



Universidad  
Carlos III de Madrid

Computer Science Engineering

Bachelor Thesis

# **ShoalUp: Development of a meeting Android application**

**Author:** Álvaro de Saavedra

**Tutor:** David Griol Barres

Leganés, September 2017



# Acknowledgment

I'd like to start by thanking my tutor, David Griol, for his support and motivation since the first moment when I asked him to be part of this project, he encouraged me to do it and helped me in several ways sharing his ideas and experience with me.

Second, I'd like to express my gratitude to my family, without them I wouldn't be here now, they helped me in so many ways, motivating me to continue when I was down, calming me when I was anxious and just being there to support me.

Ultimately, my special mention is for my granddad, who I know right now is watching me from above smiling with pride. He always supported me, motivated me to study and encouraged me to give my best at everything I do, thank you grandpa, without you I wouldn't be the person I am today.

# Abstract

The project aims to develop an Android application for putting together people with similar interests. The users will register and be able to create “meetings” or “events”, informing potential attendants with the description, location and date and time. On the other hand, the users will also be able to discover this “events” and join them and talk with the people that are assisting.

The developed Android application allows people to scan a certain range from the user position discovering potential meetups, filter by kind of activity, i.e. sports, party, trips... and chose to join them or comment in the event.

The app implements a client-server structure, using REST services to communicate, and a NoSQL management database system.

Every decision taken, from the OS to the technology stack selected will be analyzed and justified and put in context, describing the available technologies that could fit in this purpose and reasoning the election.

Also, this being a big personal project, several future features are mentioned with the correspondent description and selection reasons.

**Keywords:** Client-server, Android, NoSQL, SQLite, Java, REST, API, Google Maps, Google Places.

# Introducción

El objetivo del proyecto es desarrollar una aplicación Android para poner en contacto personas con intereses similares. Los usuarios se registrarán y podrán crear “meetings” o “eventos”, informando a los potenciales asistentes con la descripción, localización y fecha y hora. Otros usuarios podrán descubrir estos “eventos” y unirse pudiendo comunicarse con el resto de asistentes.

La aplicación Android desarrollada permite escanear un rango concreto desde la posición del usuario descubriendo posibles “quedadas”, filtrándolas por actividad, i.e. deportes, fiesta, viajes... y elegir unirse o dejar un comentario en el evento.

La aplicación implementa una estructura cliente-servidor, usando servicios REST para la comunicación, y un sistema de gestión de bases de datos NoSQL.

Todas las decisiones tomadas, desde el sistema operativo hasta el stack tecnológico escogido serán analizadas y justificadas y puestas en contexto, describiendo las tecnologías disponibles que podrían encajar para este propósito y razonando la elección.

También, al ser un gran proyecto personal, diferentes características futuras serán mencionadas con su correspondiente descripción y razones de selección.

**Palabras clave:** Cliente-servidor, Android, NoSQL, SQLite, Java, REST, API, Google Maps, Google Places.



# Table of contents

CHAPTER 1: INTRODUCTION .....	1
1.1 Introduction.....	1
1.2 Motivation .....	1
1.3 Objectives .....	2
1.3.1 Thesis objectives .....	2
1.3.2 Application objectives.....	3
1.4. Planning .....	4
1.5 Means employed .....	9
1.6 Memory structure .....	10
CHAPTER 2: STATE OF THE ART .....	11
2.1. Problem analysis .....	11
2.2 Platform and mobile OS evaluation .....	19
2.3 Design and implementation patterns.....	21
2.4 Evaluation of available technologies .....	24
2.4.1 Database evaluation.....	24
2.4.2 Backend technologies evaluation .....	26
CHAPTER 3: ANALYSIS AND DESIGN.....	31
3.1 Analysis.....	31
3.1.1 Requirements.....	31
3.1.1.1 User .....	31
3.1.1.2 Events .....	32
3.1.2 Use cases.....	33
3.1.2.1 Registration .....	34
3.1.2.2 Login .....	35
3.1.2.3 Account recovery.....	36
3.1.2.4 Profile modification .....	37
3.1.2.5 Account deletion .....	38
3.1.2.6 Log out.....	39
3.1.2.7 Create event.....	40

3.1.2.8 Modify an event .....	41
3.1.2.9 Delete event .....	42
3.1.2.10 Show events .....	43
3.1.2.11 Find events .....	44
3.1.2.12 Filter events .....	45
3.1.2.13 Discover events .....	46
3.1.2.14 Join an event.....	47
3.1.2.15 Leave a comment .....	48
3.2 Design .....	49
3.2.1 Application architecture .....	49
3.2.2 Diagram of components.....	50
3.2.3 Activity diagrams .....	50
3.2.3.1 Registration activity .....	50
3.2.3.2 Login activity .....	52
3.2.3.3 Event creation activity.....	53
3.2.3.4 Event retrieval activity .....	54
3.2.4 MongoDB schema .....	55
CHAPTER 4: IMPLEMENTATION.....	57
4.1 Login and registration .....	57
4.2. ShoalUp.....	61
4.3 Evaluation .....	69
CHAPTER 5: SOCIOECONOMIC ENVIRONMENT .....	70
5.1 Socioeconomic environment .....	70
5.2 Budget of the project .....	71
CHAPTER 6: REGULATORY FRAMEWORK.....	74
CHAPTER 7: CONCLUSIONS AND FUTURE WORK.....	75
7.1 Conclusions .....	75
7.2 Future work .....	76
7.2.1 Near future work.....	76
7.2.2 Further future work.....	76
BIBLIOGRAPHY .....	78



# Index of figures

Figure 1.1. Software Development Life Cycle .....	5
Figure 1.2. Project planning Gantt chart .....	7
Figure 2.1. MeetUp activity filter .....	13
Figure 2.2. MeetUp activity focused groups.....	13
Figure 2.3. Fever App homepage.....	15
Figure 2.4. Eventbrite homepage.....	16
Figure 2.5. Timpik homepage.....	17
Figure 2.6. Party with a Local homepage.....	18
Figure 2.7. Coachsrfing Travel App homepage.....	18
Figure 2.8. Native vs Web Apps.....	20
Figure 2.9. Android vs iOS usage .....	21
Figure 2.10. Android MVC pattern.....	22
Figure 2.11. Android MVC – MVP comparison.....	23
Figure 2.12. MVVM pattern.....	23
Figure 2.13. SQL vs NoSQL queries.....	25
Figure 2.14. NoSQL vs SQL.....	26
Figure 2.15. NodeJS event loop.....	28
Figure 2.16. Client – Server – Database interaction.....	30
Figure 3.1. ShoalUp diagram of components.....	50
Figure 3.2. Registration activity diagram.....	51
Figure 3.3. Login activity diagram.....	52
Figure 3.4. Event creation activity diagram.....	53
Figure 3.5. Event finding activity diagram.....	54
Figure 3.6. MongoDB user collection document schema.....	55
Figure 3.7. MongoDB event collection document schema.....	55

Figure 4.1. ShoalUp login screen.....	57
Figure 4.2. ShoalUp registration form.....	59
Figure 4.3. ShoalUp home screen.....	61
Figure 4.4. ShoalUp event creation.....	63
Figure 4.5. Date and time picker examples.....	64
Figure 4.6. ShoalUp Find Events screen.....	64
Figure 4.7. ShoalUp event complete information.....	64
Figure 4.8. ShoalUp discover screen.....	66
Figure 4.9. ShoalUp discover event full information.....	66
Figure 4.10. AddMarker() method example.....	67
Figure 4.11. ShoalUp profile information.....	67

# Index of tables

Table 3.1. Use case structure.....	33
Table 3.2. Use Case: Registration.....	34
Table 3.3. Use Case: Login.....	35
Table 3.4. Use Case: Account recovery.....	36
Table 3.5. Use Case: Profile modification.....	37
Table 3.6. Use Case: Account deletion.....	38
Table 3.7. Use Case: Log out.....	39
Table 3.8. Use Case: Create event.....	40
Table 3.9. Use Case: Modify event.....	41
Table 3.10. Use Case: Delete event.....	42
Table 3.11. Use Case: Show events.....	43
Table 3.12. Use Case: Find events.....	44
Table 3.13. Use Case: Filter events.....	45
Table 3.14. Use Case: Discover events.....	46
Table 3.15. Use Case: Join an event.....	47
Table 3.16. Use Case: Leave a comment.....	48
Table 5.1. Project technology costs.....	72
Table 5.2. Project's total budget.....	73



# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

In this section a brief introduction of the Bachelor Thesis will be put down, the development of a social, meeting application for users to discover events and people of their interest, with similar taste. Locating these “meetings” in a map and being able to track them by zone, activity or close location.

Below, the motivation for selecting this topic, objectives, structure of the memory and temporal planning, as well as the resources and expenses of the project will be described.

### 1.2 Motivation

The mobile device usage has increased exponentially during the last decade, smartphones have a star role in people’s everyday life, using them not just for communicating like before but for paying, reading, navigate the web...

The smartphone application development is in constant expansion, discovering new apps every day satisfying different purposes and necessities. This was one of the main motivations for me to learn a mobile app programming language, I wanted to know how to design and create apps from scratch, to forge my ideas that could appease actual community deficits.

The subject of the app was clear for me from the beginning, the modern society’s mindset is more open as time passes, is perceptible that people want to meet new people more often, experience new things, innovate... But more important than that, they don’t want to do it alone. I myself like to do sports quite often like padel or soccer, but sometimes I don’t know people that are able to synchronize their schedule with mine.

I wanted to develop a useful that people could use, and to help people with the same hobbies, travelling, sports, party... taste to get to meet each other and do things that otherwise they couldn't do, either because they don't want to do it alone, like travelling or because they can't actually do it, playing a soccer match.

## 1.3 Objectives

This section is split into two, the objectives of the thesis, and the objectives of the app itself, i.e. the requirements to be completed in order for it to be considered successful.

### 1.3.1 Thesis objectives

The main objective of the thesis was to develop a functional, application that allows users to create events for other users to communicate and join. The technology stack, the design wasn't defined yet, the developer had to make and discuss the decisions taken.

- Selection of technology stack: one of the objectives of the thesis was to increase the analytic capacity, boosted by the task of researching the available technologies and reasoning which one will better fit for achieving the goal. Important facts were taken into account, like the actual knowledge of some technology, for example, Java. The complexity of learning another language, the limitations and benefits of choosing one over the other, **SQL** vs **NoSQL**, backend languages, **Java**, **PHP**, **Python**, **Node**... The selection of the target mobile OS was simpler, due to the fact that for developing an iOS application a license is required, as well as an Apple computer, along with the fact that Android Studio uses Java as the main programming language and the information researched of **Android** vs **iOS** that will later be explained, lead to the decision of implementing the app for **Android**.
- Learn to code mobile applications: overcome the challenge of learning how to implement an app from scratch, design, learn how the components work, the interactions between client and server. Also learn how to use the Android SDK, **Android Studio**, not just for this project but for future ones, understanding how the graphic

part (XML) and the logic (Java) are associated, how to implement version control in order to always have a backup, and to acknowledge this function when working in bigger teams.

### 1.3.2 Application objectives

Despite being a big project, some limitations had to be established in order to set a feasible limit according to the magnitude of the Bachelor Thesis.

- Ease of use: the functioning of the application has to be transparent for the user, letting them just the responsibility of creating and managing their events.
- Follow, as accurately as possible, the 10 usability heuristics for user interface design [1]:
  - **Visibility of system status:** informing the user of actions happening by giving feedback in time.
  - **Match between system and the real world:** displaying the information in a logical, structured way; using common, familiar words to define everything within the system.
  - **User control and freedom:** give the users the maximum freedom regarding application navigation and content managing, always having the possibility to go back to a previous state.
  - **Consistency and standards:** same actions lead to the same result. Words have to be relatable to the action performed.
  - **Error prevention:** an error prevention design is required in order to avoid, as far as possible the error appearance on the execution. Descriptive error messages and action confirmations are also helpful.
  - **Recognition rather than recall:** minimize the memory requirements to the user by making the application's state visible to the user, in which screen he is; also all the required information for each action has to be given not

forcing the user to remember data from different sections.

- **Flexibility and efficiency of use:** use accelerators for speeding up the navigation process, adapting the user experience for both new and experienced users.
  - **Aesthetic and minimalist design:** display only the most relevant information; otherwise the irrelevant data might distract the user from the most important parts.
  - **Help users recognize, diagnose, and recover from errors:** display a message informing of the error and, if possible, the required steps to solve it in an understandable language.
  - **Help and documentation:** if required, provide the necessary documentation to be consulted, well-structured and easily accessed.
- Device resources friendly: having well implemented backend architecture, using REST services for helping to accomplish this goal, reducing the processing part that takes place on the actual device, using the appropriate servers.

## 1.4. Planning

The phases followed during the development of this project were established following the knowledge earned during the degree in several subjects focusing on software based projects planning.

Thus, the project development followed the five Software Development Life Cycle (**SDLC**) phases [2], shown in *Figure 1.1*.



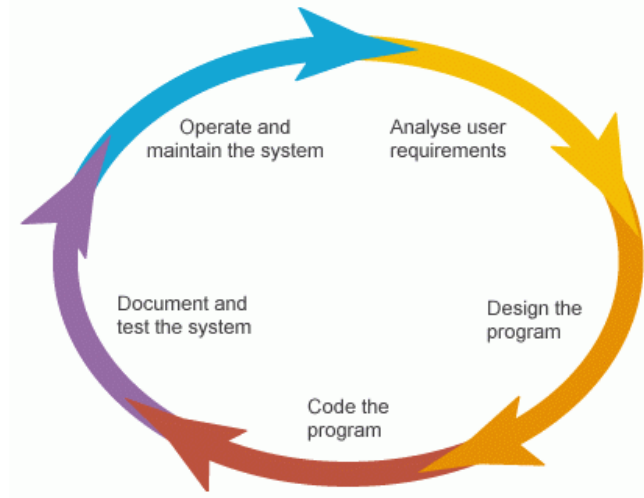


Figure 1.1. Software Development Life Cycle

- **Analyse user requirements:** this is a critical phase, where everything on the project has to be well-defined, the communication between stakeholders and the developing team takes place. Guessing what the user will expect of the application is also vital in order to later save time and resources in future enhancements.
- **Design the program:** in this phase mainly three sections are analyzed, the risks and the functional and non-functional specifications.
  - **Risks:** the potential threats and vulnerabilities that might take place during the development or after it, along with the conceivable legal aspects that the app might imply are studied.
  - **Functional Specifications:** interface requirements, backend interaction, how will the database be structured and managed are some aspects to be considered. Likewise, the workflow the app follows is studied, the interactions between user and the app.
  - **Non-functional Specifications:** here scalability of the project, future enhancements, performance and resources are some of the project facets to be analyzed. It is important to think in advance how the potential enhancements will affect the database structure. As it will later be explained, the

probable enhancements regarding this project helped in the decision of the database management system.

- **Code the program:** phase where the actual development takes place, the system is coded and the developed parts are tested to work as expected, this is usually the longest phase in the **SDLC**.
- **Document and test the system:** the fully developed project is migrated to the test environment where different kinds of evaluations are performed, not only the correct functioning is tested but also the user acceptance. Once the tests have been passed, the deployment of the project can be executed. In some cases, either the users or the technical team, will need documentation to understand how to use it or, for example, if new developers are incorporated to the team, they will need some feedback about the application in order to keep up with the rest of the team.
- **Operate and maintain the system:** once the application has been deployed, it still needs maintenance. Despite having passed through all the tests, some error might occur that will need a fast solution. Besides, some applications need the tech team to operate a part or the whole of it for the customers.

Figure 1.2 represents a Gantt chart that shows the different phases of the development of **ShoalUp** planning. It is presented in order to have a fast look at the whole process and, will also be briefly explained.

