Generic Crowdsourcing System for Dynamic Environmental Surveys

Miguel Santos Horta Antunes Leitão

Thesis to obtain the Master of Science Degree in 

Electrical and Computer Engineering

Supervisor(s):  Prof. Dr. João Nuno de Oliveira e Silva  
Dra. Ana Catarina Luz

Examination Committee
Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques  
Supervisor: Prof. Dr. João Nuno de Oliveira e Silva  
Member of the Committee: Prof. Pedro Abílio Duarte de Medeiros

January 2021
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
"There is no secret ingredient. It’s just you." [1]
Acknowledgments

First I would like to acknowledge and thank my two thesis supervisors, Prof. Dr. João Nuno Silva for allowing me, through courses like Systems Architecture and Internet Based Systems Architecture, to realize my enthusiasms for system’s architecture and web development and for the opportunity to extend my knowledge in this area by working on this project and Dra. Ana Catarina Luz for the tireless support over the entire duration of this project and for giving a real-life application and purpose to my work.

A special thank is due to my friends and family that through their support and patience helped me carry my academic path to its conclusion. It is not an easy task to write and dissertation and it becomes an even harder thing to do during a pandemic and I am aware how tiresome might have been putting up with me in more than one occasion for that and all those who did it, I am most grateful. I also would like to acknowledge Guilherme Eugénio that collaborated on this projected with me and shared this burden.

Um bem-haja a todos!
Resumo

Ecologia urbana é uma disciplina que envolve diferentes áreas do conhecimento, e.g., ecologia, biologia, geografia, sociologia. Há, no entanto, uma falta de documentação sobre Espaços Verdes Urbanos (EVU) e como eles influenciam as vidas da população, bem como de um método eficiente para recolher tal informação. Esta informação tem um enorme valor para estudos na área científica da biologia e ecologia a até mesmo para o planeamento urbano. Este relatório propõe uma solução para este problema através do desenvolvimento de um sistema genérico para a criação de aplicações de crowdsourcing. Através desta framework será possível criar uma aplicação móvel para recolher os dados de crowdsourcing, uma consola web para analisar esses mesmos dados e um servidor capaz de responder a pedidos e implementar as funcionalidades de ambos os clientes. As aplicações criadas podem facilmente ser adaptadas para resolver qualquer problema do quotidiano que possa surgir. Em particular, o sistema permitirá o desenvolvimento de uma aplicação capaz de recolher extensivamente informação detalhada sobre EVU, questionando dinamicamente os seus visitantes sobre a sua experiência enquanto nesses espaços. Esta aplicação será capaz de apresentar diferentes questionários para diferentes estados do sistema, recolher dados geográficos e relativos a actividade física a partir de sensores externos e ainda disponibilizar mecanismos para validação de dados. Todas estas funcionalidades e qualquer outra que seja necessária podem ser integradas com a parte genérica do sistema sem grande esforço a nível de programação.

Palavras-chave: crowdsourcing, espaços verdes urbanos, questionários, bem-estar, planeamento
Abstract

Urban ecology is a subject that involves different scientific areas, e.g., ecology, biology, geography and sociology. There is, however, a lack of reliable documentation about Urban Green Spaces (UGS) and how they affect the lives of the population, as well as no efficient way to collect it. This information has enormous value for biological and ecological studies and even for urban planning. This report proposes a solution for this problem through the development of a generic system to create crowdsourcing applications. With this framework it will be possible to build a mobile application to collect crowdsourcing data, a web console to analyse data and a server to answer the calls and implement the features from both clients. The created applications will be easily adaptable to solve any real-life problem that may arise. The system will allow the development of an application able to extensively gather detailed information about UGS by dynamically surveying its visitors about their experience while in those green spaces. This application will be able to deploy different surveys for different states of the system, collect fitness and geographical data from external sensors and provide data validation mechanisms. All these functionalities and any other can be integrated with the generic part of the framework with little programming effort.

**Keywords:** crowdsourcing, urban green spaces, surveys, well-being, planning
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>vii</td>
</tr>
<tr>
<td>Resumo</td>
<td>ix</td>
</tr>
<tr>
<td>Abstract</td>
<td>xi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xvii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xix</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>xxi</td>
</tr>
<tr>
<td>Glossary</td>
<td>1</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Motivation                                      | 1    |
1.2 Problem                                        | 2    |
1.3 Proposed Solution                               | 3    |
1.4 Thesis Outline                                 | 3    |

## 2 Background

2.1 Urban Green Spaces                            | 5    |
2.1.1 Visitors Profiling                          | 6    |
2.1.2 Relationship with UGS                      | 7    |
2.1.3 Visits Motivation                           | 8    |
2.1.4 UGS Related Research Issues                | 10   |
2.2 Survey Systems                                | 11   |
2.2.1 Crowdsourcing                               | 11   |
2.2.2 Online Surveying                            | 12   |
2.2.3 Crowdsourcing Taxonomy                      | 12   |
2.2.4 Smartphone as a Surveying Tool             | 13   |
2.2.5 Analysis of Existing Technology             | 15   |
2.3 UGS Surveys                                   | 17   |

## 3 Generic Crowdsourcing System

3.1 Users                                         | 19   |
3.1.1 Administrator                               | 19   |
3.1.2 Researcher                                  | 20   |
3.1.3 Normal User .................................................. 20
3.2 Functionalities ................................................... 20
  3.2.1 System Flow ............................................... 20
  3.2.2 Application Flow ........................................ 21
3.3 Requirements .................................................. 22
  3.3.1 Functional Requirements ................................. 22
  3.3.2 Non-Functional Requirements ........................... 25
3.4 Architecture ................................................... 26
  3.4.1 Context Level ............................................. 26
  3.4.2 Container Level .......................................... 27
  3.4.3 Component Level ......................................... 28
3.5 Implementation ............................................... 32
  3.5.1 Server’s API ............................................. 33
  3.5.2 Data Model ................................................ 35
  3.5.3 Technologies .............................................. 36
  3.5.4 Surveys .................................................... 38
  3.5.5 Data Validation .......................................... 40
  3.5.6 Sensors .................................................... 41
  3.5.7 Activation Notifications ................................ 42
  3.5.8 User Authentication ...................................... 43
  3.5.9 Multi-Language .......................................... 45
  3.5.10 Tutorial Presentation ................................... 46
  3.5.11 User Profile ............................................. 47
3.6 Extension ..................................................... 47
3.7 Security ....................................................... 50
3.8 Forms Tool ................................................... 51
3.9 Usage ........................................................ 54
  3.9.1 Server Usage ............................................. 55
  3.9.2 Mobile Usage ............................................ 56
  3.9.3 Client Usage ............................................. 57
4 Urban Green Spaces Application ................................. 59
  4.1 Specific Requirements ....................................... 60
  4.2 Dynamic Surveys ............................................ 60
  4.3 Area Activation Notifications ............................... 63
  4.4 Device and Web Sensor Integration .......................... 65
  4.5 Researcher Data Validation .................................. 66
List of Tables

2.1 Survey applications competitive analysis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
List of Figures

2.1 Taxonomy of crowdsourcing by Yuen, King and Leung [43] ................................. 13
2.2 Adults who report owning a ... [46] .............................................................. 14
2.3 List of activities conducted using a smartphone and corresponding percentage of smartphone owners who reported them [48] .............................................................. 15

3.1 System Flow .......................................................... 21
3.2 Application Flow ....................................................... 22
3.3 Context Level Architecture .................................................. 27
3.4 Container Level Architecture ...................................................... 27
3.5 Component Level Architecture .................................................. 29
3.6 Component Level Architecture .................................................. 31
3.7 Component Level Architecture .................................................. 32
3.8 Data Model or Code Level Architecture ........................................... 36
3.9 Dynamic and static surveys flow ............................................... 40
3.10 Administrator web console ...................................................... 40
3.11 Server’s extension usage ...................................................... 48
3.12 Mobile application’s extension usage ............................................. 49
3.13 Extensible components in the different system parts ......................... 55

4.1 Mobile application screens ...................................................... 59
4.2 Dynamic surveys basic server flow ............................................... 61
4.3 Dynamic surveys basic server flow ............................................... 62
4.4 Researcher web console ...................................................... 66
## Nomenclature

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface.</td>
</tr>
<tr>
<td>APK</td>
<td>Android Package.</td>
</tr>
<tr>
<td>ASQ</td>
<td>After Scenario Questionnaire.</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface.</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model.</td>
</tr>
<tr>
<td>ES</td>
<td>Ecossystem Services.</td>
</tr>
<tr>
<td>EU</td>
<td>European Union.</td>
</tr>
<tr>
<td>FCUL</td>
<td>Faculdade de Ciências da Universidade de Lisboa.</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol.</td>
</tr>
<tr>
<td>iOS</td>
<td>iPhone Operating System.</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation.</td>
</tr>
<tr>
<td>NPM</td>
<td>Node Package Manager.</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System.</td>
</tr>
<tr>
<td>QR</td>
<td>Quick Response.</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer.</td>
</tr>
<tr>
<td>SUS</td>
<td>System Usability Scale.</td>
</tr>
<tr>
<td>UGI</td>
<td>Urban Green Infrastructure.</td>
</tr>
<tr>
<td>UGS</td>
<td>Urban Green Spaces.</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language.</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td>Uniform Resource Locator.</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>UWP</strong></td>
<td>Universal Windows Platform.</td>
</tr>
<tr>
<td><strong>VGI</strong></td>
<td>Volunteer Geographic Information.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Nowadays, climate change and its effects on our planet and society are a main concern for most people and one of the most debated topics worldwide. It is unquestionable the role played by man as primary, if not sole responsible for these environmental changes and social transformation on a global scale.

Urban areas hold the greatest challenges and responsibilities since that is where most of the world’s population is found and predictions suggest that urban population will continue to raise [2]. These challenges encompass improving people’s health and well-being while dealing with climate change adaptation and mitigation. This improvement can be made directly correcting behaviours that cause temperature changes that has proven not to be a simple task but it can also be made indirectly by encouraging people to contact with nature and therefore acknowledge it importance and change their perceptions and habits. One of the possible pathways to overcome some of such challenges, is by improving the Urban Green Infrastructure (UGI) and the relationship between humans and nature. Specially in the urban areas this relationship is weaker, that can be altered through a better planning of green spaces. Green spaces can become more welcoming and engaging, by improving multiple functionalities that spaces offer to urban dwellers. This would strengthen their relationship with nature and consequently contribute positively to their well-being.

1.1 Motivation

Urban Green Spaces (UGS) are public spaces that have elements like trees or some other type of vegetation and possibly water bodies, like lakes, water streams or fountains. These spaces are an important part of any city or big urban area. Unlike in small towns and rural areas, urban areas’ residents do not have easy access nor frequent contact with nature and so it is important to provide them with this indispensable connection.

The existence and easy access to appealing UGS encourage urban dwellers and attendants to visit them. Besides its ecological benefits, like e.g., temperature regulation, water retention, biodiversity protection, UGS are known to be spaces dedicated to leisure, and amongst the most common uses
for these spaces are relaxation and exercising. The mentioned activities are ways to keep and improve health both at the physical (exercise) and mental (relaxation) levels. Apart from the benefits above, these spaces also provide the opportunity for social gatherings, that strengthen visitors’ sense of belonging and community, also contributing positively to their well-being [3]. Thus, people’s perception of well-being can be influenced by the layout and infrastructures found in the UGS they visit.

1.2 Problem

There is a lack of both full mapping of UGS in most mapping platforms and an understanding on people’s motivations and preferences towards UGS. These problems have been noted and registered by researchers and practitioners (e.g., planners and decision-makers), who would benefit from this missing information either purely for research purposes or to apply it in other various fields such as urban planning.

It is important to promote and facilitate the attendance to UGS through the collection of current data about urban dwellers’ uses and preference of green spaces. The deficiency in this type of data was noted and exposed by authors in several pieces of work related to this subject and can be divided in three main categories: Visitors Profiling, Relationship with UGS and Visits Motivation. This situation and how it is handled in the different studies are explored in the next sections. This issue contributes to proving that researchers and practitioners are in need of a more complete database, not only with geographic data about UGS that could be used for research and assisting in future urban planning, but also social data about the visitors themselves, all that could be used to improve UGS and promote the attendance to parks and other green spaces. More important than to solve the immediate problem, which is this shortage, is to think about a long term and broad solution because the real problem might not be the actual missing of information but the nonexistence of an efficient mechanism to collect it.

There are multiple crowdsourcing applications already developed. These apps focus on solving various problems in a wide range of subjects. There will always be new problems that will require different type of data collection and processing also requiring a different application. The existing applications allow the creation of simple surveys and the gathering of their respective answers for analysis by the survey creator or someone else with the appropriate permissions. What these applications do not encompass is the possibility of extension, the possibility of anyone with little programming knowledge to implement their own version of the existing features or even create new ones in order to help solving a different and specific problem in another context. Also not possible at the moment is information collection from external sources, sometimes the classic survey format may not be enough and sensor data could be used to complement users’ input and enrich the overall drawn conclusions. Lastly, there are no data manipulation mechanisms available, the purpose of collecting data is so that it can be studied and so it should be allowed the mechanisms to handle the data the as the people studying it see fit.
1.3 Proposed Solution

In this work it is proposed the development of a generic system that allows the creation of crowdsourcing applications that can then be specified into a wide range of themes. The system will provide the traditional functionalities found in this kind of applications but also add some additional features not currently available in other tools. Features like (a) a better definition of the gathered data so that there is full control over what information is collected in order to better fit the needs of the system’s creator, (b) a larger amount of data sources or sensors both internal and external capable of getting data not obtainable through user input and (c) data cleaning mechanisms through which data can be handle in the most relevant way to fulfill the system’s purpose, this will improve the results of the applications while allowing to apply to other areas the useful concept and tool that is crowdsourcing.

The real innovation surrounding this system is not the new features themselves but the possibility of their existence, which means that new features can be added to the ones mentioned previously. With little effort these new features can be integrated with the rest of the system making it more useful.

Starting from the provided generic system, anyone with basic programming experience and knowledge will be able, through the creation of simple extension files, to make an application with the common crowdsourcing features but also with extra functionalities dedicated to solving their specific situations and problems.

Using this generic system it will be possible to create a crowdsourcing application to solve the lack of information about UGS issues. This application will take advantage of the framework’s extensibility to incorporate modules with the functionalities that will allow the dynamic collection of data from the urban dwellers that visit green spaces.

1.4 Thesis Outline

The rest of the thesis is organised as follows. In the chapter 2 it will be presented some related work that approaches the theme of this thesis, these will be divided into work about Urban Green Spaces in the section 2.1 and Survey Systems in the section 2.2. The UGS related work will relate to the issues mentioned before while the surveying part of this chapter will point out import aspects of surveying and crowdsourcing system that were essential to the design of and development of the proposed solution.

Chapter 3 describes the information about the solution itself. Here are explored the system users, flows, requirements and components through its architecture in the sections 3.1, 3.2, 3.3 and 3.4, respectively, but also a detailed explanation of how were implemented its main features in the section 3.5 and how can they be used in the section 3.9.

Chapter 4 describes an application of the generic system, the Urban Green Spaces application and how its features demonstrate the correct operation of the system.

In the chapter 5 there are presented other examples of the system usage, tests performed and how the system requirements were met.

Finally, in the chapter 6 the main achievements of this project are described, specially in the sec-
tion 6.1, followed by a brief discussion about some challenges that arose during the project development in the section 6.2, concluding with proposals for future work in the section 6.3.

The document also contains an appendix A that contains the detailed results of the usability tests performed on the Urban Green Spaces application.
Chapter 2

Background

In this chapter the two main concepts behind this report are being explored. The first concept is Urban Green Spaces and all the challenges associated with them that motivated this project. The second concept is Survey Applications, more specifically, already existing technologies and features that they lack and how they could be improved.

2.1 Urban Green Spaces

Urban Green Spaces is in a sense a somewhat vague term but a good definition was made by Plunz et al. [4] when defined UGS as components of the Urban Green Infrastructure, which is the network of interconnected multi-functional green spaces which if strategically planned and managed it can deliver multiple benefits - ecological, social, and economic - for cities and its urban-dwellers [5]. The UGI comprises private and public spaces such as parks, urban forests, allotment gardens, green roofs, derelict lands, street trees, among others, which are accessible for leisure.

There is several literature that shows the links between UGS and their visitors’ well-being and health along with the communities where they are located. These are the most natural and semi-natural structures existing in the urban areas and so where most of the biodiversity can be found. Biodiversity and the nature itself provide a variety of services - Ecossystem Services (ES) - which are benefits nature provide to people, such as temperature regulation, air quality improvement, or even recreation, relaxation, among others. These ES benefit from the UGS planning and management and nature-base solutions promoted by the European Union (EU) directives.

Even with the need for further research about the correlation between UGS and well-being of their visitors and their communities, it is possible to infer, from work already developed in this field, that UGS affect positively the stress level and mental health, self-esteem and sense of community of those who visit them. They also promote and stimulate social interaction, through a sense of belonging and the creation or strengthening of neighbourhood ties and even assisting in ethnic inclusion.

In the following Sections some of problems related with UGS that have being pointed out throughout the years will be highlighted and correlated with studies already performed in this area and that explored
those same problems. These problems touch two main themes that are interlinked. Firstly, there is the matter of the association made between UGS and the well-being of their visitor and the surrounding population. Secondly, there is the importance of studying spaces layouts, visitors and effects in the surrounding environment in order to use those studies as a positive contribution to the UGI effective planning and management. A well planned and rich UGI can encourage a shift to urban soft modes of mobility [6], reducing this way the pollution caused by means of transport, noise and traffic stress [7].

One important step in this project will be assessing the uses and perception about the UGS by their visitors. This assessment will be done via questions asked to the green spaces’ visitors. Each of these questions were designed to get the information essential to achieving this project’s goal. This is supported by several research papers and case studies in this area that emphasize the importance of the data gathered to improve the understanding of these questions. So, after relating problems to their respective studies it will be clearer the reason for each of the proposed questions.

In the next sections will be presented the major issues related to UGS found in work related with this subject, these will be correlated with series of studies where these problems where encountered. These issues can be divided into three major themes: Visitors Profiling, Relationship with UGS and Visits Motivation. Besides describing the problems and referring the studies that faced them, it will also be described how the authors handled and solved these struggles.

