World Cities and Places - A Microservices Application

Abstract goes here

Elena Makarenko

Jose I. Retamal

B.Sc.(Hons) of Science in Computing in Software Development

May 10, 2020

Final Year Project

Advised by: Dr. Dominic Carrl
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

About this project

Abstract A brief description of what the project is, in about two-hundred and fifty words.

Authors Explain here who the authors are.

Contents

List of Figures

Chapter 1

Introduction

This project has been developed as part of the BSC(Honours) in Science in Computing in Software development course for the module minor project and dissertation on the last year of a fourth-year course. The module corresponds to 15 credits out of 60 in the year.

Two students have developed the project, which has been divided into two parts: front-end and back-end, Elena was is developed the front-end and Jose the back-end.

We have developed a microservice application using go as a principal back end programming language, with a react native client and a REST-API(Figure 1.1).

1.1 Objectives

Doing this project, we expect to learn and prepare to afront the professional software environment, so we tried to organize our work in such a way that would imitate the real-world software development process, including the use of best practices and contemporary tools. The main objective of this project is to learn as much as we can to be ready to start our professional path in the software industry. Some of the objectives that we expect to achieve by doing this project are:

- Research and learn new technologies that are utilized in the software industry.
- To improve soft skills, including teamwork, critical thinking, time management, and problem-solving.
- Apply agile techniques to develop an application based on initial requirements.

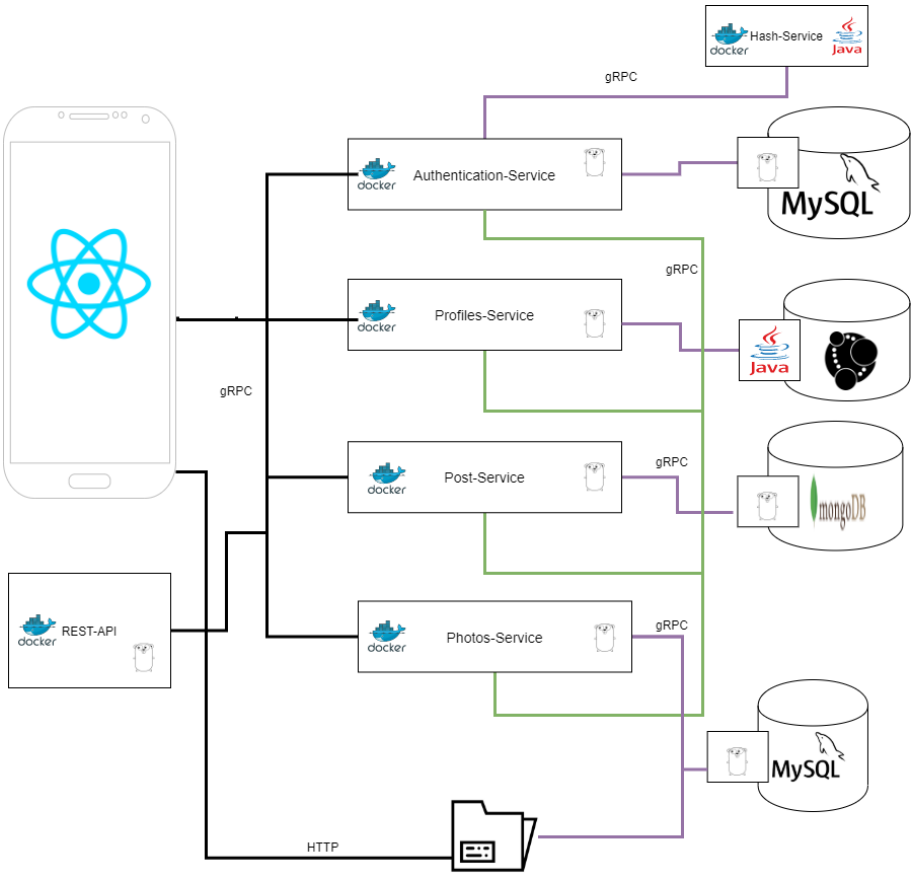


Figure 1.1: Application - UML.

- Develop a scalable microservices distributed system to manage a dynamically growing database.
- Implement the system on the internet using the most popular cloud providers. Develop an application that efficiently works with images.
- Implement a reliable authentication system that securely stores passwords using hash and salt.
- To create a CRUD (create, read, update, delete) application.
- Develop a high-performance mobile application that integrates the microservices.
- Develop a REST API that integrates the microservices.
- Works in a team utilizing flourishing software development tools.

