CoopMap: Democratizing Community Maps using Crowdsourcing

Ricardo José Cordeiro Xavier

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. João Pedro Faria Mendonça Barreto

Examination Committee

Chairperson: Prof. Francisco António Chaves Saraiva de Melo
Supervisor: Prof. João Pedro Faria Mendonça Barreto
Member of the Committee: Prof. Jacinto Paulo Simões Estima

December 2019

Dedicated to my family...

Acknowledgments

I would like to thank Professor João Barreto for supervision and guidance during the development of this project. It would not be possible without his support.

I would also like to thank my family and my girlfriend for providing the motivation and assistance I needed to finish this thesis successfully.

I want to express my gratitude to Instituto Superior Técnico, which was also responsible for the success of this project.

To finish, I want to give some credit to my colleagues who supported me when I most needed it.

Resumo

A informação é um elemento vital nos dias de hoje, e os mapas têm um importante papel na representação de certos tipos de informação. Existem inúmeras aplicações baseadas em mapas onde o utilizador pode encontrar informações sobre a via pública. No entanto, essas aplicações apresentam limitações, o que pode levar a que nem todos os utilizadores consigam encontrar os detalhes que procuram.

Nesse sentido, surge a ideia para esta tese, que consiste em desenvolver uma plataforma *mobile* e *crowdsourced*, onde os utilizadores poderão criar mapas e classificar ruas consoante atributos específicos. Deste modo, várias pessoas poderão criar e/ou usufruir de mapas personalizados, adaptados às suas necessidades, exigências e interesses.

CoopMap é uma plataforma desenhada para satisfazer esta necessidade de informação de algumas comunidades. Foi desenvolvida para telemóveis Android e usa a plataforma OpenStreetMap para obter dados sobre as ruas da cidade de Lisboa. Nesta aplicação, o utilizador consegue, em poucos passos, criar um mapa com determinados atributos, assim como juntar-se a mapas criados por outros utilizadores. Já dentro do contexto de um mapa, o utilizador consegue submeter opiniões relativas às ruas, tendo em conta os atributos definidos anteriormente e ainda visualizar o resultado dos contributos da comunidade.

Para esta tese, foi construído um protótipo que foi sujeito a avaliação, de modo a perceber se a solução proposta conseguiria responder às necessidades do utilizador. Foram realizados múltiplos testes *benchmark* e um teste usabilidade. Os resultados obtidos permitiram concluir que a plataforma cumpre com os requisitos, existindo margem para eventuais melhorias.

Palavras-chave: Crowdsourcing, Volunteered Geographic Information, OpenStreetMap, PostgreSQL

Abstract

Nowadays, we depend on information more than ever, and maps are one of many other forms in which we can represent that information. As users, we can find many map-based applications that provide specific information about public spaces. However, we may not find every detail we were looking for in those applications.

The idea of this thesis is to develop a crowdsourced and mobile platform where users can create a map and classify streets according to attributes that are relevant to a given community. This way, any map can provide any type of information, as long as there is a group of people interested in it.

CoopMap is a platform designed to fulfill this gap of information between communities and their needs. It was developed for Android devices and uses OpenStreetMap to extract data about the streets. Within a few steps, this platform allows users to create their map with a set of attributes. They also can join other users' maps. When inside a map's instance, users can contribute by submitting spatial judgments about streets or they can just visualize the results from other users' contributions.

For this thesis, we have produced a prototype that was evaluated. We wanted to know if the proposed solution could answer to the necessities of the user. We have performed some benchmarks and a usability test. From the results obtained, we were able to conclude that our platform meets the requirements. However, this does not mean it cannot be improved for future work.

Keywords: Crowdsourcing, Volunteered Geographic Information, OpenStreetMap, PostgreSQL

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A driver usually wants to know which is the fastest way to get somewhere and there is a lot of applications that assist users in that matter like Waze [1]. However, people are requiring other sorts of information which are not that accessible. An example is the disabled people that barely have any information about the streets they are going to pass through with their wheelchairs. Imagine the case where a disabled person must reach a certain address using his/her wheelchair and the sidewalk is in a bad state that might result in a trapped or broken wheelchair. What if there was a way of reporting this street's condition to other people in the same physical state so that they can avoid passing by this specific street? Mobasheri et al. [2] created a solution for this concrete scenario. However, this solution only targets one problem among many others. Another possible and similar situation can also be applied, for instance to cyclists, where they benefit from streets with low traffic and the floor in good condition. A typical person that goes for jogging usually enjoys running in a track with the pavement in a decent state (to avoid getting injured), not too crowded (to not disturb or get disturbed by other citizens) and probably with a good air quality index (low pollution levels). Many other aspects promote a good jogging activity, however, we only described three to demonstrate how complex a spatial assessment can be. A map gathering all these sorts of information would be appreciated by the jogging community.

Some solutions innovate by using crowdsourced map information to solve particular problems ([2], [3]). These types of solutions aimed at solving one single issue and are limited to the developer's perspective of the problem. Some platforms offer the user the possibility of creating custom maps with any type of information (Google MyMaps and Wikimapia[4]). However, they do not provide proper mechanisms to collect, in a crowdsourced fashion, georeferenced data that complement the base information of the underlying map.

Building one such application from scratch requires a significant amount of knowledge that most of the users do not own. A platform accessible for any non-programmer user that allows the creation of custom maps would lead to a variety of maps with every kind of information. This means that there would be maps today not available anywhere else, and in which the user can rely on when he needs.

This thesis aims at developing a crowdsourced mobile application where users can create and have access to any kind of map without requiring any special skill. These maps have a set of attributes defined at creation by the user. When the map set-up is completed, the user can start picking streets or just segments (small parts of a street). After these segments are selected, he classifies these areas regarding the attributes defined during the map creation. Hereafter, we call each classification provided by a user on a given map segment as a *contribution*. Other users can join the map and submit contributions as well. The application provides a way of visualizing the results from these contributions. Consequently, the communities can visualize, interpret and use that specific map information for personal use.

We decided to create a platform using a client-server approach. The client consists of a mobile application developed to the Android operating system, which allows our users to interact with maps in two different ways:

- **Visualization Mode:** In this mode, they can visualize an instance of a map (belonging to a specific community) and obtain information about other users' contributions. We provide an intuitive color system, which gives information about the quality of the streets based on contributions performed by the users of a certain community (contributions made directly for that map's instance);

- **Contribution Mode:** In this mode, users can obtain information about which streets or segments (parts of streets) have already been classified by themselves. This mode allows the user to select streets (or segments) and submit contributions (considering that map's attributes).

The back-end server was implemented using the gRPC technology and it is mainly responsible for processing the multiple contributions coming from users.

Regarding the database, we use two distinct databases near the server. OpenStreetMap copy database is responsible for maintaining the metadata from the streets which comes from OpenStreetMap [5] platform. This data is used used to populate or update the second database. The second database is called the CoopMap database and contains the data to be accessed in run-time by our users.

After implementing our solution, we performed a formal evaluation of our platform. During this evaluation, we made multiple benchmarks and a small usability test. From the results obtained, we were able to conclude that our platform meets the requirements. However, this does not mean it cannot be improved for future work.

The rest of the document is structured as follows:

- **2. Background and Related Work:** discusses the background work done relevant for this thesis solution;
- **3. Proposed Solution:** describes the decisions we made regarding the solution;
- **4. Implementation:** presents our technical decisions to implement the proposed solution;
- **5. Evaluation:** describes our formal evaluation of the platform where we also analyze the results;
- **6. Conclusions:** discusses the achievements of this platform and outline some work to improve the platform;

Chapter 2

Background

2.1 Crowdsourcing

2.1.1 Introduction to Crowdsourcing

There are many definitions for the concept of **crowdsourcing** [6] because the term *crowd* differs from perspective to perspective as well as the definition of *task*. For instance, Geiger et al. [7] describes crowdsourcing as an open call to an indefinite network of people to perform a function that previously was performed by the employees of a company or organization. Meanwhile, Howe [8] defines it as the concept of taking advantage of crowds by using a set of approaches to make those crowds perform tasks.

If we join all these definitions to create one we would get a general but easy to understand definition like the one presented by Daniel et al. [9]: Crowdsourcing is the concept of outsourcing a piece of work so the crowd can perform it in exchange for some kind of payment or benefit. The crowd in this context are the users of the platform that are looking for work to perform or also called the *workers*. The people interested in the work performed by the crowd are called *requesters*. The work assigned to a certain *worker* is called a *task* and the results delivered by the crowd are called *outputs*.

In the context of this thesis, a user can play the role of *requester* and *worker* at the same time. Thus a user can provide or receive information for personal use. The role of the *requester*, in this case, is a user that is interested in specific information but does not assign *tasks* directly to *workers*. Instead, *workers* should have the initiative of voluntarily choosing the *tasks* they wish to contribute to.

2.1.2 Traditional Crowdsourcing vs Participatory Sensing

Participatory sensing uses embedded sensing (e.g. mobile phones) and communication (cellular or WiFi) infrastructure to collect data ([10] [11]), resulting in a reduction of the user's involvement. When a user is moving, it is possible to get information about his speed through the accelerometer of his smartphone or other wearable mobile device and send that information to a running server to process it. This is a brief example of a sensing activity. **Traditional crowdsourcing**, on the other hand, implies

users manually performing tasks. This second way of crowdsourcing is more suitable for tasks that humans perform way better and faster than computers while **participatory sensing** is more appropriate for repetitive and straightforward tasks.

A crowdsensing *task* can be triggered manually, automatically or based on the context. A mobile phone user can allow an application to use certain sensors and, this way, the task is performed automatically when the same application is running. Some tasks are only performed when the user wants to, and there are others which are in standby until something in the environment triggers them.

Traditional crowdsourcing has some issues that are solved by the participatory sensing. However, other types of problems arise when using sensing. We now discuss some of the most relevant advantages and disadvantages, regarding this project, of both ways of collecting data.

Participatory sensing depends mainly on mobile phone's sensors as described previously and therefore there are some concerns we need to have in account [12]:

1. **Privacy concerns**. Since we are using sensors, the user may not be entirely aware of which kind of information is being provided. Consequently, it may raise some trust issues between the smartphone owner and the application;

2. If the application is not well designed, it may **consume a substantial amount of resources** while using the sensors. Some applications require a continuous use of the smartphone's sensors without owning any mechanism to save battery;

3. The information we can get from sensing is very **limited to the kind of sensors** we have. However, there is ongoing research on how to optimize the use of the smartphone's sensors. For instance, NoiseTube [13] is an application where we can measure noise pollution levels using the microphone.

The main advantage of participatory sensing is the possibility of gathering lots of information requiring the minimum human effort. However, this does not imply that the user, once in a while, will not interact with the application and provide some information manually that the sensors were not able to. A hybrid model (meaning it applies both ways of collecting information) is possible to achieve.

Traditional crowdsourcing requires human effort, which might be a big drawback since individuals need to be motivated to perform some sort of tasks [14]. This motivation can exist in varied ways, which are going to be debated more ahead in this section. But since it calls for user attention, the benefit the user obtains from performing the task must correspond to the cost of performing it. This is, in many cases, the biggest disadvantage. Besides this, there is another disadvantage we feel the need to refer to, which is: traditional crowdsourcing is more susceptible to human mistakes and as a result requires detailed and better-defined quality control measures to avoid using inaccurate data. The main advantage of this type of crowdsourcing is that we can get any type of information as long as anyone is willing to provide it.

To illustrate the hybrid model, Waze [1] is a well-known mobile application whose purpose is to help users with routes, traffic and other pertinent information to drivers. It applies both ways of collecting data. Sensors like GPS, accelerometer and gyroscope are used to gather user information regarding

the position, velocity and direction of the user. But it also uses information that is reported by users directly in the application, like the location point in a map where a car accident happened or where a stop operation is occurring, so other users can be aware.

As refereed above, participatory sensing does not mean that a user will not interact with the application once in a while. However, the goal of a system that applies this approach is to absorb most of the information through sensors. Our goal is to make users manually gather most information. Some sensors may be used to help the user with his activity. However, their outcome is not meant to be shared with other users. Because of these two points, we classify our approach as traditional crowdsourcing.

2.1.3 Quality Control of the Crowdsourced Data

In the crowdsourcing topic, one of the main challenges is to have a proper quality control of the data delivered by the *crowd*. We cannot be aware of the intentions of a certain *worker* when he is submitting his results for a certain *task* ([15], [16]). He may be trying to contribute but he may be also trying to corrupt the platform with wrong data. Other matters can affect the output data that we need to take into account like the personality or the mood of the *worker*.

Daniel et al. [9] provide a detailed taxonomy to understand and classify the state of the art in quality control techniques for crowdsourcing. They divide it into three core aspects.