2.1.1 Visitors Profiling

One of the first problems when doing any kind of survey is understanding the meaning and how to interpret each answer, which can be done by getting some context and information about the respondent. Studies show that the visitors’ behaviour is influenced by their background and profile. So, for different people with different profiles, the same question can have very different answers or even the same answer but with a different meaning. So, because the value of the surveys rests not only on the data itself but mainly in the information extracted from it, the gathering of personal attributes of the respondent can be useful for analysis and interpretation by crossing it with the uses and perception data, among other (e.g., structure of green spaces) in later stages of the study where the survey is inserted.

As mentioned previously, the lack of profiling details of respondents can limit data analysis, and that is why, many authors mentioned in their literature, the measures implemented in order to avoid the shortage of this information.

For instance, Liu and others[8] conducted a study with the objective of assessing the perception of sound and noises inside Urban Green Spaces by users during their visits. The questionnaire designed for this study was divided in two parts, the first one was dedicated to gathering social, demographic, and behavioural information about the interviewees. Specially, this information included variables such as (1) in age, where the people were divided into six different age classes ($\leq 24$, $25 – 30$, $31 – 40$, $41 – 50$, $51 – 59$, $\geq 60$), (2) educational background, where it was considered three main groups (primary school, secondary school, and higher), (3) residential status (community resident, local resident, tourist) and the final personal characteristic collected was (4) the person’s occupation, i.e., students, working...
people and others (retired, unemployed and full-time housewife).

In 2018, Romagosa et al. [9] also studied the perceptions of the UGS visitors, but this time regarding the motivation and results of their visits in terms of physical health. Age, residential area and gender of the visitors was some of the collected data. This allowed valuable conclusions to be drawn, like, for older visitors and visitors who lived closer to the UGS, they were increasingly motivated by health benefits, whilst women were the ones who appreciated most the positive impact of the visit.

Distance to the nearest UGS is another profiling information commonly collected in this type of surveys. This variable has been shown to be linked with visitors motivations to visit a UGS and positive perceptions towards nature in general as pointed by Shan [10] and Luz et al. [11]. This was even proved to be a key factor in defining the relationship between UGS and the users by Coombes, Jones & Hillsdon [12], Godbey & Mowen [13], Godbey [14] and Mytton, Townsend, Rutter & Foster [15].

Lee et al. [16], Sanesi and Chiarello [17] and Yuen [18] studied how those motivations change for each gender. Moreover, Chiesura [19], Matsuoka and Kaplan [20] and Romagosa [9] explored how age differences affected the motivations behind visits to UGS. Although many of the conclusions may be similar among different studies it is always important to gather this type of data because it can be context or cultural dependent. There are some characteristics of a visitor that may affect other features, establishing sometimes connections not so obvious with the collected data that might completely alter the result of the work. For example, all the authors mentioned before concluded that younger people visits to UGS were more motivated to conduct physical exercise, yet by changing the location of the study Shan [10] has observed the exact opposite pattern.

### 2.1.2 Relationship with UGS

When surveying about any subject it is necessary to get some context, not only about the person answering the questions but also about the knowledge and relationship of respondents with the subject as well as about the biophysical structure of the environment. In this specific case, we refer to the relationship between respondents and UGS. This provides context and valuable information to help in the data analysis and interpretation. Here is where are comprised some not very detailed questions about the subject, like more high-level questions about UGS and their impact on the lives of their visitors, resulting in a big picture view of this relationship.

UGS can be important because they allow different types of relationships with nature and the place itself as pointed out by Murphy [21]. For instance, people can feel place attachment which is when a user really enjoys a space and probably due to it simplicity it has emerged in several research pieces related with this area. Place identity is when a person feels connected with the identity of a place, and that reflects with its personal identity. Place dependence is associated with the situation where the place can be seen as a resource used by people. Lastly, place pride is common in people that visit a place in a regular basis or are in some more personal way connected to a place that possesses some features that make its users feel proud.

Murphy [21] also studied the evolution of the relationship between visitor and space trough out the
years, for that, users answered questions based on their perception of the space at the time of the study and based on what they remember from ten years before. Place attachment was measured by these questions recurring statements and Likert scales, inspired by Stedman [22]. Likert scales are scales based on forced-choice questions, their purpose is for users to classify their level of agreement or disagreement with the statement on a five (or seven) point scale. Stedman evaluated the answers to his questions with both 7-point and 5-point scales while exploring sense of place.

Sense of place is a term recurrent in research about this topic and is an important factor when defining and determining a person’s attachment to a green space. Stedman [22] described it as a “three-component view that weaves together the physical environment, human behaviours, and social and/or psychological processes”. This concept was defined in multiple ways throughout the years, for example, a place’s physical setting and human experience and interpretation by Brandenburg [23], positive and emotional bond between people and their environment by Low [24] and Moore [25]. As proved by the different definitions above there is a disagreement about the origin of the attachment to a place (which is in the root of sense of place), while some defend that this attachment comes solely from human emotion other thinks it is triggered by an already existing physical feature of the place in question.

The gathered data about these questions will allow to understand why people prefer some green spaces while avoiding others and the same for particular places and features within specific UGS.

The background of the visitors is also a determinative factor in the definition of these relationships. The mentioned background is some previous contact that a person might have had with green spaces or nature of any type. A previous contact could be a memorable visit to some UGS or a deeper connection like growing up in a rural area where the nature is plentiful. This exact point was proved by Murphy [21] that realized that green space identification by its users is highly influenced by their place of origin and upbringing, the users who described them “rural” or “green” had a more positive relationship with UGS in their present life. Resembling claims had already been made by Chawla [26].

2.1.3 Visits Motivation

Figuring out what drives people to visit UGS is usually one of the main goals for studies relating UGS planning and management. The motivation can come solely from the visitor's will or it can also be influenced by some characteristics of the green space that as appealed to that person. The first type of motivation supports measuring and correlating the connection between people’s visits to green spaces and their well-being. Inferences about the second type of motivation can also be seen as a study of the user’s favourite features and elements within the UGS, and how they work as attraction elements to visitors. It is important to collect this information that allows a complete mapping of the UGS, providing the biophysical structure of green spaces in order to draw conclusions and establish associations between the well-being of the citizens and the characteristics of the surrounding environment. The two types of motivations are not independent from each other because features of a place can trigger needs that lead people to a green space. Results driven from these data are at the core of many of UGS related studies, so it is of the utmost importance that this data is correctly evaluated.
The motives that take a person to visit an UGS have already been studied in many occasions. In general these motivations do not change a lot from study to study, although they can be context dependent as previously referred. Below is a list of motivations why people visit UGS found in the works of Shan [10], Chiesura [19], Lo and Jim [27] and Yuen [18]. The main motivations pointed out are (1) to enjoy fresh air/scenery, (2) to relax, (3) to exercise, (4) for quietness, (5) to have contact with nature, (6) to play with children, (7) for social interaction, (8) to accompany the elderly, (9) to walk pets, (10) to watch wildlife and (11) to engage in cultural activities.

The previous items are ordered by the number of times they were selected by visitors in the mentioned studies.

Liu [8] stated that, in some situations, there is not a clear boundary between the different visitors motivations, i.e., they are often interlinked. One example of this is the people who select options such as “To enjoy fresh air/scenery” or “For quietness”, could easily be include in the group of people that selected “To relax”.

The main reason for visiting UGS identified by the visitors was “To enjoy fresh air/scenery”. The favourite motivation is clearly backed up by individual features of the place in question. This is not related with any element or feature in particular but to all of them as a whole and the way they are connected. The way these elements are laid out and look, influence the aesthetics of the place and how its visitors perceive it in terms of beauty and scenery. The type of used elements (e.g., vegetation) and respective layout, as well as the location itself of the park may also contribute, or not, to give the perception of fresh air. There has been an increasing concern about the UGS aesthetics and pleasing the people who visit them, this has been fuelling several studies to figure out which elements to include, or not, when planning UGS in order to keep motivating people to visit green spaces by filling this need to enjoy the scenery and promote well-being and health to the population.

One type of elements that are considered when planning UGS is vegetation. In terms of the vegetation, some studies suggest that people prefer biodiversity and the approximation to the wilderness to aesthetics. This was concluded by Murphy [21] along with the fact that there is also a preference for diverse ES besides the recreational one. Supporting this finding, are also the studies carried out by Carrus et al. [28] and Romagosa [9] that state that higher levels of well-being were reported in parks with high biodiversity. Likewise, Krasny et al. [29] added how sense of place can be shaped by the ecosystem health and function and not only by its landscape features.

The vegetation elements can also be associated with the “For quietness” motivation because green spaces should have noise reduction function as well as providing positive soundscape experiences through “quiet areas” [8]. The quietness and tranquillity induced by vegetation was correlated with high levels of relaxation and reduced anxiety [30] and the elements used to achieve these goals vary from single plants, to green walls and hedges. The topic of sound in UGS and the issues around it can also be connected to the biodiversity issue visited above. Research conducted by Yang and Kang [31] and by Liu [32] have shown that natural sounds are preferred to artificial ones, tweaking with biodiversity can introduce more natural sounds like animal noises or even the sound of the wind in different types of trees and plants.
There is no doubt about the benefits of physical activity in maintain and even improving good health and because "To exercise" is the third most popular reason why people visit green spaces there have been some efforts in showing that places like parks, natural open spaces and green spaces are important locations to practise exercise, this statement was validated by Godbey & Mowen [13], Henderson [33] and Pretty, Pea-cock, Sellens & Griffin [34]. Romagosa [9], that studied the connection between UGS and exercise by surveying park visitors in the metropolitan area of Barcelona verified that people expected and actual felt improvements in their physical health associated to their visits to the parks where the study was carried out. This is supported by the fact that although not completely necessary some parks have elements designed to assist the visitors in practise physical exercise. These elements can range from structures specific for certain types of exercise (e.g. sit-ups, pull-ups and others) to just running/walking friendly flooring.

2.1.4 UGS Related Research Issues

By connecting the initial problem of this report (section 1.2) to the problems and studies explored in the section 2.1 a set of questions was designed to collect UGS related data from their visitors.

Some questions ought to be answered almost directly by the users when collecting information about their likes and dislikes.

- Which Urban Green Spaces do people visit the most?
- Within the visited green spaces which are the preferred places?
- Within the visited green space which places do people avoid?
- Which Urban Green Space do people avoid?
- What do people feel when visiting their favourite places in green spaces?
- Why do people visit their favourite places in green spaces?
- What makes people like a green space? The layout and design of the space, the nature, the people or other?

While other answers could be derived from information provided by the users indirectly, such as their profiling information and their position.

- Who visits Urban Green Spaces and when?
- Who prefers certain Urban Green Spaces?
- Who prefers specific elements within the Urban Green Space?

The answers to these questions intend to solve the previous problem by providing the detailed information about UGS from the urban dwellers. These questions will support designing the necessary mechanism to support research and urban planning of the UGI and human well-being.
2.2 Survey Systems

This section aims to explore the survey systems and crowdsourcing in general. After a brief description of the state of the concept of crowdsourcing, a special focus is given to the already existing system and more precisely which are their major weaknesses to support and motivate some of the following work.

2.2.1 Crowdsourcing

Traditionally, sources of information used by research and academic purpose were required to follow some standards and scientific methodologies. Yet, this vision has been challenged by the understanding of the increased value of citizens as a source of valid information, that as said by Goodchild, individuals should be considered as intelligent sensors, each of them an independent synthesizer and interpreter of local information [35].

This, triggered extensive research and development of work related with this subject to explore the true potential of this field and its application to a wide variety of problems. Work from different authors applied in different situations led to the emergence of different terminology and definition throughout the years. The terms that have arose refer both to information generated and to processes used to generate it and although the definitions might diverge, they all have the same core principle, citizen involvement in carrying out different activities while collecting data. Some of the most used terminology regarding people’s participation are Crowdsourcing, Volunteer Geographic Information, Citizen Science, User-Created Content (UCC), Public Participation in Scientific Research (PPSR), Public Participation Geographic Information Systems (PPGIS) and Map Hacking [36]. These methodologies have been applied in some of the following areas, as communications, crime, disasters, ecology, education, environmental monitoring, among others. One of the biggest advantages regarding crowdsourcing geographic information is that researchers no longer need to travel to a specific location in order to collect data, the same can be done by regular people that are already on-site.

It is common to find in literature about Crowdsourcing, Citizen Science or VGI mention of what started and fuelled all the work related to this area, this enabler being technological advancements made. What was briefly described by See [36], Haltofova [37] and Bubalo [38] was explored in detail by Goodchild [35]. They identified Web 2.0, Geotags, Global Positioning System (GPS), Graphics and Broadband Communication as being the foundations on which crowdsourcing applications are built. All these technologies are accessible using smartphones, which have become an indispensable object in our everyday life. Because smartphones have internet access from almost everywhere through mobile data, GPS signal and cameras and they can work as perfect sensors to gather data from and by their users. The previously mentioned GPS signal and geotags is a key factor in enabling geo-crowdsourcing applications. By appending the geographic location of the user to the data that has just been produced allows that location from where the data originated to be studied countless ways.
2.2.2 Online Surveying

The collection of data through surveys is not a new technique, surveys have been conducted for many years and have known many formats from paper-and-pencil, mail, telephone and now they are migrating to online surveys [39]. This change did not appear out of the blue, it was heavily motivated by technological advances made in the fields mentioned in the end of section 2.2.1, specially Web 2.0.

The clear advantage of surveying over the internet is the quick and easy attainment of data from the largest number of participants [40] since these technologies are widely available to almost everyone. Other advantages are the need for fewer physical resources, simplified logistics and elimination of data entry errors due to the possible association of software to handle or even filter the answers.

Although very distinct there seems to be some agreement in how paper-and-pencil and online surveys have some similarities in terms of the expected quantifiable attributes like personality measures with scales such as affiliative, conscientiousness, conventional, decisive, emotionally controlled, evaluative, optimism, outgoing, outspoken, social desirability and variety seeking [41]. Equivalencies were also found in terms of impression management/social desirability and data completeness.

Online surveying fills all the requirements imposed by other types of surveys while adding some of its advantages to it. It last years it has proven to be useful in large number of cases like the ones listed by Behren, Sharek and Meade [41], extending from distributing the search for a crash site in satellite photos to save lives to increasing the population engagement in academic research studies, crowdsourcing has shown its worth when time is of the essence and there is a need for input from a large number of individuals that may be separated in space, for example, during natural disasters and pandemics.

2.2.3 Crowdsourcing Taxonomy

Since 2006 when Jeff Howe coined the term in a Wired Magazine article [42] crowdsourcing has become a broad subject that collected throughout the years many different definitions due to all the different situations where it has been used. Each of these definitions is applied to a specific time, technology and situation, so maybe a simple definition may not be enough or can even be inadequate depending on the mentioned variables. In order to solve this problem it may be useful to better understand what crowdsourcing really is through a different method such as a structured view of the research associated to this topic. Through this method Yuen, King and Leung [43] were able to identify the main components of crowdsourcing proposing a taxonomy of crowdsourcing systems, this taxonomy is represented in Figure 2.1 where crowdsourcing was divided into four main parts: Application, Algorithm, Performance and Dataset.

Due to the popularity and evolution introduced by technologies like Web 2.0 [44], the web and mobile crowdsourcing Application format have proliferated and received much attention. These applications have a list of tasks to be executed, requester and workers, being the requesters the ones that provide the tasks that populate the list and the worker the ones who complete tasks. Usually, the completion of the task is rewarded either in a monetary way or by gaining credibility around the requester and other workers. Crowdsourcing applications can be grouped into four different categories: Voting System,
A crowdsourcing application is considered a voting system when the correct answer to a task or question is determined by the most given answer. Information sharing system is when an application has the main goal of sharing various types of information among its users. Game type crowdsourcing applications pioneered by Luis Von Ahn [45] collect data and metadata from problems solved by the players. Differently from the others, this type of applications rely heavily on their ability to entertain the users. Finally creative systems are the applications that require from their worker creative tasks that a computer is not able to do like drawing and coding.

Many crowdsourcing systems make use of an Algorithm in order to improve themselves. Several studies can be found on the different ways these algorithms are used in this type of systems. Some of these uses are formalization of the system's design, performance modelling, study user’s behavioural patterns and even quality management.

There can also be found several pieces of work that center their attention analysing the Performance of crowdsourcing systems. These studies fall into one of the three following categories: user participation, quality management or cheating detection.

Because the task of any crowdsourcing application is distributed and completed by somewhat anonymous users it is important to understand the workers demographics, how they respond to financial and intrinsic incentives and also their behaviour when completing the system tasks and challenges. This last point offers insight not only about the user but also about the system itself, for example, in terms of interface and the surveys complexity and relevance. To control the answers’ and task’s solution quality it is important to divide the main task into smaller and more comprehensible tasks, this way the number of invalid or incorrect answers should be lower since the majority of the workers were able to understand and easily solve the task. Cheating detection work emerged motivated by malicious workers that instead of dedicating themselves and working on the requested task prefer to produce generic answers in an attempt to maximize their profits.

Dataset literature had its origin in many real crowdsourcing systems that during their operation have gathered large datasets that were, afterwards, made available to promote future research.

### 2.2.4 Smartphone as a Surveying Tool

As briefly explained before, smartphones have played an important role in the evolution of crowdsourcing in recent times. The main reason is the fact that smartphones are being more and more used...
worldwide, and due to all the multiple they offer, these devices have found their way into everyone’s pockets. This smartphone proliferation was made clear in an article posted in the Pew Research Center by Jacob Poushter [46].

Figure 2.2: Adults who report owning a .. [46]

In this article it is shown the percentages of smartphones used by the population in a list of countries (2.2). The percentages were based on a total sample from Spring 2016 Global Attitudes Survey. Q80 & Q81 U.S. data from Pew Research Center survey conducted Sept. 29 - Nov. 6, 2016. Overall, the smartphone ownership percentage takes a considerable value, in all the surveyed countries, i.e., there are more smartphone owners than owners of other types of mobile phones and than people that do not own a mobile phone. It is also worth mention that the results presented in [46] were of surveys conducted in advanced economies.