1.2 Design overview

In this project, we utilize cutting-edge technologies to develop a microservice application. The application is developed using relational, document base, and graph databases. We look at the benefits of each database, and we use them where we can maximize performance base on how the data is store and access on each type of database.

The System is composed of a total of 10 applications: 4 primary services, 1 hashing service, 4 database access applications, and the REST service. The services use 4 databases and 1 file storage bucket (Figure 1.1). All services are encapsulated, perform a specific task, they communicate using gRPC, and run in a docker container.

The advantages that we want from developing the application as a distributed system are [1]:

- Avoid expensive queries for fast performance. We use multiples databases to store the data in a simple way that makes access very fast. We avoid joins and any expensive query.
- Scalability, each component performs a specific task; more instances of each service can be added.
- Flexibility, each service is independent and composed of independents components as well; more components to perform a different task can be added on each service can be added, and new services can be created for a completely new task.
- Maintainability, services are easy to maintain because they are encapsulated and perform simple tasks.

1.2.1 The Solution

We have designed a tourism application where users create the data. It stores data about three main types: users, cities, and places; all three have a public profile(public to register users of the application). Users create the cities and the places, and they can be created only once. Cities and places have a profile where users can post. Users can mark Cities and places as visited, so they show in their profiles and have actualization about recent posts.

In brief, it is a social media application similar to TripAdvisor. The main features of the application are

- Register/login/logout.
- Create a city if a city is not yet created (add photo and description).
- Add a place in a city (add photo and description).
- Write a post about a city (add photo and description).
- Visit a city (all visited cities displayed in his/her profile).
- Visit a place (all visited places displayed in his/her profile)read .reviews/posts about a city.
- Review all the cities that were created.
- Review places in the city that were added.
- Review all posts about a place.
- Change information about himself/herself (upload photo, change. name, and description) in settings.
- Review all the personal information in the profile (photo, geolocation, email, name, description).
- Search a city.

1.2.2 The Microservices

The application is divided into four main microservices:

- The authentication service provides secure authentication storing passwords in binary format using hash and salt.
- The profiles-service store primary data of all profiles.
- The post-service, store, and manage posts.
- The photo-service, manage, and stores photo for profiles and posts.

1.2.3 Databases Design

Each service has its database. We have research best database that fits each of the services:

- The authentication service uses a relational database. When a user login a token is generated, that token can be used to authenticate each request that the user performs. A leader/slave replication has been set up to improve performance. The read from the database is distributed on multiple databases to perform the most used operation that is to authenticate requests.
- The profiles services use a graph database. We used the graph database to create relations that avoid complicated queries.
- The post-service use a document-based database. Posts are indexed and stored in scalable documents. We can use inexpensive queries to perform CRUD operations in the posts of a city or place.
- The photo-service store images in the file system and use a relational database to store URLs. Binary images have public access from the file system, so the image is loaded directly from the client without the need to send the image through the service, the service store URL which is sent to the user.

1.2.4 Technologies introduction

After some research, we decide to use Goland as the main back end programing language; some parts of the system were also writing using Java.

We have used gRPC as the communication interface for the microservices because it provides a transparent client/ server relation. We created a go module with all gRPC interfaces that can be imported in all the components of the application.

All the services are run using docker containers, and the images are published in Docker Hub and then pulled from virtual machines to run the service.

We have work with the 3 most popular internet services providers(Azure, google cloud, and AWS), all of them offer free student credit, and we use that. In general, the 3 services have excellent performance and outstanding support, which was used several times to learn and fix errors.

We find that one of the best for us was google cloud because they live chat and convenient SSH on the browser. Also, the container optimized OS is outstanding to work with docker images.

1.2.5 Methodology

The project has been developed in an iterative approach based on agile principles, based on the original principles we have created an adaptation to feet the demands of our project.

We define stages that are reviewed during all the project development, measure progress based on the working code, continuously meet with the team, and be ready to adapt to any change in circumstances.

The solution was created component by component, integrating them after several independent testing, we keep a working solution all the time to which we integrate more components as they are developed.

1.3 Authors

Authors of this dissertation are: Jose Retamal and Elena Makarenko. Introduction and Conclusion of the current paper were written in cooperation. Each of the rest of the chapters: Methodology,Technology Review, System Design, System Evaluation, were divided into two parts: back-end side of the application and front-end side, each part was done by Jose Retamal and Elena Makarenko respectively.

