POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione


POLO TERRITORIALE DI COMO

Master of Science in
Computer Engineering

# Experience and comparison between native and hybrid development approaches for mobile devices

Candidate

Giona Colombo

Student Id. number 785986

Thesis Supervisor

Prof. Brambilla

Academic Year 2013/2014

POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione


POLO TERRITORIALE DI COMO

Laurea Magistrale in
Ingegneria Informatica

# Esperienze e comparazione tra gli approcci di sviluppo nativo e ibrido per dispositivi mobili

Candidato

Giona Colombo

Matricola 785986

Relatore

Prof. Brambilla

Anno Accademico 2013/2014

**Experience and comparison between native and hybrid development approaches for mobile devices**

Master thesis. Politecnico di Milano



This thesis has been typeset by LaTeX and the smcthesis class.

Author's email: giona.colombo@mail.polimi.it

*Dedicated to*
*...*

## Sommario

Lo sviluppo di applicazioni per dispositivi mobili è un argomento molto attuale. Si conoscono, ad oggi, tre diversi approcci per avvicinarsi a questo tema e ognuno di questi presenta vantaggi e svantaggi. Questi sono lo sviluppo di applicazioni *nativa*, *ibrida* e *web app*. L'applicazione nativa risulta essere più efficace ma è disponibile solo per un sistema operativo alla volta. l'applicazione ibrida non è efficiente come quella nativa ma può essere dedicata a tutti i sistemi operativi. La web app si trova solo online. In questa tesi prendiamo in considerazione soprattutto lo sviluppo *nativo* e quello *ibrido*. Questi due metodi di sviluppo per dispositivo mobile sono gli unici due modi per impacchettare e caricare un'applicazione su un negozio online. Non c'è modo migliore per sviluppare un'applicazione e gli sviluppatori hanno pareri contrastanti. Di solito si tengono in considerazione vari aspetti per decidere quale sistema utilizzare. Di questi, i più importanti sono risorse umane ed economiche. L'obiettivo di questo lavoro di tesi è illustrare il nostro punto di vista sull'argomento, valutare quale approccio riteniamo sia il più corretto, sviluppando un'applicazione per testare ognuno di questi approcci. Abbiamo sviluppato l'applicazione nativa per *iOS* e di conseguenza abbiamo testato l'applicazione ibrida su *iPhone* per comparare l'interfaccia dell'utente e le prestazioni. Abbiamo usato lo strumento *Xcode*. L'applicazione nativa è stata sviluppata usando il linguaggio di programmazione *Swift*. L'applicazione ibrida è, invece, stata sviluppata utilizzando *PhoneGap* che sfrutta, a sua volta, tecnologie *HTML*, *CSS* e *JavaScript*. L'applicazione che abbiamo sviluppato si trova a dover fronteggiare una questione alquanto attuale: la ripartizioni di spese comuni in un gruppo di lavoro. All'utente dell'applicazione è permesso aggiungere amici a un gruppo di lavoro, utilizzando le API di Facebook. Agli amici facenti parte di tale gruppo è concesso aggiungere i propri movimenti finanziari e ognuno di questi può sfruttare le API native della fotocamera e quelle di Foursquare del dispositivo per fare il check-in in un posto. Per tutta la durata di questo processo, l'applicazione calcola passo a passo se un membro del gruppo debba dare o ricevere denaro. Durante lo sviluppo abbiamo riscontrato varie difficoltà per ogni approccio. Quella predominante nell'approccio nativo è che il linguaggio di programmazione (Swift) è nuovo e poco conosciuto ma molto intuitivo e, inoltre, non presenta alcun problema di compatibilità con le API native. Le difficoltà maggiore dell'approccio ibrido sono, in generale, la complessità nello svolgere operazioni semplici e l'incompatibilità con le API. In conclusione diamo la nostra valutazione a entrambi gli approcci e spieghiamo dettagliatamente quale di questi riteniamo essere il più valido.

# Abstract

The application development for mobile devices is a very actual and discussed topic. There are three main approaches, and each of them has its advantages and disadvantages. These approaches are the *native* app development, the *hybrid* app development, and the *web app*. The native app is more performing but available only for one operating system at time. The hybrid app is not as performing as the native app but it can be allocated for all the operating system. The web app can only be found online. In this work of thesis we take into account especially the *native* and *hybrid* development. These two ways of developing for a mobile device are the only ways for packaging and deploying an app on a online store. There is not a better way of developing an app, and the developers have different opinions about the topic. Usually the choice of the approach is given by several factors. The most important are economical and human resources. The purpose of this work is trying to explain our idea about this matter, decide which approach we would generally choose, by developing an app with both approaches. We developed our native app for *iOS* and consequently we tested our hybrid app on an *iPhone* for comparing the user interface and the performance. We used the tool *Xcode*. The native app has been developed in the programming language *Swift*. The hybrid app has been developed using *PhoneGap* that is exploiting *HTML*, *CSS* and *JavaScript* technologies. The app we developed is facing a very modern issue: the repartition of common expenses in a group of work. The user of the app is allowed to add friends to a group of work, exploiting the Facebook APIs. The friends who are part of the group are allowed to add some financial movements they do. Each financial movement can exploit the device's native APIs of the camera and the APIs of Foursquare for checking-in to a place. During the whole process, the app is calculating step-by-step if a person present in the group must give or receive money. During the development we faced different issues for the two approaches. The main problem of the native approach is that the programming language (Swift) is new and not known. But, on the other side, it's very intuitive. Besides, with Swift there are not problems related to the native APIs compatibility. The main problems of the hybrid approach are the complexity, even for easy operations, and the compatibility with the APIs in general. Finally, we give our evaluation on both approaches, and we carefully explain which approach is, in general, better for us.

# Acknowledgments

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This work of thesis is the result of an experimental research aimed at the evaluation of the different approaches in the development of an app for mobile devices. In particular we have worked on native and hybrid approaches, we developed an app with both approaches and we tried to achieve some relevant result. We tried to establish which approach suits more the development of a simple app by evaluating the time and the issues we encountered.

## 1.1 Context

The universe of the mobile devices development is very actual and it grows day by day.[2] While for other fields like web or desktop programming there are a strengthened methodology and a clearer hierarchy, these doesn't exist, or they're not very well defined, for mobile programming. What is clear is that there are very well defined characteristics for the three main approaches of mobile programming:

- Web apps
- Hybrid apps
- Native apps

The *web apps* are multi-platform web sites developed for mobile devices. They are for sure the cheapest to develop, since most of the time they derive from an already existing web site. Plus the technology used for programming is well known. There are a lot of developers who know perfectly HTML and CSS. The main issues of a web app is that it cannot be deployed on an app store, it cannot be integrated with any of the native APIs and it's only online. This is why, in this work of thesis we will only give a superficial explanation about this technology, without going throught the details, and we will not use it for our experimental part.[1]

The *hybrid apps* are web applications packed into a native app. They are, in a sense, a middle way between web and native apps. They have some similarities with the web apps and some with the native. As the web apps, the technology used for programming is very often HTML, CSS and JavaScript, and they're multi-platform. But the can also be deployed on an app store, they can use native APIs.

The *native apps* are applications build exclusively for one operating system. These apps are very powerful and with high performance, they can be deployed on an app store and they can, obviously, use native APIs with full compatibility. The greatest lack of a native app is that is single-platform, and this fact increases the cost.
We developed two apps, using the two main different approaches:

The mobile operating system that we used for developing a native app is *iOS*. This operating system, is the only OS available for Apple's mobile devices. For building a native app is necessary to exploit *Xcode*, an integrated development environment present only for *Mac OS X*, the Apple's operating system for computers.[3]

The hybrid app has been developed using *PhoneGap*. For better comparing the two apps, we developed our app with Xcode on Mac OS X. We also used some tool like *Handlebars* for managing the different pages of the app, and *Ratchet*, a tool that gives a native user interface to hybrid apps. For our database wee used *Parse*.

## 1.2 Problem Statement

The problem we tried to solve with this work of thesis is related to the choice that a developer must take before starting to develop a mobile app. We tried to understand which issues, for both approaches, are less relevant and easy to bypass.

By developing the same app with the native and hybrid approach, we tried to understand if the known problems are real or if it's possible to find a compromise. The main problems we took into account for hybrid development are:

- Performance
- Cost
- Time
- APIs compatibility
- User interface

The main problems we took into account for native development are:

- Cost
- Time
- The inability to build the app for other operating systems

## 1.3 Structure of the Thesis

This work of thesis is structured in this way:

- In the chapter 2, with the help of some source and reference, we explain which is the state of the art of the mobile development. Then we show which technology we used to reach our goal and solve our problem.

- In the chapter 3 illustrate in detail the advantages and disadvantages that a developer might have by using the three different approaches in mobile development.
- In the chapter 4 we elucidate which problem is solving the app we developed. We describe the app by using an entity-relationship model and a use case model.
- In the chapter 5 we show how the app we developed is working and what it's doing with the help of a collaboration diagram and some activity diagram.
- In the chapter 6 we display how our apps have been implemented with the two different approaches
- In the chapter 7 we exhibit which conclusions we reached and what are the possible future implementations.

# Chapter 2

# Background

In this chapter we present the general background of the actual situation of the mobile app development. Then we present the main technologies that we used.

## 2.1 Mobile app development

Nowadays smartphones are the main handheld devices for the majority of people, which makes mobiles applications crucial in both technical and commercial fields. The approaches for developing mobile Web apps can be various, but taking into consideration the high-speed of software evolution with continuously appearing new gadgets, the comprehensive understanding of basic technologies became critically important. [1]

The previous trend was creating desktop applications specially for an OS like Windows or Unix. However sequentially the tendency is changes, as developers mainly want to make the application work for mobile devices. The decision-making process for creating mobile apps is multipart, mainly because of the significant increase in the number of platforms and frameworks. Therefore understanding the various types of mobile apps and the various options for building them became crucial [2].

### 2.1.1 Mobile web and apps

By definition, native apps run at native speed, while hybrid and mobile apps run on top of additional layers, which consumes computing resources and can decrease the app's speed.

Different users and developers use both responsive design and adhoc options. [4]

#### Responsive web

Applications with a responsive Web design handle different style solutions, mainly found on cascading style sheets (CSS). While serving the application the server can choose the design, also the chosen design can be applied at the client level or both can occur.

The intention is to have a single source of content that renders variously depending on the features of the device. It is worth mentioning that this is not only a solution

for Web apps, but also critically important for such devices as tablets, tv sets game consoles and etc.