It might also be important to understand who actually owns a smartphone to better target the crowdsourcing workers. Aaron Smith published an article in Pew Research Center where explored the smartphone adoption and where studied this issue [47]. In this article smartphones owners where studied regarding their gender, age, race/ethnicity, household income, education level and geographic location. In this article, it is possible to assess that smartphones are more popular among young people than older people, also more common among people with higher household income which makes sense due to the elevated cost of some of these mobile devices. It is also possible to conclude that people with higher education level use more smartphones as well as people who live in urban and suburban areas. The source of this table is The Pew Research Center’s Internet & American Life Project, April 26 - May 22, 2011 Spring Tracking Survey. The sample size is n=2,277 adult internet users ages 18 and older, including 755 cell phone interviews. These interviews where conducted in English and Spanish. It was
considered that "Smartphone ownership" includes those who say their phone is a smartphone, or who describe their phone as running on the Android, Blackberry, iPhone, Palm or Windows platforms.

Finally, it is necessary to realize that smartphones or more specifically all the features offered besides communication are not only used for entertainment. In Figure 2.3 there are some common uses for smartphones reported by smartphone owners [48]. We can see that among the listed uses are job or house search, more than half of these smartphone owners have used their phones to get health information and do online banking. The information in this chart was obtained from Pew Research Center American Trends Panel Survey, October 3-27 2014.

![Figure 2.3: List of activities conducted using a smartphone and corresponding percentage of smartphone owners who reported them [48].](image)

In sum, given that smartphones are nowadays broadly used in our society [35] it is clear that they are the ideal tool to use in the development and evolution of crowdsourcing systems and applications.

### 2.2.5 Analysis of Existing Technology

With the increasing popularity of crowdsourcing and technological evolution backing it up. A lot of crowdsourcing applications have been created, both web and mobile. Still there is a demand for this kind of applications because every day, people are trying to solve new problems or even optimizing solutions for old ones. Crowdsourcing can be the answer to many of these situations which means that applications will continue being developed as each problem or situation requires a different application with different or new features and functionalities.

Developing an application is not an easy job much less if it is a crowdsourcing one, it requires a large amount of resources both human and financial. Reduction of these costs would be possible by simplifying the process of development of the crowdsourcing applications. Having a system generic system for creating this type of applications would simply the process but, if there are a lot of options when it comes to crowdsourcing applications the same cannot be said in terms of generic systems to create standardized applications that would allow an effortless specialization of the application in any
subject.

Through some literature review [49] were found some tools that resemble that generic system, such as Device Magic [50], Fulcrum [51], Magpi [52], FastField [53], Zoho Forms [54] and SurveySparrow [55]. These are applications that allow the creation of multiple custom surveys, and do not require programming experience because the user relies on dragging and dropping building blocks to create the survey application. A comparison between them is made in the Table 2.1 regarding key factors like (1) having a Free package for their users, (2) if the created surveys are Cloud Based or Self Hosted, (3) whether the application is Open Source or not, (4) if there is available the possibility of implementing Dynamic Questions in the surveys, (5) if they produce Public Surveys or are the survey only shared within the same organizational accounts, (6) if the application is at some extent Extensible, (7) if the survey allow the incorporation of External Data Sources and finally (8) if the survey can be target to a specific profile o people.

<table>
<thead>
<tr>
<th>Name</th>
<th>Free</th>
<th>Cloud Based</th>
<th>Open Source</th>
<th>Extensible</th>
<th>Dynamic Questions</th>
<th>Targeted</th>
<th>Extensible Data Sources</th>
<th>Public Surveys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Magic</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>+/-</td>
<td>×/−</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Fulcrum</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Magpi</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>+/-</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>FastField</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>−</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Zoho Forms</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>SurveySparrow</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>+/-</td>
<td>✓</td>
</tr>
</tbody>
</table>

Based on the Table 2.1 is possible to realize that only Magpi offers to its users the possibility of using their services for free but even them only offer the minimal services and only Zoho Forms is open source but does not offer any easy way to implement new features through extensions.

Extending the application is not an option, none of these applications allow new features to be added programmatically.

Dynamic questions can be a very powerful feature and it is not found in any of the applications, Device Magic and Magpi has the possibility of incorporating in its surveys conditional questions which although they do not have the same versatility as dynamic questions they can mimic them in terms of functionality. These conditional questions will always be limited by lacking the automation provided by a dynamic questions module.

None of the applications allow targeted survey creation by limiting survey answers to people that fit certain profile characteristics.

Only SurveySparrow considers external data sources, although there is only a limited set of external data sources that can be integrated with the survey.

When it comes to publishing the surveys none of the applications offer an ideal solution. The applications that make the survey publicly available do it by sharing its link across several platforms while the
tools that make the survey accessible in through mobile applications only allow this survey to be seen and answered by member of the same organization of the person who created it.

Some of the table cells have not been filled because there is not enough information available about the applications' features and has not been possible to register in the system to test them manually.

Also, a shared drawback of these tools is that they are not systems, but applications. This means that all the applications that a user might creating stay within the these “parent” apps. This does not allow the creation of fully independent programs, being always limited to the standards imposed by the tools in question.

2.3 UGS Surveys

From section 2.1 is possible to say that there is a lack of information regarding UGS, specially about three main topics, which are (1) the profile of the visitors of UGS, (2) the relationships established between the visitors and the space itself, and (3) the motivation behind each visit. Data collected in UGS and the information obtained through its study has huge value due to all its possible applications and so this shortage has proven to be very inconvenient in multiple situations which let to authors writing about it over and over again. In many of the studies about this subject, researchers propose a solution for the problem by designing their own mechanism to collect data. So far, the found solutions have not been very efficient, as for the most part they involve paper surveys and large amounts of human resources to deliver them and collect the respective answers. This represents a waste of time and effort that could be dedicated to improve the survey, collect more data or more extensive analysis of it, an upgrade to this process would highly benefit researchers who need the collected data. The few cases that adopted some sort of technology to help perform these tasks struggle both with limitations of existing crowdsourcing tools or with the hardship of creating their own tool.

Which unveils another part of this problem, if not the root of it. There is still missing of a system capable of serving every researcher crowdsourcing needs. In section 2.2 is made clear that technology is the way to go at least when considering the future of crowdsourcing and although there is a wide range of crowdsourcing applications and systems available it seem that none of them is fully able to satisfy the specific needs of every researcher. All the existing technology provides its users with basic crowdsourcing features which might suffice in situations that only a simple survey is required but when it comes to scientific subject it may be necessary to implement more complex surveys and collect different types of data. These crowdsourcing tools are limited to the collections of a narrow variety of data and to not comprise the possibility of collection different types of information through methods customized by the researcher. Besides this the analysed tools are not free to use nor open-source, not allowing the people using the system to use or implement useful feature like dynamic survey deployment, data validation mechanism or complex feedback processing. All the features missing and disadvantages found in the available systems motivated the creation of a system that offered the basic crowdsourcing functionalities and allowed anyone to easily extend it in order to incorporate even more features. This system in particular is capable of being applied to the solve the UGS issue, which means that with little
programming effort an application can be created that dynamically collected user input and sensor data from UGS visitors.
Chapter 3

Generic Crowdsourcing System

In this chapter is described in detail the proposed solution for the problem that motivated this work. The different users and their roles will be evidenced as well as the system core functionalities and the requirements that must be fulfilled. The system’s architecture allows to analyse the system in different levels of abstraction and identifies the different modules or components that implement the different system features. How these functionalities and mechanisms are used is explained and demonstrated in the implementation of the system. At the end of this section is presented an auxiliary tool developed to facilitated the development of the mobile surveys module along with instruction of how to use both that tool and the framework itself.

3.1 Users

There will be different agents interacting with system throughout its operation. Each of these user interactions will happens in different moments and will be associated with different functionalities. There are three different types of users that keep the system running correctly and their roles are explored below.

3.1.1 Administrator

The first type of users are the administrators or programmers. They should have some programming skills because they will be responsible for making any necessary changes to the framework. The programmer is asked by someone who wants to create a crowdsourcing system to alter the generic system in order to incorporate a set of features defined by that person. Because the main feature of this system is being generic it will allow customization, which is achieved by programmatically extending existing system modules or developing new ones. There will be available an administrative console (only accessible with the administrator’s credentials) that will allow to manage users and assign them one of the types described below.
3.1.2 Researcher

Researchers are the type of users responsible for creating the surveys. Usually these will be people studying some subject and whose work would benefit from the collection of crowdsourcing data. So, by extending the generic and already existing system resorting to a programmer, researchers will be able to create surveying applications. These users will be able to define the survey targets in terms of both geolocation and profile, they will also choose from which sensors to collect data and most important, define the method to generate the dynamic questions. Besides the definition of all the mentioned parameters, researchers also have access to their surveys’ answers, which will be available to download in order to be applied to solving the problem that motivates each survey. Besides their active role in the definition of the system features, researchers will also have access to a console where they will be able to see the mentioned answers and make other important validations. It is expected for the system to have small number of researchers.

3.1.3 Normal User

The final type of users are normal or common users. These users are just people who download the developed application and answer surveys. This is the only function of these users, although it may seem simple it is highly important because these are the users who produce the data to be studied by the researchers, solving the problems that motivated the creation of the survey and the application in the first place. It is expected for this to be the group with the highest number of users.

3.2 Functionalities

The system must implement a set of functionalities imposed by the researchers. This is done by dividing the two main moments of system operations in a series of steps. The two main moments are survey creation (System Flow) and the submission of a survey answer (Application Flow), the mentioned steps are explained in more detail in the sections below. Through this division will be made evident that this is a generic and extensible system that allows the implementation of dynamic surveys and the collection of sensor data.

3.2.1 System Flow

In Figure 3.1 there is a high level flow of how the system will operate when building a customized survey application. This flow will illustrate the different steps the researcher and programmer must take in order to create a specific application from the generic framework. These steps will be essential in defining the main features of the application to be developed. This system and the new features it brings will innovate the process of creating crowdsourcing applications. All these new features are represented by the coloured blocks in Figure 3.1.

The system will allow the definition of the area to where the applications are destined to be used in
the **Area definition** step. While some crowdsourcing applications are not region sensitive, others may benefit from the restriction of their usage to specific geographical areas.

The system will provide a way to define the profile of the targets of the survey, just like the area definition this feature aims to improve applications that benefit from their participants fitting in a specific and customizable profile. In the **Target definition** block the way of matching the application with its target participants will be done in a similar way that Google and other platforms implement targeted advertising.

A mandatory common step in the development crowdsourcing applications is the definition of the period of time during which data gathering is enabled and also the data that is actually being collected. These steps are also present in the proposed system and are represented respectively by the $\Delta t$ and **Gathered data definition** blocks.

Besides the traditional data sources already contemplated by existing applications, such as, basic user input and some device originated data, the system will facilitate the inclusion of external data sources such as physical and web sensors (**Sensor definition** block).

After collecting all the necessary data, it is possible to subject it to some data validation mechanisms. These mechanisms should be established in the **Data validation definition** stage and can be used to filter out invalid data or solve data repetition situations as well as conflicting data occurrences.

Finally, it is necessary to define the parameters that will be used by the system to dynamically select the actual questions to be presented in the survey. This will be done in the **Dynamic questions definition** block.

### 3.2.2 Application Flow

Similarly to the previous example, in the Figure 3.2 the grey block also refer to features that are implemented in other tools while the coloured blocks indicate the innovations introduced by the proposed system. In this chart is also represented an high level view of the work flow of the usage (answering surveys) of an application built using the generic framework.

The first step of this process is represented by the **Sends questions** block and consists in providing to the participants the necessary questions in order to gather the wanted data. In this stage the questions sent can be fixed/static (same for every participant), the questions can also be dynamic or even a mix of both. The standard for this step in other tools is static questions. Here the questions are sent from the server to the mobile application. The questions sent to the application are selected in the server by the data validation method implemented by the survey architect.

Then the system will gather the data. This will be made in two steps, first by receiving the answer from the participants (**Receives answers**) and in the second step by retrieving data from sensor and
other external data sources (Gets sensor data), this last step represents the possibility of integrating external data sources as mentioned above. In these steps data will flow from the application to the server and from the external sensors also to the server.

Finally, the answers and other collected data will be stored in the system’s database and made available for analysis.

3.3 Requirements

This section is dedicated to presenting the requirements of this framework. The requirements are classified as functional and non-functional. The functional ones are directly related to system feature while the non-function are some characteristics expected in any system, such as, usability, security and scalability.

3.3.1 Functional Requirements

Sensor Integration

One of the innovations introduced by this system is the possibility of data collection from external sources. It is required of the framework to allow the integration of external data sources. The external source might be the web because of the amount and diversity of data available there that can be used to support or be crossed with data gathered from the surveys.

While web sensors collect global data available online and made available by other people, the system should also be able to integrate sensors that collect data from each user. The users smartphones are examples of this last type of sensor, each device can collect data from hardware sensors if available or user data from third-party platforms.

The integration of these different data sources differentiates this system from the all the other ones, this freedom of choice is appealing to survey creators.
Easy Authentication

The system should allow its users to register and log into it. With the increasing internet and mobile app usage also increases the amount of accounts from applications, websites and other tools accumulated by every individual user. If security is of utmost importance for the user usernames and especially passwords should vary from application to application and remembering many different passwords can be cumbersome. So, besides a easy and fast login the system should be able to register users even if they do not want to create a new account, a service should be available to handle users authentication using external platforms where the user might already have created an account in past. From these external platforms the system must gather enough information about the user so that it can be uniquely identified within the crowdsourcing system.

Multi-OS Operation

With the growing global usage of smartphones covered before in the section 2.2.4 the smartphone market has also expanded having appeared a large amount smartphone manufacturers but also smartphone Operating System (OS). From all the OS that have arisen undoubtedly Android [56] and iOS (iPhone Operating System) [57] are the most popular, Universal Windows Platform (UWP) [58] might also be considered to be part of this group. It is important for the framework to be able to reach as much OS and consequently as much users as possible. For this to happen the technologies used to develop the system's mobile application must be able to run in as many different OS as possible.

Easy Development

One of the main features of the system is its possibility of extension, with some programming effort the generic system can be modified into a system able to solve different problems. For this to be true, besides all the mechanisms intrinsic to the framework, the system should be built using an environment, programming language and framework that facilitate the programmers job by providing programming libraries and other helpful mechanisms. The programmer could be spared some effort by shortening the programming stack, using the same programming language to build more than one part of the framework would accelerate the development process and avoid some programming errors cause by constantly changing languages.

Easy Deployment

The development is not the only step in the system creation that should be facilitated. Different mobile OS have different ways to deploy applications publicly, usually to deploy an application it should be made available in so sort of application store, for the main OS this has associated costs that might be undesirable at least in the beta stage of the application life-cycle. The used technologies should also consider this situation and offer a free and easy alternative for the initial application deployment.
Multi-Language Operation

Crowdsourcing applications value depends heavily on the amount and quality of the data they are able to gather. So, it is important that the applications created using the generic crowdsourcing system are able to reach as many people as possible increasing their odds of collecting large quantity of quality data. The language barrier can be an obstacle when trying to reach a great amounts of people because if the applications is in a language unfamiliar to the user it may cause the user to provide unreliable data or to not use the application at all. Because the developed project is a generic system whose main feature is the ability to be easily adapted and used in different situations by different people, restricting the application to a single language would be limiting. So, the system should be able to operate in different languages according to the programmer’s specification.

Notifications

One more time, the utmost importance of data collection motivates a system requirement. It is especially important to collect the best data possible, the parameters that measure the data quality varies from situation to situation. Users may not always remember to answer surveys and produce crowdsourcing data or simply forget to do it when are gather the most favourable conditions to do so. So, the system should allow the definition of the appropriate mechanism to notify the application’s users when they best suitable to answer a survey. These conditions may be related to the user’s geographical location if it is beneficial to collect data from a specific area, fitness activity or can also be time related if the surveys should be answered within a defined period of time or just to periodically remind the user to submit an answer.

Tutorial

It is common for application to provide their users with some sort of tutorial or support on how to use it. The generic system allows the implementation of a wide range of applications that may vary in complexity, so it should be offered the programmer an easy way to create and deliver to the users the support pages they may require. This presentation can be optional and its content totally up to the programmer because it may vary from application to application.

Dynamic Questions

The framework should be able to deploy dynamic questions to its users. The static questions mechanism implemented by most of the existing has many limitations and has proven not to be enough for some researchers who would benefit if different surveys were to be served in different situations. These dynamic questions can be associated to a user status where the new questions will depend of previous answer, can be related to a specific period of time, deploying different surveys at different times of the day, week, month or year depending on the programmers implementation, the logic behind this dynamic questions can also restrict surveys’ deployment to a specific group of people or geographic area.
User Profiles

It is common for online platforms to provide a way to identify their user and with the proliferation of social media and other online technologies in general people have gotten used to and expect to have a profile page in the platforms they use. These pages usually have customizable information about the users that gives them a sense of identity and how they are perceived by others. Apart from this, the value and meaning of survey answers may vary depending on the person give those answers and so this framework should provide a profiling feature. Although not being a social platform, the system should collect user data from the mobile application that allows the users to identify themselves but also enable the researcher to draw some more conclusions by crossing user answer with useful profile information.

3.3.2 Non-Functional Requirements

Usability

Except in few situations, the survey applications will not target very specific groups of people, this mean that people from different backgrounds may answer the same survey, this may include people that are not very educated on the subject of the survey or even people less comfortable with technology. But this does not mean that the answers given by those people are less valid or valuable. Therefore, in order to collect as much data as possible in each survey the system and its applications must be intuitive and of easy access to every one. Even though tutorials may be made available for the user the flow and dynamic of the application must be as logical and simple to understand as possible.

Security

Fundamental security requirements must be met like in every application and service. The system will hold accounts for its users, mainly because there are different tiers of users, for example, researchers and common users have different privileges regarding the creation of surveys but also because the answers to the surveys are crossed with personal data and so there is a need to clearly distinguish between users. The need to register before contributing either by creating surveys or simple answer them, make people more accountable for their action avoid irresponsible usage of the system.

Each account will have associated credentials (username and password) and personal information because this is considered sensitive data, it must be encrypted before stored in the database. If the survey concerns a sensitive subject or the collected data from the answer is in some way confidential or just because the author considers the collected information part of its intellectual property and wants to protect it, could be advisable to also encrypt the collected data.

Connections between the different parts of the framework should also be secured through the standard mechanisms provided by the used technologies, extra work that can be done in terms of encrypting the messages send over HyperText Transfer Protocol (HTTP) and code to avoid attacks on that level and simple error handling.
Ensuring data security to the researchers will allow the also ensure it to the users, encourage this way more people to participate because they know any data they input in the platform will be protected.