1.4 Overview

The structure of our dissertation is divided into chapters, each one containing different aspects of the project. A brief overview of the content of each chapter is provided below:

1.4.1 Methodology (Chapter 2)

This section outlines the methodologies that were used in order to ensure the project's success. Mainly it describes Agile methodology, version control tool, and testing. It also explains the criteria to chose languages, platforms, and technologies that were used.

1.4.2 Technology Review (Chapter 3)

In this section, we research and decide the best technologies to use in each component of the application. We base our research on the criteria set in the methodology. Then we reviewed the technologies, describe it, and give an assessment.

1.4.3 System Design (Chapter 4)

This section provides a detailed explanation of the overall system architecture, thoroughly describing each component in the system. We describe how those components are linked together and how they communicate.

1.4.4 System Evaluation (Chapter 5)

We show how the system works and how the testing is performed. It also criticizes limitations and advantages by taking a detailed look at the pros and cons of the technologies that were used. Screenshots of the final UI can be found here as well. We also talk about the problems encountered while developing this project and solutions to those problems.

1.4.5 Conclusion (Chapter 6)

This section of the dissertation evaluates our project against the objective. We look at the achieved goals, set a path of progress, and analyze challenges encounter during the development of the project.

Chapter 2

Methodology

In this section, we describe how we develop the project, we set our basis on the start, and them follow them during the development of the project.

The section is divided in Back-end and Front-End because there where slightly different approaches in the developing process.

2.1 Back-End

Jose I. Retamal

Most successful software projects are developed using some agile methodology[3]. There is a reason for this and is because the software is an intellectual product that needs to adapt to change on requirements, libraries, and hardware. This constant change required adaptative and iterative approaches to develop.

We adapted some of the agile principles to our project [3]:

- Continuous delivery. The application is tested at all stages, and we keep a working application all the time.
- Adapt to changes at any stage. Even if we started with a design and chose the technologies to be use, they can change at any stage.
- Work together with all participants. We continuously meet with all people involved in the project, check the stage of the project, and check if there is anything that needs to change.
- Self-motivation. There is a personal interest in the project to all participants.
- Working software is the measure of the stage of the project. What is done and working in the software is the main reference point on at what stage we are on the project. Simplicity. We develop a working application most simply, but without leaving the performance aside.

- Self-organizing. Everyone in the project organizes himself in the way they think it would be the best way to have a maximum amount of productivity.

Stages of the Project

There where three stages in the development of the application. They are not a hard stage, and they overlap each other(Figure 2.1). These stages are a reference to know what to do, and they are regularly reviewed.

- System Design.
- Technology Research.
- Implementation.

System Design We create a simple design of the system at the start. This defines, in a general way, the main components of the system. This was a quick process, and the design was always subject to updates. To design the system, first, the requirements where set. Then the design was done following them.

- Design the full system in a general way, understand more or less the number of services required.
- Define the function of each component.

Technology Research After we have an idea of the system, some time was spent on looking for the best technologies that will suit the application, some prototypes where design to check if they work.

- Decide the main technologies to be used.
- Test compatibility.

Implementation The implementation has been done using an iterative process. Starting from the authentication service, the system was build block by block. After one component was developed, it was tested to check that the full system works.

On each component, we start by designing the database, the DBA, and then the main service with the endpoint to the client.

When starting to develop a new component, the full system design has to be check.