#### Mobile web

Mobile web expression is used for labelling the case when there is an exact website or login implementation for content to be sent specifically to mobile devices. In general the feel and touch of Mobile web is better than the feel and touch of responsive one, as it renders the user Interface controls , (for instance, buttons, selectors, and text-boxes) in the similar with native app way. Nevertheless, with mobile Web, the need to retain various sites is still present.

#### Hybrid apps

Hybrid apps are mobile Web applications packed into a native app. Hybrid apps perform like a native app: like native apps, hybrid are also installed from a Web store and have access to the Capabilities of native app. However, hybrid apps are developed by the similar tools used to develop Web applications (mainly, HTML5, CSS, and JavaScript).

#### Native apps

Organizations, which develop MOS (mobile operative systems) prefer apps, specific to their own field, for benefiting fully from their specific features. In its turn, it necessitates building the app using that provider's language and framework ( for example, using Xcode with Objective-C for iOS and Eclipse with Java for Android).

Therefore, one project should be maintained for each OS, which, obviously means a significant increase in the development team, and as a result, in time and financial costs. Furthermore, developers must discourse the fact that new OSs constantly appear.

### 2.1.2 Technical considerations

The most suitable development approach for mobile apps for various situations cannot be chosen based on a single solution. Different technical norms and conditions can help to decrease the amount of possible options for selecting the most suitable approach in a specific situation. [5]

#### Platforms and version support

Primarily, the platforms and versions to be supported have to be considered, counting the range of devices, the development stack for each of them, the platform's browser capabilities, and our own development skills. For instance, a hybrid or mobile web approach can be considered as a better solution than the native app, in case if the aim is to develop an app with support for multiple platforms, as native apps should be developed exactly for each mobile platform.

#### Device capabilities

Another important point is consideration of required device capabilities. For instance, if the app needs access to the came, a barcode scanner, the file system, or a Bluetooth, it is more useful to Implement a native or hybrid approach because of the direct access to the features. The most up to date browsers maintain hardware-accelerated animation features, however they are not able to use device capabilities at its maximum.

#### User experience

Worth mentioning is that native apps still offer a more comfortable and more convincing experience with a more approachable interface and superior interaction , as users are able to open them faster as well as use device-specific hand gestures. Furthermore, it is still challenging for mobile apps to access a device's native features across all mobile browsers. In its turn, Hybrid apps offer an important middle ground in terms of the deepness of experience. Hybrid apps allow the HTML code to have access to native APIs (although this comes at the cost of a nonnative user interface due to the Web technologies involved).

#### Performance

One of the key concerns of app-developers is the performance. Best performance is harder to be achieved using mobile and hybrid apps approaches, in case if the user interfaces needs heavy graphics or extreme data processing. The reason is that mobile and hybrid apps are running on top of additional layers, which uses computing resources. In any case it is critically important to test the level of performance, using a initial prototype or testing other comparable existing apps.

#### Upgrade

In case if developers choose native app development approach, they should take into account that app upgrade cannot be forced, so, by default, they will be concurrently serving various versions, adding complexity to back-end development and support, which also relates to hybrid apps when the code is in the local part of the application.

### 2.1.3 Nontechnical considerations

Several nontechnical considerations can also help to identify the best solution for choosing a type of mobile application to develop.

#### Distribution

Even though mobile apps are easy to distribute, it can be difficult for users to discover them outside the app store. From discoverability point of view, it is better to use a native or hybrid app. In case of targeting consumer or gaming segments, it is possible to significantly reduce the marketing expenses by handling the distribution through the platform store. However, taking into account the fast

increase of applications in these stores, efforts for obtaining visibility in the market are more vital. In case if there is a necessity to bound the reach of the applications (for instance, in case of developing enterprise apps) it is possible to use a private enterprise app store. Nevertheless, these stores have their restrictions (for instance, there is no chance to effect the store-management). Moreover, the store might have numerous applications, which are in a tough completion for users' attention.

#### Approval cycle

Mobile development and agile development methods can conflict with each other. Fast turnover and continues user feedback are essential for agile development. If the developers are following native or hybrid approaches, approval process should be considered as a part of the project. In case of no rejections, the time for approval and the total time generally are short. It is also possible to have a license for a particular phone, as well as to use a private enterprise app store.

#### Monetization

Together with the distribution advantage, there is a possibility to significantly improve conversion rate due to a simple, well-defines payment gateway, in case of the app development is found on platform store. The negative side of this advantage is the cost associated with it: the owner of the platform store receives the considerable part of the revenue (for instance, 30% for iOS), therefore there is a necessity to thoroughly examine if it is better off outside of the platform.

#### Frameworks

Despite the possibility to build a mobile Web, hybrid or native app without a framework, however the use of frameworks can significantly easer as well as decrease the effort for development process. In its turn, frameworks are available for various development preferences, majority of them are for HTML5 development. Nevertheless, if the choice is to develop for a hybrid or native approach with, considering a cross-platform development frame, there are various options to be reviewed.

#### Native

Native app can be developed using a cross-platform approach, but just one codebase for all devices. We can select our platform on the basis of our preferred development language (for example, JavaScript or Ruby).

#### Hybrid

For a hybrid approach, Phonegap is the predominant technology and can be used with HTML5 frameworks. Also Sencha mobile packaging can be considered, which complements the Sencha development stack but only supports iOS and Android. Nonetheless, it could be a good option when developing with Sencha tools. Other important cross-development tools are Appcelerator, Adobe AIR, and Qt.[4]

| Considerations | Native | Hybrid | Web |
|---|---|---|---|
| Effort of supporting platforms and versions | High | Medium | Low |
| Device capabilities access | Full | Full | Partial |
| User experience | Full | Full | Medium |
| Performance | Very high | High | Medium |
| Upgrade the client | Needed | Needed | Not needed |
| Ease of pubblication/distribution | Medium | Medium | High |
| Approval cycle | Mandatory | In some cases | Not required |
| Monetization in app store | Available | Available | Not available |

**Figure 2.1.** Criteria to consider when choosing a native, hybrid, or Web app approach.

#### HTML5

For choosing an HTML5 framework, it is necessary to try each of the frameworks, in order to select the one most fitting to the development practices. Various options in the HTML framework are available. There will be a significant progress as more features are added and more devices are supported. Nevertheless, as a consequence of this trend, the framework's codebase is getting bigger, which considerably influence performance and bandwidth consumption. We suppose to see some specialization in frameworks' capabilities, with different approaches to overcome this issue. (For example, JQuery Mobile can select just the necessary modules for a project.)

## 2.2 Relevant Technologies

In this section we present an overview of the technologies we adopted.

### 2.2.1 Xcode

Xcode is an integrated development environment (IDE) containing a suite of software development tools developed by Apple for developing software for OS X and iOS. First released in 2003, the latest stable release is version 6.1 and is available via the Mac App Store free of charge for Mac OS X Lion, OS X Mountain Lion, OS X Mavericks and OS X Yosemite users. Registered developers can download preview releases and previous versions of the suite through the Apple Developer website. [6]

#### Major Features

Xcode can build universal binaries thanks to the Mach-O executable format, which allow software to run on both PowerPC and Intel-based (x86) platforms, and both 32-bit and 64-bit code.

Xcode also includes Apple's WebObjects tools and frameworks for building Java web applications and web services.

Xcode includes the GUI tool Instruments, which runs atop DTrace, a dynamic tracing framework created by Sun Microsystems and released as part of OpenSolaris, an open source computer operating system.[3]

#### Composition

The elements which are componing Xcode are:

- The **Integrated Developement Environment (IDE)**, also named Xcode
- A **compiler**, the LLVM (Low Level Virtual Machine), a compiler infrastructure designed as a set of reusable libraries with well-defined interfaces.
- A **debugger**, the LLDB debugger.

Xcode also supports several languages such as C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, Rez and Swift.

### 2.2.2 Swift

Swift is a new programming language created by Apple for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility. Introduced at Apple's 2014 Worldwide Developers Conference (WWDC), Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products.

Swift adopts safe programming patterns and adds modern features to make programming easier. It is built with the LLVM compiler framework included in Xcode 6, and uses the Objective-C runtime, allowing C, Objective-C, C++ and Swift code to run within a single program.[7]

#### History

Developement of Swift began in 2010 under the guide of Chris Lattner, and it took language ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU and other programming languages.

The first app written in Swift was released on June 2nd, 2014 (WWDC app), together with "The Swift Programming Language", a free 500-page manual.

The 1.0 version of Swift and the 6.0 version of Xcode were released on September 9th, 2014. The 1.1 version was released on October 22nd, 2014, alongside the launch of Xcode 6.1.

#### Features

By default, Swift does not create pointers and other unsafe accessories, contrary to Objective-C, although pointers can be created explicitly. Additionally, Objective-C's use of a Smalltalk-like syntax for making method calls has been replaced with a dot-notation style and namespace system more in common with other modern languages derived from C, like Java or `C#`. Swift introduces true named parameters and retains key Objective-C concepts, including protocols, closures and categories, often replacing former syntax with cleaner versions and allowing these concepts to be applied to other language structures, like enums.

- **Types, variables and optionals**

Objective-C provided various bits of simplified syntax to allow an easier creation of the objects, but once created they were managed with object calls, making the code quite complicated. For instance, concatenating two NSStrings required method calls similar to this:

```
NSString *str = @"hello,";
str = [str stringByAppendingString:@" world"];
```

In Swift, many of these basic types have been promoted to the language's core, and can be manipulated directly, automatically bridging strings to the NSString. There is no need of specifying the type and the strings can be concatenated by using the "+" operator:

```
var str = "hello,"
str += " world"
```

As the most languages, Swift allows to create constraints and variables, and it does that by using respectively the keywords `let` and `var`.

Another important feature is the possibility of creating an optional pointer, or rather a pointer that may exists or that may be null, avoiding all the "null pointer errors". This can be made with this declaration:

```
var optionalInteger: Optional<Int>
```

- **Libraries, runtime and development**

  Swift uses the same runtime as the existing Objective-C system but requires iOS 7 / OS X 10.9 or higher. Swift and Objective-C code can be used in a single program, and by extension, C and C++ as well.

  It's even possible to import projects developed in Objective-C and accessing to all the functions and classes by simply using the code:

```
#import "MyApp-Swift.h"
```

  Swift also has limited the set of attributes, metadata that is read by the development environment, and is not necessarily part of the compiled code. Like Objective-C, attributes use the `@` syntax, like the `@IBOutlet` attribute.

- **Memory management**

  Swift uses Automatic Reference Counting (ARC) to allow for easier memory allocation and deallocation.

One problem with ARC is the possibility of creating a strong reference cycle, where instances of two different classes each include a reference to the other, causing them to become leaked into memory as they are never released. Swift provides the `weak` and `unowned` keywords that allow the programmer to prevent strong reference cycles from occurring.