**Scalability**

Crowdsourcing applications and surveys in general rely heavily on data from the participants and users. The more data is collected the easier is for the researchers to find results and draw conclusions about the problem that motivated the survey. Although some exceptions about very specific subjects that may be restricted to a small group of people the majority of crowdsourcing applications are open to the public as long as they fit the profile to participate the application’s survey. So, it is important for this types of system to be able to scale in order to support as many users as possible.

As mentioned before, researchers or survey creators benefit the most from large amounts of answers and collected data, by providing them with that, will motivate more researcher to develop more applications using the system.

### 3.4 Architecture

In this section will be presented and explained the system architecture, analysing each of its parts individually as well as the interaction between them. To represent the system’s architecture was used a C4 model [59] approach. This model allows to display software architecture at various levels of abstraction, containing four levels: Context Level, Container Level, Component Level and Code Level. From the context level to the code level this model will provide more and more details about system as if it was zooming into it. In this section can be found the first three levels of this model while the fourth one, the code level or data model is presented in the section 3.5.

#### 3.4.1 Context Level

In the context level the system’s details will not be clear but rather the interactions with its surroundings. The diagram of this level provides a starting point showing the system in question, other systems it communicates with but also the system’s users. This is a very high level system representation that can be used to highlight some of its requirements such as sensor integration and each user’s role.

In the Figure 3.3 the generic crowdsourcing system is represented by the coloured box in the middle and around there can be found two other boxes that represent the external systems with which is made any kind of interaction. One of these boxes represents the Third-Party Authentication Platforms available to user register and login with and the other one represent the External Sensors from which the system connects to in order to collect additional data. It is also visible the Administrator or Programmer, the Researcher and the Normal User that are the three possible user types for the system and whose roles, described earlier in the Section 3.1, are briefly summarized in the arrows representing their interactions with the system. Finally, the Normal User can also interacts with the Third-Party Authentication Platforms during the authentication process.
3.4.2 Container Level

The container level comprises of a more detailed view of the system itself. Here can be found the different applications and data stores that compose the system, the relations between them as well as their relations with external components. The shown applications and data stores are the containers that name this level which are basically separately runnable/deployable units.

In the Figure 3.4 are represented the containers within the crowdsourcing system. The Web Client Application is a console-like web application used by the administrator to give or remove researcher privileges to the system users, it can also be used by researchers to analyse and manage the collected crowdsourcing data. The displayed data on these application and all the changes made on it come
from and are sent to the server. The **Mobile Application** is used by normal users to answer surveys and therefore collect their input, can connect to **External Sensors** as part of one of the surveys and also communicates with the server to exchange surveys, their answers and also some feedback. The **Server** provides a JavaScript Object Notation (JSON)/HTTP API to the previously mentioned applications in order to serve them all the survey, data validation and administrative features that they require, interacts with OAuth Third-Party Platforms made available by the **Programmer** to validate users login and registration when such method is selected and finally it reads and write data in the **Database**, as mentioned before, the data reads are to provide data to be displayed in the apps while the writes are either new collected data from the mobile app or data updates or deletions from the web client. The **Cache** container with which the server also interacts represents a cache server used to store users' sessions while they are using the system.

### 3.4.3 Component Level

Component is a term broadly used in the software development industry and its meaning may vary from situation to situation or even from technology to technology but in this context is an functional encapsulation of part of the system with defined interface, it is also important to highlight that the components are not separately deployable.

This section is divided into sub-sections, one for each of the containers identified in the Section 3.4.2. In these subsections will be explored the components of the **Server**, **Mobile Application** and the **Web Client Application**. Although the third-party authentication platforms, external sensor, database and cache may appear as containers in the previous level of the C4 Model their components will not be explored as their architecture and implementations fall out of the scope of this project.

It it also worth mentioning that although usually at this level the components representation is accompanied by a brief description in this case, due to the amount of components and in order to provide easily readable diagrams, that description is provided in the paragraphs surrounding the figures. The connections between components are also simplified for the same reason present before.

This framework was created by a team of two students, myself and Guilherme Eugénio, another Electrical and Computer Engineer master's student and final note related to this level regards the colour schema and how it reflects the work distribution when it comes to the development of the system's different components. The colours blue and white are used to represent components and containers intrinsic to the generic system that where developed by both team members, the colour green was attributed to the components built by myself that implement innovative features developed specifically to fulfill the requirements of the main case study explored in this document where the generic crowdsourcing system was applied, finally, the red components are the ones that were developed within the scope of Guilherme's master's dissertation.
In the diagram on the Figure 3.5 it is possible to see the server’s components and some of their interactions with the surrounding system parts. The server provides a Representational State Transfer (REST) API through which it receives calls in the form of HTTP Requests from both the mobile application and web client.

The Authentication component is responsible for handling users' logins and registrations, this consists in receiving users' credentials from the mobile app or web client and looking or adding them to the database. The user authentication feature requires an interaction between the server’s authentication component and its counterparts found in the mobile app and client app, the component also connects to the database component to perform all the needed reads and writes. An extension of this component is provided by default where it was implemented OAuth authentication using some of the most popular platforms that provide this service, further extension is possible allow to add or remove any platform to assist in this type of authentication.

The Surveys component has two sub-components, one that allows the system to behave as regular crowdsourcing by serving the same survey to the users, the other sub-component allow the system to be programmed in order to dynamically server different surveys. The Static Questions component only allows the system to collect the same data from a single survey and the Dynamic Questions component enables the implementation of different surveys whose content can depend on events created by the programmer and triggered by the users or previous responses given by them. This component connects to the component responsible for displaying questions in the mobile app, to sensor components both from the mobile device and external ones and can also connect to the database or the server’s file system when getting the forms to send to the users.

The Answers component has the simple purpose of receiving the answers given by the users...
through the mobile application and storing them in the database, to do so it connects to the mobile answer from where it gets the answer data passing it then to the database component in the server so they can be saved in storage.

The Feedback component is also divided into two sub-components, the Immediate block represents some simple feedback sent over to the user after the submission of an answer and the Differentiated is a component that should be called when an answer requires some processing that will eventually generate some feedback that will be sent back to the user. This components need to interact with the database component and this asynchronous feedback transmission to the user implies a connection with a feedback component in the mobile app in order to retrieve the feedback data destined to the user from storage or simply to store the differentiated results.

The Profile and Results components are mainly data retrieving and presentation components, they are called by the mobile app to get some data from storage to be display in the user’s device. The profile component handles all the user personal information and provides mechanisms to fetch data from the database to be displayed in the app’s user profile page but also handles requests to update that same data. The result component fetches from the database all the survey answers and other data that the user inserted into the system and sends it over to the mobile app where it can be consulted by each user.

The Data Validation component serves only requests from the web client. Since this console is where researchers can validate data, in order to implement all its features the server gets data from the database to be displayed in the website and also apply some change on that same data. These change are the validations made by the researchers and can comprehend moving data from one collection to another, editing parts of the information in question or even deleting it from the system.

The DB component works as a bridge between the other components and the actual database by providing a simplified interface to it. It connects to components and situations mentioned before facilitating data fetching, update, insertion and deletion.

The Cache component connects to the cache server where are store users' sessions. When a user logs into the system its email is stored in cache and anytime a request is made to the server the user must send over its email so the server can check if the user is logged before answering the request, each user session is stored in cache under a customizable timeout. So, although not represented in the diagram the cache component interacts with all the other components in the server.

Mobile Application

In Figure 3.6 diagram presents the mobile application’s components and how they interact with the remaining system.

The mobile app has a Authentication component through which the users can register and log into the system, this component get the authentication credentials as input from the user and sends them over to the server so it can add the user to the system or check if it is already registered. If the user opts to use oauth this component will retrieve the user's credentials from one third-party authentication platform.
The Questions component presents the user with the survey’s questions that it gets from the server. It is also this component that allows the user to answer those questions and sends them to the server to be processed and stored.

Mobile devices as explained in the Sections 2.2.1 and 2.2.4 are a great source of sensor data, the Sensors component provides both access to the device’s native sensors and external ones. This component is responsible for collecting the sensor data from the available and selected sources, to process it and pass it to the questions module that will submit this data as part of the survey answer.

The previous section explained that there are different types of feedback, after sending a survey answer the mobile app triggers the Feedback component to request some feedback from the server, this component is also responsible to display the server’s response containing the feedback, both immediate or differentiated.

Finally, the Data Presentation component is generically used to represent all the components that display data from server and do not have any other major functionality, such as the components that present the profile and results data.

Web Client Application

The web client features less functionalities than the other system containers and so has the simplest architecture in terms of components as presented in Figure 3.7.

The Authentication component, similarly to the ones presented before allows user to register and log into the system, either using their e-mail and password or through one of the available third-party authentication platforms, therefore this component connects with the server and the third-part authentication platforms. In the case of this web application only researcher and the administrator can use it.

For systems where the researcher is allowed to update the crowdsourcing surveys, the Survey

Figure 3.6: Component Level Architecture.
Submission component is available. This component receives input from the researcher, either in file or text format and submits it to the server, where it will replace the old survey. This component's counterpart must be implemented in the server. It consists of an endpoint responsible for receiving the new survey and store it in the database.

The Data Validation component is responsible for presenting the researcher with data from the server and providing mechanisms for it to be validated by the researcher. This component can be extended in order to change the available data for validation. Possible validations include, accepting new data into the database, removing data from the database or even editing that data, after performing any of these validations the data is sent over to the server so the database can be updated.

At last, the Download is a component that allows the researcher to download the answer data collected from the database base onto a file. This component connects to the server's data validation endpoints in order to retrieve the data and allows the programmer to configure the file name and structure, this means that the programmer is able to filter and organize the answer data sent by the server in order to better fulfill the researcher's needs.

### 3.5 Implementation

In this section there are listed and described the most relevant aspect of the system's implementation, such as, the server's Application Programming Interface (API), data model, used technologies and system features.
3.5.1 Server’s API

The server provides group of endpoints organized as shown bellow. The endpoint where divided into routes, each one dedicated to one part of the system and its functionalities. Being this a generic crowdsourcing system, extensibility is offered to endpoints directly related to the surveys and their answers, give to the programmer, freedom to make the most out of the system. The provided endpoints take in consideration a wide range of features that are possible to implement in this types of systems. Here are the provided endpoints and the suggested usage for each one, because many have to be implemented by the programmer their purpose may change according to the system designer’s will.

Oauth Endpoints

POST 'api/oauth/login'
Handles login with oauth authentication. Calls loginHandler function from the oauthExtension file.

POST 'api/oauth/register'
Handles registration with oauth authentication. Calls registerHandler function from the oauthExtension file.

Profile Endpoints

POST 'api/profile/
Get profile information of a certain user from the database. Calls getProfile function from the profileExtension file.

POST 'api/profile/editRequest'
Gets form used to edit user profile. Calls editProfileRequest function from the profileExtension file.

POST 'api/profile/edit'
Sends edit profile form, already filled, to update the database. Calls editProfile function from the profileExtension file.

POST 'api/profile/editRanking'
Edits user ranking, can be called in different occasions or not at all depending on the app implementation. Calls editRanking function from the profileExtension file.

Researcher Endpoints

POST 'api/researcher/getData'
Gets some data to be displayed in the researcher's console in the web client. Calls getData function from the researcherExtension file.

POST 'api/researcher/editData'
Edit some system data from the researcher's console. Calls editData function from the researcherExtension file.

POST 'api/researcher/removeData'
Deletes data from the researcher's console. Calls removeData function from the researcherExtension file.

**POST 'api/researcher/validateData'**
Validates data from the researcher's console. Calls validateData function from the researcherExtension file.

### Results Endpoints

**POST 'api/results/'**
Gets results containing users survey answers. Calls getResults function from the resultsExtension file.

### Surveys Endpoints

**POST 'api/surveys/'**
Gets a survey. Calls dynamicSurvey and staticSurvey functions from the surveyExtension file.

**POST 'api/surveys/submit'**
Submit a new survey to the server. Calls submitSurvey functions from the surveyExtension file.

**POST 'api/surveys/answer'**
Submit survey answer. Calls processAnswer functions from the surveyExtension file.

**POST 'api/surveys/answerImage'**
Submit image as (part of) survey answer. Calls processImage functions from the surveyExtension file.

**POST 'api/surveys/feedback'**
Get feedback of an answer. Calls immediate and differenciated functions from the feedback module file.

**POST 'api/surveys/getInfo'**
Get info to help answer survey. Calls getInfo functions from the surveyExtension file.

### Users Endpoints

**POST 'api/users/'**
Gets all users registered in the system. The request body must contain the adminEmail to check if the user making the request (usually the admin) is logged in the cache.

**POST 'api/users/register'**
Register new users in the system. The request body must contain the name, password, email and type (normal or researcher) of the new user to be registered in the system.

**POST 'api/users/login'**
Logins user in the system. The request body must contain the email and password of the user logging into the system.

**POST 'api/users/logout'**
Logs user out of the system. The request body must contain the email of the user logging out of the system.

**POST 'api/users/changeType'**

Allows admin to change user type between normal and researcher. The request body must contain the adminEmail in order to check if the admin is logged in the system but also the email of the user whose type is being changed and the new type is being changed into.

**POST 'api/users/remove'**

Allows admin to remove user from the system. The request body must contain the adminEmail and adminPassword to validate the admin identity in such an important decision and also the emailDelete which is the email of the user being deleted from the system.

**POST 'api/users/get'**

Get user’s generic profile from the database which may contain profile picture and sends it over to the mobile application.

**POST 'api/users/edit'**

Receives updates to the user’s generic profile from the mobile application and store those changes in the database.

### 3.5.2 Data Model

The final level of the C4 Model is the Code level, here should be shown how code was implemented using Unified Modeling Language (UML) class diagrams or entity relationship diagrams. For this project it was decided that the most relevant diagram to show in this section would be a simplified UML diagram (Figure 3.8). This diagram helps understand how data was perceived and modelled during the development of the system.

This diagram represents the associations between the different data used during the system’s operation and the corresponding multiplicities but it does not contemplate the methods and attributes associated to each system’s data component.

Below it will be presented a description of the data model and the most relevant data associations. As mentioned earlier there will be three different types of users: **Admin** (administrators/programmers), **Researchers** (survey creators) and **Normal** (common users that answer surveys), this division is also clear in the UML.

A **Survey** might be created with contributions from one or more researchers which is evidenced by the 1..* multiplicity in the aggregation between these two entities. On the other hand one researcher may have as many survey as he or she wants.

Due to the dynamic nature of the surveys the presented questions may vary, and so an association is established between the Survey and Question. Each survey must have at least one question (1..* multiplicity), but each question belongs to one and only one survey (multiplicity 1).

**Sensor** data can also be associated with a survey. A survey may incorporate data from as many sensors as wanted, it is also possible not to incorporate any sensor data at all, but this collected data is
exclusive of one survey only (multiplicity 1). The sensor may be of different types, such as, Web sensors or Mobile native sensors.

The last association with the survey objects is done by the Answer entity. Each answer belongs to a single survey but each survey may have a lot of answers (0..* multiplicity). Between survey and its answer is established a composition, this means that when a survey is deleted so will all of its answers. Each answer is produced by a single normal users which if the application allows it might answer more than one time to a survey.

```
Figure 3.8: Data Model or Code Level Architecture.
```

Finally, both the questions and the answers will have some value or Data associated to it. That is why in the data model there is a composition between each of these two objects and the Data object, when either question or answer is removed from the system so must be the associated value.

### 3.5.3 Technologies

In this section it is described which tools were used to build each part of the framework and the motivations behind the choice of those technologies.
Mobile Application

For the mobile applications, React Native was the tool chosen to build the app [60]. React Native is an open-source JavaScript Framework created by Facebook and used to develop mobile applications for both Android, iOS, UWP and even web [61]. This tool allows the development native rendered apps using React and it was used in the development, fulfilling the requirement for multi-OS operation from the section 3.3.1.

The possibility of using the help of web technology to develop mobile applications, the possibility of cross platform application development and the ensured speed and agility for the built applications were what motivated the choice of this technology [62].

When it comes to building React Native applications there are two available options, the React Native Command Line Interface (CLI) or the Expo CLI [63].

The React Native CLI which is managed by the React Native team or community provides a bare-bone development setup, requiring the installation of different tools to build the actual application for Android and iOS. This method does not provide any convenience or utility features and can be very cumbersome especially during the process of setting up some native device features such as the camera, the advantage of the React Native CLI is that allows the programmer to have full control over every part of the code even allowing to write native Android and iOS code and connecting it to the JavaScript code.

On the other hand there is the Expo CLI which was the selected environment to develop the system’s mobile application. This is free third-party service which provides a managed application development workflow, which essential means that it takes a lot of the app development complexity away from the programmer by making available tools to help in various tasks like integrating native device features. The only limitation when using Expo is that it works as a wrapper limiting the programmer to the Expo ecosystem not allowing the fine tune achievable through with the React Native CLI. The need for an easy way to develop and deploy the mobile application (section 3.3.1 and section 3.3.1) was another motivator for picking Expo, this tool make available for its users servers where they can publish and share their apps free of any charge, by running a single command Expo deploys any applications as running it in the development environment. The published app is available in the public page that belongs to the developer in the Expo website, instead of downloading the app from the store as one would in a normal situation, anyone interested in trying the application must download the mobile Expo client (available for Android and iOS) and access the app through the web page mentioned previously either by clicking a button or scanning a Quick Response (QR) code, both available in said page. The application deployment to the actual App Store [64] and Play Store [65] is also made easier with Expo CLI, which provides commands to build deployable packages for both platforms as well as a Android Package (APK) file for direct installation in Android devices. Ultimately, this choice was made to facilitate not only the development of the generic system itself but mainly its extension that this way can be done by any programmer, not needing extensive knowledge in both Android and iOS native code.
Web Client

As for the web client, taking into account the choice made in terms of the framework for the mobile application and due to the easy development requirement from the section 3.3.1, it would only be logical to choose React [66] to build this part of the system. This is also an open-source JavaScript Framework created by Facebook that has as its main objective the creation of front-end web applications by rendering data to the Document Object Model (DOM).

