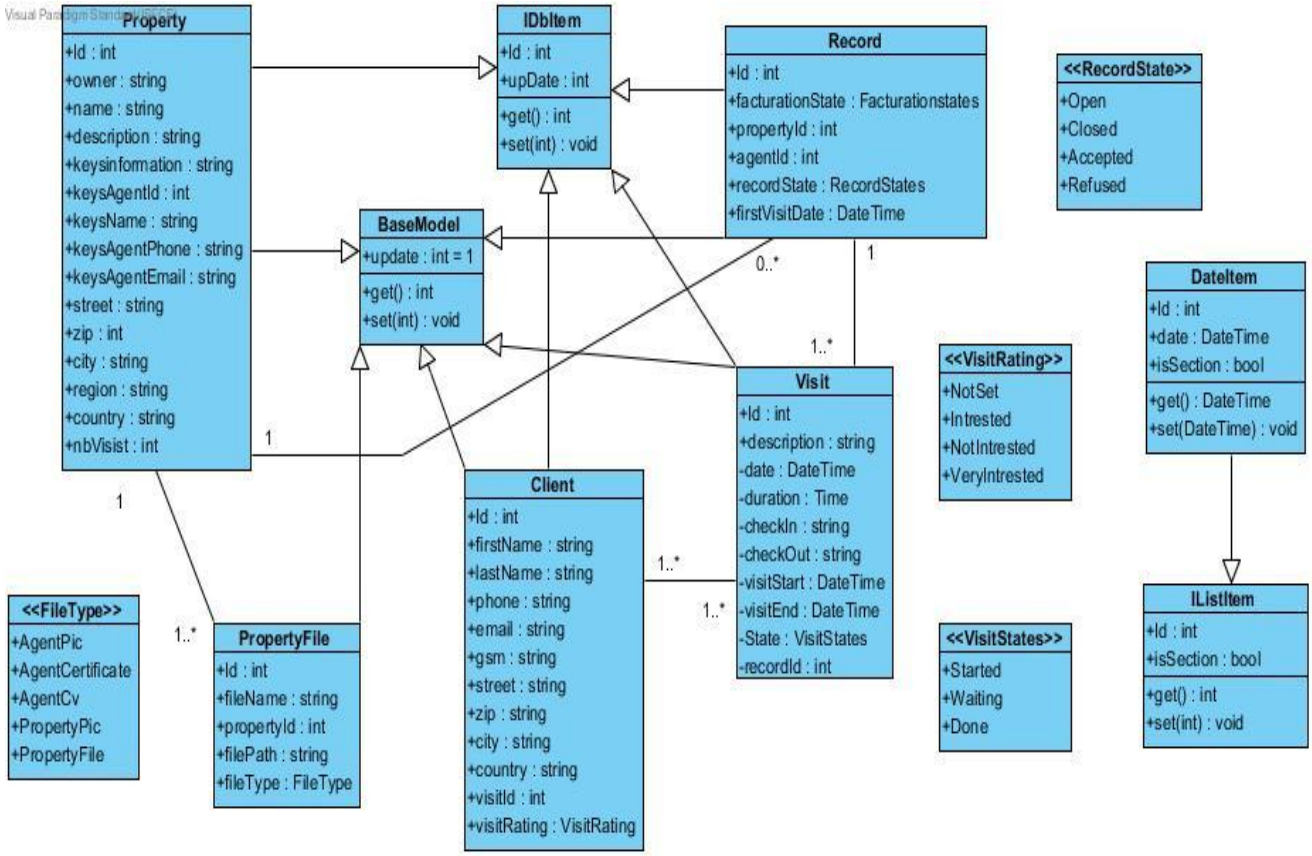
-Display list of file for property.

3) Diagram Use Case:

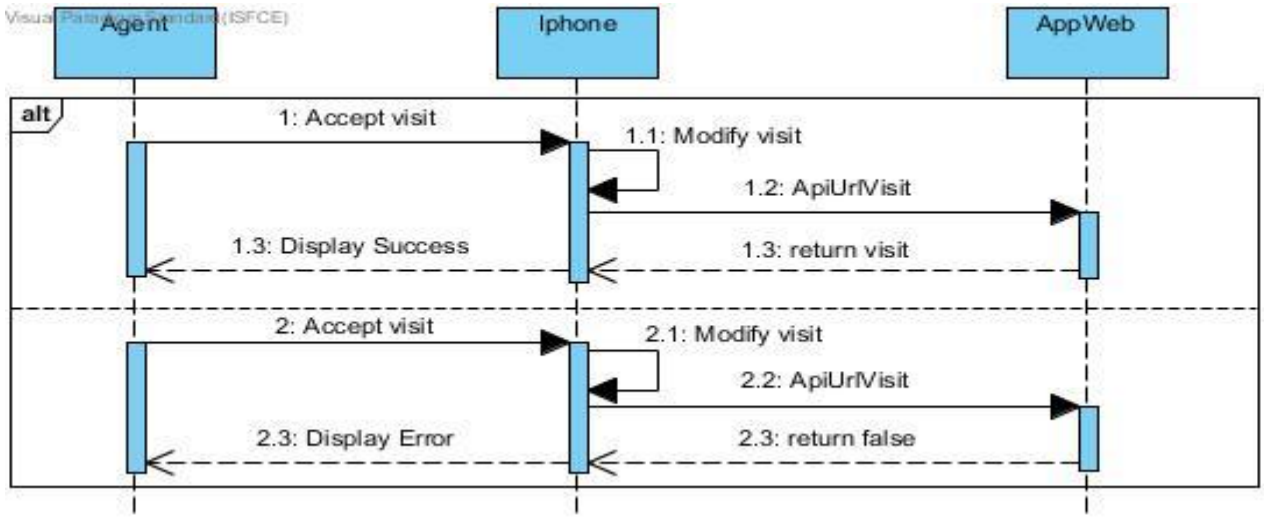


III) Database Sqlite:

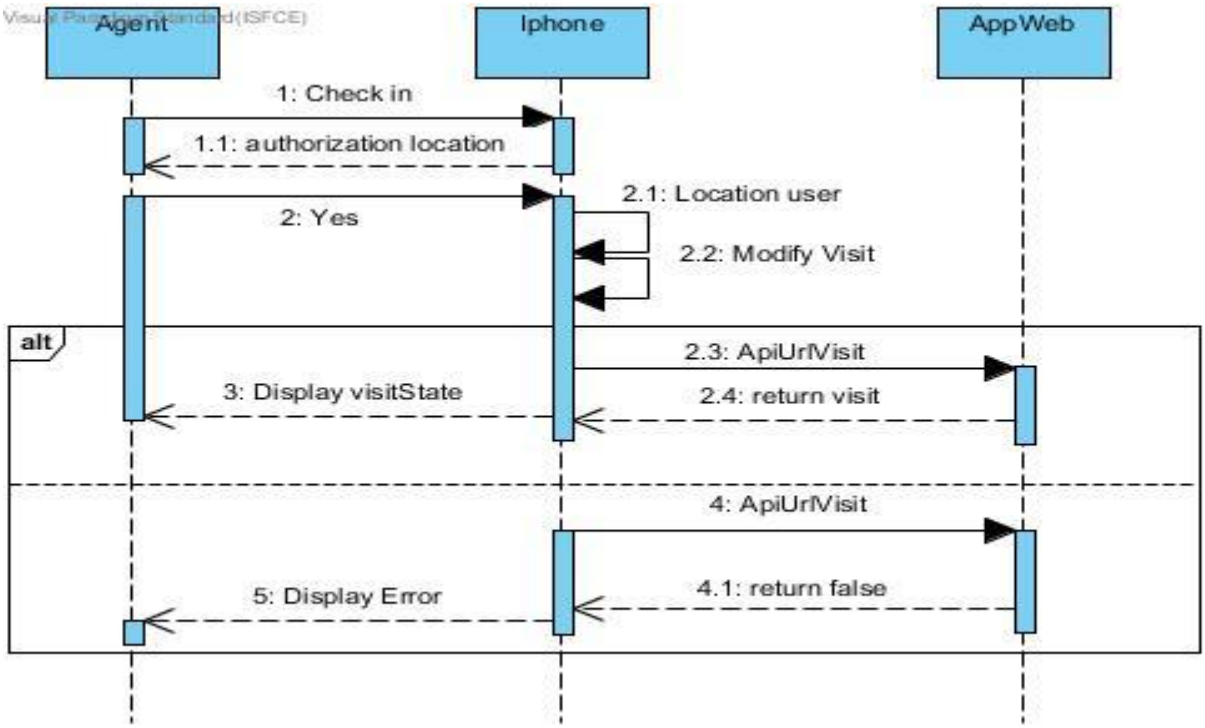
a) Diagram class:



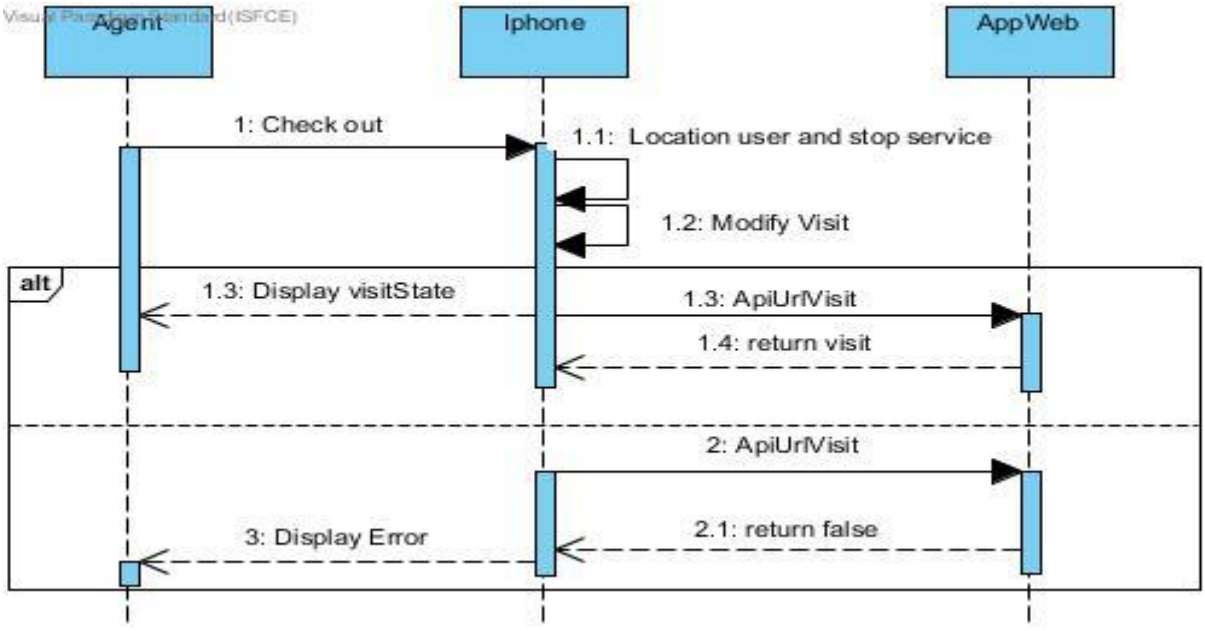
b) Diagram Sequence (Accept/Refusal Visit) :



C) Diagram Sequence (Check in) :