- **Debugging and other elements**

  A key element of the Swift system is its ability to be cleanly debugged and run within the development environment, using:

  - **REPL (read-eval-print loop):** it takes single user inputs, evaluates them, and returns the result to the user;
  - **Playgrounds:** interactive views running with the Xcode environment that respond to code or debugger changes on-the-fly.

- **Similarities to Objective-C**

  - Basic numeric types (`Int, UInt, Float, Double`)
  - Most C operators are carried over to Swift, but there are some new operators
  - Variables are assigned using an equals sign, but compared using two consecutive equals signs. A new identity operator, ===, is provided to check if two data elements refer to the same object.
  - Control statements, `for, while, if, switch` are similar, but have extended functionality, e.g. a `for in` that iterates over any collection type, a `switch` that takes non-integer cases, etc.
  - Class methods are inherited, just like instance methods; `self` in class methods is the class the method was called on.

- **Differences from Objective-C**

  - Statements do not need to end with a semicolon (`;`), though they must be used to allow more than one statement on a line
  - Header files are not required
  - Strong typing
  - Type inference
  - Functions are first-class objects.
  - Enumeration cases can have associated data (algebraic data types).
  - Operators can be redefined for classes (operator overloading), and new operators can be created.
  - Strings fully support Unicode. Most Unicode characters can be used in either identifiers or operators.
  - No exception handling (though it can be emulated through use of closures)
  - Several notoriously error-prone behaviors of C-family languages have been changed,:

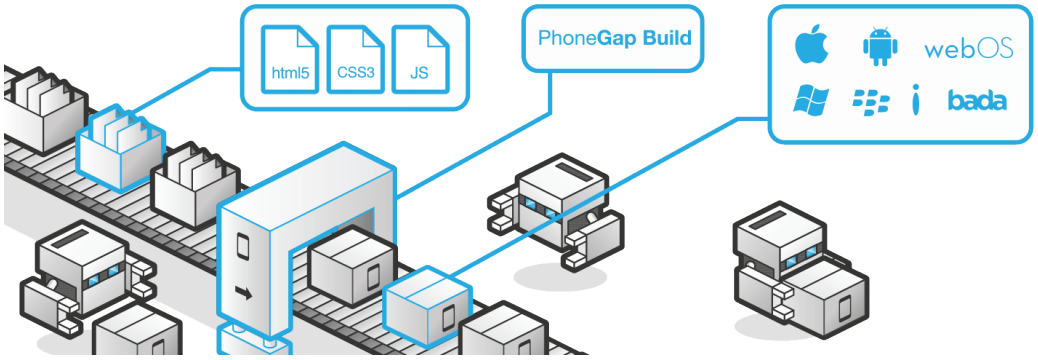


**Figure 2.2.** PhoneGap Build

- **Comparing with Python**

  Python is one of the languages that helped inspire Swift.

  - Both languages feature Read–eval–print loop (REPL) development environments
  - Conditional statements are similar `for` , `if` , `while`
  - Swift is a compiled language while standard Python is interpreted
  - Whitespace is not significant
  - Python doesn't employ `var` or `let`

[3]

### 2.2.3 PhoneGap

PhoneGap is a mobile development framework produced by Nitobi, but Adobe Systems purchased it in 2011.

Phonegap allows programmers to develop mobile applications using JavaScript, HTML5 and CSS3. It extends the features of HTML and JavaScript to work with the device, and the resulting applications are hybrid.

An hybrid application is a mix between a native application and a web-based app. The layout rendering is done via web views instead of the platform's native UI framework, but it's packaged as app for distribution and it has access to native device APIs.

A native application is a mobile application developed exclusively for one operating system, exploiting all the native APIs.

A Web-app is a mobile application that can be visited by browser, but it's not packaged and it cannot be downloaded from the online stores.

With PhoneGap has been possible to mixi native and hybrid code snippets has since version 1.9.[8]

The software underlying PhoneGap is Apache Cordova.

### History

First developed at an iPhoneDevCamp event in San Francisco, PhoneGap went on to win the People's Choice Award at O'Reilly Media's 2009 Web 2.0 Conference and had the approval of Apple Inc.

Adobe purchased PhoneGap from Nitobi on October 4th, 2011 and, at the same time, the code was distributed to the Apache Software Foundation for a project called Apache Cordova.

After September 2012, Adobe's PhoneGap Build service allowes developers to compile the code on the cloud, but before that, PhoneGap required a person for each operating system.

### Design and Rationale

The core of PhoneGap applications use HTML5 and CSS3 for their rendering, and JavaScript for their logic.

Although HTML5 now provides access to underlying hardware such as the accelerometer, camera and GPS, browser support for HTML5-based device access is not consistent across mobile browsers, particularly older versions of Android. To overcome these limitations, the PhoneGap framework embeds HTML5 code inside a native WebView on the device, using a foreign function interface to access the native resources of the device.

PhoneGap is also able to be extended with native plug-ins that allow for developers to add functionality that can be called from JavaScript, allowing for direct communication between the native layer and the HTML5 page. PhoneGap includes basic plugins that allow access to the device's accelerometer, camera, microphone, compass, file system, and more.

However, the use of web-based technologies leads many PhoneGap applications to run slower than native applications with similar functionality. Some applications developed by PhoneGap have been rejected by Apple for being too slow or not feeling "native".

### Supported Platforms

PhoneGap currently supports development for the operating systems Apple iOS, BlackBerry, Google Android, LG webOS, Microsoft Windows Phone (7 and 8), Nokia Symbian OS, Tizen (SDK 2.x), Bada, Firefox OS, and Ubuntu Touch. The table below is a list of supported features for each operating system.

# Chapter 3

# Comparison between native and hybrid development

In this chapter we present the comparison between the two main approaches in app developing: the *native approach* and the *hybrid approach*. We show the general characteristics as well as advantages and disadvantages for both the methods.

We also give a brief walk-through about the other way of accessing internet data using a mobile device: the web app.

## 3.1 Web apps

A web app is pure HTML and CSS code fitted for several devices. It's actually a website with limited content and functionality for helping the user to easily navigate within the touch screen and not with peripherals (such as mouse and keyboard), and without downloading a big amount of data (such as great images and other large graphical contents).

### 3.1.1 Advantages

We list all the advantages of building a web app.

- **Cost**

  The costs are limited. Most of the time, if there is a web app, there is also a website. This means that the code already exists and it should only be adapted for the devices. The developers who managed the website can easily manage the web app, because they already know the programming language and the structure of that specific website. In this case, there are some missing phases of the software development process: no feasibility study and limited software design and implementation make the costs very low.

- **Time**

  The time is strictly related to the cost, and vice versa. Since the code already exists, and developers already managed it, it's easy for them to work on it. They don't have to write it from the beginning most of the times, and due to this fact, the time is exponentially reduced.

- **Cross-platform**

  Certainly, being a web app the device version of a web site, it is also exploitable on any kind of device and operating system.

### 3.1.2 Disadvantages

We list all the disadvantages of building a web app.

- **App stores deployment**

  The greatest lack of a web app is the impossibility to deploy the app on any app store. The app is not packaged and it works only with a browser. Some operating systems allow to create a permanent "app-shaped" link to easily access to the contents.

- **Device features**

  There are no native device APIs that can be used with a web app. In particular, there is no possibility to use camera, geolocalization, storage, media, accelerometer and many other device features that are almost essential in the modern apps.

- **User Interface**

  The user interface of a web app is, of course, not native. This implies that the layout is not consistent with the device's one, but more important, the controls are not responsive as a native or a hybrid app, because everything must be used online.

- **No off-line mode**

  All the contents available in the web app come from the internet. This doesn't allow the user to use the app when there is no wifi or network connection.

## 3.2 Hybrid Approach

The hybrid approach is the newest approach in terms of time. It has been invented due to the large variety of mobile devices and operating systems. Developers were looking for a way to build multi-device applications with the same code, that could be used offline and that could exploit the native APIs of the device. The hybrid approach is essentially HTML, CSS and JavaScript code packaged as a native app with a native shell.

### 3.2.1 Gartner Hype Cycle

The Gartner Hype Cycle (Figure 3.1) helps to understand the tendency of the expectations for any new technology. It represents the time on the x-coordinate, and the expectations on the y-coordinate. It's not a qualitative graphic, but quantitative, so it's only a tool for understanding better the phenomenon. Since it can be used for any kind of technology, it can also be used for explaining the trend of expectations of the hybrid app development:

- *Technology Trigger:* it's the phase in which the new technology is introduced. There is excitement and the expectations start to grow. In the case of the hybrid apps, the rise of several mobile operating systems, brought the developers to think about a method to build applications by using some technology that they already knew. At this point, new cross-platform tools (as PhoneGap) were born.

- *Peak of Inflated Expectations:* the peak is reached when there are too many expectations, but probably the technology is not ready to manage them. Speaking about hybrid apps, the peak is reached when Adobe started believe in multi-platform development, and in 2011 it acquired PhoneGap, the most used framework.

- *Trough of Disillusionment:* this phase happens happens when the technology can't face the expectations because of some lack or bug, and everybody starts to lose hope. In our particular case, the lack of some important aspects brings Facebook and, later, LinkedIn to abandon the hybrid approach. These aspects are:

  - Performance not even close to the native apps
  - Lack of a remote debugger
  - Lack of a cloud-based builder
  - Impossibility to have a native aspect of the user interface

- *Slope of Enlightenment:* during this period, the lacks and bugs that brought the expectations until the bottom, are in some way fixed and/or improved. The two most spread operating system (iOS and Android) gave the chance to debug the applications by using their official browsers (respectevely, Safari and Chrome). PhoneGap and other frameworks allowed the developer to build the apps on the cloud, and, furthermore, some frameworks gave the possibility to give a native aspect to the apps. Also the performance are improved.

- *Plateau of Productivity:* we find this phase when there is a stabilization of the expectations. The biggest problems are solved, and the technology is set.

### 3.2.2 Advantages

We list all the advantages of building a hybrid app.