Server

The technology chosen to build the server was Node.js [67]. Node.js is an open-source JavaScript (JS) runtime environment that can run JavaScript across platforms without needing a browser [68]. It is asynchronous and event-driven, easily scalable and ensures high performance due to the used compiler. Besides all these advantages it still unifies and simplifies the web development through full stack JavaScript instead of having a different programming language in the client and in the server [69] as required in the section 3.3.1. On top of the Node.js environment was used the Express.js [70] web application framework, although simple and flexible this framework provides a robust set of features to build web and mobile applications like a huge variety of HTTP utility methods and middleware.

Database

The server connects to a database that was implemented using MongoDB [71]. MongoDB is an open-source document-based database, easy to develop and scale where data is represented as a collection of documents and where the schema does not need to be well-structured. Ad-hoc queries, real-time integration make MongoDB good at changing data frequently. Adding server-side data validation to the already mentioned features make MongoDB a good choice to integrate with the Node.js server [72].

Cache

To implement the cache used to store the users’ credentials during a session was used a Memcached [73] server running on the same system as the server. Memcached is a free, open source, high-performance, distributed memory object caching system. This in-memory key-value storage is of fast deployment and easy use and there are available APIs and packages of this tool in many programming language, such as the one used in this framework, memcached-promisify [74]. This is an alternative version to the most popular fully featured Memcached client for Node.js that allows to handle the memcached methods response in promises instead of doing it on callback functions.

3.5.4 Surveys

The surveys feature is one of the most important features because the generic crowdsourcing system like every other crowdsourcing system relies heavily on the ability of delivering surveys to their user
but also because for this specific a large part of the programming effort was directed to develop this functionality. This feature operates in the mobile application and the server parts of the system.

The mobile application has one or more screens available, depending on the programmer’s implementation, where the users can visualize and answer surveys, this was made possible through the usage of a surveying tool described in the the section 3.8. On the server this features takes a form of a series of endpoints dedicated to handle all the functionalities inherent to the surveys features.

There are available endpoints to serve the surveys to the users (/api/surveys/), to receive and store in the database new surveys from the researcher (/api/surveys/submit), to receive and process the surveys’ answers (/api/surveys/answer) and process the answer if it contains images or large files (/api/surveys/answerimage), there is also an endpoint responsible to provide feedback to the users’ answers (/api/surveys/feedback) and the last endpoint allow the programmer to make available any additional system data that maybe useful when answering a survey (/api/surveys/getInfo).

The most relevant endpoint to the surveys feature is the one that actually selects the survey to be sent over to the users, present in the listing 3.1. When it comes to selecting this questions the system provides two alternatives, dynamic surveys or static surveys. Examples were provided for both of these options but the programmer can and must implement its own functions. For the dynamic surveys the programmer must develop some kind of system of conditions that will return a certain survey depending on the system state as for the static surveys it is must more simple, the system must only be programmer so it returns always the same survey. In the surveysExtension file (.server/extension/surveysExtension.js) can be found the functions to implement the dynamic (dynamicSurvey) and static (staticSurvey) surveys.

```javascript
// Get a survey
router.post('/', async (req, res) => {
    var dynamicRes = surveyExtension.dynamicSurvey(req, res);
    surveyExtension.staticSurvey(req, res, dynamicRes);
});
```

Listing 3.1: Endpoint responsible for sending surveys to the users.

The programmer can integrate only one of these types of surveys by providing an implementation for the respective extension function and leave the other function (survey type not being implemented) blank. In this case, the implemented function should respond to the HTTP request that called the endpoint and made the function execute, sending inside the answer the selected survey, these surveys can be found in the database or in the server’s file system. It is also possible to implement the usage of both survey types, because the functions are part of the extension their implementation is the programmer’s responsibility.

In order to do so the programmer must consider that it must only be sent one response to the HTTP request, in the listing 3.1 it is visible that the function responsible for getting dynamic surveys runs first than the static surveys one and it was implement a variable dynamicRes that allows to passed values from from the first function to the last. This variable can be used to implement a mechanism for using both types of surveys like the one described in the Figure 3.9, where the server tries to get a survey from
the dynamic function and if any of the defined conditions are met this function returns the corresponding survey otherwise the function returns null, then when the static function receives that null value it know that it should respond with the static survey. If the static function receives a value different from null then it knows that a it is receiving a dynamic survey and should forward it to the client in the request request response.

Figure 3.9: Dynamic and static surveys flow.

3.5.5 Data Validation

The data validation mechanisms available consists of a console where researchers can analyse the collected data approve, edit or delete that same data. This console was implemented in the web client part of the system. This application was designed mainly to fulfill this need and is only available for users with researcher credentials. This type of users after logging into the website will be presented with a single page application containing all the data validation methods developed by the programmer to each specific application of the framework, while a user with no researcher credential will be served an empty page. The provided template of the system only presents the researchers with a list of the survey answers and the possibility of excluding them from the database but other mechanisms can be integrated, validation may be include moving data from provisional storage to a permanent one, editing fields of that data or totally erasing it.

This console provides yet another functionality, the administrator's console. This console allows users with administrative credentials to login and manage the registered users. The user management page is presented in the Figure 3.10. The administrator can remove user from the system, deleting their credentials from the database by clicking the red remove button but there it is also possible to promote users from normal users to researchers and vice-versa through the grey switch. This management mechanisms can also be perceived also as a sort of data validation admin another type of validation.

Figure 3.10: Administrator web console.
3.5.6 Sensors

As mentioned several times there is value in collecting data from other sources besides the inputs provided by the users. To explore this untapped potential this framework implements a sensors module. There are available different types of sensors, some may gather information from native hardware components on the device while others may be external to the system like other applications on the device or even third-party platforms from where it is possible to collect data from. The sensor data original purpose is to provide additional information to the researchers but it was found also useful to users. Users may be interested in knowing the content of the collected data, either merely out of curiosity for example for knowing the geographical coordinates or the atmospheric pressure at a certain location but the user may also have a more practical interest in this data like knowing the geocode of their location or track their fitness data like step count, speed, walked distance and spend calories.

The collection of data from the first type of sensors is limited to the device therefore limited to the mobile application but during the development of the system it was debated if the integration of the external sensors should also be implemented in the mobile application or outsourced to the server. With the creation of the react-native-json-forms tools and the easy integration of extensible form elements provided by this tool it was made clear that this could be used as an interface between all the sensors and the system meaning that all sensor data collection could be executed by the app. Other problem solved by moving all sensors to the application side was the usage of API keys for web sensors, most of the external sensors provide their information through an API which requires a key to be accessed, keys are given to users registered in that specific platform to avoid in order to avoid misuse and over use of the provided service. With this feature in the application side is easier to implement mechanisms that allows each user to access sensor through its own key rather than having a single API key to serve all the users in the system.

The forms tool from the section 3.8 creates forms in React Native from pre-implemented components listed and organised in a JSON file, this package is extensible allowing the implementation of new form elements provided that these created element comply with a set of rules defined by the tool in order to enable their integration. The form elements can be developed to fit some need of certain application that is not being done by the default form elements and that is why it was created the SensorElement (./mobile/extension/formElements/SensorElement.js). This element must be declared in the form extension in order to receive all the form elements of the type “ext:sensor” from the JSON file, this objects must also contain a subtype field that will identify to which sensor do they correspond as exemplified in the listing 3.2. In the same directory of the sensor form element there must be a folder containing all the different sensor elements that perform the data collection and a SensorArray file (./mobile/extension/formElements/sensors/SensorArray.js) which in fact contains an object with all the implemented sensors, this “array” will be used by the SensorElement to call the correct component according to the specification made in the sub-type from the JSON file.

```json
{"pages": ["name": "page1"],
```
3.5.7 Activation Notifications

For this functionality to work correctly it is necessary two main components both provided by the programmer in the extension directory (./mobile/extension). One of the components is a JSON file that has the configuration of the type of activation that the system must implement. This file must be located at the mobile application’s extension directory (./mobile/extension/activationJSON.json) and it must contain an array called ActivationModes. In this array there will be listed all the activation modes selected by the programmer. Inside that array must be an object for each of the intended modes, in those objects the mode type must be specified in the mode field. An example of the content of this file and all its possible modes is provided in the listing 3.3.

There were five activation modes that made more sense in the context of crowdsourcing application. (1) The null mode, destined to programmers that do not want to implement this feature. (2) The area mode can be useful for geocrowdsourcing applications, in this mode the programmer must provide an array of areas that when the user enters one them receives a notification. The areas in the array must contain a name, the area in whatever unit the programmer decides, an identifier (id) for that area and its latitude (lat) and longitude (long). (3) The deltat mode allow the programmer to define the time interval delta in which the system must notify the user. The value of the the interval for this periodic notifications can be in any time unit that the programmer sees fit since that is taken into account when implementing the handler for this mode. (4) The activity mode implies the usages of some sort of fitness sensors. Usually these sensors are able to identify the type of activity the application user is doing, for example, running, walking, cycling among others, the type of the activity must be specified in the activity field and in the timing field the programmer must define if the notification must be sent when the activity started, ended or during its execution. (5) The final mode is the time mode in which the programmer car define a period of the day in which the notifications are active, this period is limited by the start and end fields of this mode object.

The second part of this mechanism is a function that handles each activation mode and actually sets the notifications. Because some of these handlers may require the creation of tasks such as geo-fencing for the area activation mode, they must be called during the application’s initialization phase. So, the function activationConfig that belongs to the generic part of the system (./mobile/config/activationConfig.js) is called before the rendering of the application’s main component (./mobile/App.js). As is possible to see in the listing 3.4 this function iterates through the activation JSON file and calls the

```json
"elements": [
  "type": "ext:sensor",
  "subtype": "sensorSubType",
  "name": "Sensor Name",
  "id": "SENSOR_ID"
]
```
activationHandler function (./mobile/extension/activationHandler.js) for every mode selected.

```javascript
const activationConfig = props => {

    // Activation JSON
    ActivationJSON.ActivationModes.mape((activationMode) => {
        activationHandler(activationMode);
    });

};
```

Listing 3.4: Activation handlers usage.

3.5.8 User Authentication

As mentioned in the section 3.3.1 an easy way to authenticate and use the application is one of its requirements and to solve that problem, **OAuth 2.0** was integrated with the system. This open standard
grant to websites or applications access to user’s information from other websites but without exposing their passwords. This way there can be made available to the user a way to register and use the application without needing to create a new account. Because there are many third-party platforms that provide support for oauth this feature was developed as part of the system extension this way the programmer can use the authentication with Google and Facebook already developed in the template extension but also implement authentication with other platforms from the code already written.

In the OAuth 2.0 Authorization Framework RFC page [75] can be found the basic flow of this feature in the system. It starts when the user clicks the button to authenticate with a third-party platform in the mobile application, this will redirect the user to the platform’s page where the user must input its platform credentials. If the credentials are correct the external platform will send a token to the mobile application, this token is then redirected by the mobile application to the server. On the server will be received an HTTP request for oauth specifying the platform and the user token, the server will try to connect to the platform received in the request and confirm if the user has logged in using the given token. If the token is valid the third-party platform will send to the server the user data requested, this information can contain the user name, email and a user identifier. The server uses that information either to register the user in the system or to just to log into it, sending, afterwards, a response back to the application confirming the registration or login.

In the mobile application all the oauth operations are handled by the OAuthButtons component (./mobile/extension/OAuthButtons.js). This extension component is called in the register and main screens and it renders the buttons that will allow the user to authenticate with the external platforms, each button will call a function that will after calling the respective platform authentication function send the request with the token to the server. For the provided implementation the logins in the third-party platform were made using packages from Expo [76] [77]. OAuth usually requires previous registration of the application in the platform to use the authentication services, from that registration will result some credentials such as the application identifier in the external platform that is necessary to send over to the platform when authentication, these identifiers may be stored in the configuration file (./mobile/extension/config.js). To extend this feature the programmer only needs to add the mentioned credentials to the config.js file, add the rendering of the button of the new oauth platform and provide the implementation of the function that will contact with that platform’s authentication API.

On the server’s side is also provided the implementation of oauth for Google and Facebook in the extension directory in the oauthExtension file (./server/extension/oauthExtension.js). Depending on the intended action, login or registration, the mobile app will send the token over to the server through two different endpoints /api/oauth/login or /api/oauth/register, respectively. Before executing the tasks associated with registering and logging the user both this endpoints will call the oauthHandler function that will depending on the platform in question call another specific function responsible for connecting to the platform and checking if the user’s token is valid. In order to extend the server part of this mechanism the programmer must provide the connection to the platform’s API as well as add a new verification in the oauth handler where it will check for the name of the new platform in the request from the mobile application and call the connection function mentioned before.
3.5.9 Multi-Language

It was built in the application a mechanism that allows it to operate in multiple languages by providing the implementation of the generic part of the system in various languages, developing this feature for the extension screen must be done by the programmer. The mechanism is composed of four components.

The first part of this mechanism is a configuration array found in the configuration file in the extension directory (.mobile/extension/config.js). This file exports a config JavaScript object which must contain a field called `languages`, this field must have an array containing the codes of the languages used in the application in string format. This file and array must be modified by the programmer to select which languages will be available for the users, the generic system makes available to the programmer the following languages: German (de), English (en), French (fr), Italian (it), Portuguese (pt), Russian (ru) and Spanish (es).

The second component involved in this mechanism is a component created to interact with the application user, this component is a button that the user click to select the languages in which the application should be presented. The LanguagePicker component can be found in the `LanguagePicker.js` file from the components directory (.mobile/components), it is used in the application login screen (.mobile/screen/MainScreen.js) and it is visible in the the top right corner of this screen taking the shape of a circular button with the flag of the country corresponding to the selected language. By clicking the button the user will be able to go through the array of languages defined by the programmer.

The third element is a React state that will be used to display the application content translated according to the selected language. This state is create in the `MainScreen (.mobile/screen/MainScreen.js)` component using the `useState` hook from React. In this screen this state can be changed by the LanguagePicker component and from there it is sent to the register screen (.mobile/screen/RegisterScreen.js) and to the menu screen (.mobile/screen/MenuScreen.js), from the menu screen this state is transferred to the profile screen (.mobile/screen/ProfileScreen.js), surveys screen (.mobile/screen/SurveysScreen.js) and results screen (.mobile/screen/ResultsScreen.js), each of these three screens is then responsible for passing the states to their respective extension components. This was the state is made available in every screen throughout the application even reaching the extension screens that will be developed by the programmer extending the system.

The state described before spreads the selected language inside the application but by itself is not able to provide the text in the correct language. That is where it enters the last part of this multi-language mechanism, the dictionary. The dictionary is a JSON file available in the data directory (.mobile/data/-dictionary.json) and it contains the strings using in all the generic part of the system in all the available languages mentioned before. This JSON object contains a field for each of the languages available and for each of those fields an object containing identifiers for all the strings and the corresponding string translated to the respective language. The structure of the dictionary can be seen in the listing 3.5.

```
{
  "en": {
    "STRING_ID": "STRING_VALUE_IN_EN"
  }
```

45
The dictionary is imported to a screen where a translation is required and used as shown in the listing 3.6. In the example the dictionary is the object imported from the JSON file, the language is the state containing the selected language code by the user and STRING is the identifier of the string that must be rendered inside the text component. It must be mentioned that because the multi-language is only one of the many features of the system the translations where made using Google Translate [78] a free online tool, so some of the string values’ translations may not be most correct for all the languages.

In order to implement multiple languages in the extension part of the system the programmer can use the first three components of the mechanism but the dictionary containing the strings used in the extension screens must be build from scratch even if reusing the structure of the generic dictionary.

3.5.10 Tutorial Presentation

The tutorial pages take the shape of a carousel presentation accessible in the menu screen through a button located in the top left corner that looks like an “i” inside a circle. When pressed the button will open a modal window with the carousel containing all the pages created by the programmer.

The first step to create this presentation is the creation of the pages themselves. This pages should be regular React Native components but their content is totally customizable by the developer, they should be stored in the application extension directory or even inside an additional folder called presentation (.mobile/extension/presentation), as done in the template extension provided in the generic system. This components should then be imported into the configuration file (.mobile/extension/config.js) and grouped as an array that will be used in the config object under the field presentation as demonstrated in the listing 3.7, where the presentation1, presentation2 and presentation3 are the previously imported components containing the tutorial pages.
By removing the presentation array from the config.js file or by emptying that array the programmer is able to disable the presentation feature and the button will automatically disappear from the menu screen.

3.5.11 User Profile

In order to collect user profile information it was created a profile module in the system. This module operates both in the mobile application and the server.

In the mobile application it that the form of a screen accessible from the menu and in this screen the user is able to visualise and edit all the profile information. This screen is an application component divided in two other components, one of these components is responsible for handling the generic part of the profile while the other manages the profile extension. The generic part of the system is the base information that all users must provide when registering in the application and the profile extension is additional information that may be collected by the framework in order to help the researchers when studying the surveys answers. All the infrastructure to support the generic profile features has already been built into the system but all the methods that implement the profile extension must be developed by the programmer. On the application side the developer must implement not only a way to display this additional information but also a mechanism to collect it. The profile extension component available for modification is called in the profile screen below the generic profile information.

On server side of the system there are endpoints responsible for storing and retrieving the generic user information from the database, which the programmers do not need to concern themselves with. There also available a set of endpoint available that where designed to support the profile extension functionality. These endpoints comprise of functions to get profiling data from storage to be displayed to the user, a way to for the user to submit changes to the profile and even an endpoint so the programmer can define if and how users ranking should be implemented. It was also considered the possibility of collection of these user details through a form similar to the ones used by the surveys module, for this situation it is also available an endpoint dedicated to sending from the server to the user's application the form that will collect the extension profile data.

3.6 Extension

In this section is explored the system's extensibility, this is the capability of the generic system to be adapted to new situations, fulfilling new purposes through the collection of different types of data and implementation of various features.

The generic system developed for this project implements the basic crowdsourcing functionalities which can and should be modified to implement the new intended features. The system was built so that all the file the programmer needs to implement any new feature are inside a single directory, the extension directory. Restricting all the possible system changes to one folder facilitates the programmer's job by directing all the focus to that directory and eliminating any concern about the function of any of the
files external to the extension. There can be found extension directories in all of the three system’s parts (server, mobile and client), each one responsible for containing the specific functionalities to implement in each of the system containers. In the end the extension folder can be seen as an interface between the generic system and the programmer. The extension directories contain some required files with some required functions, these files and functions are the ones imported and called by the generic part of the system therefore their existence is mandatory for the system to be able to run although their implementation is not necessarily required in the case when the programmer does not wishes to implement the features that those elements are destined to enforce. The folder may also contain additional files used in the implementation of the extension features, the existence, naming and structure of these files is entirely up to the programmer and the specific system’s needs.