The first aspect is the **quality model** and describes the attributes and dimensions necessary to evaluate a crowdsourcing service. Those dimensions are:

- The **data**, in this case the output data, in order to be classified as good quality data needs to:
    1. be **accurate** (corresponds to the expected result) or ”correct”;
    2. be **consistent** in relation to the other user answers (depending on the task);
    3. be available in useful time to the *requester* (**timeliness**);
    4. apply the use of some **ad-hocs attributes** that are able to capture task-specific properties like *coverage* and *conciseness* in the *task* of summarizing texts;
- The ***task*** proposed by the *requester* for the *worker* to perform needs to have:
    1. a clear and complete description of the work (**clarity**);
    2. a **robust** (avoid cheaters), **easy to learn** and **easy to use** interface;
    3. **incentives** (e.g. monetary) to motivate the *worker*;
    4. have a terms and conditions that guarantees privacy, IP protection, information security and be in conformance with the laws and regulations (**compliance**);
    5. a good **performance**, which means having low cost and be time efficient.
- The **actors** involved in the tasks must also possess some attributes in order to be considered good:

1. The **requester**, when dealing with the *users*, should be *communicative*, *generous*, *fair* and *prompt*;
2. The **worker** should have a **profile** with a significant amount of information about himself, some **credentials** showing he has the right skills to execute the *task* (a certificate) and if he is an **experienced** *worker* he should have something that proves that experience like a badge;
3. The *workers* might also be grouped working together and, in that case, the **group** should be **diverse** (individuals with different attributes and skills), **available** (ready to work on the *tasks*) and **non-collusiveness** (the workers won't share information neither make joint decisions).

Some attributes mentioned above are trivial to **assess** manually or automatically, but others may be more complex. Daniel et al. [9] present three ways of assessing the tasks:

1. **Individual assessments** imply other humans (e.g. *workers*, *requesters* or experts) assessing the *tasks* such as rating an *output* delivered by someone else;
2. **Group assessment** consists in a group of people (typically *workers*) forming an assessment, usually by voting or by peer review;
3. **Computation-based assessments** do not require human hand and usually work well for activities like comparing an *output* with the ground-truth of a *task*.

After describing briefly the assessment techniques based on a quality model, the next step is quality assurance. Only by assessing we are already making a positive impact on the data quality.

Improving the data quality is the most intuitive step we can take to improve the overall quality. This means improving the quality of the *outputs* delivered by the *workers*. There are some ways of doing this. We can filter the *outputs* by rejecting the bad ones, clean eventual errors or even ask for another *worker* to improve it.

Selecting people can help us assign *tasks* for the right *workers* with a specific range of skills that might be useful for those specific *tasks*. We can also reject unqualified *workers* or even recommend *tasks* to the good ones.

Incentivize people is a good way of motivating the *workers* to get the job done. There are two kinds of motivations:

- **Extrinsic motivation**: corresponds to the idea that if a *worker* works harder, the better the reward will be. This kind of motivation is usually associated with prizes, money rewards or even promotions inside the crowdsourcing platform. A very common approach is gamification which consists of adapting a user application to follow a game workflow. The most iconic elements of this approach are the level and rewarding system;
- **Intrinsic motivation**: the *worker* knows that the *task* he is doing will have a positive impact on the community or himself so he feels he must get the work done correctly. Good examples are sites with a share purpose like Wikipedia.

Training people and getting them ready to execute their *tasks* is a way of assuring data quality. Teaching *workers* and giving them feedback motivates them to perform better.

The ***task* design** is also a key element to keep the *worker's* attention on the work he is performing. Some good ways of improving the design are to lower the complexity of the *task* or decompose it into smaller ones. Separating the duties, where a *worker* can do a certain type of job while other *worker* does a different job, but still related to the same *task*, helps in a way that a single worker does not need to learn how to perform different activities.

The last part of improving the data quality is **controlling the execution**. Many times it happens that a certain *task* is interrupted or left in standby and the crowdsourced platform must be able to detect these cases and shut down the task or transfer it to another *worker*. Otherwise the *requester* would not get the result in useful time.

Since collaborative mapping is part of the proposed solution of this project, there are some aspects of quality control which are more important than others. This means that we should not put the same weight in every component described during this section. As stated before, the community interested in a map is composed of individuals who are at the same time *requesters* and *workers*. This fact will influence some of our choices regarding data control. When analyzing the quality model, this platform must consider the following concerns:

- Since the data we are dealing with is slightly influenced by the *worker* opinion and thus not being entirely objective, it is not required to be that consistent and accurate. Slightly different answers may be acceptable.

- Regarding the *tasks*, in this situation, they are pretty simple and straightforward to perform, even though malicious intentions must have low impact.

- As a community, the actors involved have the same role. The *tasks* do not require an extensive set of skills and therefore no kind of credential, badge or certificate should be required as well.

Computation-based assessments are prioritized in this kind of platform since the *tasks* are not complex and, thus, there are straightforward ways of assessing their outputs. When a certain user delivers an output, we need to calculate its impact based on the output consistency and user background.

Regarding quality assurance, the most relevant aspects of this platform are user motivation and training. Motivation is intrinsic while the training must be fully provided by a well-designed user interface with a small tutorial.

Most platforms using crowdsourcing mechanisms follow different approaches related to data quality control. This is still a topic where we cannot find common agreement. Quality control differs from system to system depending on their needs but it is proved that lack of control on this aspect causes the projects to increase their costs [17].

2.1.4 Volunteered and Rewarding models

In the **volunteered model**, the requesters depend on volunteered work performed by the crowd. For a volunteered crowdsourcing platform to have success, it is required a strong sense of community between the requesters and workers. To achieve this, usually, there is a communication channel where these two groups can have contact with each other and share their ideas (for example an internet forum). This way, the crowd may be aware of the requesters' goals inside the platform and may decide to help. Workers must know how their contributions will affect the course of the project.

The **marketplace model** is one of the most popular approaches in the topic of crowdsourcing. Some of the most relevant platforms like Amazon Mechanical Turk [18] and Microworkers [19] apply this model. The analogy in the name makes the concept very self-explanatory.

In this kind of market, usually, we see work waiting to be performed and sometimes no workers are willing to do it [20]. This is because the payments are low and workers tend to perform crowdsourcing tasks as an extra income and not as a full-time job.

Chandler et al. [21] deliver a detailed list of the strongest and weakest aspects of this approach. Some of these points are transversal to the whole crowdsourcing topic. The strongest aspects of this kind of model are:

- it allows requesters with limited financial and technical resources to have **access to crowdsourcing techniques**. This way, the costs of maintaining the platform are shared by all requesters that use the same system avoiding the issue of having to create and maintain their platforms;

- Workers need to be **recruited only once** to integrate the market and **learn only one standardized interface**. From here, they can work for a variety of requesters and have access to a big diversity of tasks;

- A marketplace with a **large amount of requesters delivering tasks** is a leading motivation for workers to access the platform more often since they know there will be work to be performed. Consequently, this will motivate new requesters and workers to use the platform;

- **Low payments do not prevent workers from performing the tasks** and stimulates requesters to deliver more work. The most significant reason why workers still do it, even though the payments are low, is because they can choose the tasks they want to perform and prosecute them at any location.

- **Low payments do not mean low quality data** [22]. One of the reasons that support this fact is that workers do not want to harm their reputation inside the market or they might not get another opportunity to work for a certain requester.

- There is **flexibility regarding the recruitment process**. The possibility of a requester specifying which sort of worker profile he is looking for is an enormous convenience from the perspective of the work provider.

A big marketplace has its weak aspects that might raise some issues in the mind of a requester. In the moment of choosing a marketplace platform as a solution to a problem, the requester must consider the following items:

- Usually **workers tend to work the minimum required** until ensuring the payment agreed and they try to avoid performing the most difficult elements of a task;
- **People are not infallible computational agents and therefore make mistakes**. This statement is applied to both groups of actors, workers and requesters. Regarding workers, they can introduce some errors in their work which should be taken into account by the requesters. Ambiguous instructions are common and it is the responsibility of the requesters to make sure this does not happen that often;
- Sometimes **workers may not be truly honest**. Usually, this happens in tasks where the information is really hard to be verified by the requester. A common case is when a worker lies about his personal information to be able to perform some kind of action or to have access to restricted content.

When comparing both models, it is noticeable a behaviour difference on workers [23]. The level of commitment in the marketplace model is bigger than in the volunteered model. The reason behind this is because in a volunteered platform, usually, a worker does not lose much when he quits from a task before its completion. On the other hand, in a marketplace model, when a worker does this, usually, it means he is not getting paid. It is also possible, as expected, to find more workers performing tasks while getting paid than when volunteered. However, there is one aspect where the paid model loses to the volunteered one which is the reliability of the worker's answers. Redi et Povoa [23] conducted a study which proved the previous statement. The cause associated with this fact is related to the worker's motivation when performing the task. In a paid model, the worker is interested in getting paid no matter what is the result of his work, while in a volunteered model, the worker is interested in his contribution's impact on the project and therefore tends to perform better.

Regarding this document's topic, it is more common to see solutions using a volunteered model because tasks are usually simpler and faster to perform. There is also another important factor regarding this project which is: users benefit from other users' contributions which means there will be a stronger sense of community. The user motivation increases when knowing that a task's outcome will affect multiple users.

2.1.5 Example Crowdsourcing Platforms

Crowdcrafting [24] is an opensource platform where people can contribute to other people's projects in a volunteered way. This system focuses mainly on science researches, where scientists can recruit volunteers for their experiments. These recruiters are enthusiasts, interested in contributing intellectually to science activities. They can perform work like image classification and transcription. The platform owns

a community space where scientists and workers can discuss with each other. This way, Crowdcrafting allows casual citizens to have contact and feel part of the science field.

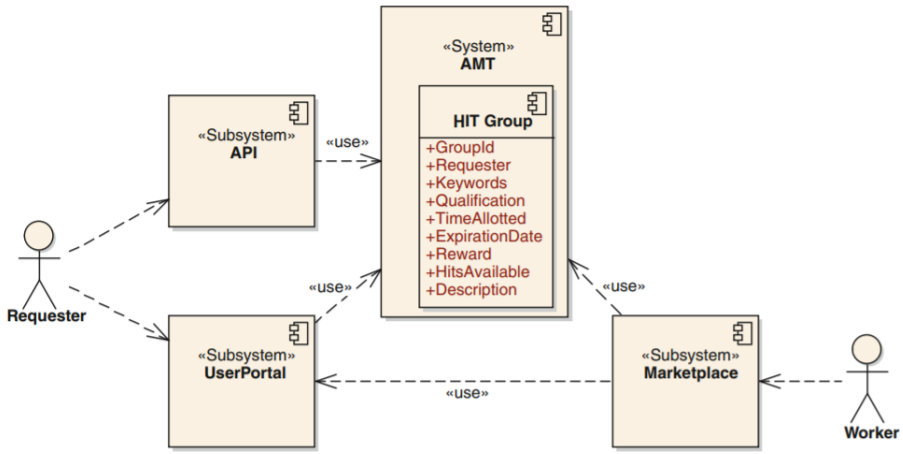

Figure 2.1: Amazon Mechanical Turk crowdsourcing system model [25]

Amazon Mechanical Turk [18] is a crowdsourcing system that was initially designed to support Amazon's business processes [26]. However, now it can be used in various ways, for instance, in natural language processing and machine learning.

It works as a marketplace, where the requesters can deploy tasks they intend to be performed. These tasks, also called HITs (Human Intelligence Tasks), are displayed to workers (also called Turkers) who can select the jobs they are most interested in.

The requesters can determine how many workers they need and how many times can a worker participate in their tasks. For instance, if the task consists of a survey, it makes sense that the user can only answer once. However, if the task consists of classification, it is fine if the worker performs it more than once. The requesters can also restrict the workers who can fulfill their tasks, by requiring them to do some tests before or restrict the tasks to a few countries. On the worker side, it is also possible to filter tasks to be displayed only the ones the worker finds most relevant for himself.

Schall [25] presents a generic system model of Amazon Mechanical Turk and a simple interaction corresponding to Figure 1. At the core, the middleware is responsible for task management and presents a model for a task group, or also called HIT group which carries nine attributes:

- Group id: unique identifier for a group of available tasks;
- Requester: id of the requester who delivered the group of tasks;
- Keywords: set of keywords used for the process of task suggestion;
- Qualification: set of requirements to restrict undesired workers;
- TimeAllotted: duration of the tasks;
- ExpirationDate: time when the tasks expire;
- Reward: monetary reward after finishing a task;

- HitsAvailable: number of available tasks in the group;
- Description: additional textual information.

The process of posting tasks on the platform can occur in two different ways. The requester can use the User Portal or a Web services-based API which can also be used to monitor tasks and to automate the process of task creation.

Concerning the quality management, Amazon Mechanical Turk is not in charge of that aspect. 3rd party crowdsourcing platforms must take care of that part.

Amazon Mechanical Turk is a reference in the world of crowdsourcing. Ipeirotis [20] conducted a study to analyze the AMT marketplace and for one year and four months collected a total of 6,701,406 tasks which is a significant number demonstrating how active AMT is.

AMT has some similarities when comparing to our proposed solution. For instance, both platforms' main objective is to promote the interaction between requesters and workers. They focus on being easy to learn, to use and fulfill most of the needs of both groups of users. With just a few clicks, it is possible to have a customizable crowdsourced application.

However, Crowdcrafting [24] and Amazon Mechanical Turk [18] are suitable for more complex tasks while this project requires them to be more straightforward and easy to answer. Therefore, they are not indicated to solve this kind of problem.