- **App stores deploy**

  Compared to a web app, this is the most important feature of an hybrid app. The fact that can be deployed on a store implies that:

  - The app can be sold
  - Advertisement can be added
  - In-app purchase items can be inserted

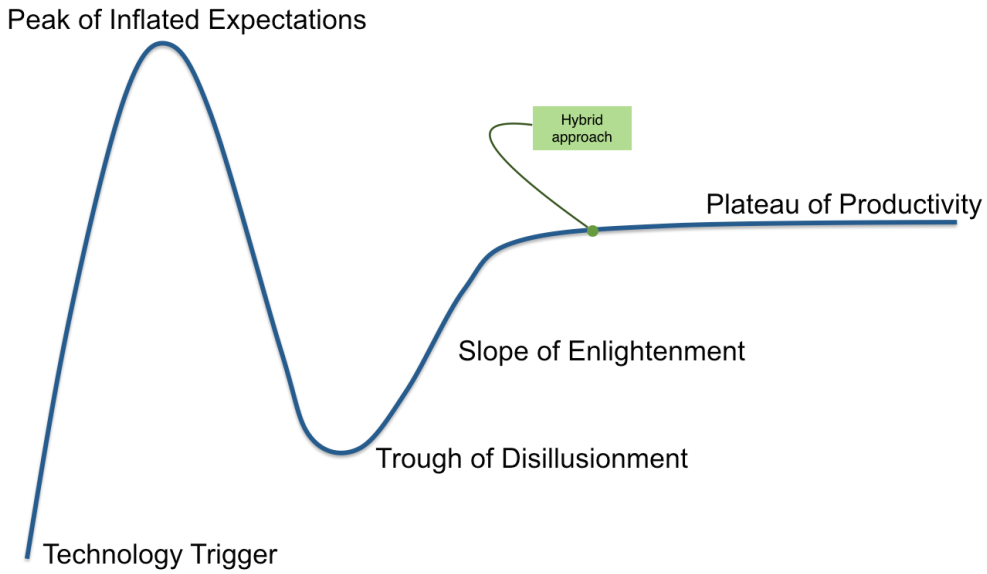


**Figure 3.1.** Gartner Hype Cycle

- **Profit**

  The direct implication of the previous point is about profit. If it is true that with a web app there might be indirect profits, with an hybrid app the earnings are direct.

- **Cross-platform**

  As a web app, a hybrid app can be dedicated to multiple platforms. PhoneGap, for example, allows the developer to build apps for iOS, Android and Windows Phone. The code is packaged with some native parts of the user interface for each operating system, so it doesn't change.

- **Device features**

  Most of the frameworks have access to all the native APIs of the device. Developers can build app using camera features, geolocalization, accelerometer, contacts, events, file, InAppBrowser, media, notification, splashscreen, storage. This implies that all the offline features can be used.

- **Offline mode**

  As said in the previous point, since the framework has access to almost all the APIs, the online mode is not essential anymore. Even though the app is developed using HTML, CSS and JavaScript, the several accessible features give the opportunity to build native-like apps.

- **Performance (compared to a web app)**

  If we take a web app as a point of reference, the performance related to a hybrid app is considerably higher. If we speak about apps that need an internet

connection, it is true that part of the content is online, but it's also true that the UI is stored on the device, and the controls are more responsive.

- **Cost and time (compared to a native app)**

  The cost and the time for a hybrid app is on average comparable to the cost and the time for a web app. The way the app is developed is very similar (HTML, CSS and JavaScript). The only difference is when the app is including other features as camera or geolocalization APIs. In this case, the time and cost increase, but also the quality and complexity of the app increase.

### 3.2.3 Disadvantages

We list all the disadvantages of building a hybrid app. Some of these characteristics are present also in the "advantages" list, but here they are compared all with a native app.

- **Device Features**

  The percentage of the compatible APIs depends on the used framework for developing the app, and the related operating system we are developing for. For example, using PhoneGap, there are no compatibility problems with the APIs for the two main operating systems (iOS and Android). Some feature, if we develop for Blackberry or Windows Phone may not be available.

- **Performance**

  Hybrid app performance can be strong, but will sometimes suffer depending on how the tools build code to interface with the native OS. For complex apps, in fact, the abstraction layer may prevent native-like performance.

- **User Interface**

  The user interface of a hybrid app can be very similar to the user interface of a native app with the help of some dedicated framework. But some difference remains. There are three main points:

  – The native user interfaces are constantly changing (for example, from iOS 6 and iOS 7), the framework we are using must be *updated* or *changed*.

  – Some graphical features (like the bounce at the end of a page in iOS) cannot be *reproduced* by using JavaScript.

  – The hybrid UI is *apparent*. The bitmap composition, when a UI web view is used, does not happen in the hardware like in a native app.

## 3.3 Native Approach

A native app is an application developed exclusively for a specific operating system, using the programming language that is originally supported by that operating system.

The main operating systems for mobile devices and their related programming languages are shown in the Figure 3.2.

| | Operating system | Programming languages | Platform | Store |
|---|---|---|---|---|
| **Apple** | **iOS** | Objective-C<br>Swift | Mac OS X | App Store |
| **Google** | **Android** | Java | Windows<br>Linux<br>Mac OS X | Google Play |
| **Microsoft** | **Windows Phone** | C++<br>C# | Windows | Windows Phone Store |

**Figure 3.2.** Main Operating Systems

### 3.3.1 Advantages

We list all the advantages of building a native app.

- **App store deploy**

  Clearly, as for the hybrid apps, there is the possibility to deploy a native app on the relative store. There are two aspects to take into account, one positive and one negative. The positive one is that the process for validating an app, if it's written with native language and not with some framework, is easier. The negative one is that, obviously, once the app is complete, it can be deployed only for one store.

- **Profit**

  The monetization is an important aspect if we want to develop a native app. The characteristics are the same as the hybrid app, with two difference, one positive and one negative. The positive one is that, most of the time, a native app is more performing and good looking than a hybrid app, and this gives to the app more chances of being a successful app. The negative one is that, developing for only one operating system, the number of downloads will be probably less.

- **Device features**

  A native app has access to all the native APIs of the device. From this point of view, it's even better than a hybrid app.

- **Offline mode**

  A native app using native APIs can be developed. Internet is not needed, and a local storage can be used to store temporary and permanent data.

- **Performance**

  Performance are much better than the performance of an hybrid app. Since the access to the APIs is direct, and the UI is native, the responsiveness of a native app, and its access to the data is very fast. Another aspect to take

into consideration is about developing mobile games. They need a very quick response from the device, and only a native app can manage some graphic animations having direct access to the GPU properties.

- **User Interface**

  The user interface is native and it's not a representation of it. This means higher performance, because most features happen in the hardware, and better looking apps, because the developer doesn't have to manage the design or layout of an app. This is already set.

### 3.3.2 Disadvantages

We list all the disadvantages of building a native app.

- **Cost and time**

  The cost and time, as for the hybrid apps, are strictly related. There are some aspects to take into consideration. To develop a native app, a developer who knows a specific programming language is needed. There are two possibilities:

  – Exploit some developers for each programming language.
  – Train some developers in order for them to learn every programming language.

  In both cases, a big amount of resources, in terms of time and money, is needed. Plus, it must be considered that programming languages as Objective-C and Swift are more "rare" and less known by the developers, on average. This might mean that the cost can increase even more.

- **Single platform**

  Clearly, the main disadvantage of a native app is related to the fact that once the app is built and ready, it can work only with an operating system, and the code cannot be reused for the others.

## 3.4 Conclusions

In this section we write a brief summary about the different approaches and we explains the choices we made.

### 3.4.1 Summary

The Figure 3.3 represents a visual summary of what has been explained in the previous sections. It is visible how the more we move from the right to the left of the diagram, the more we have a cheaper with low performances product. The more we move from the right to the left, the more our product would need more resources but with better quality.

The greatest lack of the native approach is, obviously, the single-platform support, while the greatest advantages are compatibilities and performance.

The greatest lacks of the hybrid approach are compatibilities and performance, while the big advantage is the cross-platform support.

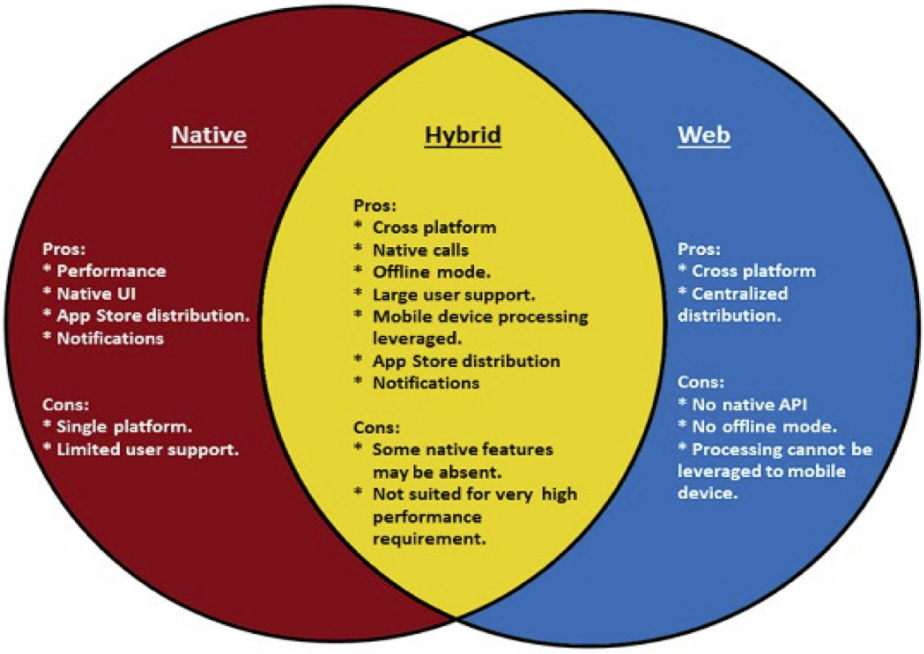


**Figure 3.3.** Pros and cons. Picture taken from [1].

### 3.4.2 Choices we made

Our initial idea was to try to develop an app using an hybrid approach, since the language is less technical and more known. The APIs we needed are supported and we didn't need very high performance. The possibility to test the app on more operating systems was a plus.

# Chapter 4

# Case study specification

In this chapter we describe the problem that we managed. We show our solution and we explain how it is implemented, with the help of an entity-relationship model and a use case diagram.

## 4.1 Description of the problem

In this section we show the overview of the problem we tried to solve. The scenario implies two main factors: a *group of people* and *money*.

In everyday life is usual to perform an activity with friends or other people, in which money is implied, such as the organization of a travel, of a home-made dinner or of an event. We show in details the "travel" case, for understanding better the problem.

### 4.1.1 The "travel" case

The classic scenario of a travel is divided in two periods. The first period is before the travel, and it's the *organizational* part. The second period is the *travel* itself. Both of these periods face some financial issues.
The *organizational* part faces these financial issues:

- Research and booking of the flight (or a different means of transport);
- Research and booking of the hotel (or a different accommodation);
- Research and rental of a car on site;

The *travel* faces these financial issues, due to everyday common expenses:

- Grocery shoppings
- Rentals
- Restaurants
- Common activities

All these activities have to be managed by someone who is calculating all the money spent by each person in the group, and then dividing and redistributing all the money. This problem can be solved in an easier and faster way.