*“A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance what the various activities are, when each activity begins and ends, how long each activity is scheduled to last, where activities overlap with other activities, and by how much and the start and end date of the whole project.” [3].*

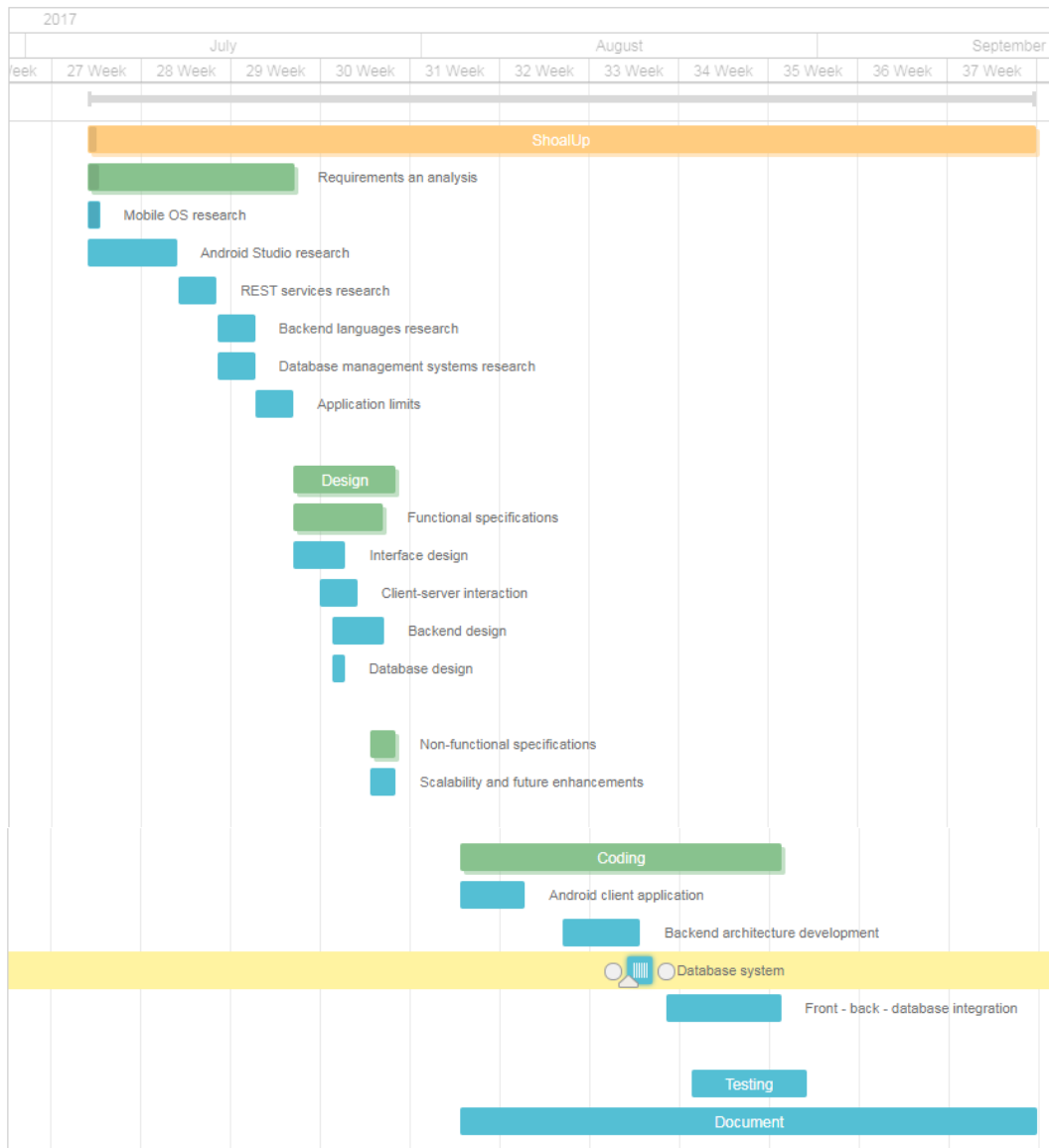


Figure 1.2. Project planning Gantt chart

- **Requirements and analysis:** the required research of the technology to be used and possibilities of development is performed at this point. **Android** vs **iOS**, how to use **Android Studio** and some tutorials to start off from, how **REST** services work and how to be implemented within the project, potential languages to develop the backend, database management systems pros and cons, **NoSQL** vs **SQL**.
- **Design:**
  - **Functional specifications:**
    - **Interface design:** design how the frontend will work, and its appearance in order to be intuitive and self-explanatory, reducing the time the user will have to spend to learn how to use the app. Also the interaction between activities and this section's workflow.
    - **Client-server interaction:** how the developed part will interact with the backend and access the database.
    - **Backend design:** the backend architecture was defined in this section, establishing the REST services to develop, its interactions with the database and the front end and the language to be used to accomplish this goal.
    - **Database design:** using NoSQL, the schema, i.e. the collections and documents that will be required for the expected functioning of the system, was define. Likewise, the relationships between them, OneToMany, ManyToMany... were designed for being implemented afterwards.
  - **Non-functional specifications:** after designing the actual features to be developed in this stage of the development process, the future enhancements and improvements were thought of and written down for a later implementation.
- **Coding:** the implementation of the previously design model is performed in this step. The frontend and backend were implemented separately, tested to work for later integrating them together.

- **Testing:** subsequently, the complete application was put into the test stage, verifying that it has been developed successfully and correcting the found errors.
- **Documentation:** once the previous stage was completed, and even though during the coding step some of the documentation was being written, it was at this point where most of it was done.

## 1.5 Means employed

For developing this software project, the subsequent resources have been required:

### Hardware resources:

- Laptop computer.
- Desktop computer.
- Smartphone Xiaomi Redmi 4.
- USB cable.

### Software resources:

- Netbeans software development Java platform.
- Android SDK (*Software Development Kit*): Android Studio
- RoboMongo (now Mongo 3T): MongoDB management tool.
- Volley HTTP library: managing the REST services more easily from the Android SDK.
- Google Places API for Android.
- AWS (*Amazon Web Services*): web server for storing the application REST services and Mongo database.
- Gantt Pro web application: for creating the Gantt chart.
- SourceTree: Git repositories managing tool.
- Git: version control software.
- Postman: Chrome extension for API development.
- Microsoft Office 2013.
- SmartDraw: flowchart creation tool.

The expenses for the above mentioned resources are calculated in section 5.2 of this memory.

## 1.6 Memory structure

**Chapter 1: Introduction** Including the motivation and objectives of the project, its planning, the resources employed for its accomplishment and the memory structure.

**Chapter 2: State of the art** The observed gap that the proposed solution would cover is analyzed, along with the evaluation of the available solutions and differences between them. Likewise, a study about the available technologies for the implementation is carried out.

**Chapter 3: Analysis and design** Includes the first two dimensions of the SDLC to be analyzed. The functional requirements, use cases and the actual design that the application will implement.

**Chapter 4: Implementation** This section includes the screenshots and explanation of the processes executed when interacting with the application.

**Chapter 5: Socioeconomic environment** Including the social and economic potential impacts of the application release. Along with the project budget and previsions.

**Chapter 6: Legal framework** Including the legal aspects related to the application.

**Chapter 7: Conclusions and future work** Lookback of the whole project, analyzing the prior objectives and whether they have been achieved or not. Also, future implementations and improvements to be made are described.

**Bibliography** References of the consulted books and webs for the development of this memory.

## CHAPTER 2

# STATE OF THE ART

The project's objective is to solve two problems related to the meetings/events creation and discovering by ordinary people. The main problem that is studied during this section is the difficulty to find an easy-to-use, intuitive and complete application that allows the user to discover events according to their interests at any moment, anywhere in a simple way.

Henceforth, the previously stated problem will be deeply analyzed, explaining the available similar apps and the differences between those and the proposed solution.

Moreover the available platforms for the project to be developed will be explained, alongside with the design/implementation possibilities and the available technologies to be taken into account for the completion of this project.

### 2.1. Problem analysis

For having a better understanding of the problem and solution, the aspects that play a major role in this situation will be put into context.

First, for a better understanding of what “event” or “meeting” are used for, in this application context, it's crucial to say that an event or meeting does not refer, necessarily, to an organized congregation of people hosted or promoted by companies, with ticket or prior registration required definition. With this in mind, an event can consist on a gathering of ordinary people willing to meet other users in order to perform an activity that otherwise they would not be able to achieve.

Society has always shown an interest for being surrounded of people with a taste similar to their own, the most compelling evidence are clubs, societies. From antique times to our days, people have been congregated in associations, organizations depending on their interests. Having that in mind, and thinking of how people are influenced to join clubs or societies where they get to know a community that better suits their hobbies, and can enjoy it sharing the experience with others, since the inception of their consciousness. It can clearly be seen that we tend, either subconsciously or on purpose the

human being is inclined to be in company of those who will better appreciate their interests.

Under those circumstances, different platforms arise, like **Meetup** [4], one of the applications that will be analyzed later on for the differences between that and the proposed solution to be explained.

But solutions to the observed problem do not necessarily have to be software based, there are plenty of societies that bring together people having alike passions. Universities, governments, town halls... all of these entities usually have a vast variety of different activity focused communities, from sports to travelling or simply for meeting new people.

Nevertheless all of the previously mentioned organizations, usually take advantage of the available software resources, as **Facebook events and groups** [5], to communicate between the parties conforming the aforesaid community.

Given these points, recently mentioned platforms and the proposed solution are further compared.

### **MeetUp**

MeetUp is a web based application that aims to help its users to “*Find local groups of people who love the same things that you do*”. Despite both solutions aspiring to reach the same goal, bringing people having akin interests together, giving them the possibility to form a concrete activity focused community where they can establish a relation and put it into practice. The idea is notably influenced by the formerly explained concept of clubs and societies, in other words, MeetUp has taken these organizations to the virtual world, increasing the opportunities of those looking to be included into one of these congregations to find their desired one.

Every club, society, association, has a limited range of action, to put it in another way, these entities are restricted by factors like location, mouth to mouth communication or physical publicity. University clubs are found to be an excellent example to illustrate the last remark, students usually only get to know associations based in the university they are studying at, being the chances of finding their coveted community more limited than if every aggrupation was available to be consulted in a digital format.

MeetUp not only successfully achieved the aforesaid goal but also facilitating the user the navigation through the multiple and different communities being able to filter them to find the ones closer to you, or by type of activity.



## Explorar

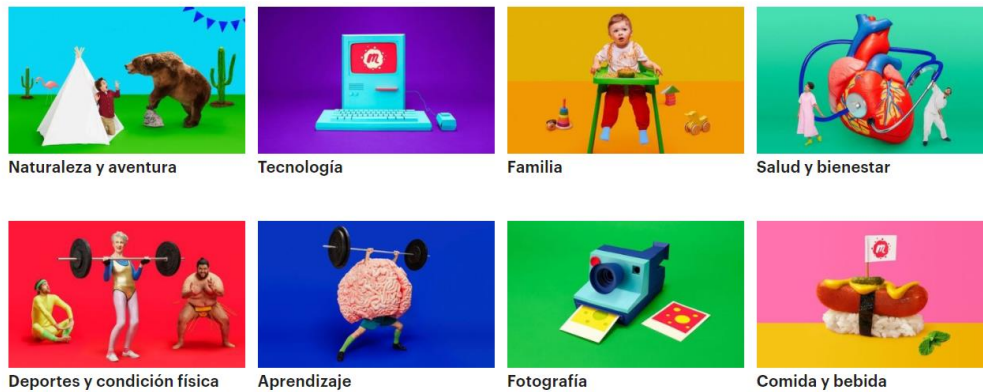


Figure 2.1. MeetUp activity filter

Figure 2.1 shows the previously mentioned activity filter, it certainly improves the user experience while navigating throughout the page, significantly reducing the time required to get through the different events, as well as being clear and self-explanatory.

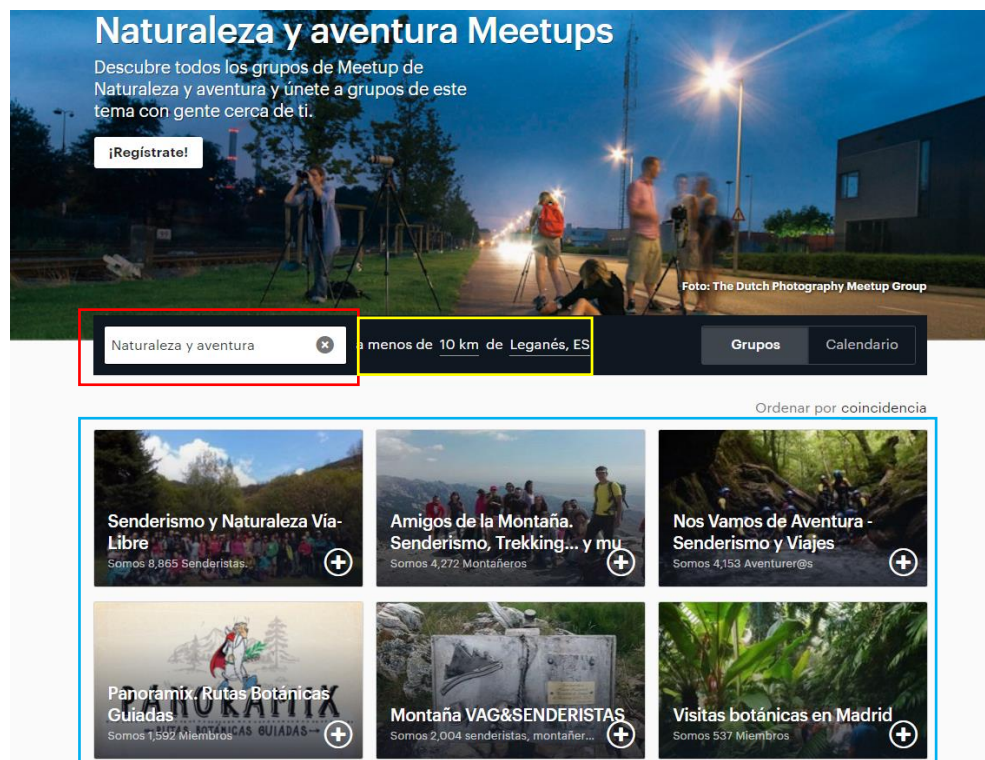


Figure 2.2. MeetUp activity focused groups

In the event of clicking one of *Figure 2.1* fields, all of the existing groups associated to that kind of activity will be shown as it is consecutively captured in *Figure 2.2*. As has been notated, the lastly mentioned figure contains three meaningful sections that have been highlighted for observation purposes:

- **Naturaleza y Aventura:** the red highlighted section shows the activity filter that the user would have previously selected. As it was mentioned in **Section 1.3.2**, one of the ten heuristics for a good user experience design states that the memory required to the user to recognize the current state of the app has to be minimized.
- **Location:** “a menos de 10 km de Leganés, ES”, yellow section is communicating the user that the events that will be found are in a 10 kilometer radius from Leganés. This filter allows the user to select, on one click, the city and the radius decreasing the time the user is required to spend to configure the parameters.
- **Events:** the last highlighted section will display the available communities found that fulfill the previously analyzed filter sections.

The point of the deep analysis of MeetUp is to point out that, despite this solution might seem similar to the one proposed on this paper, there are notable differences that make them completely disparate.

Platforms like MeetUp or Facebook with its pages and events, aim for creating this communities of alike people allowing them to communicate and arrange their own events or meetings. On the other hand ShoalUp will simplify this process granting the users the possibility of finding these events directly, without being part of a certain group.

This solution is intended to ease the process commented in the previous paragraph, getting rid of the necessity of being part of a certain community and follow the messages people post in order to acknowledge if an event is taking place, where or at what time. Users will just discover the encounters they filter to, this being activity based, city or proximity filters.

On the context of just displaying events, applications like Eventbrite [6] or Fever [7] can be found. Despite it can be thought for them to be similar to the proposed application, these solutions are design to publicize events already created, granting their users some advantages like discounts and also the implicit leverage of being able to consult all events taking place on the selected city at a glance.