2.2 Volunteered Geographic Information

The concept of users mapping geographic information on dedicated platforms is called **Volunteered Geographic Information** (VGI [27]). Users can be trained but most of the time they are not. There are two types of Volunteered Geographic Information ([28]).

Explicit-VGI is the process of a user volunteering to provide information for a project while knowing its purpose. Some platforms using this approach are OpenStreetMap (Section 2.3), Wikimapia and Google MyMaps.

In **Implicit-VGI**, the users provide information which they do not know for which purposes it will be used. An example is when applications extract geographic information from Twitter [29] posts (tweets) like the location of a music concert. Platforms like Wikipedia [30] and Flickr [31] use this way of gathering information.

For this project, it is intended that users contribute with information manually and directly, knowing it will be used to improve maps for communities. In other words, an **explicit** approach is more appropriate.

In the rest of this section, we are going to introduce two VGI platforms and also talk about two VGI-based platforms as well as their respective approaches to similar problems.

2.2.1 Wikimapia

Wikimapia ([4]) is a project where it is possible to tag places on satellite imagery. The main idea consists of users drawing polygons on a satellite background and then tagging them with a description.

The core entity of Wikimapia is called *place* or *object*, and it corresponds to a polygon on a map. Many possible categories can be assigned to an object. Some examples are the house and garden. Only users with administrative rights can create or delete categories.

An object is associated with a description that consists of a long string containing information about the specific place. Descriptions are the main source of information on this platform. They are meant to be detailed and some places have a link to their Wikipedia [30] web page.

Regarding quality management, this platform uses a gamification approach (introduced in Section 2.1) to help motivating the users. A user can go through 8 different levels of winning rewards. Experience points can be obtained by doing positive actions inside the platform, and each action has its corresponding experience points. Bad contributions result in negative experience points. Being able to use moderating tools or protect some places from being edited are some of the rewards possible to acquire.

Wikimapia also facilitates the process of creating and editing *objects* to motivate the new users to stay active in the platform, and not leave because of a limited action pool. To avoid receiving bad data, editing made from new users must be approved by moderators, just like in Wikipedia [30].

Ballatore et al. [4] made a comparison between Wikimapia and another VGI platform called OpenStreetMap. While Wikimapia uses long descriptions, OSM employs tags which are short strings containing succinct information about a place. In OSM, it is possible to have multiple tags associated with one object. On the other hand, Wikimapia only lets each object have one description. From the perspective of a developer, using tags is easier to work with, when comparing to long descriptions. Tags imply dealing with less information and consequently are easier to extract relevant data.

When analyzing Wikimapia, the fact that it uses long descriptions instead of tags is a significant drawback. In our context, it is preferable to succinct data. Long descriptions also imply better parsing mechanisms to remove non-relevant terms.

2.2.2 Google MyMaps

Google Maps is probably the most popular map platform in the world. MyMaps is an extension of this platform that allows the users to create, edit and share their maps.

In this system is possible to draw points, lines, and polygons on a map, along with a name and a description for each object. The platform also allows users to define layers on a map, and select which ones they want to see.

We, as users, can invite people to our map instance or make it public so other interested users can join. When someone joins we can select if she/he can edit the features on our map.

In Google MyMaps, the fact that it allows the selection of layers displayed on a map is a great advantage when comparing to other platforms. However, if a map owner wants to share his map with the community, he cannot choose exactly how others can contribute with information. Consequently, the owner becomes confined to two possible actions: or he allows a joined user to edit everything or he does not allow editing at all. There is no middle term between both options.

2.2.3 Example VGI-based platforms

TowIt [32] is a mobile application, website and an API that allows users to contribute with information about dangerous driving and parking violations. It is built using Google Maps API and the main goal is to fight and mitigate, dangerous driving habits and bad parking, therefore improving the community driving behaviors.

For a user to report an irregular situation, he must follow a simple three-step process which is described below:

- take a photo of the offending vehicle;
- verify its license plate information to include in the report;
- submit it to the platform;

For assuring that data is valid and of good quality, it is required that the user is connected to the internet and uses the camera from within the TowIt application.

The valid reports are displayed uncensored, in a map, associated with a location calculated by the TowIt back-end automatically. All reports are verified, to check if the content of the report consists of an actual offense. Only then, these reports can be displayed and be passed on to municipalities or the police.

The most interesting aspect of this platform, regarding the project described in this document, is the fact that it is possible to assign pictures to specific locations. When assessing a street, it would be interesting if we were able to associate a picture to our verdict and submit it to the application.

Safetipin [33] is a mobile application meant to be used in countries where women do not feel that safe when compared to male individuals. In other words, this application was created to be used in cities where there is a high rate of physical or sexual violence against women.

This application uses information gathered by its users concerning how safe a street or area might be. Attributes like lightning, people density, gender diversity among others, are significantly relevant regarding the safety of public space. So, as a user, it is possible to assess all these attributes relative to a place, and in the end, it is calculated a final score that can be shared among all application users.

Besides user assessment, the developers also designed a non-crowdsourcing way of gathering information through photographs taken by moving cars across the cities. Joining both ways of collecting information, it is guaranteed better quality information. Women who feel threatened, can now be aware of the safest zones in their cities and avoid the non-safe places.

This application possesses a great concept that will be also in our solution which is the possibility of users assessing multiple attributes of public spaces.

2.3 OpenStreetMap [5]

2.3.1 What is OSM?

OpenStreetMap is a VGI platform where a user can create an editable map of the world for free (like Wikimapia and GoogleMyMaps). It was meant to overcome some of the restrictions related to the use and availability of map information. Its data comes mostly from the users via the manual survey, GPS devices, aerial photography and other free sources. This data is then stored in a shared database which can be updated.

2.3.2 OSM Structure

OpenStreetMap has one main database which is shared by everyone. However, if a user wants a private map instance to edit, it is created an instance of another database specific for that user. The main copy of OSM is stored in the main database which is an instance of an open-source object-relational database called PostgreSQL [34]. When someone edits something, the main database is altered but the map only gets updated a few minutes later to avoid rendering whenever a user changes something.

OpenStreet map has four data primitives which are:

- **Node**: a pair of coordinates that specifies a geographic position according to the standard World Geodetic System (WGS84). It is used to mark points of interest, mountain peaks and also to create **ways**.
- **Way**: ordered list of **nodes** which represent a connected sequence of line segments. They can represent streets, rivers and other linear elements of a map. They can also represent areas (e.g. lakes or parks) if they form a closed loop.
- **Relation**: ordered list of **nodes**, **ways** and **relations**, where each element can have a role (string). A **relation** is used to represent a connection or relationship between **nodes** and **ways**. An example would be a relation between a **way** that represents a road and a **node** that represents a stop sign in that road.
- **Tag**: key-value pair that is used to store metadata about a map object (e.g. name, properties or type). It is always associated to an object (**node**, **way** or **relation**).

Each data primitive has its table and individual objects are stored as rows in these tables. Every single edit happens in this database. The user can keep his customized map by downloading a *planet.osm* file. This file is composed of all primitive data instances and its size is over 160GB. It is possible to keep a small part of the map using a tool like Osmosis Java Software [35].

Besides Osmosis, OSM also recommends and provides other tools to help developers designing their applications. The most relevant we found were **osm2pgsql** (takes an OSM XML file and loads it into a database), **ogr2ogr** [36] (facilitates the manipulation of spatial data) and **mapsforge** [37] (provides tools and APIs for rendering and downloading maps).

2.3.3 Applications based on OSM

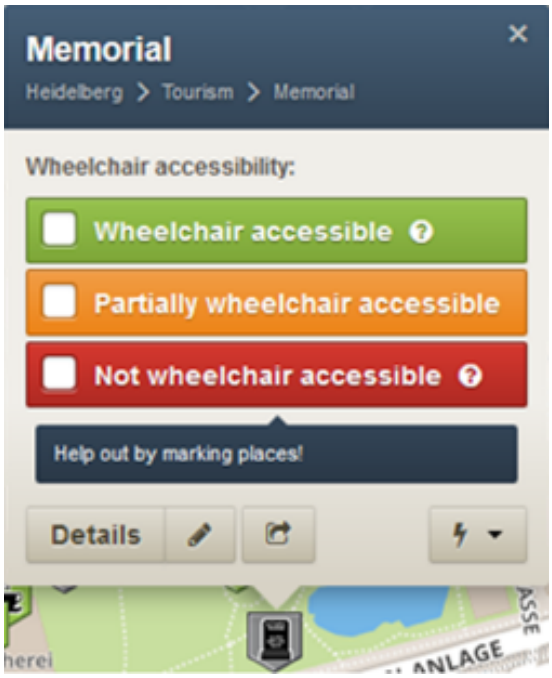

Figure 2.2: Wheelmap [2]

Wheelmap [2] is a crowdsourcing platform where volunteers contribute with information about the wheelchair-accessibility of public places. As a user, you can tag any kind of public place, like streets or parks, assessing how easy is the access to wheelchairs. Mobasheri et al. [2] introduce 3 different colors to classify the accessibility (Figure 2 shows part of a screen): green for good accessibility, orange for reasonable and red for bad accessibility. The final score of a place is calculated considering classifications from all the users.

Wheelmap provides a RESTful API to access and maintain the data within the OSM. When a Wheelmap account is created, an API key is linked to the account which is used to authorize REST requests to OSM. This account is based on a valid OSM account. One of the issues concerning our project is the bind between our solution and the VGI platform OpenStreetMap which Wheelmap has found one way to solve it.

Ciepluch et al. [3] proposes a prototype web map service that aims at delivering map-based environmental information in Ireland. It uses a customized version of the OpenStreetMap database. To keep the map updated, a Cron task [38] is scheduled to run every week. Cron runs on Linux and is a time-based job scheduler. Every week, using the Osmosis tool and osm2pgsql, it downloads the updated OSM XML file and does the necessary statements to update the PostgreSQL [34] database instance. This database carries an extension to allow the use of Geographic Information System (GIS) objects. To manipulate geographic data between different formats they use ogr2ogr [36].

Chapter 3

Proposed Solution

CoopMap consists of a mobile application where users can either create a map or join an existing one. After this initial step, they can start performing contributions to a particular map. In this context, contributing consists in giving an opinion considering some spatial attributes about a street or a group of streets. In Section 3.1 we describe the requirements for the CoopMap platform. Section 3.2 describes two use-cases focusing on the most relevant details.

This platform was built using a traditional client-server approach. Section 3.3 clarifies all the architectural decisions regarding the chosen design.

User Experience and Interface are not the main focus of this thesis. However, some issues needed to be solved regarding this field. It was required that the user could visualize information and also submit contributions. Using a mobile approach, this issue does not have a clear solution. It could get complicated and confusing for the user if it is not solved properly. Section 3.4 describes the approach chosen to solve this interface issue.

3.1 Platform Requirements

We wanted to keep our users motivated to raise the number and quality of the contributions. For this reason, we aimed at developing an application where the act of contributing is simple and effortless.

We outlined the following functional requirements:

1. Any user must be able to create a map with a certain set of attributes;

2. A user must be able to join an existing community and contribute with information;

3. The process of performing a contribution must be simple and quick.

Besides these three requirements, we also wanted the results of each user's contributions to be accessible by everyone in a low amount of time.

3.2 Use Case Scenario

To better understand the purpose of this solution, we demonstrate two use cases. In both situations, the user has similar needs but, in each case, he plays a different role.

When a user reaches the application looking for specific information about the city he lives in, he may follow two different paths. The two following subsections (Section 3.2.1 and Section 3.2.2) describe both paths and how the application behaves in those cases. In Section 3.2.3 it is explored in detail how a user interacts with the application when inside a map instance.

To achieve our goal of having a platform where it is possible to visualize information and contribute with some, we need to build a proper pipeline of information. The data acquired from OpenStreetMap needs a long formatting and filtering process until it gets easily read by a casual user. Section 3.2.4 explains the full process since data comes from the OpenStreetMap until it reaches the mobile screen of the user.

3.2.1 Creating a Community

Consider Bob as a user who likes jogging. He is interested in knowing which areas or streets are the best to go jogging. Bob decides to turn on the application and looks for already existing community maps. Two things can happen in the worst-case scenario:

- Bob does not find any map for the jogging community;
- The maps Bob finds, do not correspond to the real necessities of a jogger.

Consequently, since Bob is not satisfied with the current results of the application, he decides to create his instance of a map where the topic is ”Jogging”.

The process of creating a map is very simple. The steps required to create a map are described below:

1. Think of a topic and give the map a **name**. It is better if the name of the map is intuitive;
2. Add a map’s **description**. This will help other users understand the concrete purpose of the map;
3. Decide which aspects of a street (**attributes**) are important to consider regarding the activity or map’s topic;
4. **Submit all the information** and it is done.

In Bob’s case, he decides to name the map ”Run Forest” and adds a description saying: ”A map containing information for all the city joggers”. He considers relevant three attributes considering the map’s topic: sidewalk state, traffic density, people density. After this process, other users can join this instance of the map and submit spatial information.

3.2.2 Joining a Community

Even though creating a community map is simple, joining one is even simpler. All a user must do is go to a dedicated fragment inside the mobile application and **search for a community** he is interested in. He is free to enter the map instance and give contributions or use the map information for his use.