## 4.2 Proposed solution

In this section we explain our idea for solving the problem explained in the previous section.

### 4.2.1 Main idea

The main idea for solving the problem is the development of an app that is calculating all the movement step by step. The users can create some group of work and add some friend. Each friend, then, can add some expense. The tool, for each movement added, is calculating if a person must receive or give money.

#### The "travel" case example

In order to better understand the problem and the solution, we take as an example the "travel" case explained in the section 4.1.1.

Mark, John and Spencer are organizing a travel. Mark is paying for the flight tickets, John is reserving the hotel and Spencer is renting a car. The following table is explaining the problem:

| | | |
|---|---|---|
| Mark | 400€ | Flight ticket |
| John | 300€ | Hotel reservation |
| Spencer | 80€ | Car rental |
| Total | 780€ | |

**Table 4.1.** The "travel" case.

In this case, the total cost of the trip is:

$$400 + 300 + 80 = 780$$

The total cost must be divided fot the number of partecipants (in this case: 3), and we have:

$$780/3 = 260$$

260€is the total cost for each participant, but some of them payed more and some of them payed less. In the following table we show, for each participant, if he must give or receive money:

According to this table and our calculation, Mark must receive 140€, John must receive 40€and Spencer must give 160€.

### 4.2.2 Main features

The main features of our app are:

- Possibility to add new groups of people

| | | | |
|---|---|---|---|
| Mark | 400€ | 260€ | -140€ |
| John | 300€ | 260€ | -40€ |
| Spencer | 80€ | 260€ | +160€ |
| Total | 780€ | | |

**Table 4.2.** The "travel" case solutions.

- Possibility to add new people to the groups, by selecting them among Facebook friends
- Possibility to add new financial movements for each person present in the group
- Possibility to exploit the native APIs of the device camera to add a picture for each movement
- Possibility to check-in for each movement
- Possibility to calculate if a person should give or receive money from the other people of the group
- Record of every movement

## 4.3 Entity-Relationship model

In this section we present the entity-relationship model of the data of our app. The Figure 4.1 shows the E-R model with all the three entities, their respective attributes and the cardinalities.

### 4.3.1 Group

The *Group* entity has these attributes:

- *idgroup*: it's a unique attribute that autogenerates all the times a group is created and it's necessary to distinguish a group from another. Type: integer.
- *amount*: it's an attributes that indicates the total money that have been spent by each person in the group. Type: smallmoney.
- *groupname*: it's an attribute that indicates the name of the group. It's also the name that is displayed in any page of the app. Type: character.
- *date*: it's an attribute that indicates when the group has been created. Type: date.
- *description*: it's an attribute that allows the user to describe the characteristics of the group. Type: character.

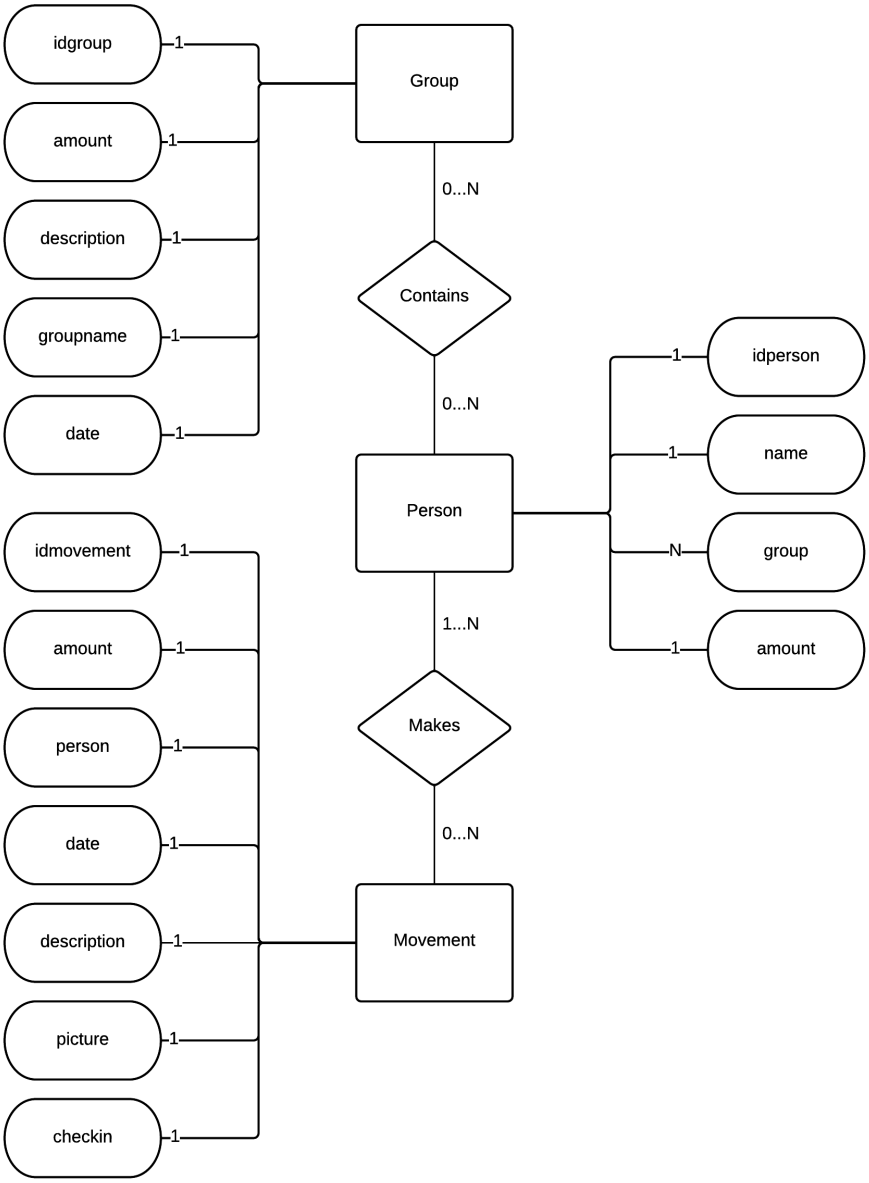


**Figure 4.1.** E-R model

### 4.3.2 People

The *People* entity has these attributes:

- *idperson*: it's a unique attribute that autogenerates all the times a friend is added and it's necessary to distinguish a person from another. Type: integer.
- *amount*: it's an attributes that indicates the total money that have been spent by a person. Type: smallmoney.
- *name*: it's an attribute that indicates the name of the person (friend). It's also the name that is displayed in any page of the app. Type: character.
- *group*: it's an attribute that links a person to a certain group. It is equal to the id of the group from which the person has been added. Type: integer.

### 4.3.3 Movements

The *Group* entity has these attributes:

- *idmovement*: it's a unique attribute that autogenerates all the times a movement is created and it's necessary to distinguish a movement from another. Type: integer.
- *amount*: it's an attributes that indicates the money that have been spent in that movement.
- *person*: it's an attribute that links a movement to a certain person. It is equal to the id of the person from which the movement has been created. Type: integer.
- *date*: it's an attribute that indicates when the movement has been created. Type: date.
- *description*: it's an attribute that allows the user to describe the characteristics of the movement. Type: character.
- *picture*: it's an attribute that contains the image taken by the user for a certain movement.
- *checkIn*: it's an attribute that contains the check in done by the user for a certain movement.

## 4.4 Use case diagram

The Figure 4.2 shows the *use case diagram*. The tables describe every use case present in the diagram:

- *Purpose* describes what the use case allows the user to do.
- *Role(s)* indicates who is using a certain use case.

- *Pre-condition* lists all the previous use cases completed before the current use case.
- *Post-condition* lists all the enabled use cases after completing the current use case.
- *Workflow* explains all the steps that the user must cover to complete the current use case.
- *Accesses data in each view* indicates which data are available to the user in the current use case. *RM* and *WR* indicates, respectevely, "read mode" and "write mode".

### 4.4.1 Sign up with Facebook

| | |
|---|---|
| Purpose | It allows the user to register to the app with Facebook credentials |
| Role(s) | User, friends |
| Pre-condition | None |
| Post-condition | It's now possible to log in with facebook credentials and to start using the features of the app:<br>1. Log in with Facebook<br>2. Add group |
| Workflow | 1. The user accesses the app<br>2. The user signs up with Facebook credentials |
| Accesses data in each view | WM: Facebook API |

