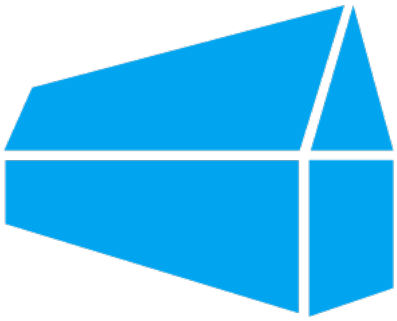
App Project Report
Design and Implementation of Mobile Applications

Filippo Pedrazzini, Emanuele Torelli

July 23, 2017



Contents

1 Introduction

1.1 Objectives

The objective of the project was to develop a professional application following the Microsoft requirements that they gave us in the beginning of the semester.

The document explains how we managed the work in order to have, just in few months, a minimum valuable product ready to be used in the field.

2 Requirements

2.1 Microsoft Guidelines

Microsoft didn't explain us in the detail all the requirements and functionalities that the app must have; infact with just few slides [microsoft.pdf] and emails it gave us an overall idea on what the app should do, without focusing that much on the technical requirements. From the slides the three key functionalities that the app should have do are:

- **Booking meeting rooms**: giving the possibility to the end user to see all the available rooms and book them in the current day or in the following ones.
- **Parking reservations**: showing free parks and allowing the personell to reserve one of them.
- **Shared calendar**: notifying the employees of new meetings and activities, that will be taking place in the office.

2.2 Requirements

At the beginning, with just these few informations, was difficult to understand and visualize clearly the app in our mind. We tried to give an interpretation of each main goal and we decided to put some assumption in order to have a good idea of what doing before starting to code. Here the explanation for each functionality.

2.2.1 Booking meeting rooms

The problem with this first requirement was identifying which kind of user can book the different rooms. In a work place there are many rooms and of course based on your function inside the company you should have more or less flexibility in doing this action. For this reason, we decided to leave this doubts and assume that all the app users could visualize and book a room selecting the time and date of the reservation.

2.2.2 Parking reservations

Here the issue was much more difficult. There are many ways to implement this specification and after thinking a lot we reached a solution. From the app the user can see the number of available parks, but it can not reserve it before getting to the place. When the user has arrived he can book and cancel a reservation using the QR Code reader in the app.

2.2.3 Shared calendar

In the specification is not clear who can create an event and which is the difference between a meeting and an event. We assumed that all the Users can create a public event in one of the main rooms of the building. These rooms are the biggest rooms in the building and they can afford to guest many people. When a public event is created all the users are notified of its occurrence through the app.

2.3 Use Case

In order to explain better the different functionalities in a visual way we decided to draw a Use Case Diagram.