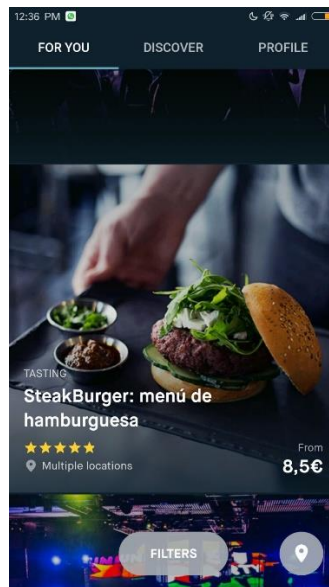


Figure 2.3. Fever App homepage

Figure 2.3 illustrates the *Fever App* homepage, displaying events or offers occurring close to the user's position. As it can also be observed, there are three tabs to be selected:

- **For you:** will advertise offers related to users preferences, previously acquired or researched plans.
- **Discover:** find plans transpiring around you. As its common in this kind of application and section, the user is able to filter by type of activity.
- **Profile:** basically the settings menu, the user is able to configure his profile, consult his obtained tickets, etc.



Figure 2.4. Eventbrite homepage

Similar to Fever, in Figure 2.4 it is shown the homepage of the previously mentioned application **Eventbrite**. Similar to formerly evaluated applications, Eventbrite will display the available events, its price. A deeper explanation is not required as its similarities with previously analyzed applications are relatable.

It has been compared the different available solutions to the problem aimed to be solved in this thesis showing the differences between them and the gaps for ShoalUp to cover. To summarize, first analyzed applications, **MeetUp** and **Facebook** aim for a community creation of alike interested people where they can debate and configure meetings. Second, applications like **Eventbrite** or **Fever**, where they advertise organized events providing their users with discounts and offers. While this being decent approaches to the problem, ShoalUp aims to ease the process with regard to MeetUp or Facebook and increase the event offer in respect of applications like Eventbrite or Fever, not just publicizing pre-organized events but allowing the common users to create their own.

Notwithstanding another kind of applications with the same objective as this project haven't been named, it's the case of **Timpik [8]**, **Party with a Local [9]** or **Couchsurfing Travel App [10]**.

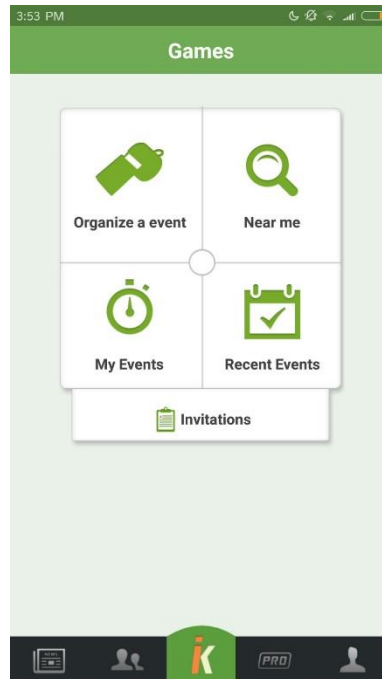


Figure 2.5. Timpik homepage

The above mentioned applications will be evaluated, showing similarities between them and analyzing the possible problem for which the proposed solution might fit.

Starting off with *Figure 2.5* and the application it illustrates, Timpik. As it is mentioned before, the applications to be analyzed from now on present a greater similarity to the one proposed within this thesis. With this in mind, it's relatable that the structure of this application is akin to ShoalUp's one. Regarding the application design, you can either organize an event, find events near to the user's position, manage the events you are attending or created. This application, despite looking simple and easy to use, requires more time than expected to get used to. Besides, the screens take longer to load than it should, this being a turn off for its users. Another flaw that was observed is the fact that it has a PRO version, i.e. features only accessible for premium users who will pay for this characteristic. In most of the cases, this aspect will make potential users to turn down the app.

The major difference with respect to ShoalUp is diversity, while Timpik is focused in sports events only, the proposed solution aims for bringing together events regardless of the field.



Figure 2.6. Party with a Local homepage

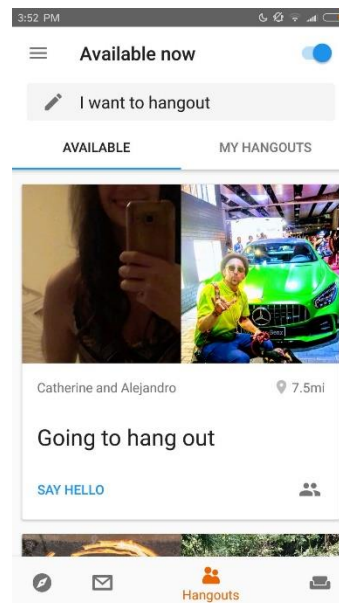


Figure 2.7. Couchsurfing Travel App homepage

Figures 2.6 – 2.7 represent the last two applications to analyze, Party with a Local and Coachsurfing Travel App. This two applications, despite being created with different scopes are currently being used for the same purpose, hang out and meet people from a certain place. To back up the previous statement it is imperative to understand the objective for which Coachsurfing Travel App was created, as it might be deduced from its name, this application aspired to help its ‘surfers’ to meet other users from the country they are travelling to, and that are offering to host them, for free, getting a deeper experience of the country by staying with a local, and the local to share its culture with foreigners. Nevertheless, and as Party with a Local, it is currently used to find people to hang out, grab a drink and ultimately socialize with strangers. This proves that either the application use and objective is not completely clear, illustrating a bad UX/UI design, or the market it aims for does not have demand enough for this application to successfully achieve its goal.

Given these points, it is discernible that the greatest problem users find regarding the discovering of events of their interest is the lack of unification. There are different applications for different kind of activities devolving upon the user the task of downloading, registering and learning how to use each one of them.

To summarize, the goal of ShoalUp is being a simple-to-use, intuitive application for fast finding any kind of events the user wishes around him or in the city of his election. Consequently, the evaluated problems, i.e. the time requiring problem of being part of a community and its necessity to reach an

agreement with those within the same in order to establish an event. Along with the second analyzed problem of application publicizing pre-organized, paying, company events and moreover the lastly mentioned problem of having to acquire, register and learn how to use several applications for different activity focused events.

The design and implementation of the proposed solution will be explained in the upcoming sections.

## 2.2 Platform and mobile OS evaluation

Regarding the application purpose and its implementation, the platform election is pretty straightforward. It will implement GPS based functions, for tracking events close to the user's position. Hence, the platform had to be mobile based.

Nevertheless, a study between web and native applications has been performed in order to support the election.

A study from *ditrendia* [11] shows that in 2016 the number of mobile devices, world like, was 7.9 thousand million, surpassing the number of habitants on Earth. In Europe 78 out of 100 people owns a smartphone, and in Spain smartphones represent an 87% of the mobile devices.

Further, in Spain, there are more of these devices than computers, 80% of the population have a mobile phone, against 73% owning a computer, either laptop or desktop.

Furthermore, the age when people start to use smartphones is being reduced year by year, 98% of the country between 10-14 years have full equipped, latest generation mobile device.

Moreover native applications are taking over the digital world, meaning a 54% of digital time spent. Not only the time spent is increasing but the use of formerly web applications like Facebook, the study shows that in Spain, 70% of the population access this social network from its mobile device.

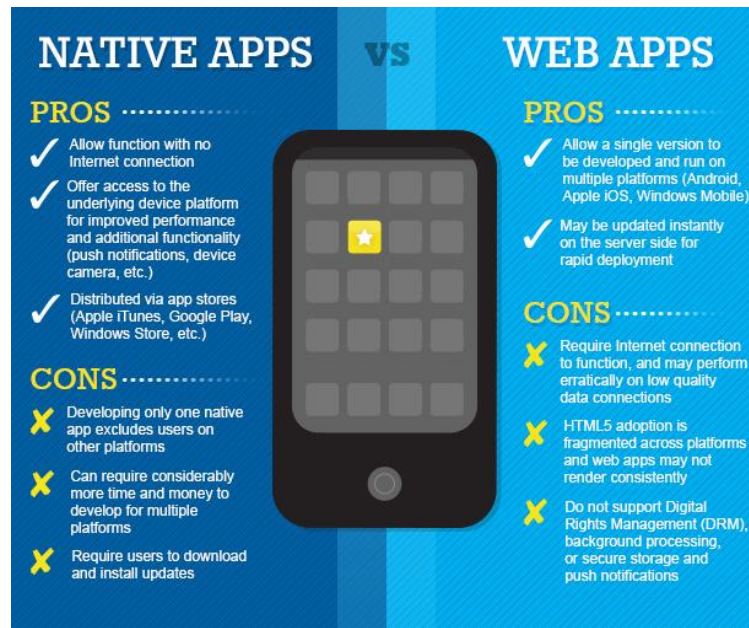


Figure 2.8. Native vs Web Apps

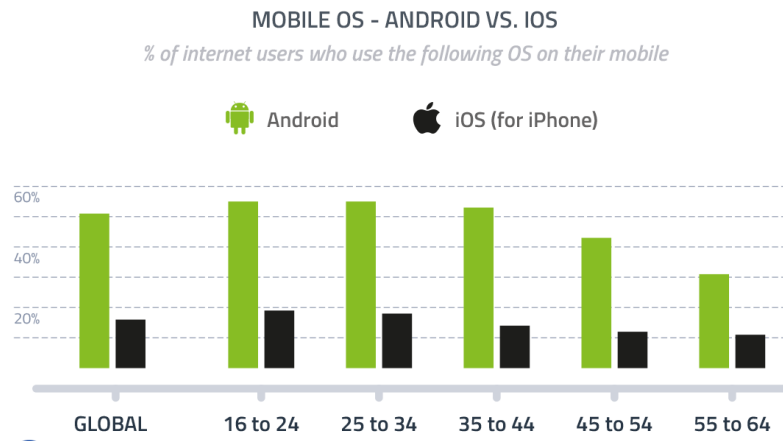
Figure 2.8 illustrates the pros and cons of native vs web apps, some of these features are fundamental for the platform election decision.

- Allow function with no Internet connection: native applications can store data locally allowing the user to have access even if without Internet connection. In this particular case, where the user might want to consult some details about the event they are attending and have no access to the Internet, having this data stored will be of great usage for him.
- Offer access to the underlying device platform: as it was commented before, the platform will use the phone's GPS in order to track discoverable event around to the user's location.

Cons like excluding other platform users will not be a problem in the future, as it is planned to be available for any platform, it will be further explained in the "Future work" section.

Once the platform decision has been defended, a comparison between both Android [12] and iOS [13] is required in order to justify this election.





*Figure 2.9. Android vs iOS usage*

*Figure 2.9* compares the usage of the mentioned mobile OS regarding the different age ranges, proving that, regarding a commercial aspect, the decision being Android the first OS for which the application will be implemented is the correct decision, as Android users are more than double with respect to iOS.

The above mentioned reason is not the only one that influenced the decision of implementing the application for Android. Again, factors like the previous experience on Android language (Java), and Android Studio utilization, during the last course. Along with the fact that for developing in iOS a MAC computer is required and the developer fee that Apple obliges to pay to anyone that develops an iOS application, 99 USD per year, led to the choice Android being the first mobile OS for the application to be developed for.

## 2.3 Design and implementation patterns

In this section an evaluation for the existing implementation patterns for Android will be carried out, needless to say that the below commented patterns are not Android exclusive but used on the development of any software based project.

### *Model View Controller (MVC)*

The MVC pattern [14] splits the tasks to be performed into 3 levels:

- **Model:** being independent from the other two components, it is the core of the model, it represents the data, state and business logic.

- **View:** the view represents the model reacting to the changes the model experience and modifying itself according to those. Having access to the model state despite not having permissions to modify it.
- **Controller:** the controller will decide which actions to perform when the view notifies an interaction, e.g. the user presses a button, changing the state of the model.

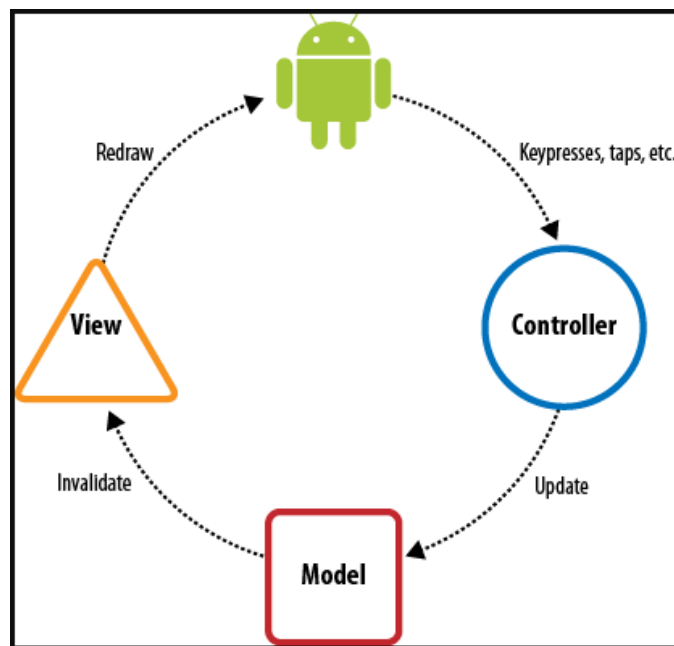


Figure 2.10. Android MVC pattern

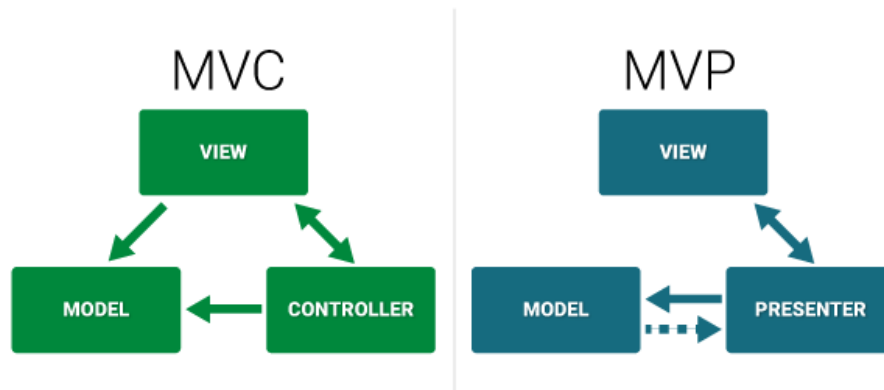
Figure 2.10 represents the MVC pattern exemplified for an Android application, where starting the user's zero state will be the logo, interacting with a concrete view, he/she performs a certain action, as it can be push a button, and the view will notify it to the controller, in this case an **Activity**, which will decide what to do, updating the model state that will, at the same time, be access by the view noticing the change, getting updated and showing the expected change to the user.

### Model View Presenter (MVP)

MVP pattern [14] derives from the MVC one, finding in the Presenter the most important difference while the Model and the View remain the same.

- **Presenter:** this component will act as the “middle man” between view and model, formatting the data obtain from the model and forwarding it to the view to be represented, and deciding as well actions to be taken when interacting with the view.

Regarding *Figure 2.11* below, differences between both previously explained models are illustrated. As it is observed, MVP presenter manages the communication between model and view, as opposed to MVC where the view had access to the model in order to consult its state and modify itself.

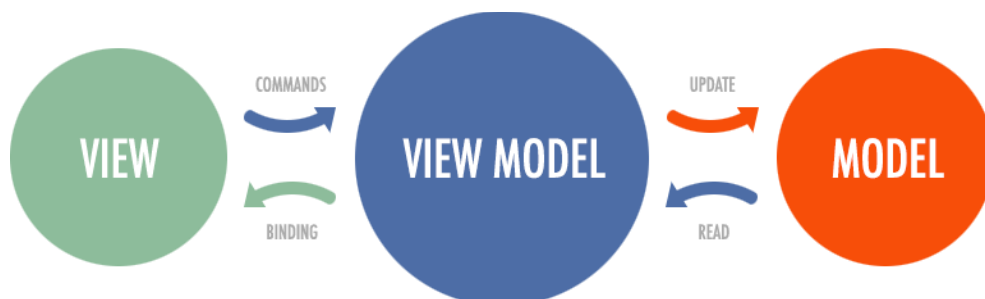


*Figure 2.11. MVC - MVP comparison*

### Model View ViewModel (MVVM)

The MVVM pattern [14] eases the testing process and increased modularity reducing the need of connections required to between model and view. Again, the model and view remain unchanged with respect to the MVC, being the controller the one that changes, exemplified in *Figure 2.12*.

- **ViewModel:** will substitute the controller being something in between a model and a view, helping in the task of passing the events from the view to the model and preparing the data required by the view.