**Table 4.3.** Sign up with Facebook

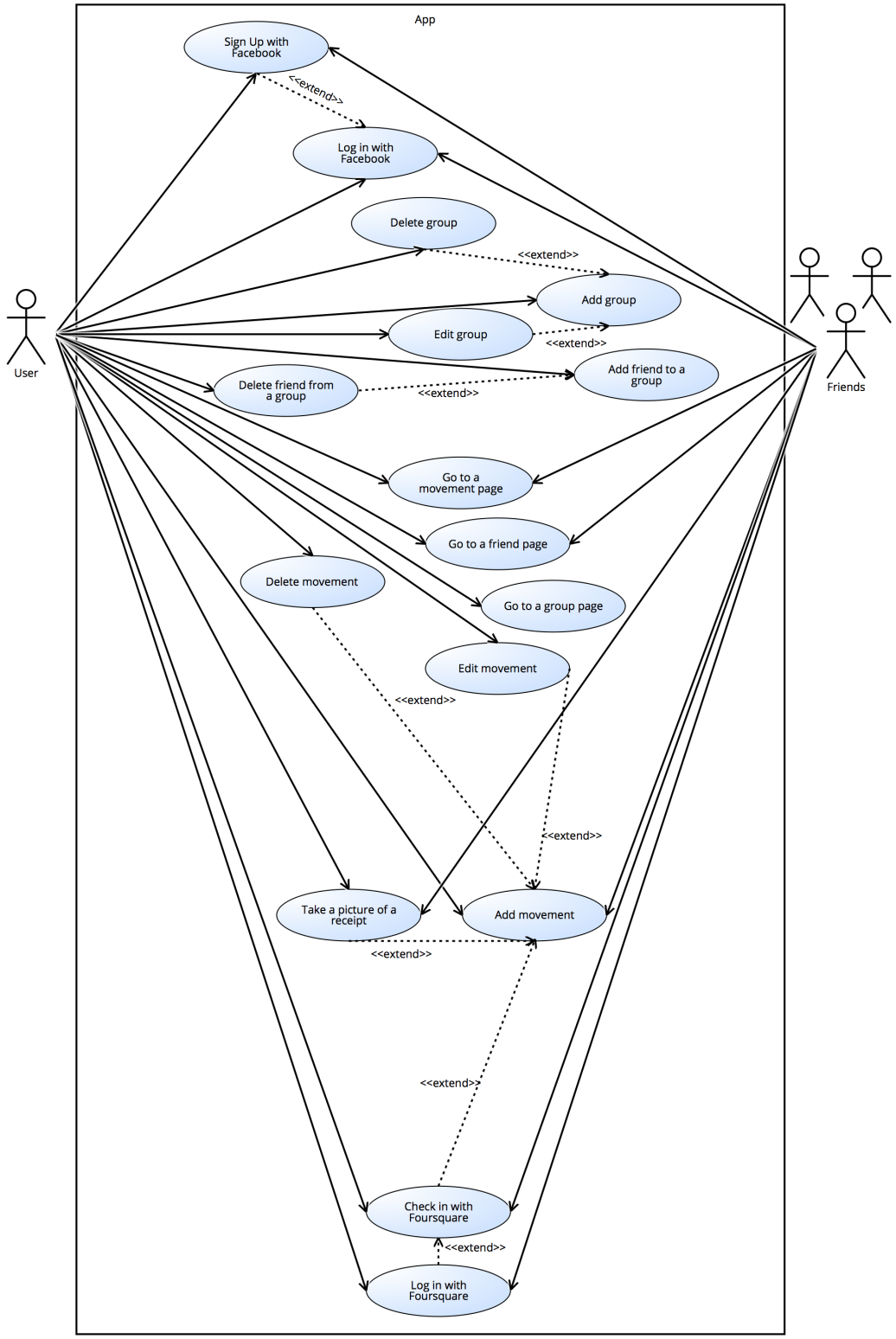


**Figure 4.2.** Use case diagram

### 4.4.2 Log in with Facebook

| | |
|---|---|
| Purpose | It allows the user to access the app with Facebook credentials |
| Role(s) | User, friends |
| Pre-condition | The user must be registered. Use case completed:<br>1. Sign up with Facebook |
| Post-condition | It's now possible to use the features of the app. Use case enabled:<br>1. Add group |
| Workflow | There are two ways. The first one is when the user access the app for the first time:<br>1. The user accesses the app<br>2. The user signs up with Facebook by tapping on the button<br>3. The user is automatically logged in<br>The second way is if the user already signed up:<br>1. The user accesses the app<br>2. The user is automatically logged in |
| Accesses data in each view | RM: Facebook API |

### 4.4.3 Add group

| | |
|---|---|
| Purpose | It allows the user to create a new group |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook |
| Post-condition | It's now possible to go to a group page and add people to the created group or delete or modify the group. Use case enabled:<br>1. Go to group page<br>2. Edit group<br>3. Delete group<br>4. Add friend to a group |
| Workflow | 1. The user accesses the app<br>2. The user taps on the "+" button for adding a new group<br>3. The user writes the name of the group in the text field<br>4. The user tap on "ok" to confirm |
| Accesses data in each view | WM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date |

### 4.4.4 Edit group

| | |
|---|---|
| Purpose | It allows the user to modify the name of the group |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in, and he must have created a group. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group |
| Post-condition | It's now possible to go to the group page, add people to the edited group, or to delete the group. Use case enabled:<br>1. Go to group page<br>2. Delete group<br>3. Add friend to a group |
| Workflow | 1. The user accesses the app<br>2. The user taps on the "edit" button<br>3. The user taps on the group he wants to edit<br>4. The user changes the name of the group and taps "ok" |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.date<br>WM: Group.groupname, Group.description |

### 4.4.5 Delete group

| | |
|---|---|
| Purpose | It allows the user to delete a group |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in, and he must have created a group. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group |
| Post-condition | It's now possible to create a new group or to go to an existing group page. Use case enabled:<br>1. Add group<br>2. Go to group page |
| Workflow | 1. The user accesses the app<br>2. The user taps on the "edit" button<br>3. The user taps on the group he wants to delete<br>4. The user taps on the "delete this group" button |
| Accesses data in each view | WM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date |

### 4.4.6 Go to group page

| | |
|---|---|
| Purpose | It allows the user to navigate from the home page to a group page. |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. At least one group must be already created. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group |
| Post-condition | It's now possible to add friends to the group, or delete the existing people. Use case enabled:<br>1. Delete friend from a group<br>2. Add person |
| Workflow | 1. The user accesses the app<br>2. The user taps on a element in the list of groups |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name |

### 4.4.7 Add friend to a group

| | |
|---|---|
| Purpose | It allows the user to add a person to the group |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. At least one group must be already created. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page |
| Post-condition | It's now possible to add new movements to the added people, to delete the added people, or to go to a friend page. Use case enabled:<br>1. Delete friend from a group<br>2. Add movement<br>3. Go to a friend page |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user taps on the "+" button for adding a new friend<br>4. The user searches the friend in his friend list and tap on it for adding it |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date<br>WM: Person.idperson, Person.name, Person.group, Person.amount |

### 4.4.8 Delete friend from a group

| | |
|---|---|
| Purpose | It allows the user to delete a friend from a group |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in, he must have created a group and added at least one person. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Add friend to a group<br>5. Go to group page |
| Post-condition | It's now possible to add new people to the group or to go to an other friend page. Use case enabled:<br>1. Add friend to a group<br>2. Go to a friend page |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user taps on the "edit" button<br>4. The user taps on the person he wants to delete<br>5. The user taps on the "delete this person from group" button |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date<br>WM: Person.idperson, Person.name, Person.group, Person.amount |

### 4.4.9 Go to a friend page

| | |
|---|---|
| Purpose | It allows the user to navigate from the group page to a friend page. |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. At least one group must be already created and one person must be added. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Add friend to a group |
| Post-condition | It's now possible to add, edit or delete a movement. Use case enabled:<br>1. Add movement<br>2. Edit movement<br>3. Delete movement |
| Workflow | 1. The user accesses the app<br>2. The user taps on an element in the list of groups<br>3. The user taps on an element in the list of added friends |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount, Movement.idmovement, Movement.amount, Movement.date |

### 4.4.10 Add movement

| | |
|---|---|
| Purpose | It allows the user to add a movement of a friend |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. At least one group must be already created and one person must be added. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page |
| Post-condition | It's now possible to go to a movement page, edit or delete a movement, to take a picture of the receipt of that movement or to check in with Foursquare. Use case enabled:<br>1. Go to a movement page<br>2. Edit movement<br>3. Delete movement<br>4. Take a picture of a receipt<br>5. Log in with Foursquare<br>6. Check in with Foursquare |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "+" button for adding a new movement<br>5. The user fill the fields of the movement (amount, date and description) and taps on the "ok" button |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount<br>WM: Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in |

### 4.4.11 Edit movement

| | |
|---|---|
| Purpose | It allows the user to modify the fields of a movement |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, one person and one movement must be added. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement |
| Post-condition | It's now possible to add new movements. Use case enabled:<br>1. Add movement |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "edit" button<br>5. The user taps on the movement he wants to edit<br>6. The user changes the fields of the group he wants to change and taps "ok" |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount, Movement.idmovement, Movement.person<br>WM: Movement.amount, Movement.date, Movement.description, Movement.picture, Movement.check-in |

### 4.4.12 Delete movement

| | |
|---|---|
| Purpose | It allows the user to delete a movement |
| Role(s) | User |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, one person and one movement must be added. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement |
| Post-condition | It's now possible to add new movements. Use case enabled:<br>1. Add movement |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "edit" button<br>5. The user taps on the movement he wants to edit<br>6. The user taps on the "delete this movement" button |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount<br>WM: Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in |

### 4.4.13 Go to a movement page

| | |
|---|---|
| Purpose | It allows the user to navigate from the person page to a movement page. |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, one person must be added and one movement must be created. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement |
| Post-condition | It's now possible to edit or delete a movement, to take a picture or check in with Foursquare. Use case enabled:<br>1. Add movement<br>2. Edit movement<br>3. Delete movement<br>4. Take a picture<br>5. Check in with Foursquare |
| Workflow | 1. The user accesses the app<br>2. The user taps on an element in the list of groups<br>3. The user taps on an element in the list of added friends<br>4. The user taps on an element in the list of a created movement |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount, Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in |

### 4.4.14 Take a picture of a receipt

| | |
|---|---|
| Purpose | It allows the user to add a picture to a movement |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, and one person must be added. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement |
| Post-condition | It's now possible to add, edit or delete a movement. Use case enabled:<br>1. Add movement<br>2. Edit movement<br>3. Delete movement<br>4. Log in with Foursquare<br>5. Check in with Foursquare |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "+" button for adding a new movement<br>5. The user fill the fields of the movement (amount, date and description)<br>6. The user taps on the "camera" button and add a picture (new picture or from library)<br>7. The user taps on the "ok" button |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount, Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in<br>WM:Movement.CameraAPI |

### 4.4.15 Log in with Foursquare (1)

| | |
|---|---|
| Purpose | It allows the user to log in with a Foursquare account |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, and one person must be added. The user doesn't have to be already logged in. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement |
| Post-condition | It's now possible to check in with Foursquare, or add, edit or delete a movement. Use case enabled:<br>1. Check in with Foursquare<br>2. Add movement<br>3. Edit movement<br>4. Delete movement<br>5. Take a picture of a receipt |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "+" button for adding a new movement<br>5. The user fill the fields of the movement (amount, date and description)<br>6. The user taps on the "Foursquare" logo and logs in with his credentials |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount<br>WM: Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in, Movement.FoursquareAPI |

### 4.4.16 Log in with Foursquare (2)

| | |
|---|---|
| Purpose | It allows the user to log in with a Foursquare account |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook |
| Post-condition | It's now possible to add a group or to go to a group page. Use case enabled:<br>1. Add group<br>2. Go to group page |
| Workflow | 1. The user accesses the app<br>2. The user taps on the "Foursquare" logo and logs in with his credentials |
| Accesses data in each view | RM: Group.idgroup, Group.groupname<br>WM: FoursquareAPI |

### 4.4.17 Check in with Foursquare

| | |
|---|---|
| Purpose | It allows the user to check with Foursquare |
| Role(s) | User, friends |
| Pre-condition | The user must be registered and logged in. At least one group must be already created, and one person must be added. The user must be already logged in with Foursquare. Use case completed:<br>1. Sign up with Facebook<br>2. Log in with Facebook<br>3. Add group<br>4. Go to group page<br>5. Add friend to a group<br>6. Go to a friend page<br>7. Add movement<br>8. Log in with Foursquare |
| Post-condition | It's now possible to add, edit or delete a movement. Use case enabled:<br>1. Add movement<br>2. Edit movement<br>3. Delete movement<br>4. Take a picture of a receipt |
| Workflow | 1. The user accesses the app<br>2. The user accesses to a created group<br>3. The user accesses to an added friend<br>4. The user taps on the "+" button for adding a new movement<br>5. The user fill the fields of the movement (amount, date and description)<br>6. The user taps on the "Foursquare" logo and checks in |
| Accesses data in each view | RM: Group.idgroup, Group.amount, Group.description, Group.groupname, Group.date, Person.idperson, Person.name, Person.group, Person.amount<br>WM: Movement.idmovement, Movement.amount, Movement.person, Movement.date, Movement.description, Movement.picture, Movement.check-in, Movement.FoursquareAPI |

# Chapter 5

# High level design

In this chapter we describe the app from an high level using some UML diagrams. Specifically we use:

- a *collaboration diagram* to understand the relations between the pages of the app;
- an *activity diagrams* to understand the path of the actions that a user can do;

## 5.1 Collaboration diagram

The Figure 5.1 shows the *class diagram* of our app. Each class represents the pages of the app and how they're related.