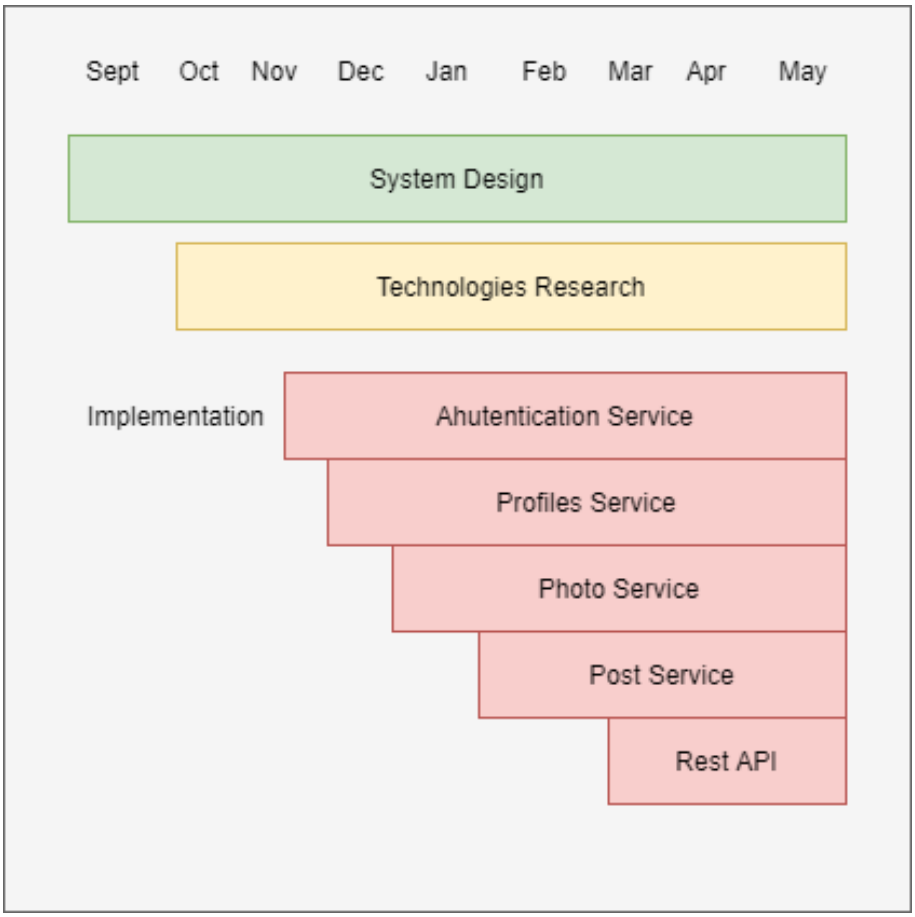


Figure 2.1: Methodology- Project Gueneral Gant Chart.

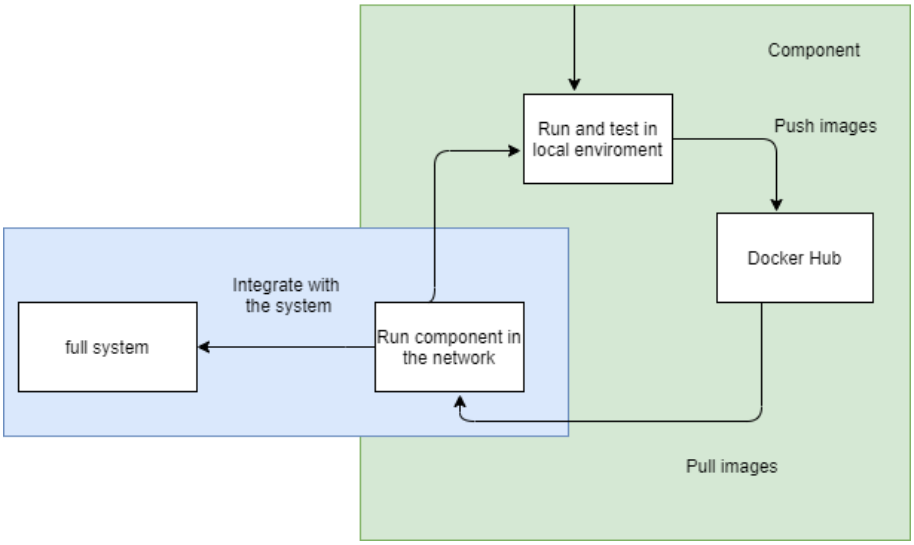


Figure 2.2: Methodology- Component Iteration.

2.1.1 System Development life cycle

Each component is developed iteratively after some functionality is implemented the component is first tested locally, them docker images is created ann updated to docker hub, them the image is downloaded in the server, and the whole system is tested. The life cycle steps are below(Figure 2.2):

- Build and test the component locally.
- Create/update a docker image and update it to the docker hub.
- Pull from docker hub, run the component in the network, and test.
- Test the full system.

2.1.2 Selection Criteria

We have applied the following criteria to chose the technologies used in the project.

- Accessibility. Technologies must be accessible without a cost; they can be free and open-source or offer student free usage.
- New to us. One of the ideas of the project is to learn how to work with new technologies and learn. So we look for technologies that are not familiar to us.

- Popular and used in They are professionally used. We want to learn how to use technologies that are popular and used in professional software development.
- Mature and recognized. We prefer technologies with background and maintained/developed by a recognized organization or a big open source community. Even if there are always new technologies coming out, we prefer those that are already established.
- It adapts well to the purpose need in the system. Simple and with resources to learn. We need to learn the technologies, so we instead chose those that are fast to learn and with many resources(tutorials and documentation) to access.