Considering a different scenario from the one above, when Bob turns on the application, he finds out an active community with a significant amount of information already gathered by its members. He decides to join this map and since the area he lives in has just a few contributions, he submits some opinions regarding some streets he knows well. This results in other users interested in the jogging activity having a little bit more information about Bob's living area.

3.2.3 Interacting With a Map

The application allows two different kinds of interaction with a map instance. A user can either visualize information or submit contributions. In CoopMap, both interactions are independent of each other. This means that a user can only either **submit information** or **visualize** it. However, the application allows the user to switch between the two modes easily.

Bob is new to the jogging community map. He already submitted some information regarding the area he lives in. However, he is interested in going for jogging the next day but he did not decided where yet. So he switches to the Visualization Mode and checks for areas with good reviews nearby that he has not been.

3.2.4 Information Flow

When a user enters a map on his smartphone through CoopMap mobile application, he will find streets represented as lines with different colors. These lines and colors deliver a certain meaning depending on the topic of the map. However, to achieve this result, there is a considerable amount of data processing, which the user cannot perceive.

The whole process starts when a file with a partial copy of the OpenStreetMap database is downloaded. This partial copy contains all the information represented in OSM but for only a certain area. The following step consists of importing this information into an empty database on the server. This raw data contains all the information about streets, buildings, points of interest and other infrastructures. Our only and current interest is extracting information about the streets. But OpenStreetMap provides a variety of information for each object. Therefore, we do not only need to filter between all the OSM objects but we also need to filter which information we want about each object. In our scenario, we require the set of coordinates for every street. After gathering this information we import all the streets into the CoopMap database. This process described only happens when either we want to populate the CoopMap database or update it.

The server only interacts directly with this new database. Every time a user enters a new map, the mobile application sends a request to the back-end server. The server will process the coordinates of the user and build multiple statements for the database. These statements will lookup for street coordinates

and their score considering the map object. The result is returned to the user and is displayed as lines with different colors depending on the scores.

When submitting a contribution, the mobile application sends a request with a group of street segments and the score the user has given them. The server will process it and update the respective street segments database information.

3.3 Architecture Overview

As refereed at the beginning of the chapter, CoopMap was built using a client-server approach. During this section, we will describe the three components that compose this solution: Server (Section 3.3.1), Mobile Application (Section 3.3.3) and Database (Section 3.3.2).

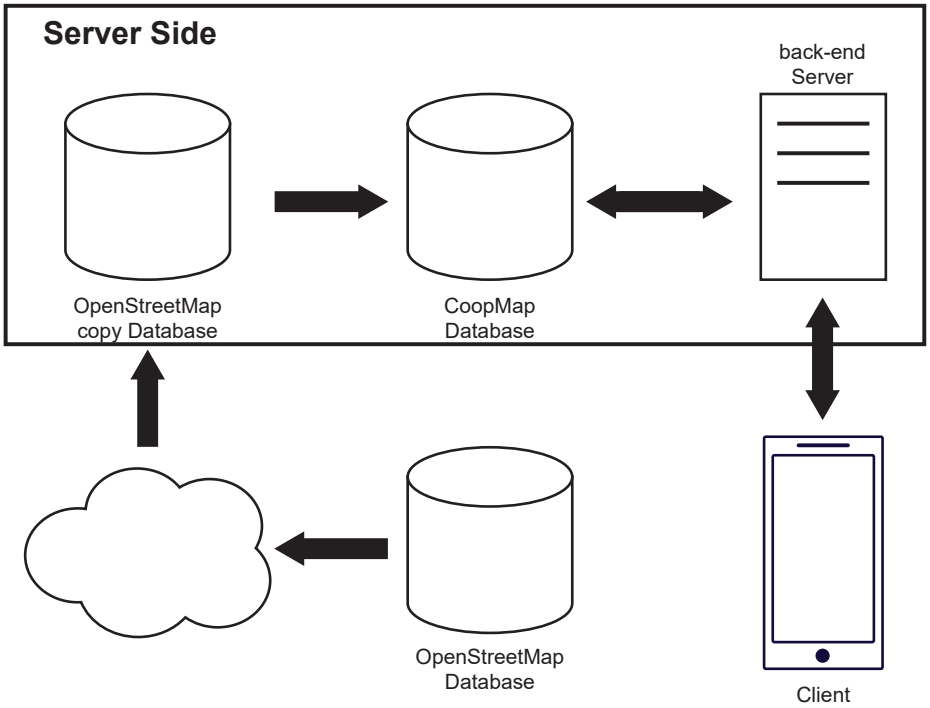

Figure 3.1: Architecture Overview.

3.3.1 Back-End Server

In the CoopMap platform, the back-end server is divided into three sub-components (Figure 3.2). The first component is responsible for receiving requests coming from users and for building and sending the respective responses. The second component is responsible for processing the requests. This processing consists of requests being decomposed in a group of calculations (which are performed in this component) and in a group of database statements, which are taken care of in the third component. This last component is responsible for the interaction with the Database.

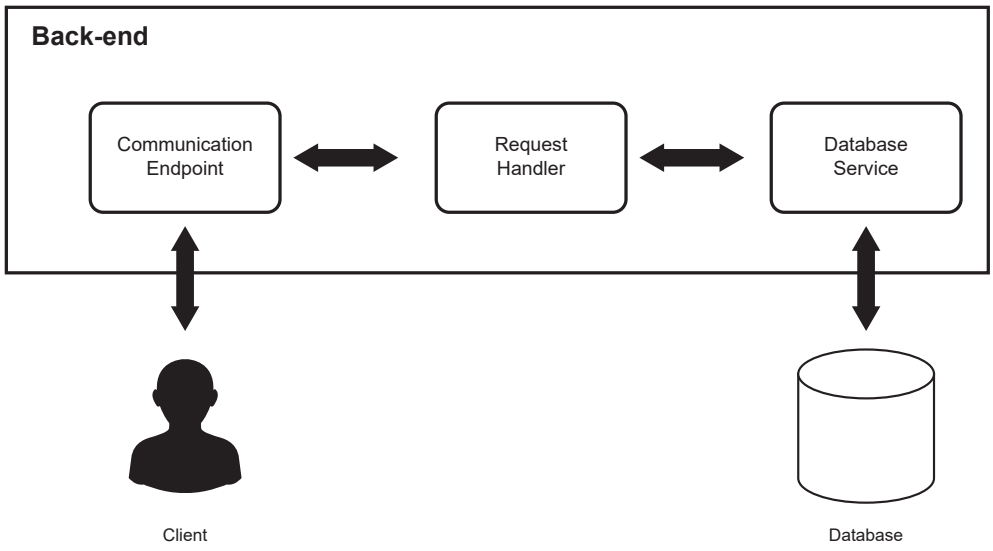

Figure 3.2: Back-end Server

Contribution:

A contribution, in the context of this platform, consists of a subjective evaluation of the attributes defined considering some specific street or area. Attributes are spatial characteristics defined by the map creator. Consequently, this evaluation (or contribution) is associated with only one map's instance. During the process of contributing, for each attribute, the user has to give a score with the minimum value of one and a maximum value of five. These values were inspired by how users can evaluate mobile applications on the Google Play Store. When a user submits a contribution, the value decided by the user for each attribute is merged with other users' values. This will induce a variation in the original value. It is important to refer that each user can only submit one contribution for each street segment. The score of a street segment s for a determined attribute h follows an Arithmetic mean defined below:

$$AttributeValue_h(s) = \frac{\sum_{i=1}^{n} Avalue_i(s)}{n} \tag{3.1}$$

where:

$AttributeValue_h(s)$ = value of attribute h considering street segment s

n = number of contributions involving street segment s

$Avalue_i(s)$ = score given by an user i to s considering attribute h

In a map, the streets' overall score is fully decided based on users' contributions. This overall score is calculated considering an **average between all attributes** defined by the map owner. Because the city is always changing, it was decided that we should give **more weight to contributions made in a more recent period**. To obtain a value for a certain interval of time we calculate the average of the contributions submitted (by any user) in that interval:

$$StreetSegmentValue(s, Interval) = \frac{\sum_{h=1}^{n} AttributeValue_h(s)}{n} \tag{3.2}$$

where:

$StreetSegmentValue(s, Interval)$ = overall value of street segment s in interval $Interval$

$Interval$ = number of the interval of time

n = number of contributions made in that interval of time

The concrete duration of one interval of time can be defined on the server setup (default duration is set to one month).

We wanted to make a recent contribution more valuable than an older one without making the older one total meaningless. Also, we did not want to have a binary approach where a contribution is either old or recent. Considering these two conditions, we decided to implement a system where we attribute weights to different time intervals. All contributions are distributed between all these time intervals taking into consideration the moment these contributions were submitted. Newer contributions will have higher weights while older ones will have lower weights.

To better understand our solution, consider a time scale from the first day of one year until the last day of the same year. Consider as well that we are currently on this last day of the year and that each time interval coincides with each month of the year. For a certain street segment A, if there were twelve contributions submitted, one for each month, the result would be twelve contributions, all with different impacts on the overall score of the street segment. The first contribution, which was made in the first month of the year, will have a lower impact than all the following contributions made in the next months. The last contribution (made in the last month of the year) will have a higher impact than all the previous ones. While the time passes, contributions will shift to the next interval of time and will have a lower impact on a street's segment overall score.

The formula applied is described below :

$$StreetSegmentValueAllTime = \frac{\sum_{i=1}^{max_i} StreetSegmentValue(interval_i) * weight_i}{\sum_{i=1}^{max_i} weight_i} \tag{3.3}$$

where:

max_i = maximum number of intervals until a contribution becomes irrelevant

$weight_i$ = weight given to a certain interval of time

It can be noticed that there is a $weight_i$, which is used to raise the influence of newer contributions. The values of the weight for each i and the value of max_i can be defined when setting up the server.

In the long term period, some works can alter some of the attributes of a street. For instance, if one of the attributes of a map instance is the state of the sidewalk, that can easily change with some street works. Distributing weights between these intervals, in the long term will achieve better results.

3.3.2 Database Structure

This platform uses two different databases, which are located near the back-end server. The main one, also called the CoopMap database, is responsible for saving information to be fetched by the application. The second database serves as a support for the main one. It keeps raw OpenStreetMap data, which can be extracted to update the streets from the main database.

CoopMap Database

The CoopMap database saves all the information required by the application and the back-end server. Regarding **user information**, the database only holds information about which maps each user possesses and his contributions.

Maps information is also saved in this database. For each map, there is a name, a description, and an attribute list. Besides these three elements, the user who created the map is also associated.

The most critical information to be fetched in run-time is the **streets metadata**. Every time a user enters a new area on a map, the mobile application needs to request that area's streets from the server. We decided to divide the streets in multiple segments. This way, the user can select one or more segments of a street before submitting a contribution. This allows a higher level of precision since the user can select small parts of a street. Regarding each segment, the respective street is stored as well as the coordinates of the segment and values of the contributions.

OSM Copy Database

The secondary database maintains a copy of the data from OpenStreetMap (Section 2.3). This data allows the extraction of streets metadata, which is used in the CoopMap database. However, it holds information about every kind of object present in OSM in a very detailed way and requires filtering. This results in a significant amount of unused information considering this project's goal.

Our interest in this database is to **extract the streets and their list of coordinates** and export them to the CoopMap database. This information flow between these two databases is a complex process and it is explored in Section 3.2.4.

3.3.3 Mobile Application

The client-side application consists of a mobile approach where the user, using his smartphone, can interact with his communities as described in the previous section. It is composed of three sub-components:

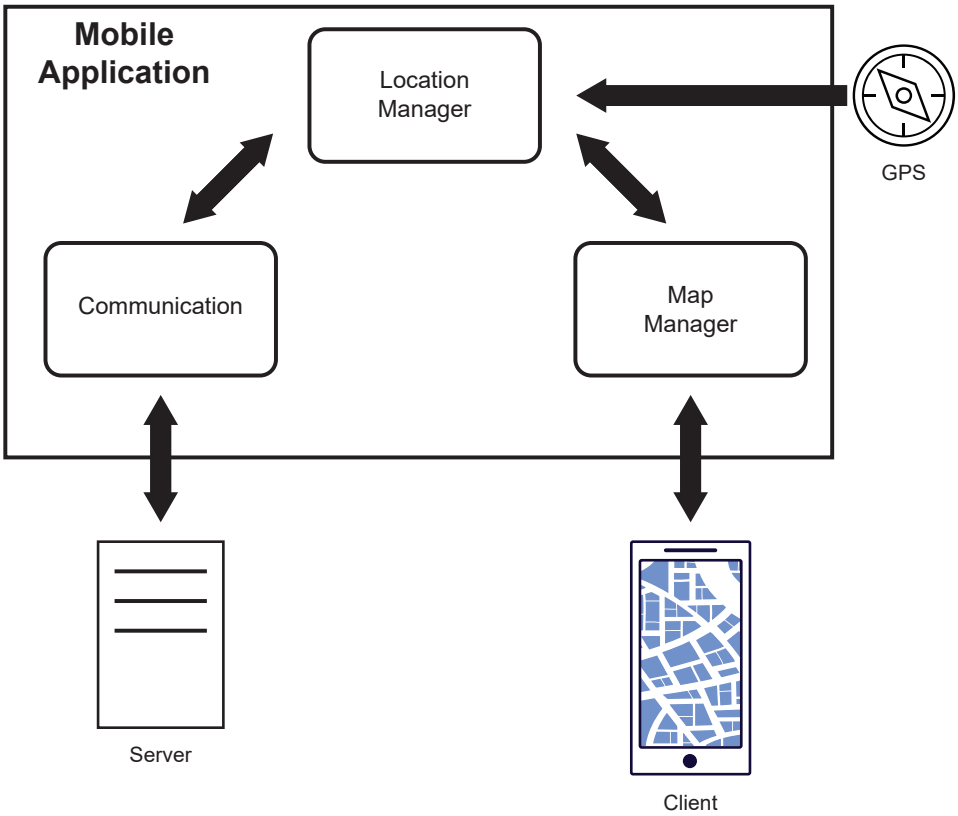

Figure 3.3: Mobile Application

Location Manager Component

Location Manager has the purpose of keeping track of the coordinates of the user's position. This allows some important features like the automatic positioning inside a map's instance and a notification system designed for this platform:

- When a user enters a map's instance, the application automatically centers the map view on the point where the user is in that moment (**automatic positioning**);
- The **notification system** is something designed to notify the user every time he is close to an area that has not been receiving contributions lately. As a quick example, consider a user walking on the streets. He reaches an area where a few or no people use the application and consequently there are no contributions in that area. The application, if turned on, will notify the user about this situation and possibly lead him to submit some new information to the platform.