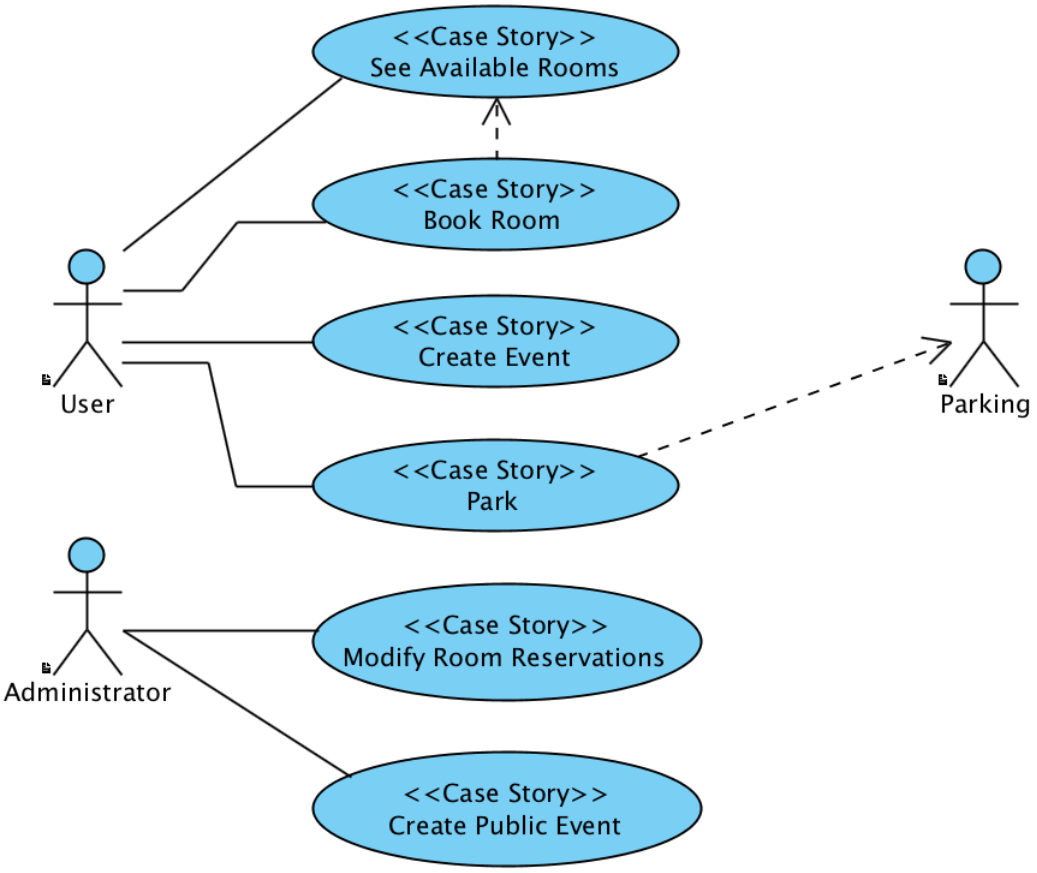


Figure 1: Use Case Diagram

3 Design Phase

3.1 Architecture

The choose of the architecture and the design was forced to the fact that we had to implement the application using Xamarin. Xamarin is a tool used for developing cross platform applications and the implementation is based on the MVVM pattern.

3.1.1 High Level

Showed below the High Level Component View in which are placed the main big component and their high level interaction.

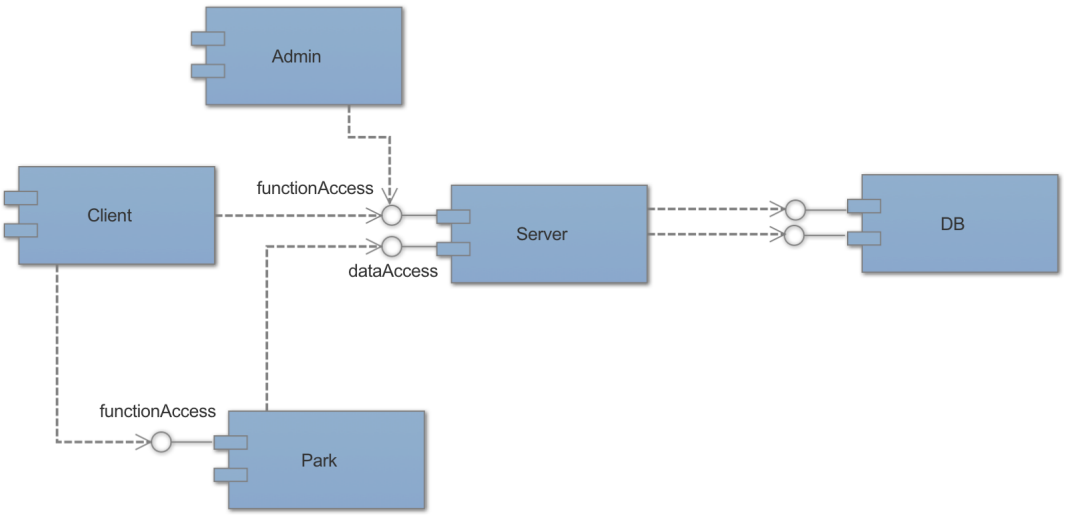


Figure 2: High Level Component View

3.1.2 Low Level

Here a brief description of the architecture of the different big components focusing our attention on the two main components.