*Figure 2.12. MVVM pattern*

## 2.4 Evaluation of available technologies

### 2.4.1 Database evaluation

In this section an analysis of relational vs non-relational databases [15] will be carried out, reasoning the election of using a non-relational database, MongoDB [16], as the one to implement the project in.

Relational databases, SQL, are positioned as the main model when studying database systems. Until the last decade, this database model was the only feasible one for the organizations to implement their data storage systems in. However, lately more and more organizations are changing his mindset as a new, more flexible, higher scalability with superior performance model has come into play, the non-relational database model or, as it is commonly called, NoSQL. The reason non-relational databases are also referred as NoSQL is because an expressive query language is not required, this system can be managed using other commonly use backend languages like Java or NodeJS. NoSQL is considered to have higher scalability due to the way it scales compared to SQL, the non-relational databases have a horizontal scalability, i.e. the way to scale is adding machines to the resource pool, whereas SQL scales vertically, i.e. adding power to the current machine (CPU,RAM).

For the evaluation, two dimensions will be analyzed in order to determine the right choice to make.

#### **Data Model**

While SQL uses a fixed data model, using tables for storing the data into rows and columns, with the columns representing different attributes while the rows store instances of the object. Being a fixed model means that this tables have to be predefined and the relations between them have also to be establish prior to their usage.

On the other hand, NoSQL databases store data in documents, using a JSON structure. Documents are more intuitive to use than former SQL tables, each document is associated to an object, this having different fields that can vary in type, strings, integers, arrays... A relatable advantage comes with data association, as it is mentioned above, SQL needs different tables connected with foreign keys to establish data relationships, on the contrary, NoSQL records and the associated data is usually stored in the same document, reducing considerably the number of required data accesses. Another advantage when it comes to scalability is the database schema, again, and as mentioned above, SQL uses a fixed schema, opposite to this, NoSQL use a dynamic schema, meaning that each document can contain different fields.

Having this in mind, it is comprehensive the statement that assures that this new database model is superior to SQL in flexibility and scalability.

Document based databases use a management system such as **MongoDB** or **CouchDB**.

Two other less commonly, different purposed data models that are used in NoSQL are the graph model and the key-value and wide column models. The graph model data representation is made by means of nodes, edges and properties. Although it is counter-intuitive and hence, less commonly used, it has specific situations to be used in like navigating social network connections or supply chains.

The key-value system is similar to a Java HashMap, where each value is represented and accessed by its unique key, being this, the only way to query the system. Along with the key-value, the wide column model stores data in a sorted map, being possible for each record to vary the number of columns it is stored at. Data will be recovered by primary key per column family, being the last mentioned ones an aggrupation of columns.

### Query Model

As it is stated in this section's introduction, SQL uses an expressive query language is required in order to interact with the database, commands like *SELECT*, *INSERT* or *DELETE* are an example of this, being used for retrieving, inserting or deleting data from the database.

NoSQL document model databases also allow to query any field within the document, some systems like the previously mentioned MongoDB, provide different set of indexes in order to facilitate the query process.

Regarding their query model, both SQL and NoSQL appear to be very similar, lying its greatest difference in the efficiency.

```
-----
SQL
-----
SELECT * FROM Customers
WHERE name = 'Alvaro';

UPDATE Customers
SET shift = 'Morning';
WHERE name = 'Alvaro';

-----
NoSQL
-----
db.Customers.find({"name":"Alvaro"});
db.Customers.update({"name":"Alvaro"}, {$set:{"shift":"Morning"}});
```

Figure 2.13. SQL vs NoSQL queries

Figure 2.13 shows an example of how the selection and update of data is performed regarding both systems being evaluated. In NoSQL, the *find()* function is the equivalent to the SQL's *SELECT*. In absence of tables, non-relational databases use collections to store the documents, and in the same way as SQL specifies the table like *FROM table*, NoSQL will select the collection to be queried in the following way, *db.Collection.command()*. The update command is similar, in the case of NoSQL, there are two possible update modes, updating the whole document, done by not writing the *\$set*, just the field to be queried for and the fields that the documents that fulfill the conditions will be substituted for. Or as it is done in Figure 2.13, updating only the parameters that follow the *\$set* command.

	NoSQL	SQL
<b>Model</b>	Non-relational Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Relational Stores data in a table
<b>Data</b>	Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
	New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
	Relationships are often captured by denormalizing data and presenting it in a single record	Relationships are often captured in a using joins to resolve references across tables
	Good for semi-structured data	Good for structured data
<b>Schema</b>	Dynamic or flexible schemas Database is schema-agnostic and the schema is dictated by the application. This allows for agility and highly iterative development	Strict schema Schema must be maintained and kept in sync between application and database
<b>Transactions</b>	ACID transaction support varies per solution	Supports ACID transactions
<b>Consistency</b>	Consistency varies per solution, some solutions have tunable consistency	Strong consistency supported
<b>Scale</b>	Scales well horizontally	Scales well vertically

Figure 2.14. NoSQL vs SQL

For the conclusion of this section, Figure 2.14 is used to summarize all of the above mentioned dimensions along with others like consistency or transactions that were not deeply analyzed as the repercussion it had on the current project was not as critical as the evaluated ones.

## 2.4.2 Backend technologies evaluation

In this section the possibilities to develop the backend architecture in will be analyzed for having a better perspective of the election. It is important to realize that the technologies here explained are not the only valid ones, the three languages considered for this concrete project and that are, consequently, explained hereunder have been extracted from a greater gamut of disposable languages.

With this been said, the languages considered that could better fit in this concrete project were: **Java** [17], **NodeJS** [18] and **PHP** [19]. Needless to say that this languages are heavily consolidated between the developing communities based on the years they have successfully served to the required purposes like Java or PHP, counting with a robust backup and support. Despite the relatively newness of NodeJS it has a large acceptance between developers and it is raising to be one of the most used languages in the near future.

### JAVA

Java is a programming language developed by Sun Microsystems, it's one of the most used languages in the software community due to its "longevity", as it was released in 1995. It was created with a purpose, for the applications developed to only need to be compiled for a platform, avoiding the recompilation for running it on a different one. This is commonly denoted as **Write Once, Run Anywhere**.

Java is a simple, dynamic, platform independent, portable, object oriented language. It also assures a strong security as it uses public-key encryption for authentication. Programs developed with Java are considered to be robust as it double checks for errors both at compile and runtime, not only these programs are capable of performing many tasks at the same time, as Java is developed to enable the multithread feature, but also make them run smoothly and with a great performance due to Java's Just-In-Time compilers.

### PHP

Being released in 1994 as an open source project, PHP's popularity across the software developer community exponentially increased. Despite being commonly used for web applications development, as it is embedded in HTML, it is not rare to find mobile device applications using this language to establish connections with the database and in general for its backend architecture.

It is a server side scripting language used for managing databases, web dynamic content... Integrated in a vast number of databases like MySQL, PostgreSQL, Oracle or Microsoft SQL Server, it is one of the best options when building an application that will require interaction with one of the previous databases. It also can collect form data, manage cookies data, encrypt data or perform system operations (create, open, delete...) against server stored files.

As a result of the previous evaluation, PHP has been consider as a potential development option for this project as it also can run on a variety of

platforms (Windows, Linux, MAC OS...), it's compatible with almost every existing server, it is free and efficiently runs on the server side.

### ***NodeJS***

The last language to be considered is the most recently released one, NodeJS. Being launched in 2009, it has been less than a decade in the programming scene but it has risen to the top positions when it comes to backend languages. Its development was influenced by technologies like Ruby's Event Machine or Python's Twisted.

Taking the event model to the next level, in spite of being single-threaded, NodeJS implements a non-blocking I/O operations event loop, this is made possible by sending operations to the kernel as frequently as possible.

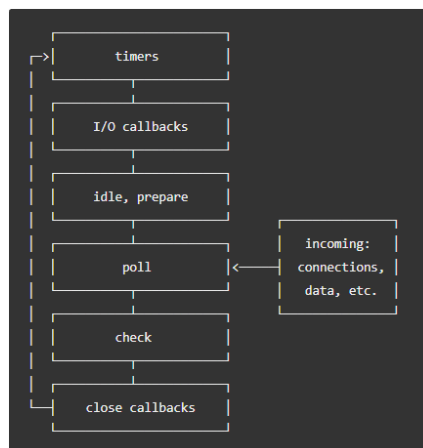


Figure 2.15. NodeJS event loop

For a better understanding of the NodeJS event loop, it will be deeper explained. *Figure 2.15* displays the event loop's order of operations, when reaching a certain phase, the event loop will execute the N operations waiting in the FIFO callback queue, existing a queue for each phase. The loop will only advance to the following phase when the queue has been exhausted or the callback limit has been reached.

The consideration of this NodeJS as a possibility for this project is explained by its architecture. Being an asynchronous, event-driven engine, when making a request to the database, it keeps working on other tasks rather than stalling until a response is received. This feature couples the best with the selected non-relational database system for this project, MongoDB, as it is also designed to work asynchronously.

Having all of the previous considerations in mind and taking into account that an implementation in any of the above analyzed languages would



perfectly fit the correct development of the application, it was concluded that the best fitting language for this application's purpose was Java. Having both frontend (Android) and backend sides implemented under the same programming language will increase the overall consistency of the project. Likewise, the prior knowledge of the cited language would considerably reduce the time investment required by the project.

### **REST Services**

Once the backend language has been justified, a brief analysis of the services that will be used is to be performed.

Representation State Transfer (REST) architectural style [20] was presented in 2000. Interface like, it uses, but it is not only related to, the *Hypertext Transfer Protocol (HTTP)* for data obtaining or to execute data operations.

Despite REST being ubiquitous, it is not a standard but a style of the HTTP protocol. Nevertheless, its services benefit from the security, caching, service routing through DNS of the HTTP protocol from the web architecture itself. The key to REST simplicity and velocity resides on its four principles:

- **Resource identification through URI:** providing a global addressing space for the potential discovery of resources and/or services.
- **Uniform interface:** use Create, Read, Update, Delete operations (CRUD) to manage its resources. This operations are also mapped to HTTP requests.
- **Self-descriptive messages:** from the message representation it is possible to obtain its content in different formats, HTML, JSON, XML...
- **Stateless interactions:** saving the client context on the server would limit scalability, REST avoids this problem by being the client the one holding the state.

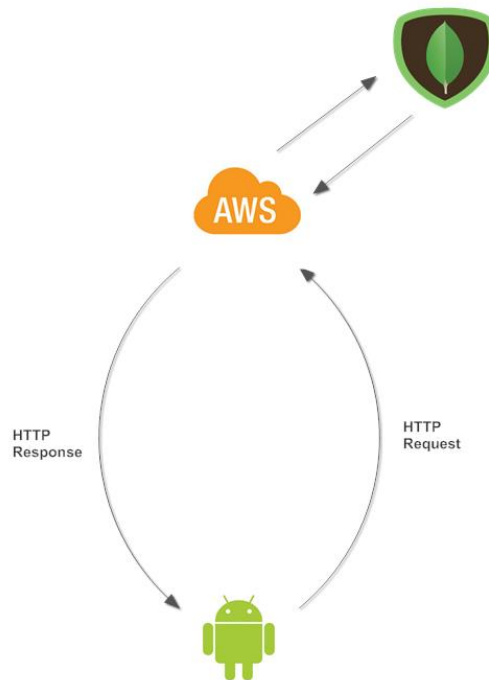


Figure 2.16. Client - Server - Database interaction

As it was pointed out above, REST is a software architectural style designed for data transferring between distributed systems, such as the Web. Opposing to the Simple Object Access Protocol (**SOAP**), REST does not need another layer for the mentioned data transmission, resulting in a lighter and faster and less complex method that has replaced SOAP in almost any case, especially when it comes to an extensive number of users, as the former main data transfer protocol is based on Remote Procedure Call (RPC), being this more suitable for a controlled environment.

With this being said, and having in mind REST HTTP transfer protocol, regarding *Figure 2.16*, the interactions between client server and database are better understood. The client will send a HTTP request to the server, which will address it, this request can be a data treatment, an insertion, get, update or delete from the database... At this point the server will performed the required actions returning the necessary data to the client in a HTTP response message.

## CHAPTER 3

# ANALYSIS AND DESIGN

In this section two steps of the *SDLC*, concretely the analysis of the requirements and the design followed for a further implementation will be deeply analyzed, including use cases, flow charts and any extra material to support and clarify the followed processes.

### 3.1 Analysis

Following a linear criteria, the first dimension of the *SDLC* to be developed is the analysis. Hence, the requirements and use cases will be are to be presented on the following subsections.

#### 3.1.1 Requirements

In this section the functional requirements for the application to fulfil in order to be considered complete will be stated.

##### 3.1.1.1 User

1. **Registration (USR-00):** in case of the user being new to the application, he will be required to register, fulfilling some basic information fields.
2. **Login (USR-01):** the user will be able to, in case he has been previously registered, log into the application with his credentials, i.e. email and password.
3. **Account recovery (USR-02):** in case the user's account has been stolen or the password forgotten, there will be an option to send a new password to his email.
4. **Profile modification (USR-03):** the user will be able to manage his profile, updating his personal data or his profile image.
5. **Account deletion (USR-04):** the information related to the user executing this feature will be eliminated.

6. **Log out (USR-05):** the user will close his session being redirected to the login/register screen.

### 3.1.1.2 Events

1. **Create event (EVT-00):** this functionality refers to the creation of a new event fulfilling the mandatory fields like date, time, title or location.
2. **Modify event (EVT-01):** events created by a user can be modify by himself only, by this modification a change on the previously mentioned parameters at **'create an event'** point.
3. **Delete event (EVT-02):** the creator of each event will have the possibility to cancel it, removing all of its information.
4. **Show events (EVT-03):** events either created or marked to be assisted by the user will be displayed.
5. **Find events (EVT-04):** a list of all the available events will be shown to the user, sorted by distance.
6. **Filter events (EVT-05):** when on find event, the user will be granted the possibility to filter by different fields like activity type, distance, date or location.
7. **Discover events (EVT-06):** event closer to users position will be displayed in a map, being possible to navigate around discovering at the same time the events scoped.
8. **Join an event (EVT-07):** users are given the possibility to join an event marked as attendant and receiving updates.
9. **Leave a comment (EVT-08):** users either attending or not to an event can write a comment on the event.

### 3.1.2 Use cases

Being defined the functional requirements, the use cases will be hereafter described [21]. *Table 3.1* represents the structure that the use cases will follow.

<b>Stakeholder</b>	Actor/s intervening
<b>Description</b>	Objective
<b>Pre-conditions</b>	Conditions to be met for prior to the use case
<b>Regular flow</b>	Actions that are executed during the duration of the use case
<b>Alternative flow</b>	Actions that can also occur during the use case
<b>Post-conditions</b>	Conditions to be met posterior to the use case
<b>Frequency</b>	Commonness of the use case

*Table 3.1. Use case structure*

### 3.1.2.1 Registration

Following the general use case structure presented in section 3.2, *Table 3.2* illustrates the use case of a new user registering into the application, corresponding to functional requirement **USR-00**.

<b>Stakeholder</b>	Anonymous user.
<b>Description</b>	Create a ShoalUp account.
<b>Pre-conditions</b>	The user is not registered in the database.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. The stakeholder opens the application in the register/login screen.</li> <li>2. Not having registered account, the user goes to the register form.</li> <li>3. The stakeholder fills out the mandatory fields of the form.</li> <li>4. The application will check the correctness of the fields and if there is no account associated to the introduced email, this being valid the user account will be created.</li> <li>5. The user will be redirected to his home screen.</li> </ol>
<b>Alternative flow</b>	In case the input data isn't correct, e.g. not valid email address, the user will be notified to make the correspondent changes.
<b>Post-conditions</b>	The user is information is stored in the database.
<b>Frequency</b>	Every time a new user wants to register into the application.