The location manager allows these two features, which may raise the number of contributions coming from users, therefore resulting in more enriched maps.

Communication Component

The Communication Module is responsible for establishing and keeping a communication channel with the server-side. It builds and sends requests and receives the respective responses. When a response arrives, this module translates the data coming from the communication channel with the server into instances of classes (adopting an object-oriented approach). In other words, this module provides an abstraction layer (considering the communication details) for the above layers of the client application.

Map Manager Component

Map Manager is responsible for saving and updating a map's state during user utilization. Some of the main operations handled by this module are:

- When a user scrolls within the map, this module decides whether it is or not necessary to **fetch new map information** from the server;
- This module keeps **track of all contributions** made by a user, which is relevant to avoid multiple contributions from the same user for the same streets;
- When new map data arrives from the Communication Module, Map Manager **updates the map view** with the corresponding new data;
- When a street segment is pressed by the user, this module **triggers a visual change** to inform the user the application is aware of his action (explored in Section 3.4).

Map Manager is the core module and it is constantly interacting with the two other modules of the mobile application. The Communication Module is mainly requested to obtain map updates and send contributions. The Location Manager is often called to obtain information about the position of the user and consequently to update his position on the map view.

3.4 Interface

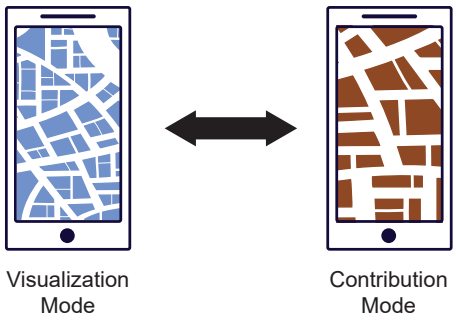

Figure 3.4: Visualization Mode and Contribution Mode

As refereed before, inside a map, the user has **two ways of interacting** with it. Either he is visualizing the results of the community contributions or he is submitting contributions. The mobile application provides an easy way of swapping between the two modes.

In the **Visualization mode**, the user cannot submit any kind of information to the server. He can only observe the street segment's colors. The color system is presented at Figure 3.5. The overall score of a segment varies between one and five where one is the worst score and five is the best. Each value is associated with a color in the figure.

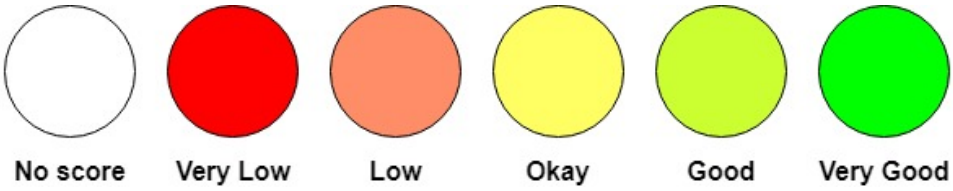

Figure 3.5: Visualization mode color system

In the **Contribution mode**, the user can click on multiple segments and submit an opinion (contribution). After this, they become closed avoiding the same user making multiple submissions. The color system for this mode is introduced in Figure 3.6.

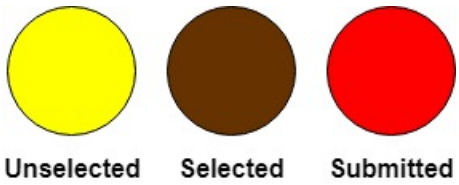

Figure 3.6: Contribution mode color system

Different colors mean different states. Consider a certain street segment and a user Bob:

- **Yellow** color (unselected) means the segment was not clicked by Bob but also means it has never been reviewed by him. In other words, it is open for a new contribution coming from the user Bob.

- **Brown** color (selected) means it is currently selected and ready to be reviewed by Bob.
- **Red** color (submitted) means it is closed for new contributions coming from Bob. He has done a contribution containing this segment before.

The visualization mode color system is intuitive and it is very easy to comprehend. However, the contribution mode requires a little more time for the users to get accustomed to the colors and the mechanics.

Chapter 4

Implementation

The CoopMap platform is composed of three main components: a database, a server, and a client. Before deciding which schema would be implemented for the database, we had to analyze the data coming from the OSM platform to determine which options were available. In section 4.1, these aspects are discussed in detail. The server and the client sides are explored in section 4.2. Both sections will go in depth about the tools and technologies used as well as the technical decisions.

4.1 Database

4.1.1 Technologies and Tools

The database management system chosen for this project is a relational DBMS called **PostgreSQL**. This was a straightforward decision due to two factors:

1. **OpenStreetMap** uses the same technology to store their spatial data.
2. PostgreSQL contains an extension called **PostGIS**, which adds additional geospatial types and functions making it more easy to handle spatial data.

Besides PostgreSQL and its extension, we also required a tool called **Osm2pgsql**, which allows the conversion of OpenStreetMap data into our PostGIS-enabled PostgreSQL database.

4.1.2 OSM Copy Database

As described in the previous chapter (subsection 3.3.2), we manage two distinct databases near the server. In this subsection, we will describe the one containing a copy of the OpenStreetMap data.

It is impossible to maintain an always up-to-date copy of the OSM's database because it would require frequent requests to their servers. Since OSM is a non-profitable platform, it cannot maintain high capability servers to answer all these requests. Our solution consists of downloading manually an OSM file (`Portugal.osm.pbf`) from a website called geofabrik [39] containing the most recent map version of Portugal.

After this step, osm2pgsql is used to import this file into CoopMap database. Every object in the OSM platform has tags associated, which contain a low quantity of information. By defining a `default.style` file (Figure 4.2), we are able to import the tags in which we have interest and discard the rest. The tag (or key) `highway=*` is responsible for identifying any kind of road or path. To achieve our goal of keeping data about streets, we decided to include all the lines with a non-empty `highway` tag. Since streets can contain multiple roads or paths they will contain multiple lines as well. We denominate the lines present in a street as **street lines**.

The Figure 4.1 shows how osm2pgsql organizes some of the most relevant tables inside our database. The Table `planet_osm_line` has information about all the lines represented in the OSM platform. These lines can represent, for instance, a border between two countries, a road or even a bicycle path. A group of lines combined can represent the limits of an area, building, country, etc. If we filter this table looking for non-empty `highway` tags, the result will correspond to a set of lines, which contain street information. The data returned from this query is crucial to populate (or update) the primary database of this platform (CoopMap database).

Schema	Name	Type	Owner
public	planet_osm_line	table	ricar
public	planet_osm_nodes	table	ricar
public	planet_osm_point	table	ricar
public	planet_osm_polygon	table	ricar
public	planet_osm_rels	table	ricar
public	planet_osm_roads	table	ricar
public	planet_osm_ways	table	ricar

Figure 4.1: osm2pgsql resulting tables

# OsmType	Tag	DataType	Flags
node,way	highway	text	linear
node,way	name	text	linear

Figure 4.2: osm2pgsql style file

4.1.3 CoopMap Database

Using a script (`bat` file), we can generate a text file containing data from all the streets of Lisbon. This data results from the query to the previous database (Section 4.1.2). The extracted text format for each street line is the following: `osm_id;street_name;coordinates`. Figure 4.3 exemplifies a small street with a single road using this representation.

To achieve the desired format, the way element requires a double transformation, which is represented bellow:

```
st_asText(st_transform(way, 4326))
```

This statement is responsible for transforming the coordinates into the World Geodetic System 1984 (WGS84) and subsequently in a recognizable text format. This transformation allows us to work with the traditional coordinate system, which is required when drawing the lines on the device screen.

```
71125730;Rua Francisco Baía;LINESTRING(-9.1787564 38.7550951000904,-9.1777355 38.7539746000902)
```

Figure 4.3: Street line format example

With the data in this format, we can populate (or update) the new database.

Populating/Updating the Database

To extract the street data from the generated text file and importing them into the database we use a **Python** script. To populate or update the database, we start by iterating over all the street lines in the text file. Every line contains a list of coordinates. The minimum number of coordinate pairs that each line can have is two (a start point and an end point). There are street lines containing more than two pairs of coordinates (usually when they contain curves). All streets have at least one street line. However, there are some cases where streets contain multiple lines (Figure 4.4).

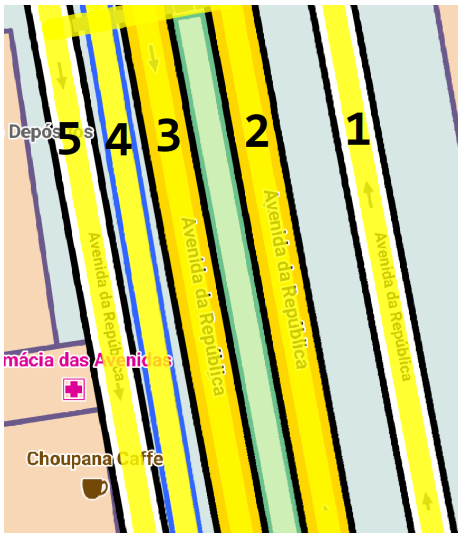

Figure 4.4: Example of a street containing multiple parallel roads

In our solution, we decided to divide street lines into multiple segments. For each line, from the list of coordinates, we built multiple lists containing only two elements. These two elements consist of two consecutive points of the same line and together are what we called a street segment.

The process described above is similar for populating and updating a database. However, the final step varies depending if we are updating or populating a database:

- Considering that we are populating, for the final step we only need to insert the segments into our empty database;
- If the purpose is to update the database, the last phase of the script consists of comparing the old segments with the new ones. If an old segment does not match with any of the new ones, it

means that it might be outdated. In this scenario, the street containing that outdated segment gets replaced by the new one. This implies losing the contributions made so far for that street.

Database Structure

The table where all the streets are stored has the structure defined at Figure 4.5.

Column	Type
street_id	bigint
street_name	text
segment_id	integer
way	geometry(LineString,4326)
attributes	bigint[]
reviews	bigint[]
reviews_count	bigint[]

Figure 4.5: Streets table schema

This table stores information about all the street segments of the city. Each entry consists of one street segment and keeps a `street_id` and a `street_name`, which come directly from the street lines text file. It also includes a `segment_id` generated during the execution of the script and it is attributed to each segment to differentiate it from others within the same street. The `way` keeps information about the pair of coordinates from the start and the end of the segment. Each entry also contains three arrays:

- The `attributes` array contains all existent attribute ids in the platform. It is ordered in such a way that the first element of the list is the first attribute ever created. Every time a new attribute is created (on a map creation), its id is added to the list of every street segment;
- The `reviews` array contains the average values for each attribute originated from multiple contributions coming from users.
- The `reviews_count` array contains the number of contributions made regarding a certain attribute. This array is used to calculate the overall score of the street segment whenever a new contribution arrives.

4.2 Back-end Server and Mobile Application

4.2.1 Technologies and Tools

Some technologies and tools were commonly used in the development of both client and server-side. The programming language used during the development of these two components was **Java 8**. For configuring the project we used **Gradle**.

To build the server, we used a framework called **gRPC** (gRPC Remote Procedure Calls [40]), which delivers a simple way of building connected systems. Using this technology we can define services and methods which can be called by the client-side remotely.

On the client-side, we decided to create a mobile application for the **Android** operating system targeting devices with Android 4.1 or higher. Developing in this OS facilitates access to a variety of open-source libraries like **mapsforge** [37], which facilitates with some map-related concepts like map rendering and layers management. Both **Java** and **Gradle** have plenty of support in the Android development.

To manage users personal data, we use an external service called **Firebase** [41]. This service allows us to focus on the main modules of the application avoiding handling user personal information required for registrations or logins.

4.2.2 Mobile Application

User Authentication

Regarding user authentication, we do not keep any user personal data (ex: email, username, password). To simplify we decided to include a login/register using the **Google Identity Platform** [42] (Figure 4.6). Google email is commonly used and since we do not keep any personal data, we feel like this was the best approach. **Firebase** is responsible for generating a user id and storing the user email.