The diagram shows also some simple units that represents the APIs used by the device. These classes are *Facebook*, *Foursquare* and *Camera*.

### 5.1.1 Home

From the home page it is possible to:

- See all the created groups
- Edit all the created groups
- Delete the existing groups
- Access to a group page

### 5.1.2 Groups

From any group page it is possible to:

- See all the added friends
- Delete the added friends
- Access to a person (friend) page

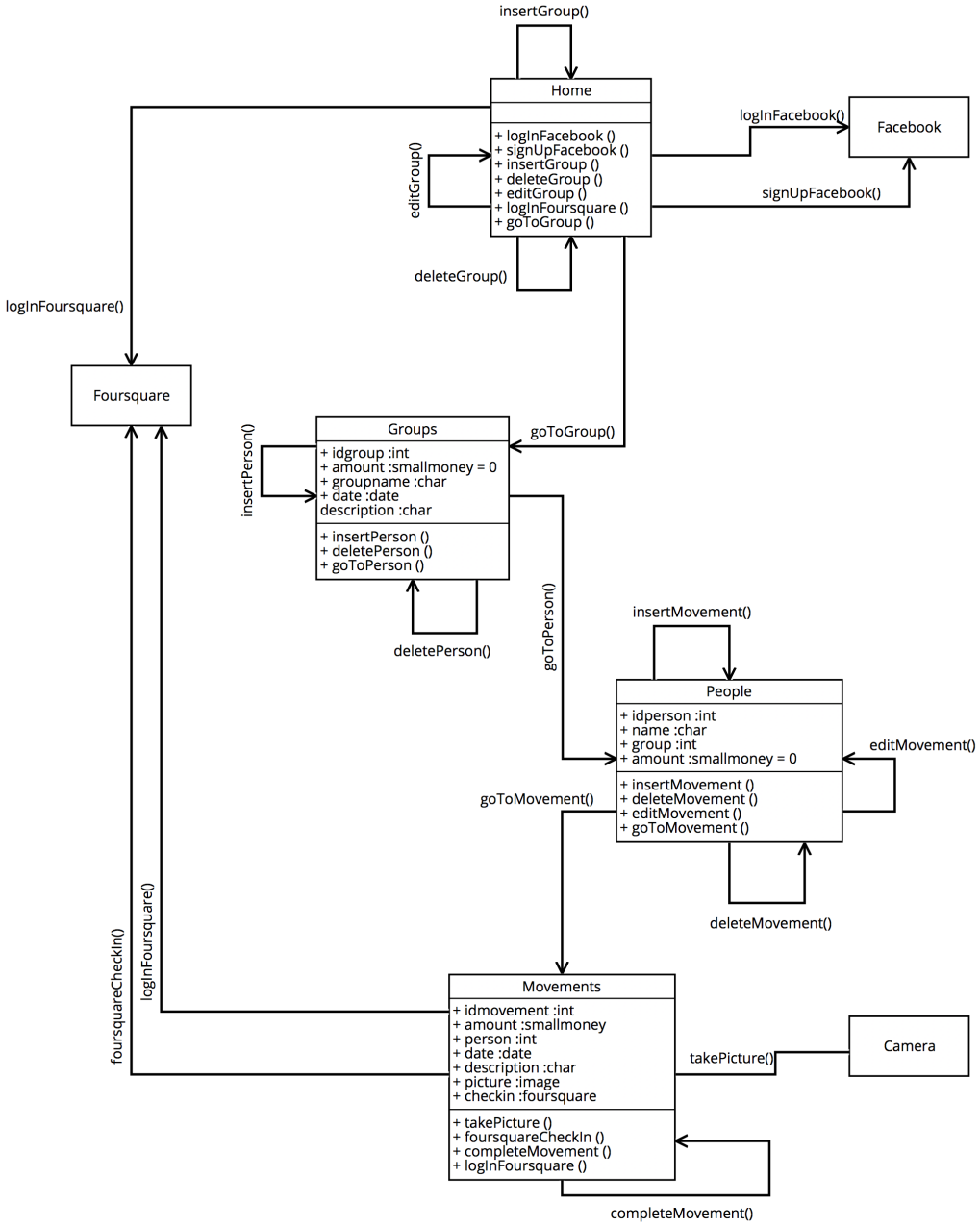


**Figure 5.1.** Collaboration diagram

### 5.1.3 People

From any person page it is possible to:

- See all the added movements
- Edit the added movements
- Delete the added movemements
- Access to a movement page

### 5.1.4 Movements

From any group page it is possible to:

- See all the added friends
- Delete the added friends
- Access to a person (friend) page

## 5.2 Activity diagrams

In this section we present an activity diagrams. It explains how the user can create a movement of a person.

### 5.2.1 Add a movement

The Figure 5.2 is the activity diagram that shows the steps needed to add a movement. This activity diagram also includes other use cases, and these are:

- Add group
- Go to a group page
- Add friend to a group
- Go to a friend page

In this activity diagram is shown the activities of the main actor (the user) and those of the system.

#### Actor

1. *Tap on "+" to add a group*: the user taps on the "+" button on the right part of the header to open a pop-up with a text field and the "Ok" button.
2. *Write the name of the group and tap "Ok"*: the user writes the name of the group in the text field and taps on the "Ok" button in order to save the group.
3. *Add another group?*: the user can now choose to add another group or to enter in the group page that he has just created.

4. *Tap on a created group*: the user enters in the group page that he created.
5. *Tap on "+" to add a friend*: the user taps on the "+" button on the right part of the header to open the Facebook's friend list.
6. *Tap on the selected friend*: the user taps on the name of the friend in order to add it to the group.
7. *Add another friend?*: the user can now choose to add another friend or to enter in the friend page that he has just added.
8. *Tap on "+" to add a movement*: the user taps on the "+" button on the right part of the header to open a pop-up with two text fields and the "Ok" button.
9. *Write amount and description of the movement and tap "Ok"*: the user writes the amount of the movement and its description in the text fields and taps on the "Ok" button in order to save the movement.
10. *Add another movement?*: the user can now choose to add another movement or to enter in the movement page that he has just added.

**System**

- *Check among Facebook friends*: the system, with the Facebook APIs, list all the user's friends.
- *Send notification to the added friend*: the system sends the notification to the added friend. He has the option to accept or decline the invitation.
- *Send notification to the people in the people in the group*: anytime someone is adding a movement, the system sends the notification to all the members of the group.
- *Make calculations*: anytime someone is adding a movement, the system makes the calculations and displays the results.

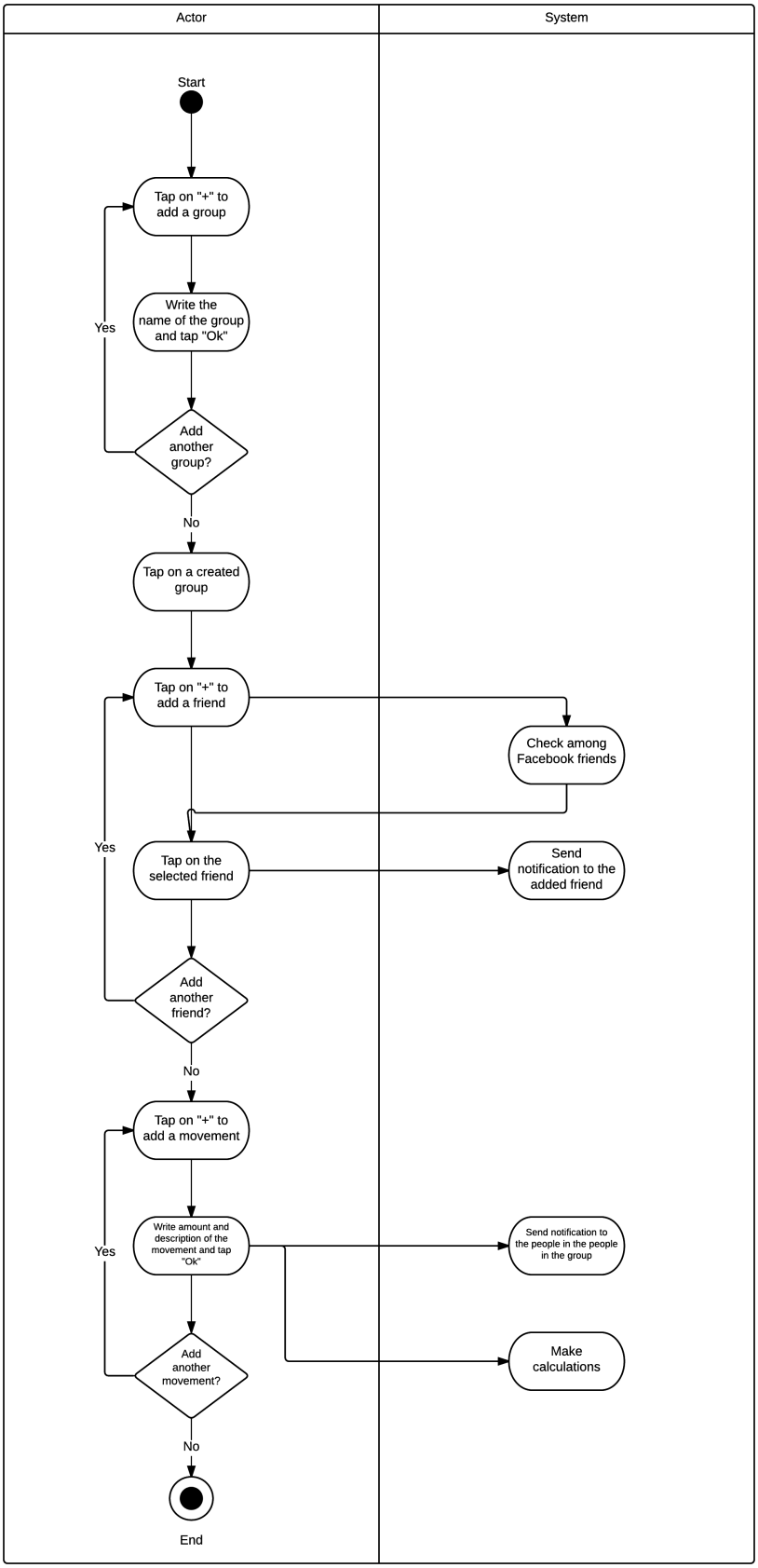


**Figure 5.2.** Activity diagram for adding a movement

# Chapter 6

# Partial implementation experience with the two technologies

TODO Per il 6 devi descrivere la soluzione implementata. Descrivi le tecnologie usate e spiegare la struttura dell'applicazione. Spiega quali scenari hai coperto e se possibile le criticità rispetto al livello modeling.