*Table 3.2. Use Case: Registration*

### 3.1.2.2 Login

Following the general use case structure presented in section 3.2, *Table 3.3* illustrates the use case of a registered user logging into the application, corresponding to functional requirement **USR-01**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Log into the application.
<b>Pre-conditions</b>	The user has a ShoalUp account.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. The stakeholder opens the application in the register/login screen.</li> <li>2. The user enters its credentials, email and password.</li> <li>3. The application checks if the introduced credential are stored in the SQLite database.</li> <li>4. If there is no register of the introduced data in the device's database the application will check it on the Mongo database.</li> <li>5. Being step 3 or 4 correct, the user is redirected into his home screen.</li> </ol>
<b>Alternative flow</b>	In case the credentials are not correct, an error will be displayed and the user will be asked for entering them again.
<b>Alternative flow</b>	In case the user closed the application without logging out, it will remember his credentials not being necessary to entering them every time the same user is logging in.
<b>Post-conditions</b>	N/A
<b>Frequency</b>	Every time a registered user logs into the application.

*Table 3.3. Use Case: Login*

### 3.1.2.3 Account recovery

Following the general use case structure presented in section 3.2, *Table 3.4* illustrates the use case of a registered user retrieving a lost password, corresponding to functional requirement **USR-02**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Retrieve a lost password.
<b>Pre-conditions</b>	The user has a ShoalUp account.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. The user starts the password retrieval process.</li> <li>2. Enters the email to which his account is associated.</li> <li>3. The application checks for the existence of the email address in the user collection.</li> <li>4. If the email is correct, the application will update its password for a random generated one.</li> <li>5. The new password will be sent to the user's email address.</li> <li>6. The user will log into the application with the new password.</li> </ol>
<b>Alternative flow</b>	N/A
<b>Post-conditions</b>	The password is updated.
<b>Frequency</b>	Every time a user no longer has access to his password.

*Table 3.4. Use Case: Account recovery*



### 3.1.2.4 Profile modification

Following the general use case structure presented in section 3.2, *Table 3.5* illustrates the use case of a registered user profile modification, corresponding to functional requirement **USR-03**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Update user's profile information.
<b>Pre-conditions</b>	The user has a ShoalUp account and he is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. The user accesses to his profile.</li> <li>2. Presses the modification button.</li> <li>3. Changes the fields he wants to.</li> <li>4. Confirms the changes by pressing the update button.</li> <li>5. Application processes the changes updating the information in the device and database.</li> </ol>
<b>Alternative flow</b>	If the data to be updated is not correct, an error message will be shown to the user to correct the conflicting fields.
<b>Post-conditions</b>	Fields changed are correctly updated.
<b>Frequency</b>	Every time a user wants to change his user information.

*Table 3.5. Use Case: Profile modification*

### 3.1.2.5 Account deletion

Following the general use case structure presented in section 3.2, *Table 3.6* illustrates the use case of a registered user account deletion, corresponding to functional requirement **USR-04**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Delete an existing account.
<b>Pre-conditions</b>	The user has a ShoalUp account and is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the profile modification screen.</li> <li>2. The user presses the delete account button.</li> <li>3. The user will be asked to confirm that he wants to delete his account.</li> <li>4. Upon confirmation the application will delete that account's data from the device and the database.</li> </ol>
<b>Alternative flow</b>	N/A
<b>Post-conditions</b>	The account information is successfully deleted from both the database and the device.
<b>Frequency</b>	Every time a user wants to delete his account.

*Table 3.6. Use Case: Account deletion*

### 3.1.2.6 Log out

Following the general use case structure presented in section 3.2, *Table 3.7* illustrates the use case of a user logging out of the application, corresponding to functional requirement **USR-05**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Disconnect from the user's currently logged account.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User presses the log out button.</li> <li>2. User will be asked to confirm that he wants to log out.</li> <li>3. Upon confirmation he will be redirected to the login/register screen.</li> </ol>
<b>Alternative flow</b>	He cancels the confirmation staying inside the application.
<b>Post-conditions</b>	The user is taken to the login/registration screen.
<b>Frequency</b>	Every time a user wants to log out of this account.

*Table 3.7. Use Case: Log out*

### 3.1.2.7 Create event

Following the general use case structure presented in section 3.2, *Table 3.8* illustrates the use case of event creation, corresponding to functional requirement **EVT-00**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Create an event to be seen by other users.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User presses the new event button.</li> <li>2. The user is taken to the new event form.</li> <li>3. The user is asked to fulfil the form, being some fields mandatory and some other optional.</li> <li>4. The user clicks the create event and a confirmation message is thrown.</li> <li>5. Upon confirmation the event is created.</li> </ol>
<b>Alternative flow</b>	The introduced data is not correct, an error message will be displayed to the user asking to correct the conflicted fields.
<b>Alternative flow</b>	The user decides not to create the event, presses the cancel button and he is taken to the previous screen.
<b>Post-conditions</b>	The event is successfully created, being discoverable for other users.
<b>Frequency</b>	Every time a user wants to create an event.

*Table 3.8. Use Case: Create event*

### 3.1.2.8 Modify an event

Following the general use case structure presented in section 3.2, *Table 3.9* illustrates the use case of event modification, corresponding to functional requirement **EVT-01**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Modify the information regarding an event created by the user.
<b>Pre-conditions</b>	The user is logged into the application and has previously created an event.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the “My Events” screen.</li> <li>2. The user clicks on an event created by him.</li> <li>3. User presses the edit button.</li> <li>4. Modifies the desired fields.</li> <li>5. The user clicks on the update button.</li> <li>6. A confirmation message is shown to the user.</li> <li>7. Upon confirmation the modified fields are updated.</li> </ol>
<b>Alternative flow</b>	The introduced fields are not correct, showing an error message to the user asking to correct the conflicted fields.
<b>Alternative flow</b>	The user decides not to update, presses the cancel button and he is taken to the “My events” screen.
<b>Post-conditions</b>	The event information is modified and saved in the database.
<b>Frequency</b>	Every time a user wants to modify an event that he created.

*Table 3.9. Use Case: Modify event*

### 3.1.2.9 Delete event

Following the general use case structure presented in section 3.2, *Table 3.10* illustrates the use case of event deletion, corresponding to functional requirement **EVT-02**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Delete an event created by the user.
<b>Pre-conditions</b>	The user is logged into the application and has previously created an event.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the “My Events” screen.</li> <li>2. The user clicks on an event created by him.</li> <li>3. User presses the delete button.</li> <li>4. A confirmation message is shown to the user.</li> <li>5. Upon confirmation the event will be deleted.</li> </ol>
<b>Alternative flow</b>	The user decides not to delete, presses the cancel button and he is taken to the “My events” screen.
<b>Post-conditions</b>	The event information is eliminated from the database.
<b>Frequency</b>	Every time a user wants to delete an event that he created.

*Table 3.10. Use Case: Delete event*

### 3.1.2.10 Show events

Following the general use case structure presented in section 3.2, *Table 3.11* illustrates the use case of displaying user events, corresponding to functional requirement **EVT-03**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Display the events the user is either attending or created.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User navigates to the “My Events” screen.</li> <li>2. The application obtains all the events where the user is part of either as an attendant or as the creator.</li> <li>3. The obtained events are displayed to the user.</li> </ol>
<b>Alternative flow</b>	There system find no events, so the user is asked to create his first one.
<b>Post-conditions</b>	The events are correctly shown.
<b>Frequency</b>	Every time a user wants to list his events.

*Table 3.11. Use Case: Show events*

### 3.1.2.11 Find events

Following the general use case structure presented in section 3.2, *Table 3.10* illustrates the use case of event finding, corresponding to functional requirement **EVT-04**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	List the available events to be attended.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the “Find Events” screen.</li> <li>2. The application retrieves all the available events and sort them with respect to the user’s position.</li> <li>3. The user is shown the available events.</li> </ol>
<b>Alternative flow</b>	There are no available events, displaying an information message to the user.
<b>Post-conditions</b>	The events are correctly shown.
<b>Frequency</b>	Every time a user wants to find a new event.

*Table 3.12. Use Case: Find events*



### 3.1.2.12 Filter events

Following the general use case structure presented in section 3.2, *Table 3.13* illustrates the use case of event filtering, corresponding to functional requirement **EVT-05**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Filter the list of events to be shown.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the “Find Events” screen.</li> <li>2. The user clicks on the filter button.</li> <li>3. Selects the parameters to filter for.</li> <li>4. The application retrieves all the available events that pass the filter.</li> <li>5. The user is shown the available events.</li> </ol>
<b>Alternative flow</b>	There are no available events after the filtering, displaying an information message to the user.
<b>Post-conditions</b>	The events are correctly filtered and displayed.
<b>Frequency</b>	Every time a user wants to find a new event with certain features.

*Table 3.13. Use Case: Filter events*

### 3.1.2.13 Discover events

Following the general use case structure presented in section 3.2, *Table 3.14* illustrates the use case of event filtering, corresponding to functional requirement **EVT-06**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Navigate through the map discovering events.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is in the “Discover events” screen.</li> <li>2. A map showing the events in the focused location is loaded.</li> <li>3. The user navigates around.</li> <li>4. The application takes uses the coordinates to establish a radius and obtain the available events in that radius, displaying them in the map.</li> </ol>
<b>Alternative flow</b>	There are no events in the current zone, hence, the map will be shown empty.
<b>Post-conditions</b>	Available events are displayed in the map.
<b>Frequency</b>	Every time a user wants to navigate around to discover new events.

*Table 3.14. Use Case: Discover events*

### 3.1.2.14 Join an event

Following the general use case structure presented in section 3.2, *Table 3.15* illustrates the use case of event filtering, corresponding to functional requirement **EVT-07**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Mark an event as attending.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is either in the “Find Events” or in the “Discover events” screen.</li> <li>2. The user clicks on an event to be expanded.</li> <li>3. User presses the join button.</li> <li>4. The application adds him to the list of the attendants.</li> <li>5. Whenever a modification is made on that event the user will receive a notification.</li> </ol>
<b>Alternative flow</b>	The user decides not to be part of the event anymore, clicking on the leave button. The application will delete the user from the attendance list.
<b>Post-conditions</b>	The attendance list is correctly updated.
<b>Frequency</b>	Every time a user wants to join an event.

*Table 3.15. Use Case: Join an event*

### 3.1.2.15 Leave a comment

Following the general use case structure presented in section 3.2, *Table 3.16* illustrates the use case of event filtering, corresponding to functional requirement **EVT-o8**.

<b>Stakeholder</b>	Registered user.
<b>Description</b>	Leave a comment on a certain event.
<b>Pre-conditions</b>	The user is logged into the application.
<b>Regular flow</b>	<ol style="list-style-type: none"> <li>1. User is either in the “Find Events”, the “Discover events” or the “My Events” screen.</li> <li>2. The user clicks on an event to be expanded.</li> <li>3. User presses the text box and writes a comment.</li> <li>4. The user clicks on the post button.</li> <li>5. The application will update the event adding the comment to its appropriate section.</li> </ol>
<b>Alternative flow</b>	The user decides not to post a comment anymore, presses the cancel button being taken to the previous screen.
<b>Post-conditions</b>	The event comments are updated from the database.
<b>Frequency</b>	Every time a user wants to leave a comment on any event.

*Table 3.16. Use Case: Leave a comment*

## 3.2 Design

Following the SDLC flow, the next dimension to get into is the design of the application. In this section the infrastructure of the application will be explained supported by component diagrams and its relation, illustrating the backbone of the application.

Further, flow charts representing the most significant use cases will be presented and the necessary clarifications will be made in order to arrive at the implementation section having a better understanding of how the application works.

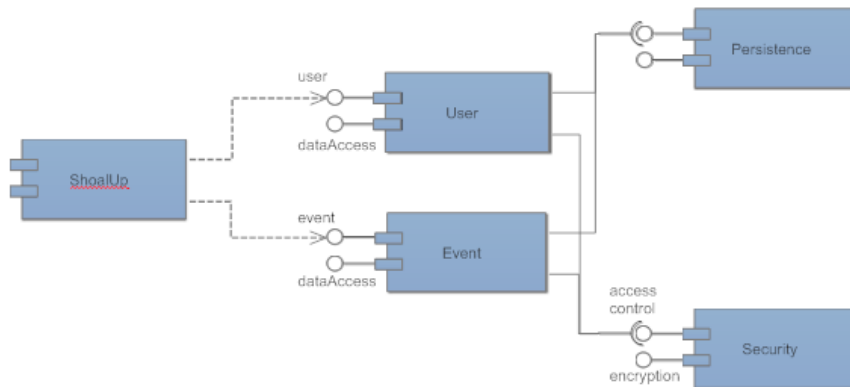
### 3.2.1 Application architecture

The Android project is structured as follows:

- **Android application:** main component of the system, the client application represent the user interface (UI) and the logic part (controller). All of the interactions to be made between the user and the application will be collected by this component. Likewise, the backend calls to the server and database will be executed and the data received from those calls gathered and treated.
- **SQLite database [22]:** allocated in the personal mobile device, will be in charge of storing some critical data in order to avoid calling the server as many times. Also the stored data, e.g. user credentials, or the events created or attended by the user.
- **AWS server [23]:** cloud server where the REST-ful web services are allocated, it is in charge of addressing the calls coming from the client, access the database performing the required operations depending on the call parameters.
- **MongoDB:** database also allocated in the server. It is in charge of the application data persistence along with the SQLite database. This database will store the whole data application, i.e. as it was mentioned before, SQLite will be in charge of storing a particular set of data, whilst it will be the Mongo database the one in charge to store all of the information, including the one stored in the complementary database.

### 3.2.2 Diagram of components

On the present section an overview of the components that compose the application is introduced. *Figure 3.1* shows the main components of the application in a UML 2.0 diagram [21]. The purpose of the diagram is to have an idea about what it will be later deeper explained in section 4.



*Figure 3.1. ShoalUp diagram of components*

### 3.2.3 Activity diagrams

An activity diagram [21] is used to display the sequence of activities that a concrete process will follow. From a start point to a finish point this diagrams illustrate states, activities, decisions, paths and more information that, as in this concrete case, helps to understand the activity flow when it comes to studying a new project.

#### 3.2.3.1 Registration activity

The following figure represents the activity diagram of the process of registration followed by a new user to the application. These diagrams are self-explanatory requiring no further explanation.

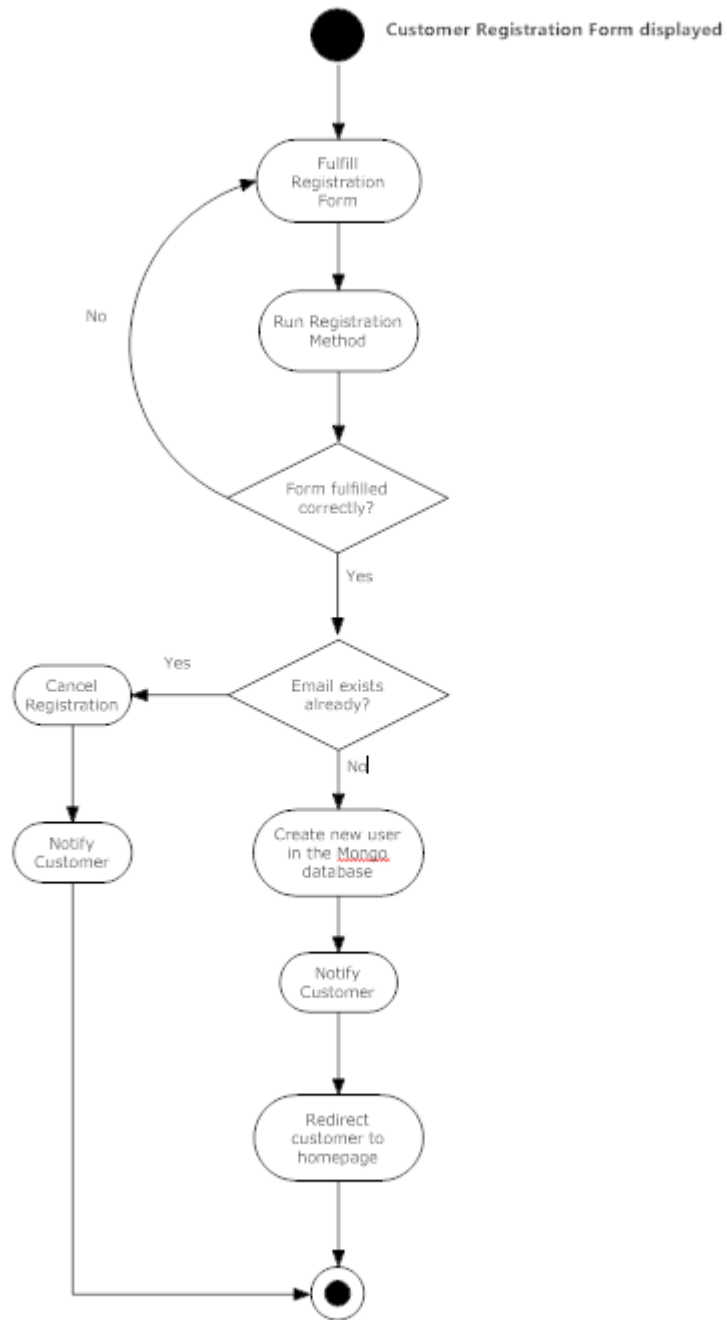


Figure 3.2. Registration activity diagram

### 3.2.3.2 Login activity

The following figure represents the activity diagram of the login followed by an already registered. These diagrams are self-explanatory requiring no further explanation.