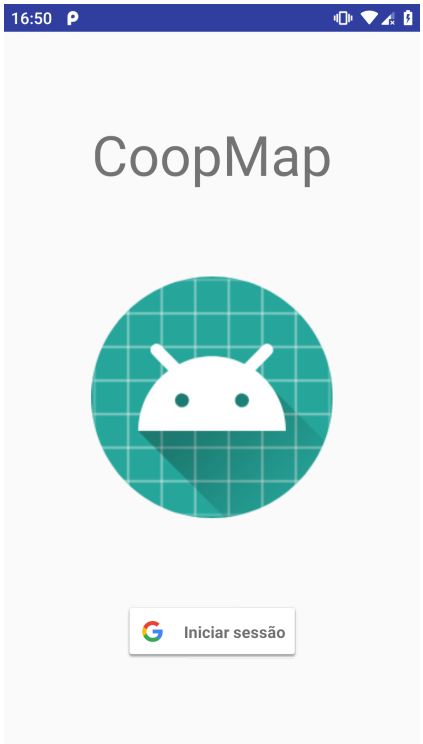

Figure 4.6: Login screen

Mobile Application Structure

The mobile application contains a Navigation Drawer Activity called `Home_Activity`. It is responsible for managing the state of multiple Android fragments. The provided menu (Figure 4.7) allows the user to navigate between all the sections of the application. Figure 4.8 presents the fragment where the user can create a map. `MyMaps` fragment is presented at Figure 4.9, and it keeps track of the maps created by the device owner. Beyond these sections just described, the user also has access to `CommunityMaps`, which contains other users' maps.

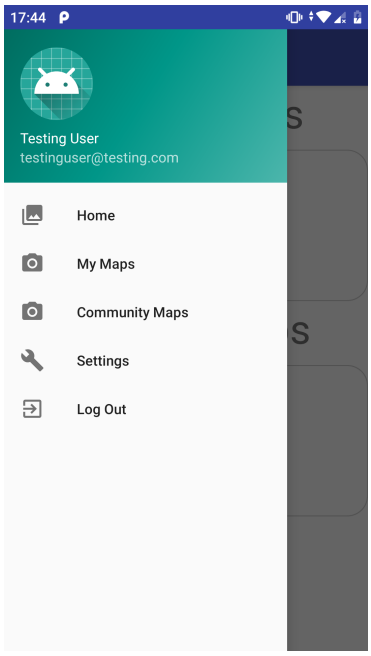

Figure 4.7: CoopMap Navigation Menu

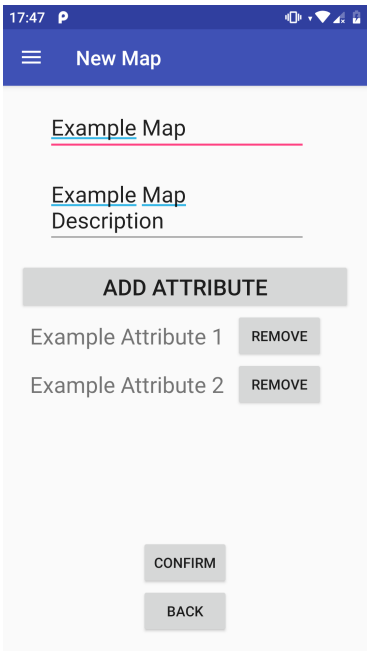

Figure 4.8: Create Map Fragment

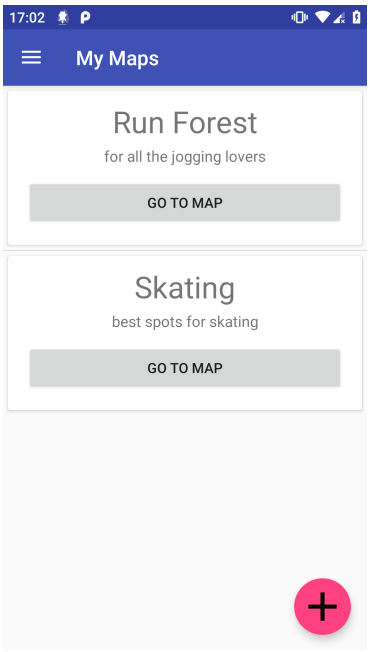

Figure 4.9: User Maps Fragment

Visualization Mode and Contribution Mode

Both Visualization and Contribution modes are Android activities that inherit from the same superclass (`Map_Activity_Template`). This superclass is responsible for setting up the initial `mapview` with the necessary controls to enable user interaction. In this set-up, it is also defined the map bound limits and the zoom limits. During the setup, two tasks are initiated:

- **FetchMapUpdateTask**: This task checks if a new map update is required. If the received response is positive, a new request is sent to the server fetching for map data;
- **GetMyLocationTask**: This task asks the device for a new user location. The returned data is used to update the position of the user icon within the map.

Each task contains a parameter to determine how frequent the process gets repeated. The values for these parameters can be defined during the setup of the mobile application.

`Map_Builder` is a static class which manages the layers of each `mapview`. While the user is moving around the map, new layers are being added or modified. The layers are composed of `polylines`, which

represent the street segments. One circle representing the user's GPS location is also added to the map.

In the **Visualization Mode** (Figure 4.10), the color of the lines (or polylines) varies accordingly to the overall value of the street segments (considering the map instance).

In Figure 4.11 (**Contribution Mode**) the colors of the lines are used to inform the user of the current contribution status. This information is detailed at Section 3.4. In the Contribution Mode, the user can select multiple segments, one by one, to classify. However, he can also select two segments of the same line and automatically all the intermediate segments get selected. This feature decreases the effort during the process of selecting segments to be classified.

When the user finishes selecting segments, he can press a button to open a `Dialog` where he can perform his final evaluation (Figure 4.12). If the server returns a success message, the segments become closed for newer contributions coming from this user.

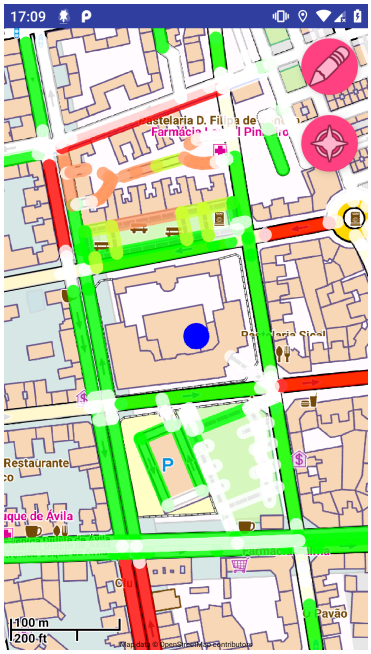

Figure 4.10: Visualization Mode

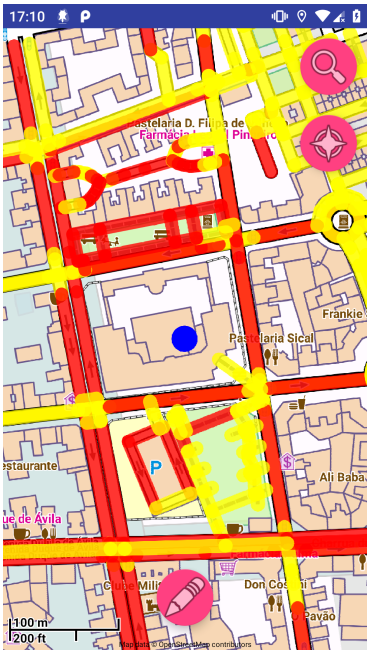

Figure 4.11: Contribution Mode

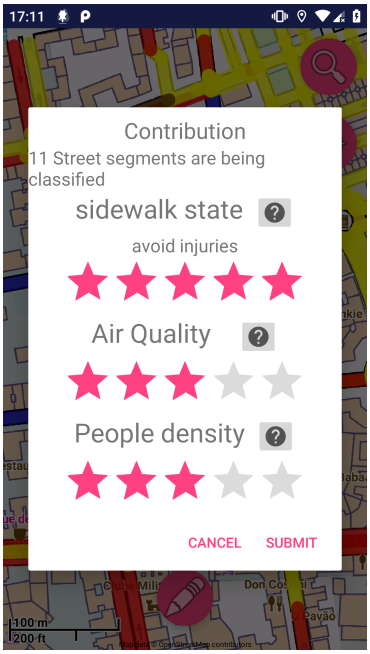

Figure 4.12: Example of a contribution

4.2.3 Back-End server

gRPC technology allows the establishment of a communication channel between the server and the client using a remote procedure call approach. Using a `service.proto` file we can generate two of the most important endpoints:

- `getClosestSegments`. This endpoint is called whenever a client application requires a map update (when the user reaches new areas inside a map). The request contains a `mapId` and a `latlon` (coordinates of the user's map view center). The returned result is a group of street segments in a close distance to the point received (default value is set to a radius of three hundred meters). These segments contain coordinates and a score considering the map instance. For each segment is also held a `boolean` to inform the client application if the user has already submitted a contribution for

that segment considering that map instance.

- `submitContribution` is responsible for processing all contributions coming from users. To avoid database conflicts from multiple contributions, it has a dedicated thread to hold and process contributions considering the order of arrival.

The back-end server does not receive user locations. Only the mobile application has access to this kind of information, which is useful to position the user within the map or to inform the user of close segments with a low amount of contributions.

Chapter 5

Evaluation

This chapter presents our evaluation of the proposed solution. The goal of this evaluation is to question if the following platform functional requirements were successfully achieved:

- Any user can create a map with a certain number of attributes and submit contributions;
- Other users can join a map and submit contributions as well;
- For any user, submitting a contribution is a simple and quick process.

Besides these functional requirements, we also want to ensure that the results of submitted contributions are accessible to every user in a relatively short time.

These are the main goals of the platform which should take the most attention during the process of the evaluation. However, this platform has some relevant features which can also be evaluated. The way we switch between the two map modes (Visualization Mode and Contribution Mode) was designed to work efficiently so that the user does not lose much time during his experience. The process of updating the CoopMap database can be evaluated as well to detect if it can be reproduced for larger areas.

5.1 Evaluation Methodology

All the tests were performed on a Xiaomi M1 A1 smartphone (Android 9.0) with the specifications: Snapdragon 625 CPU; 4GB of RAM. The computer used to run the database update has the following specifications: i5-7200U CPU; 8GB of RAM. This system is also responsible for hosting the back-end server during the tests with the mobile device. All tests were performed with a USB cable connection between the device and the server which means network latency was not an issue. Every test was performed 5-10 times to balance the effects of external conditions across all cases.

During this chapter, we will present four test scenarios. In the first scenario (Section 5.2), we evaluated the overall performance and the amount of data transferred between the client and the server. At Section 5.3, we performed tests with multiple users. In this case, the goal was to understand how multiple users, using the application at the same time, would affect the performance. In the third scenario, we decided to perform usability tests with five distinct users to collect information about the positive and

negative aspects of our user interface and overall experience. In the last section (Section 5.5), we decided to test our database scripts to understand how it behaves when dealing with different amounts of street segments.

5.2 Single-User Benchmark

During this section, we will describe our benchmark tests, which were performed in a scenario containing a single user. At Section 5.2.1, we measured the time spent to process contributions with different dimensions. At Sections 5.2.2 and 5.2.3 we collected information about the amount of data transferred between the server and the client. As introduced during the chapter 3, the application contains two ways of interacting with a map. The transition between these two map modes was tested and discussed in Section 5.2.4.

5.2.1 Request Time

We wanted to allow user contributions to be accessible by everyone in a relatively short time. To evaluate this condition, we decided to measure the time since the button to submit the contribution is pressed until the database gets updated with the value of this contribution.

Every contribution contains an array of street segments. We have submitted contributions with 100, 250, 500, 1000 and 2000 street segments (contributions with distinct dimensions).

In Figure 5.1 we can notice a linear growth of the time spent to perform each kind of request concerning the number of segments.

We expect the typical usage of the application to consist of making multiple small contributions (contributions with 1-100 segments). Contributions containing higher numbers of segments may lead the user to lose track of the already selected segments and consequently may lead to wrong judgments of the spatial attributes. The average time to process a request containing 100 street segments was 406 ms, which we consider as a reasonable amount of time.