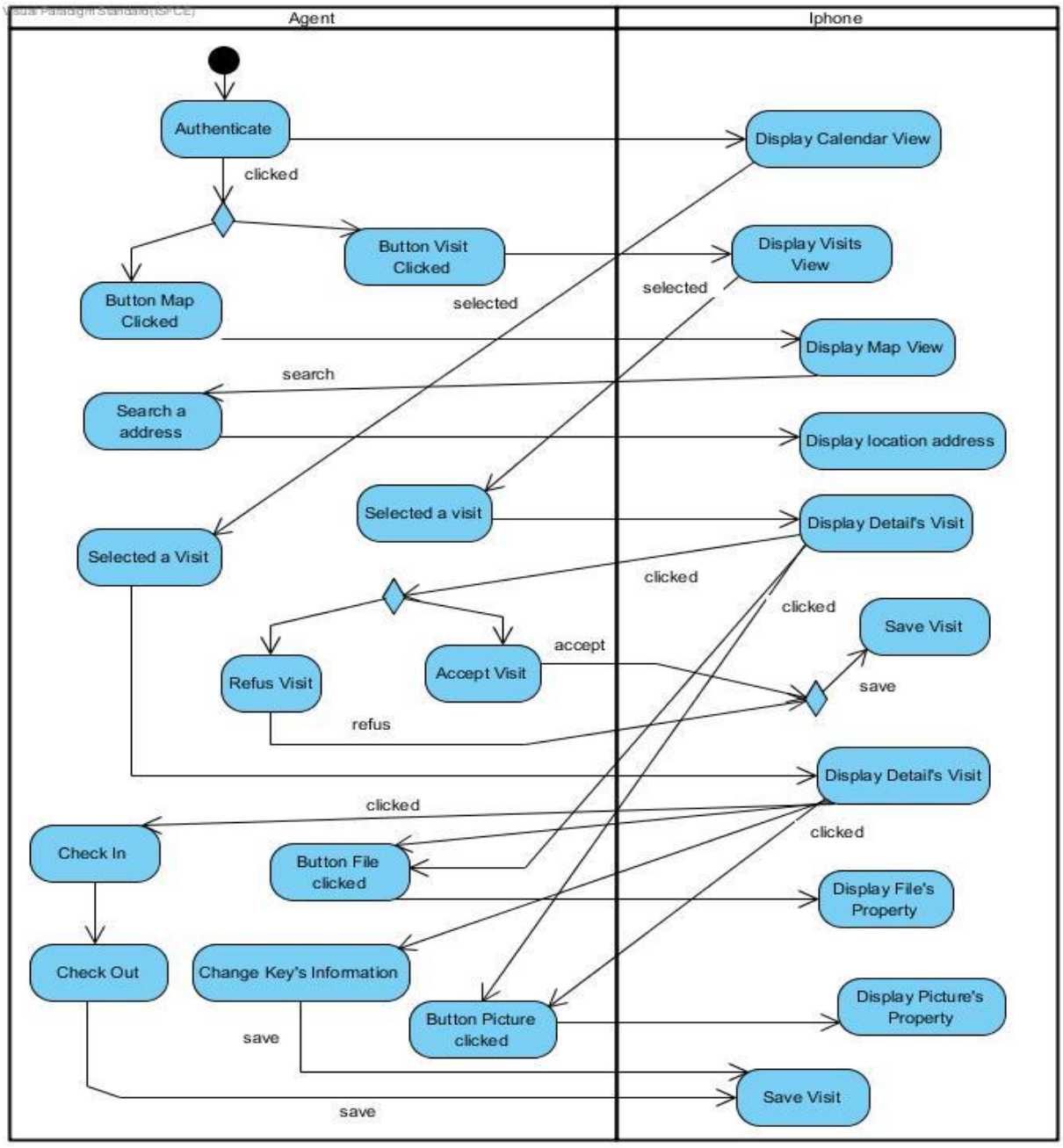


D) Diagram Sequence (Check out):



IV) Application Flow :

a) Diagram activity :



V) Assembly in project :

For my application mobile, i have used some libraries from Framework.NET:

1) SQLite-net : This assembly helps to create a database local with model class in project portable class library Core and the class CertDatabase in folder Data, then his place (where the database will be stock) is defined in the folder Context. It's lightweight, it doesn't require an extra

active services, it's fast and fully supported by Apple. Sqlite contains an embedded SQL engine and able to execute all the standard queries (select, insert, update, delete) that define in folder Repository of project. The data that I have receive from Api will be convert to model class by a special class (DataConvert) in folder APiclient before stoking in database Sqlite.

2) RestFul web Service:

This assembly is use to call Web service. REST requests are made over HTTP using the same HTTP verbs that web browsers use to retrieve web pages and to send data to servers. The code is implemented in folder ApiClient of project.

RESTful web services typically use JSON messages to return data to the client. JSON is a text-based data-interchange format that produces compact payloads, which results in reduced bandwidth requirements when sending data. The sample application uses the open source NewtonSoft JSON.NET library in order to serialize and deserialize messages.

3) RestSharp:

This assembly is use also to call web service. RestSharp makes it easy to consume the wide array of services on the web over HTTP, like: Amazon, Facebook, or even Twitter... It has a system automatic XML and JSON deserialization and detect automatically the type of content returned. It support all request standard web Api (Get, Post, Put, Delete).

4)System.Collection.Generic,Foundation,Coregraphics, UIKit, MapKit:

These are the assemblies secondary used to create user interface and map View.

5) Security.Cryptography, Runtime.CompilerServices:

These are the assemblies to use for decrypt the password of User while the service authorization. The code is implemented in folder Services of project by Crypto class.

6) Quicklook:

It's a new application introduce on the Mac OS X, allows users to look at the contents of a file in the Finder at full or near-full size, depending in the size of the document relative to the screen resolution. It can preview files sush as PDFs, HTML, plain Text, RTF, ODF, RAW.

7) What I haven't implemented in project:

- Implement a service to detect if Wifi is working. My project actually is working fine when the iphone is online but not in situation offline.

- I have implemented a system to read the files of property but it's not work yet. It defines in folder Services by PDFItem class and PDFPReviewControllerData. I try to use it in ViewFiles file which bounds to my FileController.

- Reload Data visit in VisitController. If I accept a visit, when I come back to my page VisitController, it hasn't to display the visit which I accepted, in new list of visits.