There are two extension mechanisms used in the system that are worth mention. The first is the extension of the server and the other is the extension of the mobile application and web client, because of its similarities the extension of mobile and client parts of the system will be handled as single mechanism.

The server extension mechanism operates at the endpoint level. The server endpoints are divided according to their functionalities. Apart from the users endpoints, whose implementation is part of the generic system, all the other endpoints do not implement any programming logic per say, instead they call function functions from the extension that are expected to implement the logic that would allow the programmer to achieve the wanted features as outlined in the Figure 3.11.

![Server's extension usage.](image)

In the listing 3.8 there is an example of this extension. Here can be found a survey related endpoint responsible for receiving the user’s survey answer, processing and storing it. This endpoint named `api/surveys/answer` call the `processAnswer` function from the `surveyExtension` that should be found in the extension directory and contain the necessary logic to handle user’s answer according to the programmer’s specifications.

```javascript
1 // Submit survey answer
2 router.post('/answer', async (req, res) => {
3     surveyExtension.processAnswer(req, res);
4 });
```

48
Listing 3.8: Server’s endpoint extension example.

When it comes to the mobile application and web client, the extension is made in terms of the screens and pages. The extensible screens and pages instead of returning visual elements from React and React Native they will be returning components from files in the extension folder. Inside these components the programmer is able then to return the necessary visual components as well as the logical functions in order to created the application’s screens and pages and their respective functionalities. Similarly to the routes in the server the screens and pages are also divided into the different system functionalities and logically a certain mobile and client part should interact with its homologous part in the server. For instances, the screen responsible for displaying surveys in the mobile application should get and send data to the server via the survey related endpoints.

Besides the screens extension the mobile application has additional extension elements required by the App.js file in order to implement the activation notifications features described in the section 3.5.7. All the mechanism explained are depicted in the Figure 3.12.

In the listing 3.9 there is an example of a screen extension found in the mobile application. In the figure there is the SurveyScreen component responsible for displaying to the user all the survey related features and this component returns the SurveyScreenExtension which was previously imported from the extension folder.

More details about how to extend every part of the system can be found in its repository wiki pages [79]. In the wiki pages can be found the default structures of the extension directories along with brief descriptions of the files that compose those directories.

```javascript
const SurveyScreen = props => {
  // Renders the survey screen component from the extension
  return (<SurveyScreenExtension navigation={props.navigation} />
  );
};
```
3.7 Security

To address the raising concerns about security and to fulfill the requirement indicated in the section 3.3.2 the user passwords stored in the database are encrypted. The passwords are encrypted in the server which do not prevent against any network based attack like man in the middle but it would help protect against attacks on the server since password reuse is a common practice. In a future iteration of the system it could be considered full database encryption to totally protect the data against attacks on the server.

Since each user’s password should only be know by the user the passwords are encrypted using themselves as the encryption key and initialization vector. To handle the encryption was used a Advanced Encryption Standard (AES) cipher with a 256 bits key resorting to the Node’s native module crypto. In the listing 3.10 is the server’s encryption function which first creates the cipher key using the createCipheriv function with the aes-256-cbc algorithm, a 32 byte long key and 16 byte long initialization vector, both generated from the password itself. After creating the key is added to the cipher (with the update function) the data that needs to be encrypted, found in the password variable and it also needs to be specified the input and output encoding, UTF-8 and Hex respectively. Finally the encryption is closed using the final function which means that the cipher object can no longer be used to encrypt data.

Before was mention that the key and initialization vector were generated, since these two values must have 32 bytes and 16 bytes respectively and the encryption is made using only the password, this variable may need some modifications to have the required sizes. The generate function was used to transform the user password into a key and initialization suitable value, if the password exceeds the required size it is slice, on the other hand is the password is shorter than it needs to be it is padded with zero until it reaches the intended size, as visible in the listing 3.11.

```
const encryptAES = (password) => {
    var mykey = crypto.createCipheriv('aes-256-cbc', generate(password, 32), generate(password, 16));
    var mystr = mykey.update(password, 'utf8', 'hex');
    mystr += mykey.final('hex');
    return mystr;
};
```

Listing 3.10: Encryption function.

This encryption process occurs at moments of the system operation. The first time when the password is encrypted is when the user register in the system, after being sent to the server when the credentials ought to be stored in the database it is stored not the password but its encrypted version.
The second moment when the encryption functionality is used is when the user tries to log into the system, here the user sends its password to server again and before being compared to the one stored in the database this password is also encrypted, otherwise it would not match the cipher text previously stored in the database.

```javascript
const generate = (password, bytes) => {
  if (password.length > bytes)
    return password.slice(0, bytes);
  else if (password.length < bytes)
    return password.padEnd(bytes, '0');
  else return password;
};
```

Listing 3.11: Key and initialization vector generator.

### 3.8 Forms Tool

During the development of the generic crowdsourcing system arose the need for way to display customizable forms in the mobile application. After some research no tools was available online and open-source that seemed a viable option. There were found some Node.js packages that could be used but they either hard to use, too restricting in terms of the form that they could create, poorly documented or incompatible with the development tool being used by the system.

It was decided to built a forms display and answer tool for React Native from scratch to fulfill this need. The [react-native-json-forms](https://github.com/alexanderschaffer/react-native-json-forms) open-source tool was created for React Native apps developed with Expo because it was the technology being used to develop the generic crowdsourcing system and it was then made public through a Node Package Manager (NPM) package [81]. The package as prove to be very helpful in the development of this system as well as to the rest of the community, having, to the date, over 3000 downloads.

Two important features were prioritized when designing this tool. The first one is related to the survey format and it was motivated by the fact that the surveys in the generic system being implemented by the researchers, who, most of the times, do not possess any programming skills. These surveys are stored as JSON either in files located in the server or in the database and rather then following standardized structure like the one provided by JSON Schema the data is structured according to the SurveyJS tool [82]. SurveyJS is a online web site that provides a drag and drop tool to create surveys that can then be exported to JSON, besides from this clear advantage given by this compatibility to the researchers, the JSON data generated is organized in a simple way so that even with little programming knowledge is possible to identify its content. The second import feature of this tool is extensibility, the forms are composed by elements and although SurveyJS already offers a wide variety of elements the tool user may require to integrate some new custom element to the form. After providing the implementation of its own form elements and specifying them in the form extension the programmer can add those elements to the form.
// Import stuff from react and react-native
import React from 'react';
import { ScrollView } from 'react-native';

// Import component from the package
import { Form } from 'react-native-json-forms';

// Import JSON file with the form
import data from './data.json';

// Import JS file with the extension
import FormExtension from './FormExtension';

// Parent component
const FormScreen = props => {
  // Handle form answer data
  const onSubmit = (data) => {
    console.log(data);
  }

  // Render component
  return (
    <ScrollView>
      <Form json={data} extension={formExtension} onSubmit={onSubmit} />
    </ScrollView>
  );
};

export default FormScreen;

Listing 3.12: Usage example for the react-native-json-forms tool.

The react-native-json-forms tools is installed in a react native application and after imported the
component can be rendered as a self closing component that takes three required props: `json`, `extension` and
`onSubmit`. The `Form` component should be rendered with a ScrollView so that if the created form is too
long the app user can scroll through it as demonstrated in the listing 3.12.

The `json` prop of the form component receives a previously imported JSON file containing the form
to be presented. This file must follow the structure from the listing 3.13, this is also the structure followed
by the form created using the SurveyJS tool as mentioned before. The `pages` array is used in SurveyJS
to implement forms with multiple pages, this feature was not implemented in the react-native-json-forms
tool so the array must contain only one element, if more elements are added they will be ignored. In
the `pages` array can be found the page object which as a field called `name` used to identify each page
but due to the fact that multi-page forms functionality being disabled this field is rather irrelevant. On
the other hand the `elements` array is of utmost importance because it will contain all the elements that
compose the form, each element will be represented by an object which contains the necessary fields
to the well-functioning of that form element.
The elements found in the SurveyJS tool are listed below. These are the basic element with which the researchers can build form from the SurveyJS tool. All but three of these elements were implemented and are available to be used by anyone who installs react-native-json-forms. The element compatible with this tool are **Boolean**, **Checkbox**, **Comment**, **Dropdown**, **Expression**, **File**, **Html**, **Image**, **Image Picker**, **Matrix Dropdown**, **Matrix Dynamic** (not yet implemented), **Matrix**, **Multiple Text**, **Panel Dynamic** (not yet implemented), **Panel** (not yet implemented), **Radio**, **Rating** and **Text**.

During the development of the generic crowdsourcing system and some of its case studies where developed some more element. Elements like **Camera**, **Date Picker**, **Time Picker** and **Range** were also integrated in the react-native-json-forms tools due to their relevance.

The possibility of extension is what give this tool a major part of its value. When rendering a form the tool will process the JSON file containing the actual form and match the elements there describe to the ones listed above, after this the tool will try to match the elements in the files with the ones specified in the extension. This allow both the inclusion of new elements implemented by the programmer using the tool or the overwrite of already existing ones to better fit the needs of each situation. An extension file example is provided in the listing 3.14, this file must contain an array of JS objects, each object must contain its type that must match the **type** then used in the form JSON file and in the **component** field must be found a pointer to the component to be mounted if this extension element is found in the form.

```javascript
import ElementXYZ from './ElementXYZ';
const ext = [{
  type: 'ext:elementxyz',
  component: ElementXYZ
}];
export default ext;
```

Although this tool was build specifically to be integrated in the generic crowdsourcing system developed for this project, it was recognized some value in making it available to be used by anyone, promoting the development of open-source software. The tools is constantly in updating and open to improvement suggestions from everyone, more information about its usage, details about each component and description of other functionalities is available in the wiki pages of the repository.
3.9 Usage

In this section will be briefly explained how the system can be used. The actions in this stage as mentioned will be carried out by the programmer since they required some level of programming skills and knowledge.

The generic crowdsourcing system was made available in a GitHub repository [79]. The public access to the system’s code allows to explore its main and most valuable feature, the fact of being a generic system that can be extended by different people, allowing the easy creation of a crowdsourcing tool capable of solving various problems.

The first thing anyone interested in creating a crowdsourcing application using this system needs to do is to clone the repository. After creating a copy of the code in their devices the programmers need to start modifying the extension files in order to implement the required features for each application. By default the code provides template implementation of the system, the purpose of this implementation is to provide guidance when modifying the system but in last resort it can be used to create a very rudimentary version of the system. This simple system has the structure of any other system, with only the generic profile information, only serves one static survey to the users, the results available simply tells the user how many answers where given on a specific day not providing the details of those same answers and it can be achieved just by altering the JSON file in the server that contains the survey.

When in their local copy of the files the programmer will be presented with three folders: server, mobile and client. These folder contain the code for the Node server, the React Native mobile application and the React web client. For each of these folders the programmer will need to install the dependencies defined in the corresponding package.json files which can be done by running the command npm install in each directory.

The system was designed to simplify its own extension and so in order to extend the system the programmer only needs change the files within the extension folders in each of the directories mentioned above. Because it is an open-source tool the programmers are free to alter any file they see fit, the restriction of all modifications to one folder is only a matter of simplification. The next step is to fill or modify the config.js files, these files contain important values used throughout the system which are stored in a config object.

After defining the important system parameters the programmer must alter the extension files to implement the wanted features. Below are explored the main possible modifications to the system’s containers and the Figure 3.13 help visualize where those changes will that place in terms of the different system containers and their respective components. This diagram follow the same colour schema used and explained in the section 3.4.3 but here are introduced yellow coloured components. These components represent the logical modules where are inserted the JavaScript functions and React/React Native components that the programmer must modify in order to achieve the wanted features.
3.9.1 Server Usage

In the server’s case the config file should contain information like the administrator credentials, the database name and Uniform Resource Locator (url) and the user session cache timeout.

To implement new features or modify the existing ones the programmer should provide its own implementations for the functions called in the server’s endpoints.

In the surveyExtension.js file can be found the functions responsible for serving surveys to the users (dynamicSurvey and staticSurvey), processing the answer given to the surveys (processAnswer and processImage), submitting new surveys to the database (submitSurvey) and getting additional info from the system to be used in the surveys (getInfo). Note that the dynamic and static survey can be mutually used and in that case only the staticSurvey function should respond the HTTP request.

In the resultsExtension.js file can be found the getResults function that should get the results such as previous survey answers from the database and sending them to the users.

The researcherExtension.js file contains functions that get data (getData) to the researchers’ web console but also allow the researcher to edit (editData), remove (removeData) or validate (validateData) that data.

The profileExtension.js is where are located the functions that get (getProfile) and edit (editProfileRequest and editProfile) the detailed profile of the users if the programmer chooses to implement that feature, there is also available the function editRanking to handle any ranking update if any type of mechanism is implemented to rank the users.

In the oauthExtension.js through the functions registerHandler and loginHandler the users can be logged or registered in the system, if the programmer wants to add new oauth platforms it must add an else if block to the oauthHandler function when it will be called a platform specific function similarly to what was made for the platforms already implemented (Google and Facebook). This platform function
must validate a token received from the user with the platform and return some sort of unique user identifier retrieved from the platform. This user identifier must afterwards also be returned in the new else if block created in the oauthHandler so it can be stored in the database and used to identify the user.

The storeAnswer and storeForm functions in the dbExtension.js file provides a possible way of storing answers and surveys in the database but also they can be altered as the programmer wishes.

If multiple surveys are to be used by the system, for example, to implement the dynamic surveys features, these surveys can be added to the surveys folder. Similarly, feedback modules should be added to the feedback folder and used in the feedbackArray.js file. Note that the immediate and differentiated feedback can be mutually used and in that case only the last of the differentiated feedback function should respond the HTTP request.

3.9.2 Mobile Usage

In the mobile application's the information stored in the config file is the oauth credentials, the server url, used languages and also the tutorial pages that should be previously imported in that same file.

The profile, surveys and results should be implemented in the ProfileScreenExtension.js, SurveyScreenExtension.js, ResultsScreenExtension.js, respectively.

If the programmer decides to implement a multi-language application it should be created dictionary entries in the dictionaryExtension.json file for all the texts strings which require translation in the extension screens. The language code will be available in each screen as explained in the section 3.5, if the wanted language is not among the provided ones it should be add also to the generic dictionary (dictionary.json file in the data folder) along with a its corresponding flag to the assets folder.

The OAuthButton.js contains the implementation of the provided oauth for Google and Facebook, this file should be modified if some other oauth platform is to be added or if there is a need to remove any of the provided ones. To remove the existing oauth buttons from the login and register screens simply remove the corresponding credential from the config.js file.

When using the forms tool described in the section 3.8 it is possible to include in the surveys custom elements, besides the implementation of these elements it is necessary to add them to the array in the FormExtension.js file as explained in the section mentioned above.

In the ranking.json file should be specified the user’s ranking levels, if such features is active. This file contains a ranks array, the array is composed of object that hold the ranking level id, minimum and maximum number of points to belong to that level and also the name of the level.

It is also possible for the programmer to incorporate in the system the activation notifications. The selected activation type should be indicated in the activationJSON.json file which contains an example of all the considered activation types and the actual notifications mechanism must be implemented in the activationHandler function in the activationHandler.js file where can already be found the implementation of the area activated notifications.
The **assets** folder should contain the application icon in a file named **icon.png**, this icon should be squared, in the folder should also be the **landing_icon.png** file containing the image to be displayed in the login, register and menu screen. The **presentation** folder must contain the screens to be presented as the different pages of the tutorial found in the upper left corner of the menu screen. In order to remove the tutorial feature it is only necessary to remove or empty the presentation array in the **config.js** file.

### 3.9.3 Client Usage

In the web client the stored values are the **oauth credentials**, the **server url** and the **server's homepage**.

In terms of extension files that can be modified, they are in fewer number comparing to the other system parts due to the limited features implemented in the researcher's web console.

Similarly, to the mobile application the researchers can login and register in the system using oauth third-party platforms. In the **OAuthButton.js** in the client folder can be found the React implementation of the buttons to authenticate via Google and Facebook and that exactly the way described for the mobile application.

The **Extension.css** file is where all the styles used in the extension can be found.

The **ResearcherPageExtension.js** is the main component of this application. It contains the component mounted when a researcher logs into the system. Here should be presented all the data validation mechanisms that are available to the researcher. In the provided example, it is possible to load surveys to the database through a text input box that receives the survey to be uploaded in JSON format and also a button that when pressed displays all the given answers so far. This page can be altered to have all the data validation mechanism that the programmer wishes to implement, all the data handling should be done through the researcher functions mentioned in the section 3.9.1.

The **List.js** is a component used to render lists of data, in the provided implementation example this component is used to render the list of the answer. This component may be reused to present other relevant data lists depending on the system application. The **AnswerItem.js** is the component used to display each answers details but it can be easily modified in order to display all the values of an answer from any other survey.
Chapter 4

Urban Green Spaces Application

The best way to test a system is to put it to use. In this section is presented how the generic crowdsourcing system was used to engage the situation exposed in the section 2.1 where was identified a lack of an efficient way to collect data about Urban Green Spaces. To do so it was deployed a crowdsourcing application capable of mapping UGS and register user’s preferences about these green spaces, a web console was also made available for the researchers to validate and analyse the collected data and a server was brought online in order to support all the back-end need of both these clients. This application of the generic system is available at its Github repository [83].

Members of the Faculdade de Ciências da Universidade de Lisboa (FCUL) who have also encountered the mentioned problem suggested and collaborated in the design and development of an implementation of the system that would help gather data about the UGS in Lisbon. The needs and requirements of these collaborators were taken into account to develop a system able to delivery dynamic surveys, integrate sensors and provide a way to validate the collected data.

Figure 4.1: Mobile application screens.

In the Figure 4.1 are presented the most relevant mobile application’s screens. From left to right: (1) login screen; (2) menu screen; (3) base survey; (4) mapping survey; (5) details survey; (6) profile screen; (7) results screen; (8) tutorial presentation.
4.1 Specific Requirements

In order to fulfill the researchers’ requirements, the mobile application should be able delivery to the users dynamic surveys that may collect different data depending on the users’ state and previous answers. Besides that it should also collect additional profile data from the users and provide them with a way to see all the data they have provided so far.