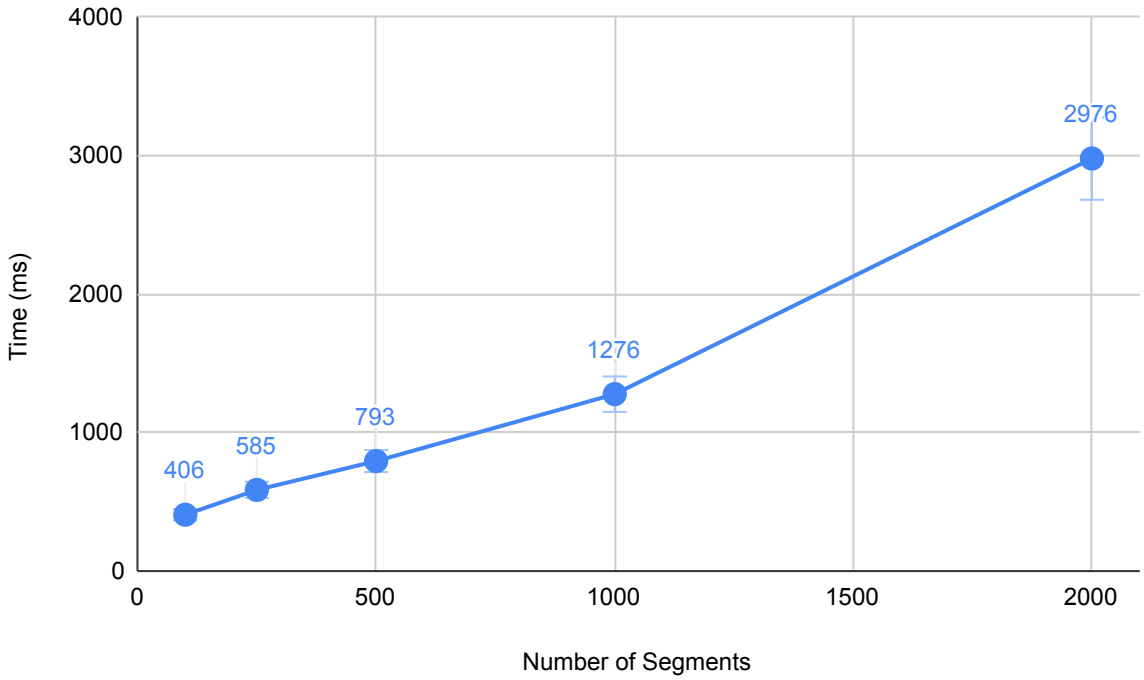

Figure 5.1: Time spent by contributions with different dimensions

5.2.2 Upload

To evaluate the amount of data transferred from the user's mobile device to the server, we measured the size of each contribution in Bytes. Figure 5.2 presents the different data sizes for each contribution considering the number of map attributes and the number of segments selected.

We can notice a proportional relation between the number of segments and the size of the contributions. However, there are two factors which may also influence the final size of the contribution:

- For each attribute, the contribution carries a value containing the user's subjective evaluation. In most cases, this means that contributions with a higher number of attributes will have a bigger size compared with contributions with a lower number of attributes;

- gRPC serialization is not deterministic ([40]). This means that contributions containing the same kind and number of objects may end up having slightly different data sizes when serialized.

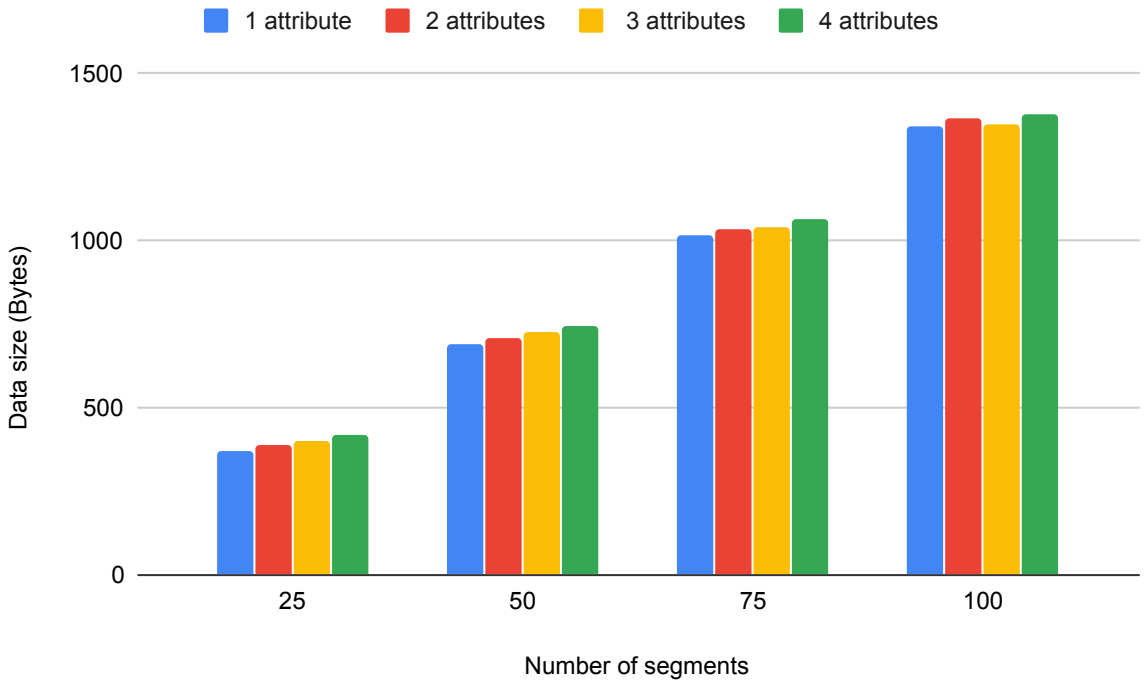

Figure 5.2: Data transferred by multiple contributions with different amounts of segments considering maps with one, two, three and four attributes. Based on these results, the average size for each segment was 14 bytes.

5.2.3 Download

When the user enters the map for the first time, he can visualize the overall score of all the segments loaded in the map through the color system described at Section 3.4. For this feature to be allowed, the client must first fetch this information from the server. The application contains a thread that handles all the requests related to map updates. These requests contain a reference point which is going to be used by the server to query the database. The query will select all the segments within a 300 meters radius from that reference point and will return this data to the client.

Number of Segments	data size (kB)
332	15.4
388	18.4
445	20.7
530	25

Table 5.1: Example of map updates responses and their respective sizes (kB)

On Table 5.1, it is possible to find some information about the responses returned to the client. As expected, areas with higher road density result in a higher number of street segments and consequently in bigger requests.

For future work, there are two clear ways of improving these values. The first consists in reducing the number of segments. By doing some processing of the data we may be able to join small consecutive

segments and delete non-relevant segments. This way, we will decrease the amount of data transferred. The other solution consists in keeping a version of the map in the user's mobile device. This will avoid sending large amounts of segment information to the user's device every time he moves around the map.

5.2.4 Switching between Contribution and Visualization Modes

One positive aspect of the mobile application is the easy way of switching between the two map modes. It was carefully implemented so that the user does not lose much time during the transition. Nielsen [43] has refereed that a user can wait a maximum of one second before noticing a delay in the application. Our goal was to not surpass this limit during the transition between the two modes.

Our efforts to achieve this goal consisted mainly of efficient management of the Android activities with some support of multi-threading as well.

During this test, we performed the two possible map mode changes (visualization to contribution and contribution to visualization) multiple times. We measured the time between clicking the button that triggers the map change and the end of the `onPostResume()` of the new activity. Our final result consisted of an average value of 130 ms, which is not a problem for the normal user.

5.3 Multi-User Experiment

We wanted to test if multiple users using the application at the same time would affect the overall performance of the platform. In other words, we wanted to measure how the platform scales when receiving requests from multiple users at the same time.

Our back-end server contains a queue (FILO approach) which accumulates unprocessed contributions. To perform this test, we decided to simulate a map context where multiple mock users would submit contributions at the same time for the same street segments. When processing a contribution, the server performs a read and consecutively a write in the CoopMap database. We need to ensure that the data remains consistent between the two operations. The point of submitting contributions for the same segments is to check if the server can process these contributions correctly avoiding data conflicts.

The conditions to run these performance tests are defined below:

- All requests were made within the same map, which contained two attributes;
- The number of segments for each contribution was generated randomly between 1 and 10 segments;
- The time was counted since the moment where the first request was sent until the last request was processed by the back-end server;
- We tested four different scenarios with four different amounts of users (100,300,500,1000);

- All requests were sent from the same mobile device.

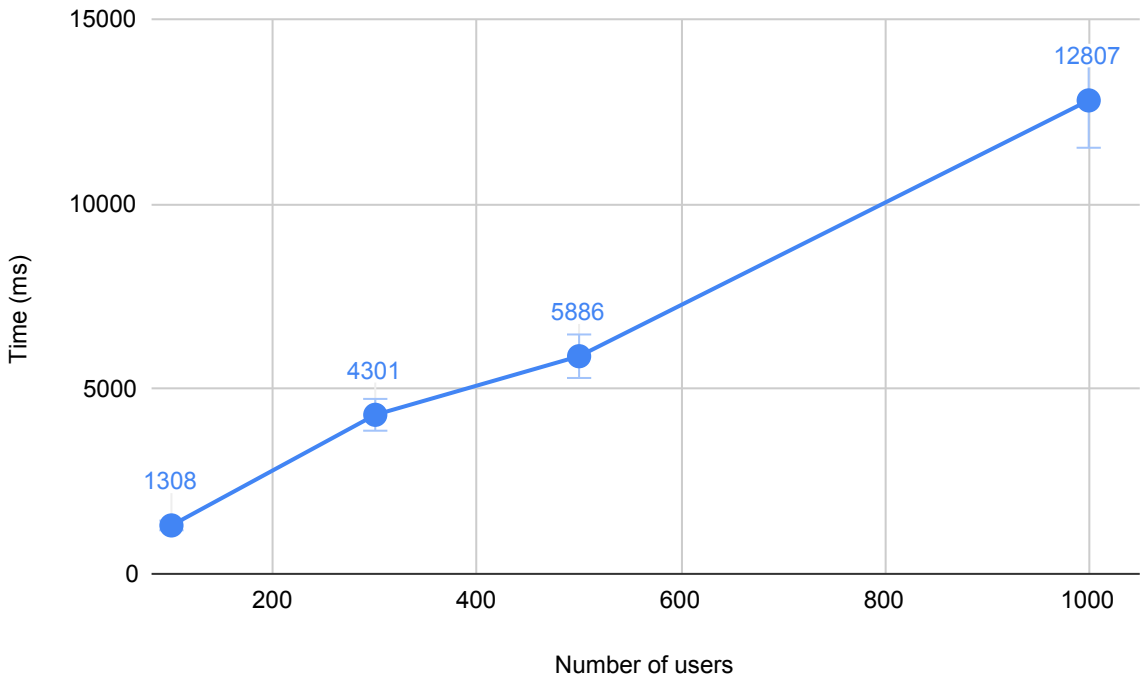

Figure 5.3: Time spent to process multiple user requests

As expected, time grows linearly as the number of users increases (Figure 5.3). Minor variations in the results come mainly from external processes related to Android.

To optimize this process, we would need to improve our queue. Instead of holding every contribution to wait, this queue should check for the existence of segment conflict to allow the processing of multiple contributions at the same time.

5.4 Usability Test

To prove that our application is easy to use and that any user can create a map and start making contributions, we decided to perform a usability test.

We defined a group of small tasks which are described below:

1. **Create a map with three attributes:** This task includes selecting a name, a description and three attributes (each one containing a name and a description);

2. **Select 1-10 segments and submit a contribution:** This task implies changing the map mode to be able to select the segments;

3. **Visualize the results of the contribution:** This task is simple and consists mainly of returning to the Visualization mode of the map and assimilate the results of the user contributions;

These tasks represent the core of the application and are meant to be performed with a low number of mistakes.

To perform this test, we picked a group of five users with ages between twenty-two to fifty-nine years old. They also had different levels of experience using mobile applications. They were informed about the goal of the application and the detailed tasks. Because there is currently no tutorial, we also revealed the existence of the two map modes.

Task	Average Time (s)	Average Number of Clicks	Average Number of Miss Clicks
1	74.3	20	1
2	34.3	14	3
3	4.9	1	0

Table 5.2: Results of the usability test

The average results are represented on Table 5.2.

Considering the **task 1**, we did not count the number of clicks while the user was writing (map description, map name, attribute names and attribute descriptions). The timer started to count when the user entered the home screen. This means that he had to navigate through our menu and reach the "MyMaps" tab to create a new map. During this task, most of the time was spent filling the `TextBox`'s as expected. There was no noticeable difficulty found by the users.

During **task 2** we noticed an issue. Some streets contain a significant amount of lines. This situation can difficult the process of choosing the desired segments. Since OpenStreetMap does not contain one line per street, we are not able to solve this problem. However, as mentioned before (in subsection 5.2.3), it is possible to mitigate this issue by removing unnecessary segments or joining small consecutive segments.

Task 3 was, as expected, easy to perform by our users. They all observed the results of their contributions without any difficulty.

The process of joining a community is simple. After that, the process of submitting contributions is identical to the scenario described above. For these two reasons, we decided to not simulate the case where a user joins a community map.

From these results, we can confirm that any user can create a new map instance and submit one or more contributions within a few seconds.

5.5 Database Update

To update the database, a `.pfb` file is necessary. This file contains a version of the OSM for a determined area. The process of updating the database follows two major phases:

1. The first phase consists of the extraction of the OSM data to our OSM copy database using osm2pgsql;

2. In the second one, the data from our secondary database is transformed and is used to update our current version of the CoopMap database.

Evaluating the first phase is not of our interest because we would be testing mainly the external tool osm2pgsql. It also depends highly on the file we are using to update our copy database (to perform this evaluation we used a file containing 186MB). Due to these two reasons, we are only evaluating the updating process between the OSM copy database and the CoopMap database.