Figure 3.3. Login activity diagram



### 3.2.3.3 Event creation activity

The following figure represents the activity diagram of for the creation of an event followed by a registered and logged user. These diagrams are self-explanatory requiring no further explanation.

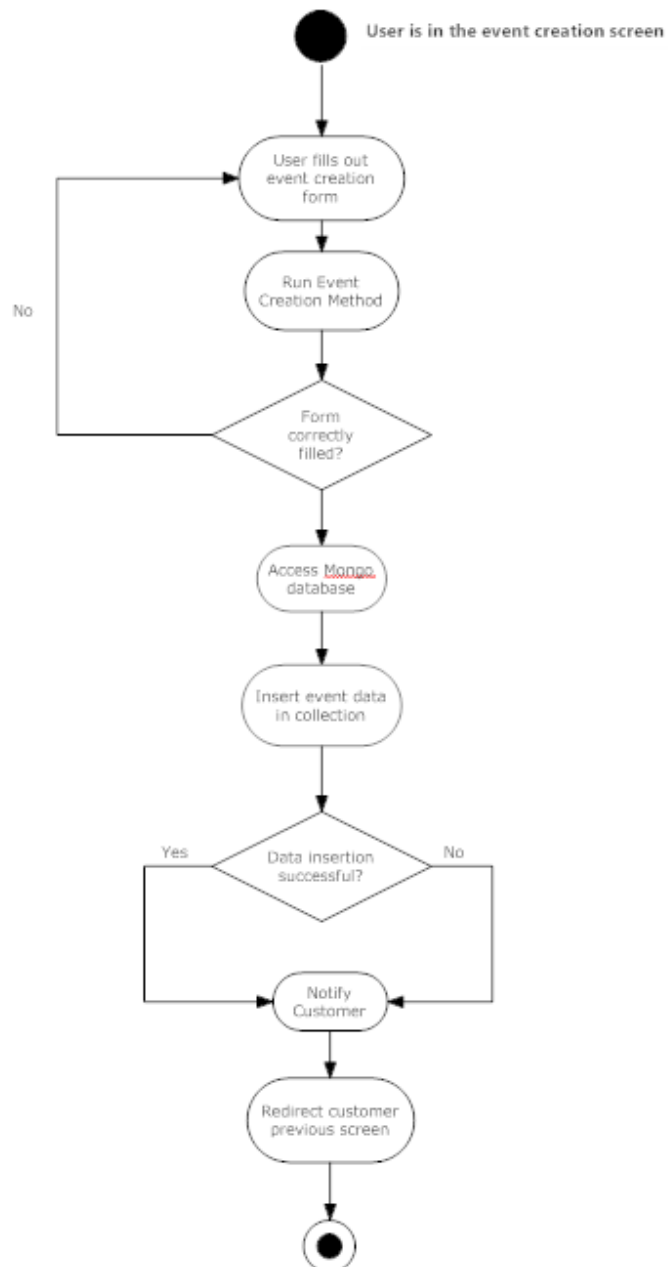


Figure 3.4. Event creation activity diagram

### 3.2.3.4 Event retrieval activity

The following figure represents the activity diagram of the login followed by an already registered. These diagrams are self-explanatory requiring no further explanation.

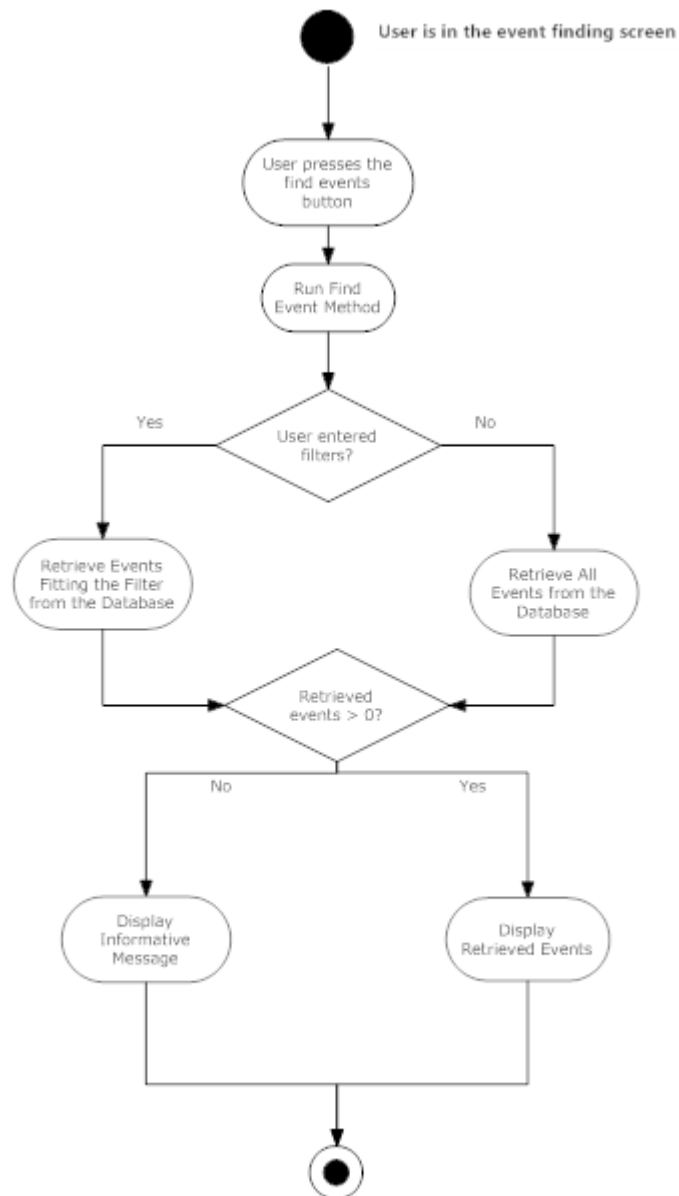


Figure 3.5. Event finding activity diagram

### 3.2.4 MongoDB schema

As it was clarified in **section 2.4.1** non-relational databases are dynamic, i.e. the former concept of defining the tables, relations between them using foreign keys... is no longer required. Nevertheless in this section, despite it can be modified at any time, the document and collections used for the storage of user and event data is hereunder explained.

```
{
  name: 'Alvaro',
  surname: 'De Saavedra',
  email: '100306200@alumnos.uc3m.es',
  password: 'aj921AI244AsHs5',
  events : [
    ObjectID ('AAAA')
  ]
}
```

Figure 3.6. MongoDB user collection document schema

```
{
  creator: 'Alvaro de Saavedra',
  description: ' ',
  attendants: [
    ObjectID ('F1K2'),
    ObjectID ('A4J9')
  ],
  location: {
    lat : '40.714232',
    lng : '-73.9612889'
  },
  date: '10/10/2017',
  time: '17:30',
  comments: [
    ObjectID ('A990'),
    ObjectID ('B898')
  ]
  image: {
    "_id" :
    {
      "$oid" : "4dc9511a14a7d017fee35746"
    },
    "chunkSize" : 262144 ,
    "length" : 22672 ,
    "md5" : "1462a6cfa27669af1d8d21c2d7dd1f8b" ,
    "filename" : "mkyong-java-image" ,
    "contentType" : null ,
    "uploadDate" :
    {
      "$date" : "2011-05-10T14:52:10Z"
    },
    "aliases" : null
  }
}
```

Figure 3.7. MongoDB event collection document schema

As it is observed, documents follow a JSON structure, being able to store arrays of data for the different fields of an instance. Regarding the previous statement, there are two possible ways of proceeding, being both illustrated in *Figures 3.5 – Figure 3.6*. Regarding the first, the events field contains an array of the events which that concrete user will attend or have created, instead of inserting every event in the array, the unique ObjectID corresponding to each

event is referenced in there. The drawback of this technique is that for getting the information about those events, a join between both tables is required, nevertheless when having more than a few elements the efficiency increase of this technique makes it worth the effort.

On the other hand *Figure 3.6* exemplifies the other technique, when having a limited array, as in the location case where only the latitude and longitude will be stored and there is no possibility for this to increase, the elements are referenced directly.

For summarize this section, the application will have three collections, the equivalent to SQL tables, in the Mongo database: users, storing the personal information and the user events; events, holding everything related to the event, attendants, comments, location, date...; also a third one for the comment storage, with the comment, user and event. The decisions that led to this schema were thoroughly thought for achieving the best application performance.

# CHAPTER 4

# IMPLEMENTATION

In the present chapter a deeper, more technical analysis of the application will be carried out, including screenshots and moreover, explaining every process that it follows with code examples and the necessary clarifications as libraries used.

This section will be structured as follows, several subsections regarding the different dimensions of the application, within each section the mentioned screenshots are found along with its process explanation.

## 4.1 Login and registration

In the current section the actions performed to manage the user access to the platform are expounded.

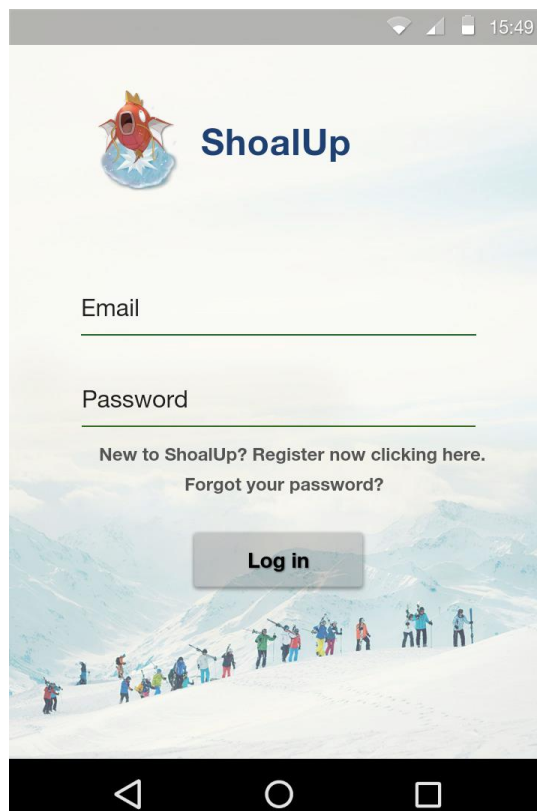


Figure 4.1. ShoalUp login screen

*Figure 4.1* illustrates the login screen of the application, where the credentials are directly asked to the user for speeding up the process of getting into the application.

There are three possible flows to be followed:

1. The user enters the credentials and presses the *Log in* button.
2. The user is new to the application and does not have an account, going through the register process by clicking the corresponding section.
3. The user has forgotten his password and wants to recover it.

### **Case 1**

In the scenario of a registered user trying to log to the application, the application will first check the SQLite database, i.e. the device's database, to see if the user's information is stored there. The reason of this first implementation is efficiency, in case it is the same user who connects from the same device, his credentials will be stored in that database, not being necessary the REST call, not only decreasing the process's time but also allowing the user to log into the application and have access to the available offline content such as his events.

It is important to remark that despite not a profound security measure plan has been established yet, it is to be discussed in the **Future work** section, some adjustments have been already implemented. It is the case of the password encryption, despite not being the most secure one, and having points to re-analyze, an **AES-256** password encryption has been implemented as the one learned during the Mobile Security subject. The method uses a **salt**, i.e. a string stored on the device to encrypt the password and store the encrypted one. For checking the validity of the credentials, the encrypted password is retrieved from the database, decrypted with the said salt string and compared with the one introduced by the user. The problem that comes with this method is that if the user tries to log from another device, the salt will be different and, even if the is introducing his password correctly, the system won't accept it as valid, this problem is contemplated in the Future work section also.

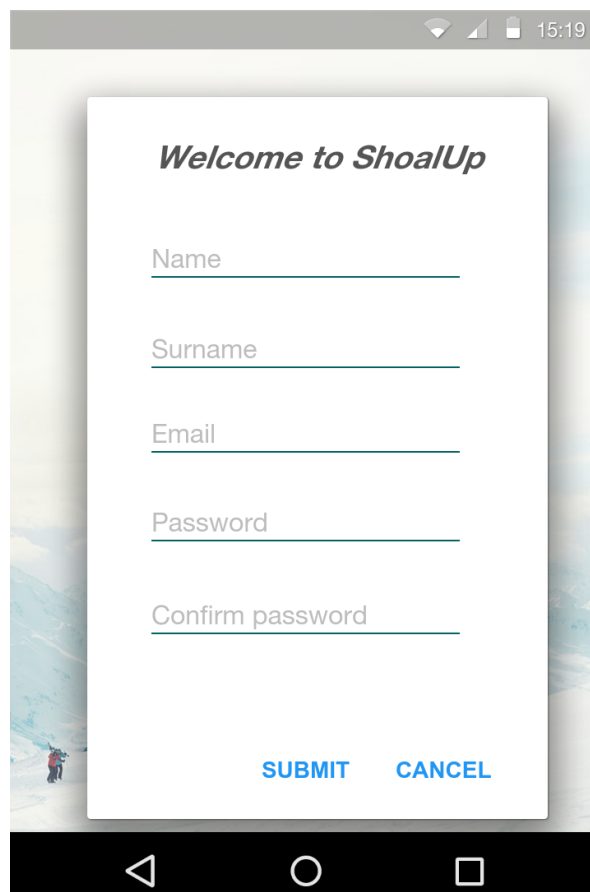
The next case scenario is the user logging from a new device, or it is his first connection to the application, his credentials won't be available on the SQLite database so it will be necessary to check the MongoDB database. To do this a REST call is invocated from the login process, this call will be addressed

by a method stored in the server, and perform the necessary actions, and send back the required information\*. Upon receiving the stored password, and in the same way it is done with the SQLite database, it will be decrypted and compared with the user's input, if passwords match, the user will be redirected to his home screen. If, on the contrary, both passwords do not match, an error message will be displayed and he will be asked to re-introduce his password.

*\*As many processes are using REST call for similar processes the REST functioning will be fully explained at the end of the chapter including examples for all the presented cases.*

### **Case 2**

If the user is new to the application, and he wants to register into the system, after pressing the register button, the correspondent form, *Figure 4.2*, will be displayed to be filled out.



*Figure 4.2. ShoalUp registration form*

*Figure 4.2* illustrates the mentioned form, as it can be observed it is a simple one, not asking for unnecessary data, keeping it simple will make the application more attractive to the user being this one of the main objectives to achieve during the development of this project. Some restrictions are applied to the form, the **Email** has to be in the correct format, i.e. *something@something.something*, and the field **Password** and **Confirm password** have to coincide in order to continue with the registration. When the form has been filled out the user will continue the process by submitting it, at this moment a check on the previous fields is run and, in case everything is correct, the form will be sent in JSON format to the server via REST. The server will perform the later explained actions and send back a code meaning that everything was done the way it was supposed or that an error occurred. In case everything is correct the user will be redirected to his home screen, otherwise, depending on the error code different error messages can be displayed, e.g. if the email already exists in the database the user will be notified and asked to introduce another one; or if there is an error regarding connectivity with either the server or the database, he will be asked to try again or wait and try later.

### Case 3

If the user can no longer remember his password, there is an option implemented for him to get a new one. As it is observed in *Figure 4.1*, the last option given is the one analyzed in this case, pressing it will pop a form similar to the registration one, being in this case single fielded, and the user will be asked to introduce the email associated to his account. Submitting the email will call a REST service that will check if the email exists on the database, in case it does, a new password will be randomly generated, updated in the database to substitute the lost one and sent to the introduced email address.