Since the crowdsourcing results depend of collected data the system should have mechanisms to maximize the amount the data collection opportunities and also maximize the amount of collected data. The goal of this application is collection data about UGS, so the application should be able to detect when a certain user is in a favourable situation for answering a survey and feed crowdsourcing data to the system. In this case, a favourable situation to answer a survey would be the user being inside or near a green space.

Although very valuable sometime the user input may not be the most accurate way to collect a specific type of data. There is data that is easier and more reliable if it is collected from sensors, either if it is geographical location, an image or fitness data. These sensors and external data sources can be found in both the mobile device used to answer the surveys or in the web where can be accessed a large amount of data ranging from even larger variety of subjects.

The last main requirement for this application is the possibility of data validation. The system should allow the researchers that requested the development and extension of the system and will actually use from the collected information to access that information, study it and most important have at their disposal methods to validate that data. To do so it must be provided an interface through which this type of users can interact with the system and the data stored in the database.

4.2 Dynamic Surveys

As mentioned before in this application was used the dynamic surveys features. This feature was designed in the extension to serve the users with the most appropriate surveys improving the quality of the collected data. The user can be presented with surveys requesting personal information if that information was not provided to the system previously (details surveys), one other type of survey that the user may receive will ask to mark on the map relevant points of the UGS where the user is located (mapping surveys) and finally, the users can be asked to answer a survey about their experience and preferences in a specific green space (base surveys). This last type of surveys is a sequence of sub-surveys that will also be presented dynamically depending on the answer that user has been giving.

The dynamic surveys functionality requires communication and data flow between the mobile application that will serve as interface for the user and the server where all the data is house as well as the logic responsible for selecting the servers dynamically. The Figure 4.2 allows to visualise this interaction and explain the mechanism used to quit these two system components synchronised.

In the mobile application’s survey screen, the user can trigger requests for both the base and mapping surveys by clicking in different buttons. On the other hand, the presentation of the details survey is
decided automatically by the system. The details and mapping surveys are fairly simple but the base survey uses a status mechanism presenting different questions depending on the status. This status is a variable that can take values from 0 to \( n \) parts of the base survey, copies of this value are kept in the mobile application as a React state and on the server side as a simple variable while also being stored in the database. This value is passed back and forth between these components through the HTTP requests, maintaining them synchronised.

There is another situation where the status mechanism can be useful, if for some reason a survey is left incomplete, the next time the user uses the application the base survey should start from the beginning, since the user may not be located in the same UGS. Every time the user enters the survey screen the status on the app is set to zero and if the status stored in the database from the previous usage does not match this one the system knows that the user cancelled or did not complete the last survey, resetting the status while also deleting the incomplete answer.

![Diagram](image)

**Figure 4.2: Dynamic surveys basic server flow.**

In the Figure 4.2 it is possible to see the mentioned interaction between mobile application and server. It all start when the application requests a survey, the request is made through the `/api/surveys/` endpoint and the application sends over the type of survey it expects to receive and the current user status on the app. This request reaches the server that based on the received parameters and will check the type of survey requested and for the base survey will check which part of the survey to sent according to the user status. The server then responds to the HTTP requests with the selected survey that is then displayed on the mobile app's screen. In that same screen the user can input the answer to the survey which will be sent to the server through the `/api/surveys/answer` endpoint, then depending on the answer, the user status will be updated and the answer stored in the database. The new status is then sent to the application as a response to the request and there it will be used to update the React state used to keep track of the user status on the mobile side. For the base survey the status will say
if the user has finished the survey and should request feedback from the server or if it should continue requesting survey questions.

In the Figure 4.3 is a flowchart of the server’s `dynamicSurvey` function implementation from the survey extension (./server/extension/surveyExtension.js). The first thing done by this function is to check the type of requested survey, if the application requested a mapping survey the server simply returns the survey if the survey type is base the function will start by checking if the user has already provided is personal details. If not it means that is a new user and that a details survey should be sent otherwise a base survey is selected.

Before selecting the appropriate base survey the system check what is the user status in stored in the server if the status indicates that the user is starting a new survey it is created an object to store this new answer in the database and returned the first question from the base survey, if that is not the case the `getBaseSurvey` is called to evaluate which part of the base survey should be sent. This function is used to return the correct base survey but also to handle some exceptions like marking an answer as finished in the database if a response to the last part of the base survey was received and the special situation of the UGS list question.

At a point in the base survey the user is asked to select the UGS from where the survey is being answered, the user is then presented with a list of UGS from the database from where to select the correct one. In order not to overwhelm the user with all the UGS stored the content of this question is dynamically selected depending on the user’s location.

The selection of a new status made by the server after receiving and processing an answer is made through a set of conditionals that for each status consider all the possible answers and the next status depending on each of those answers. The status take numeric values as mentioned earlier, the value of each status must be kept synchronised in the server and in mobile application so that the interaction
between these component happens without any inconsistency. For future work done in this project could be considered a way to gather these values in just one place in order to avoid said inconsistencies and spare some of the programmer's work when adding or removing status.

All the surveys in the server are stored in the file system and organised into an array. This facilitates the deployment of surveys based on the user status because the survey corresponding to the status zero will be at the position zero of the array, these array are contained inside a dictionary like object that allows to serve multi-language surveys making this feature compatible with the system's multi-language functionality.

The storage of answers in the database is made in objects containing the user who provided that answer the date when it did so, the state of the answer (finished or not) and most important the actual answer data. This data, in the case of the base answers, is stored in an array, so every time the server receives part of the base answer it looks for answer in storage and pushed the new data into the array. This may not be the most efficient way to implement the storage of multi-part answers due to high number of accesses but at the time of the system development this was the selected method because it provided advantages in post-answer processing of the data, it could be considered for improvement in future iterations of this system application.

### 4.3 Area Activation Notifications

This section is dedicated to describing how the system's activation notifications feature was used and extended to adapt to the UGS application. As mentioned in the section 3.5.7 the generic system was designed to support a set of activation types amongst which can be found the activation by area that allows the application to send a notification to the user whenever a certain area is entered. This would be a great activation mode to be used in the UGS application because it is useful for the users to actually be present in a green space when answering the surveys since the answers triggered by the space itself could differ from answer given from a memory about a green space. To integrate this feature with the application the activationJSON file was filled and provided an implementation for the activationHandler function.

In the activationJSON file (./mobile/extension/activationJSON.json) it was added to the Activation-Modes array an object corresponding to the area activation mode. In this object can be found another array that contains a list of many urban green spaces located in Lisbon, each of the spaces is an object containing the `name`, `area`, `id`, `latitude` and `longitude` of each location, as required.

In the activationHandler file (./mobile/extension/activationHandler.js) the `activationHandler` function was implemented in order to send a notification whenever the user entered one of the area specified in the activationJSON. To achieve this were used three libraries from the Expo development tool, the `Notifications`, the `TaskManager` and `Location` libraries. This function will be called for all the activation modes listed in the JSON file so the first step was to create a conditional block that would only run if the activation mode was the `area` mode. The most important piece of this implementation is a Geofencing API provided by the Expo Location library that notifies the app when the device enters or leaves geo-
graphical regions. More specifically the `startGeofencingAsync` method is responsible for starting this service, it does so resorting to an array of circular areas (fences) and task (taskName) that must be called when the device passes over one of these geofences as illustrated in the listing 4.1.

```javascript
1 // Creates geofence around every area in the array
2 // When fence is jumped, taskName is called
3 Location.startGeofencingAsync('taskName', fences);
```

Listing 4.1: Geofencing with Expo’s Location library.

The task must be previously defined with the TaskManager as shown in the listing 4.2 through the `defineTask` function. The definition of tasks must be made in the application initialisation phase so it is because of the line of code from this image that this handler must be executed before the application is actually rendered. The taskName is an identifier for the task and it must be unique string, no two task can have the same name, the task is a reference to the actual task function.

```javascript
1 // Define the taskName task with the task manager
2 TaskManager.defineTask('taskName', task);
```

Listing 4.2: Definition of a task with Expo’s task manager.

In this case the task function is a previously implemented function that sends out the actual notification by calling the `presentLocalNotificationAsync` function from the Notifications Expo library as presented in the listing 4.3. This method sends a local notification instead of the popular push notifications, not need to resort to any external third-party platform to do so. It takes as argument a `localNotification` which is a JavaScript object containing the notification title and body text and for android it can also be included an icon to be displayed in the notification banner.

```javascript
1 // Present mobile local notification
2 Notifications.presentLocalNotificationAsync(localNotification);
```

Listing 4.3: Presenting a local notification on the mobile device.

For the fences array that must be fed to the geofencing function it was not possible to simply provide the areas array from the JSON file because this method is limited to an array of 100 areas. A workaround this limitation was found by creating an array with the 100 areas closest to the user’s location from the the list of areas provided in the JSON. This was done by creating an array with the first 100 areas of the JSON file and then ordering that array by proximity to the user. For the rest of the areas in the file their distance to the user was compared to the furthest area already in the array and if the new area is closer one area (the furthest) with be popped out of the array, the new area pushed in and the array reordered. All this results in an array with the 100 areas closer to the user’s location.
4.4 Device and Web Sensor Integration

From the device were implemented a set of sensors with the support of the Expo’s Sensors library that provides various APIs for accessing device sensors to measure motion, orientation, pressure, magnetic fields, and step count. There were created accelerometer, gyroscope, barometer, magnetometer and pedometer elements which provide numeric value for the corresponding variables, the implementations of these sensors was adapted from the one provided in Expo’s documentation for this library and only work in the devices containing the hardware to support them. From the Expo Location library, also available through Expo, it was created a geolocation sensor that give the user’s latitude, longitude and geocode (address), the first two are obtained from the device’s GPS while the geocode results of a call to an Expo API that translate GPS coordinates into addresses.

Fitness data has also found to be popular and useful in applications probably due to the increase in ways to collect this data, whether it is from smartphones or smartwatches. For this application the user is asked to share the fitness data from the Google Fit [84]. This data can be useful for both researcher and users since it is display when the users answer a survey so they can keep track of their activity while using the UGS application.

When adding this type of sensor to the JSON the programmer must add a “config” field that will allow to specify the data to be collected as it can be seen in the lst 4.4. The programmer must also add to the object a data array that can have any of the following values, "steps", "distance", "calories", "activeMinutes" and "activity". Besides the data it must also be specified the “timeInterval” from which to collect data, this interval must specify the amount of "hours", "minutes" and "seconds". The form element for this sensor will authenticate with the Google API using the user’s credentials and requests the values for the selected activities within the defined time period.

```json
{
  "type": "ext:sensor",
  "subtype": "googlefit",
  "name": "googlefit",
  "config": {
    "data": ["steps", "distance", "calories", "activeMinutes", "activity"],
    "timeInterval": {"hours": "10", "minutes": "0", "seconds": "0"}
  },
  "id": "GOOGLE_FIT"
}
```

Listing 4.4: Google Fit sensor JSON example.

A weather sensor was also implemented, the system gets the information from this sensor through the Open Weather Map’s [85] API. This is an online global weather service that provides weather forecast based on a geographical location and it is necessary a previously obtained API key to access its weather data. To simplify the UGS app user’s job the an API key was obtained and made available for all the users to use in the application’s configuration file.
The react-native-json-forms tool was updated to allow the integration of these sensors. In its previous iteration this tool only allowed form elements that only submitted data after clicking in a submit button, it is now possible to integrate sensor without any graphical interface that simply collect data in the background, without needing user interaction and that submit an answer automatically when they gathered all necessary information.

4.5 Researcher Data Validation

The researcher data validation is available in the web client part of the system, reachable at http://146.193.41.162/ugs/client. The React web application was deployed using and Apache HTTP server and provides a page where only users with researcher credentials can access the data validation mechanisms. After authenticating with the website the researcher will have access to a single page containing two buttons as shown in the Figure 4.4.

The first button on this page is the NEW UGS LIST button is responsible for loading the list of new UGS added by the users. If the user’s are in an Urban Green Space that is not registered in the database they can submit an application to add this new UGS to the system. In order to prevent direct access to the UGS database these new spaces must first be approved by the researcher. By clicking this button the researcher will load the list of green spaces and for each one will be able to visualise the location of the space, its GPS coordinates and the name, area and photo provided by the user. Upon this information the researcher will have available three buttons that will allow to validate the space adding it to the database, delete the application or edit some of its details.

The second button is the ANSWER LIST button that, when clicked, loads the list of the answers given by the users to the base survey. The researcher is able to filter these answer using the available filter checkboxes. These filter cover the majority of the parts of the answer to the survey. Once presented with the answers list the researcher is only able to delete answers from it, for those situations where the answer is clearly not a valid one. It did not made sense for the researcher to be able to edit the answers from the user therefore that feature was not included in this console. Besides this built-in features for data validation the researcher can always download all the answer from the database through the Download Answers also available in this console.
Chapter 5

Evaluation

After implementing the system is necessary to test it and evaluate its correct operation. The system evaluation will focus on the implementation of the generic system and the specification module as well as the results provided in terms of solving, or not, the problem that motivated the development of the system. It will also be evaluated some features common to the majority of the systems and that are considered requirements of the created system, this features are usability and scalability.

5.1 UGS Application Usability

Extraneous events to the elaboration of this project encumbered the system and application testing that was planned to be carried out. Nevertheless, it was possible to perform some tests and collect feedback about the UGS app from a small but miscellaneous group of people.

It was asked the people participating on the test to explore the application features, such as, answering the different surveys, try their profile, visualise the obtained results and test the activation notifications. After testing the application the users were asked to answer a form designed to evaluate the system's usability, this form contains an After Scenario Questionnaire (ASQ) about the task of answering a survey and a System Usability Scale (SUS).

According to ISO 9241-11 standard usability is “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”. There is not only one right way to evaluate usability and so what happens for the majority of the cases is that the choice of the measurement method is left at the discretion of the evaluator.

There are well structured metrics to evaluate both effectiveness, efficiency and satisfaction. In this section is advisable that for this case study the usability evaluation should focus on usability metrics for satisfaction. For these metrics, user satisfaction is usually measured through standardized satisfaction questionnaires and should be considered two possible moments for evaluation. The whole experiment being evaluated is considered a test and each test can be divided into task, satisfaction evaluation can occur after completing each task or only after the completion of the entirety of the test [86].

The group of participants in the test contains people with both technical and non-technical, scientific
and non-scientific backgrounds being the majority of the elements university students. Females represent 52.9% of the group and males are only 47.1%, 58.8% are iOS user and the remaining 41.2% use Android. A 41.2% of people found the application hard to install which may be related to the deployment in the Expo's servers that require the installation of the Expo client in order to access the app, these are extra step that people are not used to in the app installation process. The majority (64.7%) were first time users of a crowdsourcing application while 17.6% have used crowdsourcing applications from two to four times and another 17.6% have used this type of applications five or more times.

5.1.1 After Scenario Questionnaire

Task level satisfaction is measured after users attempt a task, independently of task goal being achieved or not. They are usually five item questionnaires (can vary), based on Likert scale ratings an their main goal is to obtain insight into task difficulty from the participant's perspective. There are many popular questionnaires of this type but the selected for evaluating this application is ASQ [87].

The ASQ questionnaire's items are either treated as constituent items for summative or Likert scales. The graphic scales have seven points items, these 7-point graphic scales are evaluate in terms of "Strongly agree" for 1 and "Strongly disagree" for 7, and a Not Applicable (N/A) point outside the scale. For these questionnaires are usually chosen questions that refer some of the following characteristics : ease of task completion, time required to complete tasks, and satisfaction with support information. The question are derived from these three categories as they have been found the more prompt to influence perception of usability.

In the ASQ about completing a survey, 52.9% of the users answer 2 when asked about how easy it was to complete a survey, 17.6% answered 1, 11.8% answered 4 and 5.9% answered 3, 6 and 7. Overall it is safe to say that the majority of the users found the surveys easy to answer. Not only easy but also fast to answer was the opinion of the users since 29.4% of the users answered 1 and 2 when asked about the amount of time spent answering a survey, 17.6% answered 3 and 11.8% answered 4 and 7. When asked about the satisfaction regarding the available support information 41.2% answered 1, 23.5% answered 2 and 11.8% answered 3, 4 and 7 which allows the conclusion that the support material for is more than enough to answer a survey in the app.

5.1.2 System Usability Scale

Test level satisfaction is only measured at the end of the test session. The main goal of these metrics is to assess the overall ease of use of the system. The selected questionnaire to evaluate the project regarding these type of satisfaction was SUS [88]. The SUS is favoured in many occasions because it is a very easy to use scale that has been found to give very accurate results, it is also considered a reliable and low-cost usability scale.

The creator of this scale debated with the notion that "Usability is not a quality that exists in any real or absolute sense" and so in order to find a solution to this problem, develop SUS as a "quick and dirty" method to assess usability. SUS based questionnaires are 10 item long and give a global view of subjective assessments of usability. This scale is also based on Likert scales and generate
questionnaires with carefully selected questions that should be answered after the test is finished but before any debriefing or discussion, the answers to the questions must be given as immediately as possible, this means that the person answering the questionnaire should not have much time to think about each item. All the items of the questionnaire must be checked and if the interviewee can not reply to one of the items the center point of the scale must be marked. The SUS score is only meaningful if considered the totality of the questionnaire opposite to considering the score of each individual item, which is meaningless. It must be a value between 0 and 100 and it can be obtained by adding the item's individual score and then multiplying the sum by 2.5. It is also worth mentioning that questions with a positive connotation are measured differently from the ones with a negative one.

The UGS application obtained an average value of 67.94 in the SUS which is an average value according to Usability.gov [89]. The main reasons for this score are that 35.3% of the users answer 5 when asked if they would need the support of a technical person to be able to use this app, meaning that the application complexity may need to be rethought. Also, when asked if they found the app very cumbersome to use 35.3% answered 3 which may be motivated by the application flow and complexity.

Besides this data, users reported map related bugs and suggested for the application to be made available in the App Store and Google Play Store. All this information is available in the Appendix A.