To perform this test, we selected five different radius resulting in different numbers of street lines (Table 5.3). These radius will determine the area from which we will extract our segments.

Radius (km)	number of street lines
1	1017
3	8202
5	18672
10	34271
20	84832

Table 5.3: Radius values and their respective number of street lines

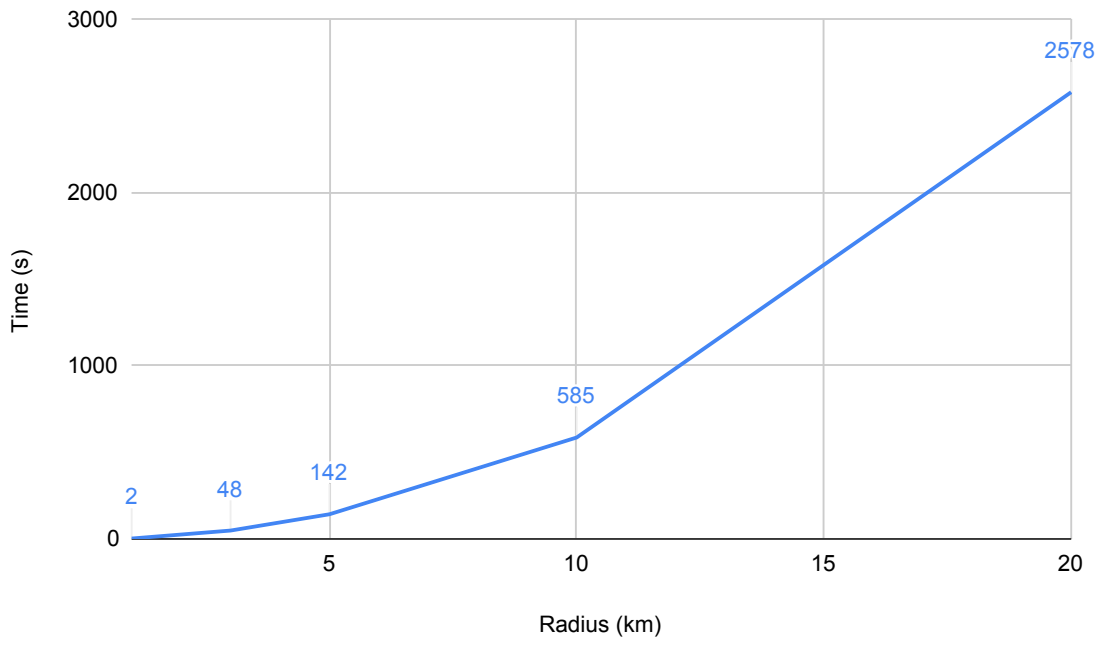

Figure 5.4: Radius values and the time they took to complete the database update

From Figure 5.4 we can notice an exponential growth of the time concerning the radius values. Our **Python** scripts iterates over all the street lines and compares them to each other which creates a time complexity of $O(N^2)$. This fact explains this polynomial growth when the number of segments increases. These comparisons are necessary to find lost lines and avoid repeated osm_ids.

A proper way of optimizing this database update would be dividing the city into tiles. Instead of running the full process for the entire city at once, we could run the script for smaller areas. Consequently, this would result in a significant decrease in the number of comparisons (lines from different tiles would not be compared).

Chapter 6

Conclusions

Not every kind of spatial information is available in a map format. Some communities may find useful spatial details related to their interests. For instance, the jogging experience can be improved if the streets hold a certain number of attributes. Thus, if there was a platform where users could report their spatial judgments of the streets, other people could use it to improve their decisions.

6.1 Achievements

The goal of this thesis was to provide an easy way of creating thematic maps where communities could work together to gather information not easily available. Based on the requirements defined at Chapter 3, we developed a platform where anyone could create or join a map and classify streets according to attributes defined at map creation.

We started by processing the data coming from the OpenStreetMap platform. As the original data was in a raw format, it was not easily readable by the rest of the platform. Therefore, we decided to create a separate database (CoopMap database) containing the data in an easy-to-read format so that our platform did not spend significant time dealing with the formatting process. Consequently, we were able to develop a way of updating our database without losing contributions made for unchanged street segments.

With the help of gRPC, we were able to implement a server able to process multiple contributions efficiently, allowing these contributions to be observed at run-time at every users' map.

In the end, we have developed a native mobile application, which delivers an easy way of interacting with the street lines provided by the OpenStreetMap platform. The user can visualize the map data in a very intuitive way and make contributions in a separate Android activity.

6.2 Future Work

Our platform is composed of three separate components: database, server and client. All these components contain certain aspects which can be improved:

- **Database**: The introduction of **Osmfilter** [44] could improve the time necessary to perform a database update.
- **Back-End Server**: Optimizing the queue to process multiple contributions from the same map in parallel would improve the overall performance of the platform;
- **Mobile Application**: Reducing the number of segments would optimize the process of selecting the desired segments in the device.

This thesis was developed considering the context of a city. We would be able to scale this platform to larger areas by improving the database update script, which would use tiles instead of updating the whole city at once. It would be interesting to apply the concepts developed for this platform to a whole country.

Bibliography

[1] Waze. https://web.archive.org/web/20191024175015/http://waze.com, 2019.

[2] A. Mobasheri, J. Deister, and H. Dieterich. Wheelmap: the wheelchair accessibility crowdsourcing platform. *Open Geospatial Data, Software and Standards*, 2(1):27, Nov 2017. ISSN 2363-7501. doi: 10.1186/s40965-017-0040-5. URL https://doi.org/10.1186/s40965-017-0040-5.

[3] B. Ciepluch, P. Mooney, R. Jacob, and A. C. Winstanley. Using openstreetmap to deliver location-based environmental information in ireland. *SIGSPATIAL Special*, 1(3):17–22, Nov. 2009. ISSN 1946-7729. doi: 10.1145/1645424.1645428. URL http://doi.acm.org/10.1145/1645424.1645428.

[4] A. Ballatore and J. J. Arsanjani. Placing wikimapia: an exploratory analysis. *International Journal of Geographical Information Science*, 0(0):1–18, 2018. doi: 10.1080/13658816.2018.1463441. URL https://doi.org/10.1080/13658816.2018.1463441.

[5] Openstreetmap. https://web.archive.org/web/20191026111558/https://www.openstreetmap.org/, 2019.

[6] E. Estellés-Arolas and F. González-Ladrón-De-Guevara. Towards an integrated crowdsourcing definition. *J. Inf. Sci.*, 38(2):189–200, Apr. 2012. ISSN 0165-5515. doi: 10.1177/0165551512437638. URL http://dx.doi.org/10.1177/0165551512437638.

[7] D. Geiger, S. Seedorf, T. Schulze, R. C. Nickerson, and M. Schader. Managing the crowd: Towards a taxonomy of crowdsourcing processes. In *AMCIS*, 2011.

[8] Howe and Jeff. The rise of crowdsourcing. *Wired*, 14, 01 2006.

[9] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Comput. Surv.*, 51(1):7:1–7:40, Jan. 2018. ISSN 0360-0300. doi: 10.1145/3148148. URL http://doi.acm.org/10.1145/3148148.

[10] D. Yang, G. Xue, X. Fang, and J. Tang. Incentive mechanisms for crowdsensing: Crowdsourcing with smartphones. *IEEE/ACM Trans. Netw.*, 24(3):1732–1744, June 2016. ISSN 1063-6692. doi: 10.1109/TNET.2015.2421897. URL https://doi.org/10.1109/TNET.2015.2421897.

[11] D. L. Estrin. Participatory sensing: Applications and architecture. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 3–4, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. doi: 10.1145/1814433.1814435. URL http://doi.acm.org/10.1145/1814433.1814435.

[12] A. L. D. Moraes, F. Fonseca, M. G. P. Esteves, D. Schneider, and J. M. de Souza. A meta-model for crowdsourcing platforms in data collection and participatory sensing. In *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 429–434, May 2014. doi: 10.1109/CSCWD.2014.6846883.

[13] N. Maisonneuve, M. Stevens, M. E. Niessen, and L. Steels. Noisetube: Measuring and mapping noise pollution with mobile phones. In I. N. Athanasiadis, A. E. Rizzoli, P. A. Mitkas, and J. M. Gómez, editors, *Information Technologies in Environmental Engineering*, pages 215–228, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-540-88351-7.

[14] J. Tweddle, L. Robinson, M. Pocock, and H. Roy. *Guide to citizen science: developing, implementing and evaluating citizen science to study biodiversity and the environment in the UK*. NERC/Centre for Ecology & Hydrology, Wallingford, 2012. URL http://nora.nerc.ac.uk/id/eprint/20678/. Freely available online - click on Official URL link for full text.

[15] P. Minder and A. Bernstein. Crowdlang: A programming language for the systematic exploration of human computation systems. In *Proceedings of the 4th International Conference on Social Informatics*, SocInfo'12, pages 124–137, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-35385-7. doi: 10.1007/978-3-642-35386-4_10. URL http://dx.doi.org/10.1007/978-3-642-35386-4_10.

[16] T. W. Malone, R. Laubacher, and C. Dellarocas. The collective intelligence genome. *Sloan Management Review*, 51(3):21–31, 2010. URL http://www.lhstech.com/chair/Articles/malone.pdf.

[17] K.-J. Stol and B. Fitzgerald. Two's company, three's a crowd: A case study of crowdsourcing software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 187–198, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568249. URL http://doi.acm.org/10.1145/2568225.2568249.

[18] Amazon mechanical turk. https://web.archive.org/web/20191024175425/https://www.mturk.com/, 2019.

[19] Microworkers. https://web.archive.org/web/20191019080606/https://www.microworkers.com/, 2019.

[20] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17(2):16–21, Dec. 2010. ISSN 1528-4972. doi: 10.1145/1869086.1869094. URL http://doi.acm.org/10.1145/1869086.1869094.

[21] J. Chandler, G. Paolacci, and P. Mueller. *Risks and Rewards of Crowdsourcing Marketplaces*, pages 377–392. Springer New York, New York, NY, 2013. ISBN 978-1-4614-8806-4. doi: 10.1007/978-1-4614-8806-4_30. URL https://doi.org/10.1007/978-1-4614-8806-4_30.

[22] J. Rogstadius, V. Kostakos, A. Kittur, B. Smus, J. Laredo, and M. Vukovic. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets, 2011. URL https://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2778.

[23] J. Redi and I. Povoa. *Crowdsourcing for rating image aesthetic appeal: better a paid or a volunteer crowd?*, pages 25–30. Association for Computing Machinery (ACM), United States, 2014. ISBN 978-1-4503-3128-9. doi: 10.1145/2660114.2660118.

[24] Crowdcrafting. https://web.archive.org/web/20121130022503/http://crowdcrafting.org/about, 2019.

[25] D. Schall. *Service-Oriented Crowdsourcing: Architecture, Protocols and Algorithms*. SpringerBriefs in Computer Science. Springer New York, 2012. ISBN 9781461459569. URL https://books.google.pt/books?id=yJGxlcJafJgC.

[26] K. Crowston. Amazon mechanical turk: A research tool for organizations and information systems scholars. In *Shaping the Future of ICT Research*, 2012.

[27] S. Elwood, M. Goodchild, and D. Sui. Researching volunteered geographic information: Spatial data, geographic research, and new social practice. *Annals of the Association of American Geographers*, 102:571–590, 05 2012. doi: 10.1080/00045608.2011.595657.

[28] H. Senaratne, A. Mobasheri, A. L. Ali, C. Capineri, and M. Haklay. A review of volunteered geographic information quality assessment methods. *International Journal of Geographical Information Science*, 31, 2017.

[29] Twitter. https://web.archive.org/web/20191026145550/https://twitter.com/, 2019.

[30] Wikipedia. https://web.archive.org/web/20191026045238/https://www.wikipedia.org/, 2019.

[31] Flickr. https://web.archive.org/web/20191026054437/https://www.flickr.com/, 2019.

[32] Towit. https://web.archive.org/web/20190403013241/http://towit.io/, 2019.

[33] Safetipin. https://web.archive.org/web/20190828110236/http://safetipin.com/, 2019.

[34] Postgresql. http://web.archive.org/web/20191206152023/https://www.postgresql.org/, 2019.

[35] Osmosis: A command line java application for processing osm data. available for download at http://wiki.openstreetmap.org/wiki/osmosis, 2009.

[36] Ogr2ogr. https://web.archive.org/web/20191024131558/https://gdal.org/programs/ogr2ogr.html, 2019.

[37] Mapsforge. https://web.archive.org/web/20190128132920/https://github.com/mapsforge/mapsforge, 2019.

[38] Linux cron. https://web.archive.org/web/20191010024546/https://en.wikipedia.org/wiki/Cron, 2019.

[39] Geofabrik. https://web.archive.org/web/20191019103737/https://www.geofabrik.de/, 2019.

[40] grpc. https://web.archive.org/web/20191019131951/https://grpc.io/, 2019.

[41] Firebase. https://web.archive.org/web/20191021103717/https://firebase.google.com/, 2019.

[42] Google identity platform. https://web.archive.org/web/20191002204741/https://developers.google.com/identity/, 2019.

[43] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.

[44] Osmfilter. https://wiki.openstreetmap.org/wiki/Osmfilter, 2019.