Then we show how the app is developed, the scenarios and some results and criticality.

## 6.1 Technologies we used and structure of the app

In this section we present the technologies we used to develop our app. For each technology we show the structure of our app.

### 6.1.1 Structure with PhoneGap

The first version of the app is hybrid, and developed using *PhoneGap*. PhoneGap is exploiting the potential of HTML, CSS and JavaScript. This means that it's creating web pages that are packaged and used as a native app.

Once PhoneGap is installed and a new project is started, we can work on our app using *Xcode*. The advantage of Xcode is the possibility to emulate the device and test our app. The emulator is very fast and responsive. We can choose any kind of device among Apple's devices. In our case, we chose the iPhone 4.
PhoneGap gives us 3 main files to work on:

- `index.css`: a CSS file used to describe the look of our application. In our case we didn't work on it. The reason is that we wanted our app to look as a native app. We used the framework *Ratchet* that provides a native app look alike file CSS.
- `index.html`: a HTML file. This is the most important file for two reasons. The first reason is that this file is where all the pages of our app are. Each page is exactly like a web page. It's divided in:

  - *Header*: the upper part of the page, usually used for the title of the app, the name of the page or some useful button (like "back" or "add" buttons).
  - *Body*: the central part of the page, where all the data and contents are. In our case, the bodies of our pages always contain a list ("groups", "people" and "movements") or some text fields with button for adding a new element to our lists. For easier manage these lists, we used the framework *Handlebars*.
  - *Footer*: the lower part of the page, usually used for some permanent button. In our case, the "home" button is always present.

  The second reason is that this file groups all the other files. All the scripts and links are imported to this file, `index.css` and `index.js` included.

- `index.js`: a JavaScript file that contains all the functionalities of our app. If the HTML file is used to describe "how" our app looks, the JavaScript file describes "what" our app is doing. The functions that are in our app can be grouped:

  - `OnDeviceReady()`: it's the first function that is called once the app is launched. In our app is calling the function `renderHome()`.
  - `insert()` functions: it's a group of function thanks to which the user is allowed to add an element to "groups", "people" or "movements";
  - `edit()` functions: it's a group of function thanks to which the user is allowed to edit an element in "groups", "people" or "movements";
  - `delete()` functions: it's a group of function thanks to which the user is allowed to delete an element from "groups", "people" or "movements";
  - `list()` functions: it's a group of function thanks to which the system is listing all the elements in "groups", "people" or "movements";
  - `render()` functions: it's a group of function thanks to which the system is rendering all the pages of "groups", "people" or "movements";
  - API calls: these functions are calling some services as camera's API (native) and Facebook and Foursquare APIs (external).

### Handlebars

Handlebars is a framework that is allowing the developers to set some templates. These templates are filled with the data given by the database. In our case it has been very useful. To list all those data wouldn't be easy without a template.

### Ratchet

Ratchet is a framework that is allowing the developers to set a user interface that looks alike a native user interface. The downloadable Ratchet's package gives to the developers a set of CSS and JavaScript files. Importing this file in our project, we are allowed to use some customized commands in our HTML pages.

### 6.1.2 Structure with Swift

The second version of the app is native, and developed using *Swift*, the new Apple's programming language. Once *Xcode* is installed, we are ready to develop a project.
A *Swift* project is divided in this way:

- `Main.storyboard`: this file has a big potential. It gives us a graphical view of the layout of each page. With the help of this file, we are allowed to create layouts by drag-and-dropping elements from the "Object library" of Xcode. We are allowed to directly link different functionalities to each element that we drop in our layout. Furthermore, we can see the path of our app.
- `Proups.swift`, `People.swift`, `Movements.swift` that manage the data of our app.
- `TableViewController.swift` files: in these files are included all the functions and links of each page.

## 6.2 Criticalities

In this section, we explain which problems and criticalities we faced with both technologies.

### 6.2.1 Criticalities with PhoneGap

We list all the issues we encountered by developing the app using PhoneGap.

- *Installation*: for a developer, this is not a hard procedure. But PhoneGap needs to be installed from the terminal using some commands.
- *Utility frameworks*: in order to develop our hybrid app, and make it similar to a native app, we needed some frameworks:
  - *Handlebars*: this framework was necessary. Since our app is a set of nested lists, writing the dependency in HTML wouldn't have been possible.
  - *Ratchet*: this framework was not necessary for the functionalities, but it was necessary for the user interface. We wanted our app to be as similar as possible to the native app. The result is good, even if the two apps are not looking exactly the same. The bounce at the end of page, when a user is scrolling a list, is missing. Some element is a little different. The app is not responsive as the native one.
- *APIs*: we had some trouble with the APIs:
  - *Facebook and Foursquare*: it's not a real issue, but we had to decide if the API we needed were the web version (since our pages are built as web pages) or the iOS version.

  - *Camera*: with these APIs we had some problem we couldn't solve. The camera was not working even if the app was correctly calling the function. We tested this with the help of a debugger, and, with some "alert", we realized that we entered in the function. But nothing happened.

- *Languages*: our knowledge of HTML and JavaScript was a beginner knowledge. Starting from this point, we needed to learn how the markup language and the programming language were working together and by themselves.

### 6.2.2 Criticalities with Swift

We list all the issues we encountered by developing the app using Swift.

- *Language*: our knowledge of Swift was a beginner knowledge. Starting from this point, we needed to learn the logic of the programming language.

This is the only problem we encountered. We didn't have problems with the other features.

## 6.3 Results

The results of our experiments are not qualitative, but quantitative. The only datum we can estimate is the time we dedicated to the development of our app. Here the results: With Swift we didn't encountered any problem with the installation since it

| | |
|---|---|
| PhoneGap | 3-4 weeks |
| Swift | 1 week |

**Table 6.1.** Results in term of time

is already included in Xcode. We didn't need to use any kind of framework that helped us to build our app or that helped us to make our app to look good. We didn't have any problems with the APIs. The external APIs had a clear explanation on Facebook's and Foursquare's website on how they should be used. The native API of the camera are working since they're native and made exactly for Apple's devices and Swift programming language, while with PhoneGap the problem is still unsolved. Furthermore, Swift is very intuitive and easy to learn. It's only one language, while with PhoneGap we needed to learn two of them.

Our opinion is that, it is true that with PhoneGap we can build the app for another operating system with only a click.

# Chapter 7

# Conclusions and Future Works

In this chapter we lead to the conclusions of the work we have done. Then, we explain which are the possible future works.

## 7.1 Conclusions

After the results obtained in the chapter 6, we can list our conclusions:

- **Performance**: it's confirmed that the hybrid approach brings to a less powerful and responsive app. The delay after each tap is significantly longer, the page loading is slower especially during the retrieve of some data.
- **APIs compatibility**: even though there shouldn't be any compatibility problem with the native APIs of the device for the hybrid approach, it's not true for our case. We encountered some issue with the camera API that we couldn't solve. The process is apparently easy, but there are no tools for solving our problem. The emulator and the browser cannot use the camera.
- **User interface**: the user interface is very similar to the original one, but it's slower. In addition, during the development, even adding a button, is not as natural as for the native app. Xcode offers to the native app a tool to build the layout of each page, and the link of a page to another one. For hybrid app, the layout is led only by coding.
- **Time**: even if it's true that, once an hybrid app is developed, it can be build for any operating system, it is not true that it's faster to develop. In fact, there are less tools that are helping PhoneGap during the developing and debugging phase compared to the tools given to Swift. The time wasted to face all the issues during these phases can be used to learn a new programming language (like Java, for Android) and to develop a new native app for another operating system. The precise time is not known, but in the end the result will be a more performing app.
- **Cost**: the costs are strictly related to the time.
- **Cross-platform**: as we said for the time, it is a problem that we are allowed to develop our native app only for one operating system at time. But we

also showed that the time for solving all the PhoneGap issues can be used for developing two times the same app in the native way.

The hybrid approach is only apparently a faster way to develop a mobile app. It's true that for an expert web developer, building an app by using HTML, CSS and JavaScript is easier. But starting from the same level, as in our case, learning a language like Swift, or learning a language like JavaScript led us to face with the same problems. Our conclusion is that, starting from the same knowledge of the programming languages for the hybrid app (as PhoneGap with HTML, CSS and JavaScript) and the native app (Swift for iOS), it is better to start the project by using the native language.

## 7.2 Future works

What we concluded in the section 7.1 is the actual situation of the native and hybrid development. The hybrid approach still presents too many bugs. It's true that the technologies used for developing hybrid apps are improving day by day, but it still lacks of performance and user experience. The native approach is improving, but from a conceptual point of view. Swift is a new programming language, made to improve the old and less intuitive Objective-C. The good aspect of the hybrid approach is that it has substantial room for improvement.

# Bibliography

[1] P. R. M. de Andrade, A. B. Albuquerque, O. F. Frota, R. V Silveira, and F. A. da Silva. Cross platform app: a comparative study. *ArXiv e-prints*, March 2015.

[2] Nicolas Serrano, Josune Hernantes, and Gorka Gallardo. Mobile web apps. *IEEE Softw.*, 30(5):22–27, September 2013.

[3] M. Knott. *Beginning Xcode*. Apress, 2014.

[4] Andre Charland and Brian Leroux. Mobile application development: Web vs. native. *Commun. ACM*, 54(5):49–53, May 2011.

[5] Anthony I. Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 397–400, New York, NY, USA, 2010. ACM.

[6] Wikipedia. Xcode — wikipedia, the free encyclopedia, 2015.

[7] Wikipedia. Swift (programming language) — wikipedia, the free encyclopedia, 2015.

[8] Wikipedia. Phonegap — wikipedia, the free encyclopedia, 2015.