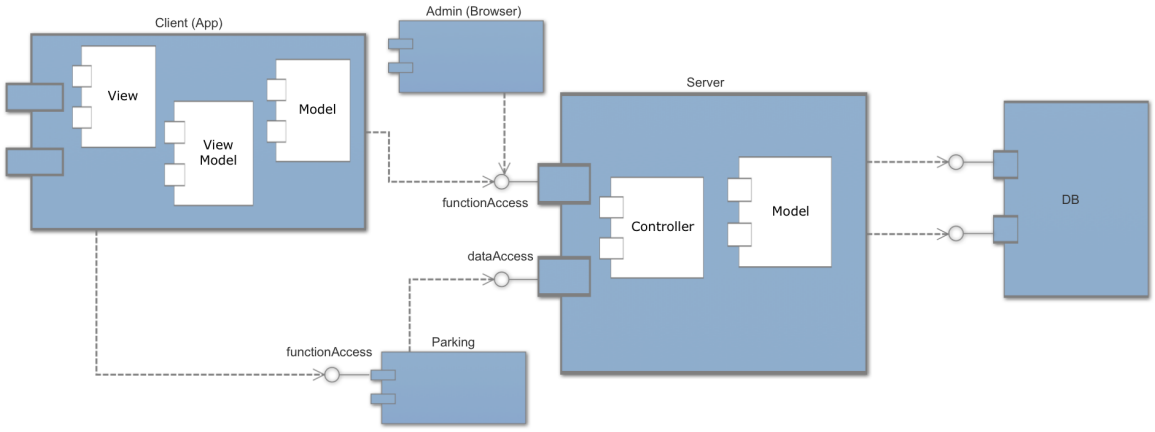


Figure 3: Low Level Component View

Client The Client which corresponds in this case to the User Application has composed by different components:

- **View**: corresponds to the classes needed to build the GUI.
- **View Model**: classes used to bind the view objects to the logical aspects. It has the objective to call all the different methods of the server.
- **Model**: rappresents the data objects placed on the server.

Server The Server has all the methods and functions needed to modify and update the DB.

- **Controller**: it has all the methods to update and modify the Data Objects.
- **Model**: rappresents the Data Objects on the server, each one with a map with the tables on the DB.

3.1.3 MVVM Paradigm

Xamarin is based on the MVVM Paradigm, which is an improved Pattern of the well known MVC. MVVM abstracts a view's state and behavior in the same way, but a Presentation Model abstracts a view (creates a view model) in a manner not dependent on a specific user-interface platform. That is why using this pattern it is possible to create cross platform applications.

"The view model of MVVM is a value converter; meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. In this respect, the view model is more model than view, and handles most if not all of the view's display logic. The view model may implement a mediator pattern, organizing access to the back-end logic around the set of use cases supported by the view."

3.2 User Interface

3.2.1 Mockup

Just to have an idea, here we show our initial mockup with some screens we did before starting to implement the real app.