## 4.2. ShoalUp

This section encapsulates everything regarding event creation, finding, discovery and modification along with the user's profile modification.

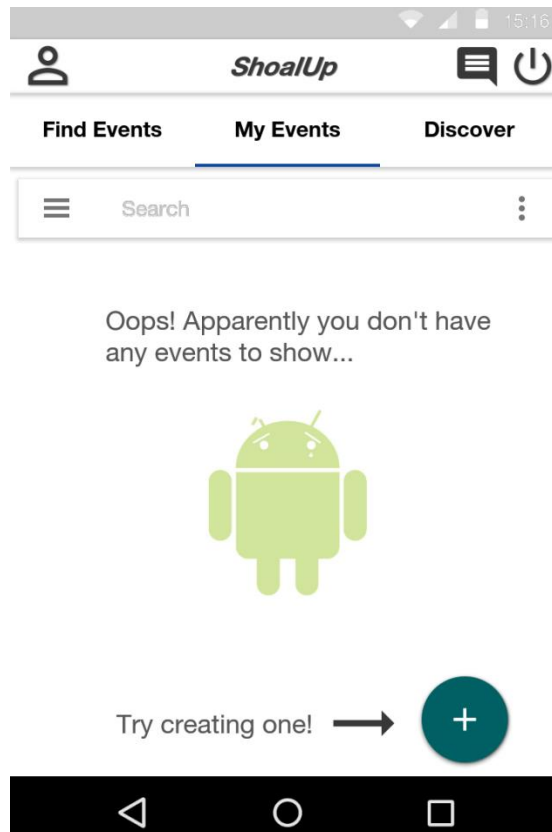


Figure 4.3. ShoalUp home screen

Figure 4.3 displays ShoalUp's home screen, the one the user is redirected after successfully logging in or registering. There are many aspects to be analyzed here.

The top menu bar encapsulates four elements: the application's title, the access to the user's profile situated on the top left, and the notifications and log out buttons on the top right corners. Being these elements self-explanatory there is no need for further commentaries but mentioning that despite the icon being already displayed, the notification system is not implemented yet, it is one of the branches of the future work and it will be explained in the correspondent section.

On the second level of the screen a tab bar to intuitively navigate through the application is found, as it is clearly observed it displays the current state of the application, i.e. the activity where the user is at every moment, this being a blue bar under the concrete tab. Changing tabs will change the executing activity changing the programmed methods as it is explained below.

Next element to be analyzed is the search bar. The filter feature that has been mentioned on previous chapters referred to this element. Again, and as the notification system, there are some future implementations to be developed here, at the moment it works as follows, the user introduce the desired word to filter for, e.g. an activity type (sports, party...), a username, a place, event title... associating a *onTextChangedListener()* to the filter bar, will detect when the user has introduced a text to filter for displaying again the event *listview*, but this time only the elements containing the filter text in any of its fields will be displayed.

Last, the event area and the event creation button. For the description of this section it is important to understand how it works, the different event activities, i.e. My Events, Find Events and Discover have a common *onCreate()* method, this method encapsulates a series of actions to be executed the moment the activity starts. The most essential one is the retrieving events one, once the activity has started it makes a call to the database via REST service sending different parameters depending on the activity. In this concrete case the user ID is sent, the server's method will query the database to find all the events where the sent user's ID appears either in the creator or the attendant fields. Sending the obtained information back to the client, it will be mapped and displayed as a *listview* being possible to sort it by location, activity number of attendants... In case there are no elements to display, the screen illustrated in *Figure 4.3* will be shown informing the user and encouraging him to create one, *Figure 4.4*.

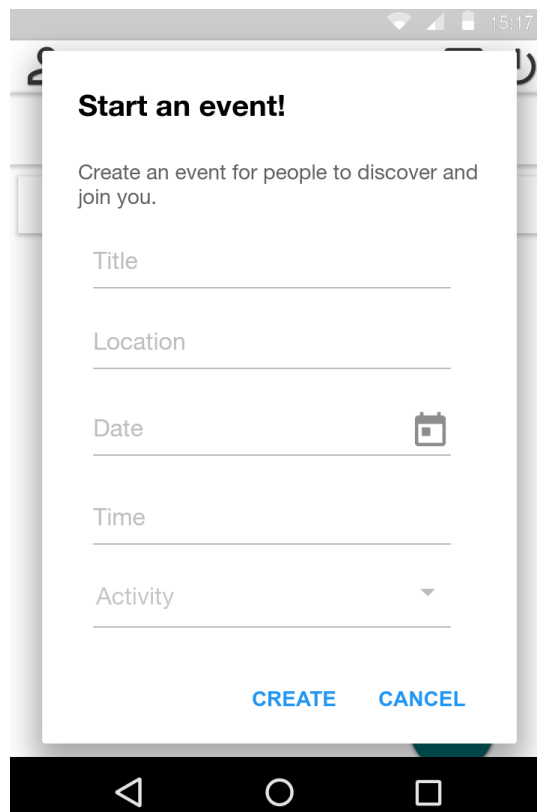


Figure 4.4. ShoalUp event creation

The above image illustrates the event creation form, the placeholders are self-descriptive leaving no room for the user to mistake the information to introduce. For the date and time completion, in order to avoid possible errors like introducing an invalid date or non-existing time, date and time pickers, *Figure 4.5*, have been implemented. The recently mentioned components consist on *DialogFragments* implementing a *TimePickerDialog.OnTimeSetListener* interface that will receive a callback when the user introduces the desired time. The same implementation is made for the *datepicker* but this class implementing the *DatePickerDialog.OnTimeSetListener*. The Activity element consists on a *spinner*, i.e. a list of pre-established values to be picked by the user, it contains the type of activity for which the event corresponds best, e.g. hang out, sports... The most complex behavior corresponds to the Location element, for a better implementation and in order to avoid as many errors as possible, e.g. people introducing a place that does not exist, this text element is connected to the Google Places API [24]. There are two possible ways of implementing this functionality, by adding a *PlaceAutocompleteFragment* or adding an intent to being an autocomplete activity, for this project the first one has been elected as the best fitting option. For its implementation it is required to add a new fragment connected to the API to the .xml of the activity and regarding

the activity, a *PlaceSelectionListener* will complete the location introduced by the user being a completely reliable direction. The rest of the fields, are plain text, it will store whatever the user introduces. Once the user has filled out the form and pressed the *Create* button, a check is run on the fields, making sure that none has been left blank and that the formats are correct, if there is an error the user gets notified and asked to correct it, otherwise it is sent to store in the database via REST service, and as in every REST call it will return a code meaning that the operation was successful or an error code for the application to interpret and display a message informing the user about the occurred error.

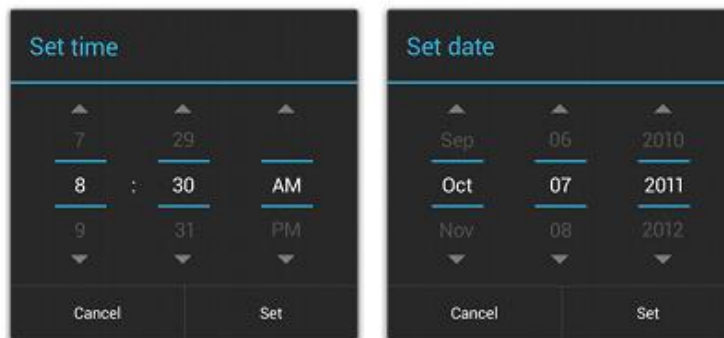


Figure 4.5. Date and time picker examples

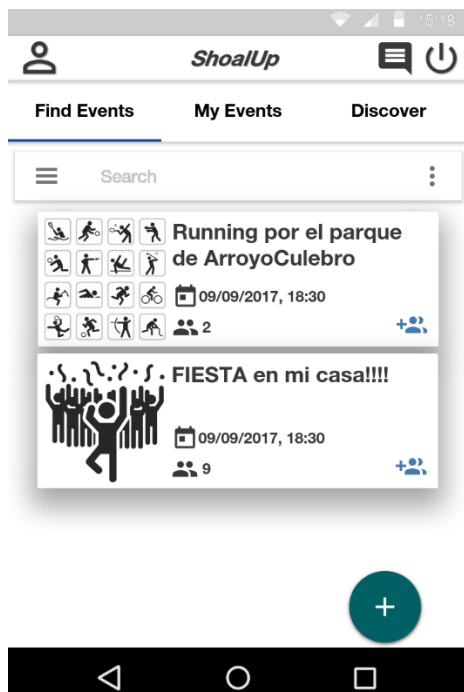


Figure 4.6. ShoalUp Find Events screen

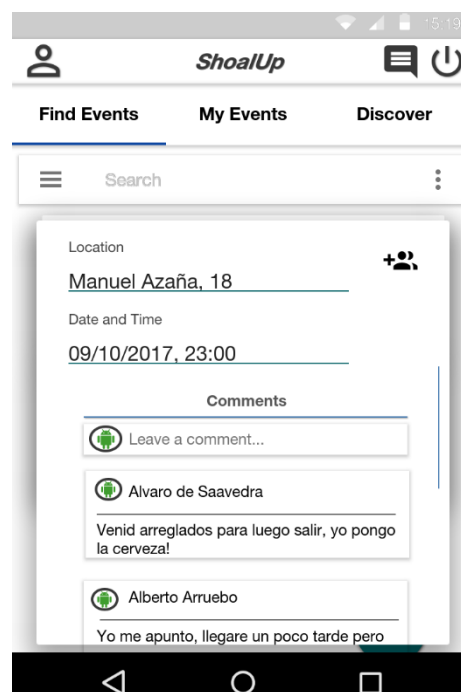


Figure 4.7. ShoalUp event complete information

Next analyzed section is Find event, *Figure 4.6 – Figure 4.7*. Maintaining the structure of *Figure 4.3*, the similarities between both screens are relatable, the whole structure is maintained, the only difference comes to the apparition of events; in this case *Figure 4.6* displays a set of events around the user's position. These events are obtained via REST service, as in the previous screen case, when the activity starts the *onCreate()* method is executed calling the server, in the previous case the user's ID was sent, now the position of the user obtained by the phone's GPS functionality, is sent to the server obtaining first the elements in the city and after filtering them by the distance from their coordinates to the sent ones. For obtaining the distance two Location objects are created, one being the position of the user and the other the location of the event, afterwards the *distanceTo()* function is used to obtain the distance in meters between both points and filter it by the radius. For this first implementation, the radius to filter for is fixed, established by the developer in 25 kilometers, in the future this parameter will be established by the user. After all this process the events meeting the conditions are sent to the client in JSON format, mapped to string and display them in a *listview* formatted as observed in *Figure 4.6*. The events consist on a representative image, being this static at the moment, it is planned to change as it is explained in the Future work section, and the basic information, the title, date and time of the event and number of attendants and the join button highlighted in blue. On the event of the user joining the event by pressing the mentioned button, a REST call will be executed informing the server with the users ID to be inserted in the database as an attendant and sent the code to the application to be interpreted.

*Figure 4.7* represents the application state when pressing an event, displaying its full information and the comment section. When inserting a comment a similar process to the inclusion of the user to the event is followed, when the user writes the comment and hits the Comment button, it is not observable as due to the full information about the event size a scrollable element was required showing just a part of it, *Figure 4.9* shows a different view of this element, the comment is sent via REST along with the event ID and the user ID to the server storing the three of them in the comment collection as a document and the comment ID being added to the array of comments in that event's document within the event collection. This complete view of the event also allows the user to join, being this icon dynamic changing depending on if the user is already a part of the event, allowing him to leave it or if the user is the creator becoming an edit function, this last feature is to be after explained.

Last, the Discover screen, *Figure 4.8*, allows the user to navigate through the map displaying the events enclosed in the focused zone of the map. This was the most complex process implemented using the Google Maps API [25]. Again and as in the previous screens, when the activity starts the *onCreate()*

method is executed sending the user's location to the server querying the database to obtain the events taking place in that concrete city and sending the list to the application. Once the application receives the response it obtains the coordinates of each element received and prints them as markers in the map using the Google Maps function `addMarker()`, *Figure 4.10* illustrates an example similar to the one used for this project's goal. Also using the Google Maps API the inclusion of the map within this screen was straightforward using the tutorial provided by them, as in the Google Place autocomplete location function, adding a fragment in the `.xml` associated to this activity was required in order to establish the attribute that will define the connection to Google's MapFragment section of its API. The `OnMapReadyCallback` interface has to be implemented by the activity and configure the callback instance to a MapFragment object, for managing the previous fragment, i.e. the map itself, the method `findFragmentById()` receiving as argument the resource id of the map fragment. The fragment's callback is managed by the `getMapAsync()` function.

Once the map is ready and the events markers have been placed, for displaying the full event information on clicking the marker an `onMarkerClickListener` is implemented to listen to the click events on each marker. If this happens, the `onMarkerClick(Marker)` function is executed passing the marker as an argument, retrieving its whole information and displaying it as it is shown on *Figure 4.9*.

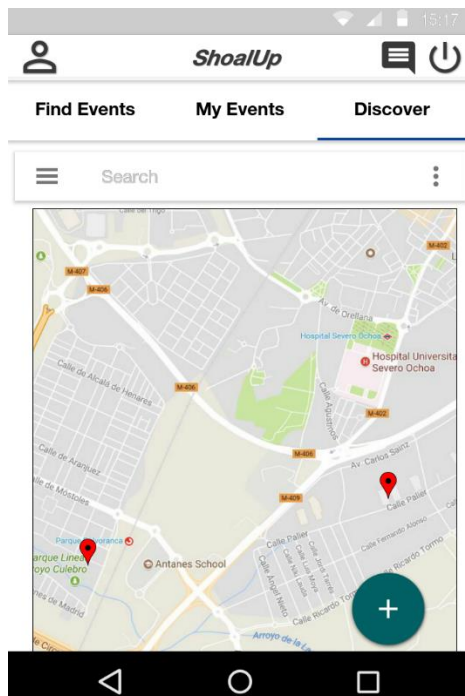


Figure 4.8. ShoalUp discover screen

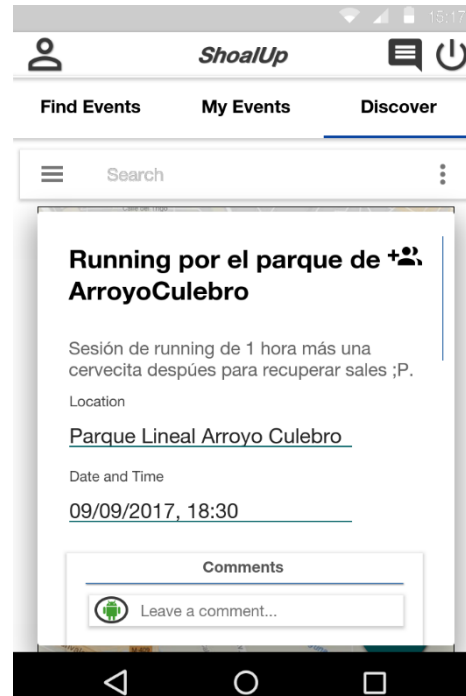


Figure 4.9. ShoalUp discover event full information

```

static final LatLng MELBOURNE = new LatLng(-37.81319, 144.96298);
Marker melbourne = mMap.addMarker(new MarkerOptions()
    .position(MELBOURNE)
    .title("Melbourne")
    .snippet("Population: 4,137,400"));

```

Figure 4.10. AddMarker() method example

Regarding profile information modification, *Figure 4.11*, accessed by the top left person icon, allows the user to change any of the details introduced when registering to the application. This modification procedure works in the same way for events.