5.2 Lichens App

A great proof of concept for this system is the parallel creation of the Urban Green Spaces mobile application and the eFlechten Lichens App [90]. This application was developed by Guilherme Eugénio within the scope of his dissertation and it was also built using the generic crowdsourcing system. Guilherme collaborated in the creation of the generic system and his application for the system was motivated by the effects and dangers associated with climate change and the fact that the current data collection mechanisms are not enough to solve this issue. Based on lichens being sensitive to air pollution and frequently used as bio monitors the application collects from its users information about lichens' appearance and through some predefined models the system is able to calculate some pollution indexes and levels about the region where the data collection happened. The answers are geo-tagged and therefore the calculated metric are associated with a geographical region, the different metrics are then represented as a heat map in the results section of the mobile application.

In order to extract the mentioned indexes it is necessary to do some calculations based on a set of parameters regarding the lichens. The models that do the calculations do not change and neither do necessary parameters to collected and so the system's data collection mechanism is rather simple and static. This application uses the surveys module developed for the UGS system and implements the static survey option. Due to its simple data collection needs the server send the mobile application always the same survey load directly from the database since this was the survey storage method selected for this system extension. The successful implementation of the dynamic surveys in the UGS app and the successful implementation of static surveys by eFlechten proves that the generic crowdsourcing system, in particular the surveys module is able with little programming effort to be implement in two completely different way and have two different sets of features.
The same can be said regarding the system's feedback module. Besides calculating the pollution data derived from the surveys' answers, the eFlechten app gives some feedback to the users. This feedback consists of the calculated results presented in a map format in order to be more attractive to the users. Normal crowdsourcing systems have simple feedback just to ensure the user of the successful submission of an answer or to thank for it similar to what is done by UGS. To insert the model calculations in the feedback mechanism Guilherme created a module able to provide some immediate feedback as well as initiate complex differentiated feedback operations. Exactly like it was done for the surveys module the feedback is used by two different applications to implement two distinct variations of the same feature.

Is the extension and usage of these same two models in two different applications two achieve two different goals that can be used to assure the veracity of the generic system's main feature, from the same generic system is possible to create a variety of applications each with its own purpose by extending modules and creating different versions of the same feature with little programming effort.

5.3 System Evaluation

The generic system evaluation was made according to the ability of the system fulfilling the set requirements.

Code Reuse

It is important for the generic system to encompass as many functionalities as possible to make its extension an easy task. The system is designed to reduce the programmer's effort, this can be proved through the analysis of the code from both the generic system and the UGS extension. While the system consists of 35764 (80.52%) lines of code the extension was developed with only 8654 lines (19.48%).

Sensor Integration

The integration of external data sources such as sensors as achieved in the generic system through the mechanism describe in the section 3.5.6 where it was created not the data collection mechanism itself but a feature that allows the integration of customizable sensors. This mechanism is a part of the forms tool from the section 3.8 which was tested and evaluated in the section 4.4 where are described the sensors developed for the UGS application and how they were used by the framework.

Easy Authentication

In the section section 3.5.8 is described how oauth was integrated in the framework. There were made available two buttons that allow the application users to register and afterwards log into their accounts using credentials from other platforms such as Google and Facebook. This way users can skip some cumbersome steps in the registration and login processes by clicking in the pop-up windows generated when the oauth buttons are pressed. In the UGS case study presented in this document 50% of the users chose this method to authenticate with the system which confirms the preference and successful implementation of the easy authentication requirement.

Multi-OS Operation

It was required that system would be able to operate in devices running different mobile operating systems, this was achieved by developing the app with React Native and the evaluation of this features
was carried out along with the usability tests when the users were asked which OS they where running on their device when testing the mobile application. The most popular operating system was iOS with 58.8% of the users running on it, Android was used by the remaining 41.2% of the users.

**Easy Development and Deployment**

Expo was the key factor in achieving the easy development and deployment due to all the reasons mentioned in the section 3.5.3. The evaluation of the easy development requirement was made by the device sensors components since the React Native application uses Expo libraries to access each of the sensors, other example of the usage of these libraries are the oauth buttons that make use of Expo functions in order to display the pop-up windows for authentication in each of the third-party platform and finally the forms tool from the section 3.8 contain many form elements that where built using the Expo’s version of native components. The successful deployment of the beta version of the UGS application in the Expo servers and the fact that this deployment was done through a single command in the Expo’s CLI is the confirmation that this tool also allow easy deployment of applications.

**Multi-Language Operation**

Although the framework makes available seven languages in its generic code only two where used in the Urban Green Spaces application. Throughout the testing phase of this app, 25% of the provided answers where in Portuguese and also 25% of the users that registered with the app provided non-Portuguese names, these values that there was the need for at least two different languages, Portuguese and English.

**Activation Notifications**

The activation notification mechanism was implemented in the generic part of the system and although it is prepared to accept about five types of activation the actual function that activate the notifications are to be implemented by programmers using the system as mentioned in the section 3.5.7. This feature was put to test in the UGS application with the development of an area activation handler that took a list of areas and created geofences around those areas as explained in detail in the section 4.3.

**Tutorial**

The main purpose of the tutorial presentation is to provide the programmer a way to make accessible support documentation to the users within the app. For the evaluation of this feature the results of the usability available in the appendix A tests must be referenced again. In the After Scenario Questionnaire about the task of answering a single survey it was asked the users to rate following statement: *Overall, I am satisfied with the support information (on-line help, messages, documentation) when completing the task*. Only 11.8% disagreed with the statement while 41.2% answer that they strongly agreed with the sentence and 23.5% answered 2, 11.8% answered 3 and also 11.8% replied 4.

**Dynamic Questions**

The dynamic questions is another functionality that is must be left to the programmer to implement since the logic that leads to the deploy of each survey can and should vary from application to application in order to better fit the requirements. A successful implementation of this feature can be found in the UGS case study where the users are provided with different types of surveys depending on their status, users receive either the application’s base survey in a normal situation, a mapping survey if they select
the mapping function of the app or even a survey dedicated to gathering personal details in the user
have not entered that information in its profile page yet. The base survey of the application beside
deploy questions based on previous answers may also change those questions based on user location.
The mechanism responsible for this dynamism is described in detail in the section 4.2.

**User Profiles**

The need for collecting let to the creation of a profile page in the mobile application dedicated to
execute this specific function. This page displays some generic user information collect at the moment
of the user registration and its complemented by the profile extension. This extension mechanism was
implemented in the UGS application and is explained in the section 3.5.11, it allows researchers to
collect additional user information such as the individual habits that allow to better understand and draw
conclusions about the users relationship with green spaces.

**Security**

As mentioned before the only security measures taken where in terms of password encryption and
separation of features among the different groups of users. These features were test through framework
usage and the correct operation of the authentication module but also the proper distribution of features
and information according to user permissions.

**Scalability**

The technologies and tools used to build both the generic system and the UGS application are not
exotic nor new. The different parts of the system use common programming languages and frameworks
therefore the scalability of the application and the system will not differ must from all the systems alike.

Systems built with the same tools will be subjected to the same type of limitations, which are imposed
by those same tools. For example, since the mobile application was deployed on the Expo servers its
scalability is restricted by the amount of server power that Expo allocates for each published application.

The server is Node.js and Express.js application running on Apache and therefore constricted by the
technologies limitations. A possible way to scale the server is horizontally by distributing the server load
for different instances, a possible division of this load would be by geographical location. In addition, it is
also important to consider that for the UGS application in particular the server is not heavy computation
or complex operations which means that me majority of the server power can be dedicated to serve a
larger quantity of clients.

The database in MongoDB also has available some scalability mechanisms available. Similarly to
the server the database could scale horizontally and geographically, having different replicas of the
database for different regions. MongoDB also supports sharding that consists of distributing data across
several machines and facilitating high throughput operations with large sets of data. Using this method
the database is only constricted by the number of machines that can be connected.

There is room for improvements and optimizations that could be made concerning scalability es-
pecially when it comes to the interaction between the server (Node.js) and the database (MongoDB).
Because this systems innovation is found at programming and architectural level this issues were not
give a high priority nor explored in detail.
Chapter 6

Conclusions

6.1 Achievements

With this project it was possible to create a framework able to fill a gap in the world of crowdsourcing applications. This gap is for the existence of a tool with the ability of easily developing a crowdsourcing system that allows full-stack customization. The system makes possible the creation of crowdsourcing applications dedicated to any subject by integrating features that prove to be more adequate and helpful to achieve each application’s goal.

The implemented system is composed by three main parts, a mobile application used by the normal users to collect the crowdsourcing data, a web client console application where the researchers responsible for creating the system are able to analyse the collected information and a server that answers the calls from both of these applications. The server should manage the data read and wrote to the database and may be responsible for some computation required to satisfy some of the requests from the clients.

The existing tools to create systems like this offer some limitations that the generic crowdsourcing system is able to overcome. The system stands out from existing technologies for implementing innovative features such as dynamic surveys, integration of external data sources like sensors and data validation mechanisms. When extending the system the programmer is able to select the desired features and also provide its own implementation for those features as well as for new ones. The mentioned full-stack customization comes from the possibility of the extension of all the system parts, the application components can and should be extended to implement the screens where the user can input the crowdsourcing data, the web client must be altered in order to implement the functionalities required for data analysis and validation by the researcher and the server’s endpoints can be modified to serve all the back-end data and computation required for all those features to be used by the system. This gives the system programmer and consequently the researchers complete control over the system, the data it collects and the processing done to that data.

The tools and technologies used in the actual development of the system were chosen in order to facilitate the fulfillment of all the set requirements. JavaScript is the dominant programming language in
this system since the server was built in Node.js and Express.js, the web client and mobile application where created resorting to the React and React Native frameworks, respectively and the database is MongoDB to take advantage of its document-based characteristics. The mobile application was developed using the Expo framework and platform which also facilitated the publication of the app online by doing it onto its servers. The server and web client parts of the system were deployed on a private Ubuntu server running Apache.

In particular this system should be able to solved the real-life problem proposed by researchers from FCUL. This problem expects the system to facilitate the development of a crowdsourcing tool that collects data about Urban Green Spaces. There is a lack of documented mapping and general information about these spaces and the users’ perception of them. The researchers and practitioners pretend to collect data about the UGS general characteristic such as the existing elements in those spaces, both natural and artificial. It is also of the interest of these researchers to collect profiling data about UGS visitors, and finally information about the relationship the visitors establish with those spaces. The relationship data should allow researchers to understand which UGS are preferred and which are avoided, also when and why are these places visited and who visits them. These was accomplished through the development of an extension/application of the system. This extension implements a series of surveys deployed dynamically to the user, the dynamic feature was customizable in the mobile and server extension files and was based in a user status system that allow to assess when is the most convenient moment for a user to answer each specific question, this assessment is based on the user’s previous answer and other factors such as user location.

The development of other applications from the generic crowdsourcing system by other collaborators of the system worked as a proof of concept for the ability of the framework to have different usages and the possibility of easily integrating extension modules. The eFlechten lichen app uses simplified version of the surveys module created for the UGS application where instead of serving dynamic surveys, it presents always the same static survey. On the other hand, the UGS application uses an immediate feedback which is a part of the feedback module developed for eFlechten that besides immediate feedback allows the implementation of complex differentiated feedback methods.

The surveys are a cornerstone for crowdsourcing application and so this task received additional attention during the development process. The lack of a tool that allowed the system to explore the full potential of the surveys feature led to the development of a tool to display and answer surveys on the mobile application. The React Native JSON Forms is Node.js package that allows to create forms in React Native from a JSON file containing the form description. This tool is compatible with forms created using SurveyJS, allowing people without programming knowledge to design surveys. The SurveyJS tool only make available the most common types of input in form but in the React Native JSON Forms can be found some extra features that allow the integration of extensions containing new form elements.

In conclusion, the objective of this project was to develop a generic system able to create crowdsourcing applications that may vary in subject but are able to implement features different from the ones available in other applications and systems of this type. The system was used to solve the UGS problem, through the development of an application able to collect reliable data about users’ experiences in green
6.2 Discussion

The sections below present two issues that arose during the implementation of this project. They do not relate directly to any of the technical part of the project but rather to the experience of designing and developing the generic crowdsourcing system. These issues are more philosophical than practical and are not restricted to this project, they can be apply or found in almost any technological project but on a personal note, they triggered some reflection and consumed some of the time allocated to this dissertation.

6.2.1 The Generic Problem

The first issue is the one more related to this particular project because it is bonded to the main system feature, its capability of being generic. A concern that was constant during the development of the entire system was to make it generic. Because the system was being developed by to collaborators each trying to built a completely different final application, while developing each system component it was discussed what features should be included in the component in order to fit idea for both final applications. Many time instead of a compromise from one or both parts involved it just seemed easier to totally remove a certain feature from the generic part of the system and leave it to be implemented by the programmer in the extension if it is necessary. Following this path would lead in the worst case scenario to an empty file where the programmer could implement the entire system. This resulted in the rewrite of a considerable amount of code in order to implement the crowdsourcing essential features that could be extended to adapt to different situations. This could only be achieved through compromise from the collaborator and reshaping of the initial idea that each one had for their applications.

6.2.2 Tech-centrism

Every technological project is created with a purpose, in this case the lack of generic crowdsourcing system capable of implementing more customizable features mentioned before and the nonexistence of a reliable and easy way to gather and process data about Urban Green Spaces motivated the creation of this system and its subsequent extension to create the UGS app. For developers who actually implement project and who may not be the owners of the idea for it can be some time difficult to keep their priorities in check. Most programmers are passionate about technology and when it comes to programming there are so many options either in terms of languages, frameworks, methods or other concept from a seemingly endless list that it may become overwhelming and hard to prioritize what really matters. There are some common behaviours that are responsible for this lost of focus, some are caused by the attachment of the programmer to a specific language which can impede to see beyond that language capabilities and look for an alternative that might benefit the project, on the other and there are those who are always looking for developing the most recent languages and frameworks and are willing to spend
large amounts of time migrating code, the waste of time can have other origin such as implementing unnecessary automation mechanisms or code rewriting in order to make each line as clever as possible. A solution to this tech-centrism or at least to prevent that it harms the project is a responsible approach where every decision should be discussed and well thought in so the pros and cons for both project and programmer can be weighed and the best decisions can be made. During the development stage of this project this tech-centrism and lost of the project’s initial purpose was felt due to the fact that the mainly used programming language (JavaScript) and its frameworks where all new to the developers. This triggered a feeling of need to explore the potential of the new technologies, other bad practices where also in play such as unnecessary spent time in optimizations and automation.

6.3 Future Work

Regarding future work there must be considered two different types of ideas to improve this project. There are some ideas destined to add value to the generic system but there are also ideas to enrich the UGS application of the system.

The biggest idea to implement in future iterations of the generic crowdsourcing system it is not a new idea but rather the improvement of an existing feature that is making easier the usage of the integrated features. The system integrate as many crowdsourcing features as possible in order to try to accommodate as many situations that programmers may need in their applications. This features should be made easier to access, ideally through a single configuration file, leaving as less work as possible to the programmer. It could also be interesting to integrate some business intelligence tool with the web client researcher console in order to make data analysis and conclusion drawing easier for the researcher.

As far as the UGS application goes there are also improvements that could be made such as integration of machine learning in order to create some automatic data validation mechanisms, the creation of a sensor board containing physical sensors to deploy in the green spaces and collect meteorological and pollution related data to be integrated with the rest of the collected information and also the introduction of some more dynamism to the surveys module of this extension. The surveys could be improved by the creation of an events system, that should be concert with the researchers teams and based on the user’s activity events would be created and the surveys deployed accordingly.
Bibliography


[54] Zoho forms official page. Available at: https://www.zoho.com/forms/, last accessed in 02/10/2020.


Memcached official page. Available at: https://www.memcached.org/, last accessed in 22/12/2020.

Memcached node.js client repository (memcached-promisify). Available at: https://github.com/flightstats/memcached-promisify#readme, last accessed in 22/12/2020.


Expo google authentication package. Available at: https://docs.expo.io/versions/latest/sdk/google/, last accessed in 30/11/2020.


Open weather official page. Available at: https://openweathermap.org/, last accessed in 04/12/2020.


Appendix A

Usability Statistics

In this section are presented the usability form statistical results in graphical format.

Gender:
17 respostas

52.9%
47.1%

M
F

Age:
17 respostas

94.1%

< 18
18 - 25
26 - 40
41 - 65
> 65

A.1
Mobile operating system:
17 respostas

- 58.8% iOS
- 41.2% Android

Your mobile device brand:
17 respostas

- iPhone: 3 (17.6%)
- Samsung galaxy ≤10: 1 (5.9%)
- Samsung: 1 (5.9%)
- OnePlus: 1 (5.9%)
- Huawei: 2 (11.8%)
- Apple: 5 (29.4%)
- Apple: 2 (11.8%)
- Android: 1 (5.9%)
1. Overall, I am satisfied with the ease of completing the task of answering surveys about UGS.

17 respostas

2. Overall, I am satisfied with the amount of time it took to complete the task of answering surveys about UGS.

17 respostas
3. Overall, I am satisfied with the support information (online help, messages, documentation) when completing the tasks.

17 responses

1. I think that I would like to use this app frequently.

17 responses

2. I found the app unnecessarily complex.

17 responses
3. I thought the app was easy to use.
17 respostas

4. I think I would need the support of a technical person to be able to use this app.
17 respostas

5. I found the various functions in this app were well integrated.
17 respostas
6. I thought there was too much inconsistency in this app.
17 respostas

7. I would imagine that most people would learn to use this app very quickly.
17 respostas

8. I found the app very cumbersome to use.
17 respostas
9. I felt very confident using the app.
17 respostas

10. I needed to learn a lot of things before I could get going with this app.
17 respostas

About the app’s layout... Was it user friendly?
16 respostas

A.7
<table>
<thead>
<tr>
<th>Suggestions</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple troubles with downloading, unfortunately, too heavy and probably not working on the older phones. Didn't get to the survey due to repeating errors.</td>
<td></td>
</tr>
<tr>
<td>Map wasn't working very well, but I enjoyed it.</td>
<td></td>
</tr>
<tr>
<td>Sometimes when clicking it didn't identify.</td>
<td></td>
</tr>
<tr>
<td>I would like the app to be available on the App Store.</td>
<td></td>
</tr>
<tr>
<td>Era melhor a app estar disponível na PlayStore em vez da Expo.</td>
<td></td>
</tr>
<tr>
<td>Não existia na App Store o que dificulta o processo ter de usar o Expo.</td>
<td></td>
</tr>
</tbody>
</table>