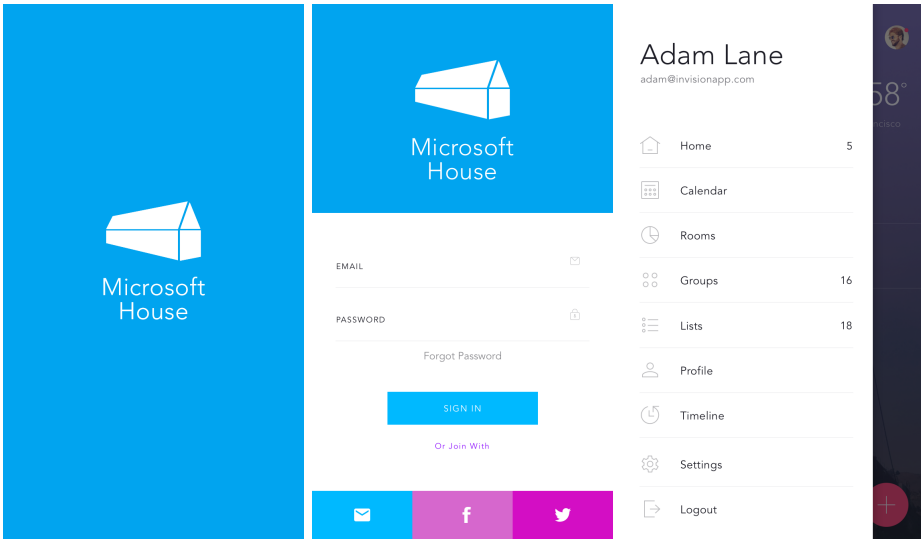


Figure 4: Splash Screen - Login - Navigation

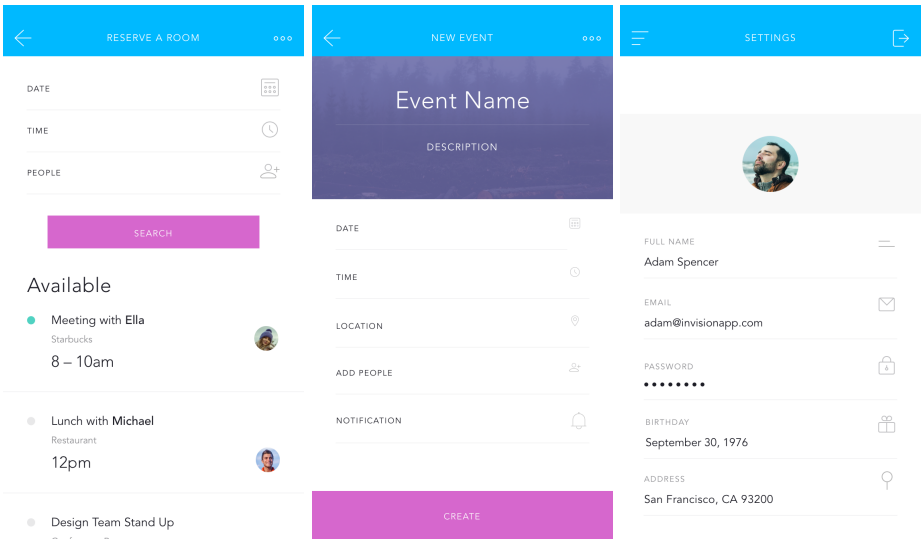


Figure 5: Search Room - New Event - User Settings

4 Implementation

In this section we are going to explain the different tools and methodologies we used to implement the app. In the end we will explain also the difficulties and the issues we resolved during the development phase.

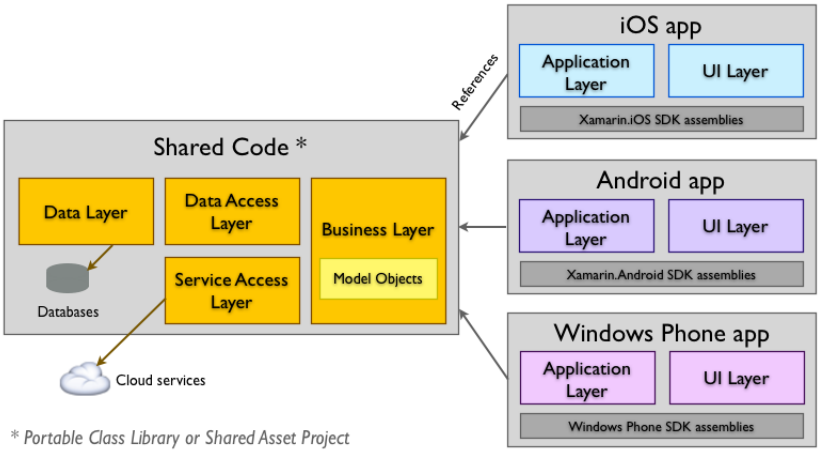
4.1 Tools

4.1.1 Xamarin

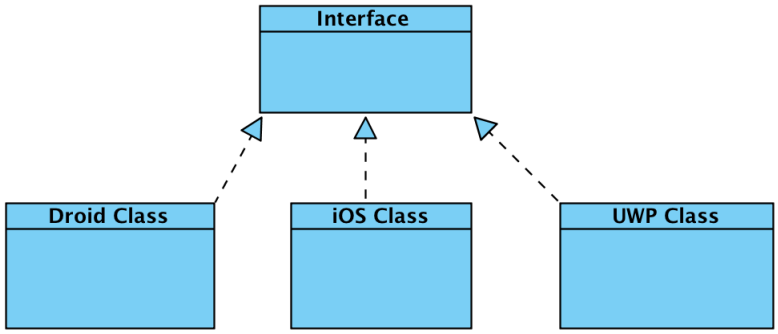
In order to develop the application for each mobile OS we used Xamarin which is a recent ide based on C# to implement cross-platform application. They say that using the PCL project you can share the 80% of the lines of code among the different platforms. The problem is that if you want to create a professional product you have to adjust and modify many things for each platform in order to make all working. Moreover, during the last year Microsoft acquired this platform and now they are doing a merge between Xamarin and Visual Studio. This is not an advantage for the programmer, which is constrained to keep the platform update hoping that after it, the software will keep working.