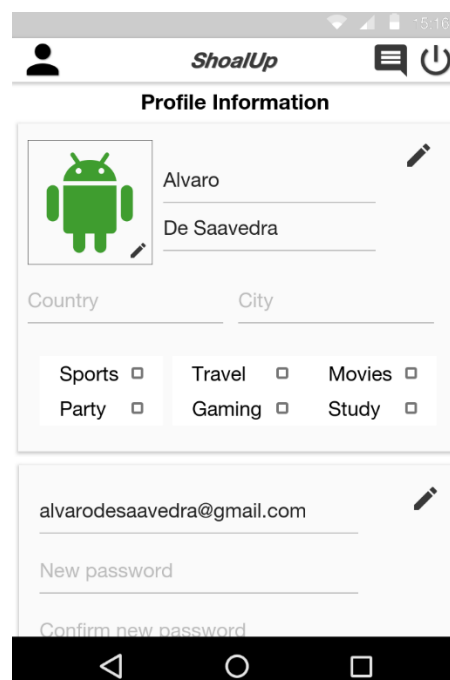


Figure 4.11. ShoalUp profile information

As it is observed there are extra fields like country, city or activity selection, as other interface options, its functionality hasn't been implemented yet as it was out of the scope of this project. Nevertheless, their inclusion is the result of the in-advance planning and in order to require less UI changes when implementing the upcoming features. The user is allowed to modify its details by clicking on the edit icon, i.e. the pencil, this will take him to the edit mode gathering the changed fields and when clicking on the Update button, again and as in prior cases, not all of the interface can be displayed due to its size, the application will send the profile's information via REST service to the server that will perform an update query against the database persisting the

modified fields, sending back the confirmation code to be interpreted by the application.

### ***REST services***

In this section as it was mentioned earlier in the chapter, an explanation of the REST implementation made to manage the server calls. REST has been implemented in order to reduce the operations to be performed at the user's device, increasing the efficiency of the application. For the REST calls the Volley library [26] has been used, being this library a high-level interface between the user and the thread management. Volley is petition focused, managing different petitions concurrently getting rid of the problem of having different asynchronous tasks at the same time and having to repeat the code. The HTTP calls are taken into a petition queue, from this queue the different calls are selected by the cache dispatcher checking if the results of each call exists already in cache, in that case that result will be used, otherwise the petition will be sent to the pendant connection queue waiting to be executed. Finally the network dispatcher is in charge of selecting those awaiting petitions and execute its respective HTTP transaction to the server.

The HTTP requests are directed against the server's URL where the REST services are stored, each service has a concrete URL, e.g. for user registration the URL ends with `/register/user`, this way the user register service is accessed. There are two possible methods regarding REST for data transmission, GET and POST. GET method data is embedded within the HTTP URL, e.g. user registration would be `/register/user/Alvaro/De Saavedra/100306200@alumnos.uc3m.es/123/123`. On the method's definition it is established that the first element after `/user` is the name, the second the surname and so on, the problem with the GET method comes when having, as in this case, a big set of parameters to transmit, and also that as for now, the encryption has not been implemented but for the password, in the case of an attack, elements like the users complete name or his email can be obtained. On the other hand, the POST method, the one used in this project, receives the data in the request body, denoted in the method as `@RequestBody`, the method will obtain the sent information, in this case a JSON object, and execute the programmed actions, e.g. insert the user in the database. When the application requires to receive data, as in the case of gathering the closer events, the method call will return a `@ResponseBody` object, this being a JSON containing the different elements to be returned.



## 4.3 Evaluation

All of the requirements established in section 3.1.1 following the use cases of section 3.1.2 have been put into test phase, checking the correct execution of each of the processes, making sure that the interactions between the client and the server worked as it was expected, the data persistence by means of both databases, SQLite and MongoDB, was correctly stored. Checking it by accessing the application from different devices.

Several users were created in order to ensure that the visualization of the events created by one of them was possible for the rest, testing the join and modification systems. Errors in retrieving the events associated for a concrete user were solved in this section.

Also the map functionality was where most of the problems aroused, leading to an extra effort for analyzing and correcting those. Nevertheless it was expected as working for the first time with an extern API can take harder-to-resolve problems than a code you are familiar with.

## CHAPTER 5

# SOCIOECONOMIC ENVIRONMENT

In the present chapter, divided in two subsections, the potential socioeconomic repercussions of the proposed solution are analyzed along with the project's budget.

### 5.1 Socioeconomic environment

Regarding the application objective, to bring people together to perform activities they like with people with similar taste, the first dimension that might be thought it can have impact on is both local and bigger businesses. If the application becomes popular, the number of events being organized anywhere will increase, affecting the utilized infrastructures for these to be celebrated, e.g. in the event of a football match meeting, a pitch will be required. Hence, potentially, the closest sports center, soccer pitch... where it can be celebrated will be booked becoming a potential social and economic impact.

The social impact might be observed in the inclusion of people that might not know people to execute his hobbies or preferred activities with into a group of alike people eliminating this barrier.

Further, the application becoming viral will exponentially increase the number of events celebrated, accordingly increasing the reservations, bookings and sales related to those events. One of the potential side-effects of this might be the increase of staff members in the affected businesses decreasing the country's unemployment rate. Likewise, the economy growth will potentially lead the owners of those businesses to invest into new ones, or expand their own to provide a better service to the customers.

Potentially, the most affected sectors would be the hospitality, sports, and party (bars and nightclubs) sectors. Having this in mind, the application would be monetized offering this entities to be publicized before their competitors, e.g. when a user looks for an Italian restaurant, those paying the most will appear first on the list. Also, as it is explained in the future work section, we will offer those business the possibility to cooperate with us

allowing our users to book directly from our app getting special discounts or offers for a pre-established percentage of each operation.

## 5.2 Budget of the project

Based on the project planning, section 1.4, in the present section a breakdown of the project's budget will be calculated. Only the actual costs will be describe below, mentioning free tools like NetBeans or Git is not required, henceforth, won't be reflected.

### Operational costs

Calculated using the formula:

$$\text{Cost} = ((\text{days} * \text{hours}) / \text{manDedication}) * \text{manCost}$$

- Duration in days: 160
- Daily hours invested: 4
- Man dedication per month: 131.25
- Man cost per month: 2000

$$\text{Cost} = 160 * 4 / 131.25 * 2000 = 9752.38$$

The total operational costs would ascend to 9752.38 €

### Technology costs

Both software and hardware means are taken into account in this section.

- Hardware:
  - Laptop: 1200€
  - Smartphone Xiaomi Redmi 4: 180€
  - AWS: 0€ \*

- Software:
  - All of the used software for this project’s development was open source, meaning that no monetary investment was required.

Table 5.1 shows the equipment amortization, being the imputable cost calculated following the next formula:

$$\frac{A}{B} * C * D$$

- **A:** number of months that the equipment has been used since the billing date.
- **B:** depreciation period (60 months)
- **C:** equipment cost (without VAT)
- **D:** equipment percentage of dedication to the project (usually 100%)

Description	Cost (€)	% project dedication	Dedication (months)	Depreciation period	Imputable cost
Laptop computer	1200	100	5	60	100
Smartphone Xiaomi Redmi 4	180	100	5	60	15
<b>TOTAL:</b>					<b>115</b>

Table 5.1. Project technology costs.

\*It is worth mentioning that AWS has a free plan with several limitations for projects like this, in order to be able to test connectivity and store a limited set of thing on the server. As the project advances, a different plan will be required.

<b>Project's total budget</b>	<b>Project's total budget (€)</b>
Operational costs	9752.38
Technological costs	115
Subcontracting	0
Indirect costs (20%)	1973.48
Total not including VAT	11840,856
Total including VAT	14327.44

*Table 5.2. Project's total budget*

*Table 5.2* reflects the project's total budget, it being a total of 14327.44€.

The application being free to download and advertisement free complicates the calculation of product amortization, as it will depend only in the number of user it has. This strategy has been proven the best in order to make the application more attractive to users and encourage their loyalty to it, i.e. ensuring that they use it on regular basis. This strategy has been followed by loads of successful startups such as Uber or Airbnb.

## CHAPTER 6

# REGULATORY FRAMEWORK

In this section the legal concerns that affect the proposed solution will be presented. Being sort of a social network where the users are required to enter their personal data, the most significant applicable legal treatment comes with the Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal (LOPD) [27]. This law's objective is to guarantee and protect the information and liberties of the people.

Due to the extensity of the LOPD, a summary containing the most concerning concepts with regard to the current project is presented hereunder.

- **Information rights:** the person requested to introduce his personal information has to be properly informed of the purpose, addressee, consequences that might entail of this data gathering along with his right of access, rectification, cancellation or opposition.
- **Consent of the affected person:** the person must agree unequivocally to its consent to provide his personal information.
- **Security, secrecy and communication:** the person or entity in charge of the acquired data, is responsible of his implement the required technical or organizational measures to assure its protection. Further, the aforesaid information is protected by the professional secrecy, being forbidden to communicate it to non-authorized entities.
- **User's age:** the application won't be allowed to make public any kind of information regarding users with less than fourteen years of age. An exception to the previous statement would occurs if he/she has the express consent of the parents or legal tutor.
- **Identity fraud:** constituting a crime, if an account is suspected to be false, it would immediately be banned and denounced to the competing authorities.

Everything privacy and security related will be clearly defined under the section named in the same way before the application release.

## CHAPTER 7

# CONCLUSIONS AND FUTURE WORK

### 7.1 Conclusions

The objective of the project was to develop a simple, intuitive interfaced application working with REST services for data transmission between the client and the server, being this last one in charge of managing the database accesses, persisting or retrieving data for the application.

The application solved the existing gap between the community creating applications like MeetUp and similar applications like TimPik being activity specific, ShoalUp's solution allows its users to avoid the effort of registering amongst different apps, having every kind of event available contained in a single application.

The application UI follows as close as possible the ten rules for a good UX design, being possible to always cancel actions, e.g. in user registration, event creation or modification... having a consistent design, robust trying to reduce as much as possible the potential errors and, in the case of an error occurrence, information different information messages are prepared to guide the user through it. As well as elements reflecting the state of the application helping the user to acknowledge where he is, e.g. the blue line under the tab bar displaying if he is in the "My events" section, the "Find events" or the "Discover".

Further, the whole system is prepared for future improvements and implementations, as seen in chapter 4 with some elements of the interface, being these functionalities explained in section 7.2.

For the conclusion of the section it is to say that the objectives aimed to be achieved at the beginning of this project were successfully completed, experiencing the whole process of developing a software project from its design to its testing phase. The most complex part was the inclusion of the Google Maps and Google Places APIs into the system; nevertheless it was worth the effort regarding the obtained results.

## 7.2 Future work

In this section future implementations and improvements to be developed are explained hereunder divided into two sections.

### 7.2.1 Near future work

The next upcoming implementations would consist on the ones mentioned in the previous chapter. Regarding the events, a notification system is to be implemented for the attendants of each event to be notified whenever there is an update, e.g. a new comment or attendant or whether it has changed location time or if it has been deleted.

Also, the scalability of the event finding and discovery is one of the closest goals to achieve, right now it obtains the events of the current city for efficiency matters, the next step is to be able to efficiently load the events within a whole country. This improvement will require a larger implementation and a higher computing power regarding the server. However, a viability study will be carried out, evaluating how this improvement might work on different devices having different computational power.

Regarding personalization, as observed in *Figure 4.11*, the user will be offered to establish a preference system based on his country and city along with the kind of events he is interested in the most. The expected result is a more attractive application displaying the best fitting events for each user. Also the radius to search for, mentioned in chapter 4, is an improvement to be soon implemented giving the user a higher degree of freedom.

The last implementation planned is security wise. A deeper analysis of the required measures to be taken is to be carried out, having sensible user information will require measures like database encryption or an authentication system based on tokens.

### 7.2.2 Further future work

Once the application has been proved complete and secure, the implementations aiming monetary profit will be developed. The procedure to be followed has not yet been decided. Nevertheless, the objective is to allow the users to book the infrastructure or resources to perform the event activity



from our application, e.g. when planning a football match, the attendants will be able to reserve the pitch in advance from the application obtaining a discount or an offer by doing it this way. This will require a further study as it becomes more complex, having to negotiate with local business, agencies...

# BIBLIOGRAPHY

- [1] S. Duggirala, "10 Usability Heuristics with Examples – prototypr", prototypr, 2016. [Online]. Available: <https://blog.prototypr.io/10-usability-heuristics-with-examples-4a81ada920c>.
- [2] "What is the Software Development Life Cycle (SDLC)?", Airbrake Blog, 2013. [Online]. Available: <https://airbrake.io/blog/sdlc/what-is-the-software-development-life-cycle>.
- [3] "What is a Gantt Chart? Gantt Chart Software, Information, and History", Gantt.com. [Online]. Available: <http://www.gantt.com/>.
- [4] Meetup.com. [Online]. Available: <https://www.meetup.com/es-ES/> [Accessed: 11- Sep- 2017].
- [5] "Facebook Events", Facebook Events. [Online]. Available: <https://events.fb.com/> [Accessed: 11- Sep- 2017].
- [6] "Eventbrite", Eventbrite. [Online]. Available: <https://www.eventbrite.es/> [Accessed: 11- Sep- 2017].
- [7] "What things to do, tourism, attractions and tours in New York City - Fever", Feverup.com. [Online]. Available: <https://feverup.com/> [Accessed: 11- Sep- 2017].
- [8] "Timpik: Practica deporte en tu ciudad", TIMPIK: Practica deporte en tu ciudad. [Online]. Available: <http://www.timpik.com/> [Accessed: 11- Sep- 2017].
- [9] P. Local, "Party with a Local | App Connecting People Who Want to Party", Party with A Local. [Online]. Available: <http://partywithalocal.com/> [Accessed: 11- Sep- 2017].
- [10] "Mobile | Couchsurfing", Couchsurfing.com. [Online]. Available: <http://www.couchsurfing.com/about/mobile/> [Accessed: 11- Sep- 2017].
- [11] Ditrendia, "Informe Mobile en España y en el Mundo 2016", 2017.
- [12] P. Deitel, H. Deitel, A. Wald and P. Deitel, Android 6 for Programmers: An App-Driven Approach. 2015.
- [13] C. Keur and A. Hillegass, iOS Programming: The Big Nerd Ranch Guide. 2016.
- [14] E. Maxwell, "MVC vs. MVP vs. MVVM on Android", Academy.realm.io, 2017. [Online]. Available: <https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/>.

- [15] [B. Moschetti, "MongoDB vs SQL: Day 1-2", MongoDB, 2014. [Online]. Available: <https://www.mongodb.com/blog/post/mongodb-vs-sql-day-1-2>.
- [16] Top 5 Considerations When Evaluating NoSQL Databases. MongoGB, 2017. [Online] Available at: [https://webassets.mongodb.com/\\_com\\_assets/collateral/10gen\\_Top\\_5\\_NoSQL\\_Considerations.pdf](https://webassets.mongodb.com/_com_assets/collateral/10gen_Top_5_NoSQL_Considerations.pdf)
- [17] H. Schildt, Java: The Complete Reference, 9th ed. 2014.
- [18] B. Syed, Beginning Node.js, 1st ed. 2014.
- [19] L. Ullman, PHP for the Web: Visual QuickStart Guide, 5th ed. 2016.
- [20] "Building REST services with Spring", Spring.io. [Online]. Available: <https://spring.io/guides/tutorials/bookmarks/> [Accessed: 19- Sep- 2017].
- [21] D. Pilone, UML 2.0 pocket reference, 1st ed. 2006.
- [22] "SQLite Home Page", Sqlite.org. [Online]. Available: <https://www.sqlite.org/> [Accessed: 20- Sep- 2017].
- [23] "¿Qué es AWS? – Amazon Web Services", Amazon Web Services, Inc.. [Online]. Available: <https://aws.amazon.com/es/what-is-aws/> [Accessed: 20- Sep- 2017].
- [24] "Getting Started | Google Places API for Android | Google Developers", Google Developers. [Online]. Available: <https://developers.google.com/places/android-api/start> [Accessed: 15- Sep- 2017].
- [25] "Getting Started | Google Maps Android API | Google Developers", Google Developers. [Online]. Available: <https://developers.google.com/maps/documentation/android-api/start> [Accessed: 15- Sep- 2017].
- [26] "Transmitting Network Data Using Volley | Android Developers", Developer.android.com. [Online]. Available: <https://developer.android.com/training/volley/index.html> [Accessed: 15- Sep- 2017].
- [27] Ley 5/2015, de 27 de abril, de fomento de la financiación empresarial. BOE-A-2015-4607