Shared Project The shared project configuration allows the developer to use most of the code for all the different platforms.

It creates 4 different floders (Shared, Droid, iOS and UWP) where Shared rappresents all the GUI classes and the logic behind, while the others are just used in case we need to touch a specific task and we have to implement it in three different ways (one for each platform).



There are some aspects in which you can not use the default Xamarin libreries, but you need to modify the specific classes for each platform. In these few cases the approach is always to create an interface in the Shared Project and then implement the interface in the different platforms, using native controls. In the figure beyond we show how the classes for a specific task are linked.



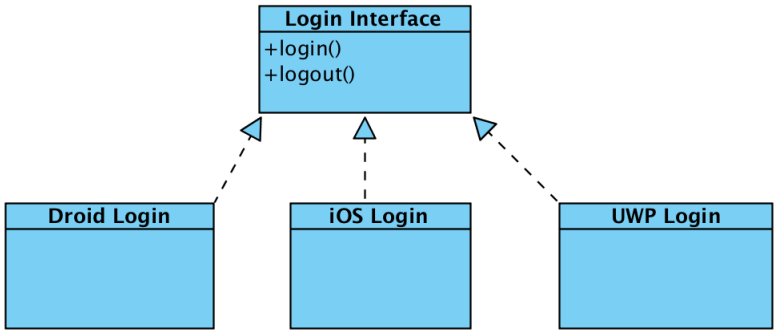
An example in which we used this kind of approach was in the render phase of each UI control (Time Picker, Date Picker...)

4.1.2 Azure

Azure is the most famous cloud service in the world and we decided to use it because of the documentation concerning the implementation with both Azure and Xamarin. Infact, both are part of Microsoft and what it is doing is a king of sinergy between the two platforms, in order to create the best development product. But we had many problems with Azure because of the new big update they did in the last few months. Due to this fact all the documentation was incomplete or obsolete.

4.2 External Services

In order to add additional services to our application, we inclueded some features provided by external services. Just to have a clear idea about what we wrote above, here an example of classes used to implement the Login functionality. The exact same thing happened with the other services.



4.2.1 Maps

Xamarin has the possibility to integrate the native platform apps just calling the right interface.

4.2.2 Login

Azure offers different kind of pre built logins; we decided to use it in order to simulate the Microsoft House App as best as we can.

4.2.3 Notifications

There is also the possibility to manage notifications using a third party interface that can be configured with Azure.

4.3 Issues

4.3.1 Requirements

The design of the user interface starting from the explained requirements was not that easy; infact most of the applications usually have just one main functionality, while this application is really complex and merging the different requirements keeping the usability and a good look was our main issue. It was a progressive and continuos and validation; due to this fact our first mockup was completely different from the today real app. The solution was to use different interface techniques in order to keep the usability and the speed that an app must have.

4.3.2 OS Rules

Figure 6: TabPage visualization in each Platform

Another problem that arises during the implementation phase was the fact that each mobile OS has its own User Interface rules, which means that even doing a sort of mockup in the beginning, it was

impossibile to keep and follow it for the duration of the entire project. We were forced to have a different GUI for each platform. In Figure 6 you can see how the navigation pattern (TabPage) has displayed in the different OSs.

5 Results

Here some screen shots in order to have an idea on the final result we obtained.

6 Conclusions

In the conclusion we can say that using Xamarin was worthy, but in the other hand due to the problems we had during the implementation because of the updates, we can say that it is not yet ready for the large scale. Moreover, to create a professional product you need to modify many native things and you loose in some way the strenght of being cross platform.