2.1.3 Testing

Because of the time frame, we do not develop the application using automated testing. We know there are many advantages to it, but we need to trade off this feature. Some aspects that we considered to decide that automated testing is not for the project: Most of the benefits of automated testing are long time benefits. Automatested testing is practical another application that needs to be maintained.

Instead, based on the principle of self-motivated development, we develop the project using a test culture that relies on developers: Each functionality needs to be tested by the developer at developing time. Bugs found needs to be personally reported to the developer. When integrating new components, the full system needs to be checked.

2.1.4 Security considerations for authentication

- Feedback for password strength.
- hash+ salt using bcrypt function on server and client.
- constant time function for compare hashed passwords.

2.1.5 Tools

Integrated development environment

- Eclipse for Java development. Solid IDE with just the right performance, we choose it because its well knows for us and free to use (`https://www.eclipse.org/`).
- Goland, for Go development. Excellent IDE is free for students(`https://www.jetbrains.com/go/`).

- Visual Studio Code, general code review, and write docker images(`https://code.visualstudio.com/`).

SHH Client

- Putty, free shh client, open-source, and simple to use(`https://www.putty.org/`).

Version Management

- Git, very popular with string community, we use the prevalent GitHub server(`https://git-scm.com/`).
- Docker Hub, Manage containers, free for public containers (`https://hub.docker.com/`).

2.2 Fron-End

Elena Makarenko

The Methodology section describes actions that were taken and tools that were used in order to ensure that all of the project goals are achieved within the given constraints.

2.2.1 Agile

Sometimes we underestimate the importance of planning and analysis. Time spent on these activities might seem to be a waste of time. Nevertheless, real-world experience shows that impact of the planning phase on project success is crucial.

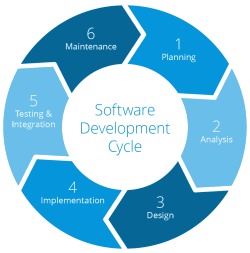


Figure 2.3: Agile Methodology

The project management in general is important as it helps to achieve objectives more efficiently and reduce the probability of failure. It also helped us to organize and prioritize our work.

Even though it was quite tempting to start writing the code straight away before doing that we set up the scope of our project. We tried to collect user stories or requirements in order to write a prioritized list of work that needs to be done - Product Backlog. These are the desired functionality that an application should have. Next, the work was divided into small portions in order to minimize the amount of up-front planning and design.

At the planning stage Jose also drew up the architecture design of our project. As well at this stage we agreed on technologies and languages that we were going to use.

Throughout the entire project we tried to meet every week together with our supervisor in order to define the scope of work that needs to be done over the next 7 days. Consequently, our sprints were a week long. At each iteration we tried to add more functionality and reduce the amount of bugs.

I think from the very beginning it was clear that Agile software development style Fig. 2.3 best matches our needs. The advantages of using Agile over traditional waterfall model or others are numerous. It advocates iterative approach and continuous improvement. Although, it implies a lot of planning, it is also adaptive to changes. It acknowledges that requirements can evolve and change over time.

2.2.2 Project Management Tool

As most of the project management tools like Jira or Wrike have only free demo versions for a limited amount of time and our project was a year long, we decided to use github project board. It is free, has all of the functionality a project management tool has and it is very convenient as everything is located in one place.

2.2.3 Version Control Tool

Even a small software project cannot do without a version control manager.

The question whether to use GitHub Fig. 2.4 or any other tool did not even arise as we used it for most of our college and personal projects and were already familiar with it. GitHub is a version control tool that provides hosting for software development version control using Git. It tracks all the changes to documents and files, has tools for basic task management and has excellent documentation. Last but not least it is used by the majority of software companies.

In our project we tried to create a new branch for every new feature. As soon as the code was written and tested the branch could be merged into master. This approach helped to isolate development work without affecting other branches in the repository (not to mess the master branch and each others code).

Figure 2.4: GitHub

As already mentioned in the previous section it was also used as a project management tool.

2.2.4 Testing

Testing is an significant part of a development process especially if you are using agile methodology. In order to test UI manual testing was used, which is also known as a Black box testing (a method of software testing that examines the functionality of an application without peering into its internal structures or workings).

Chapter 3

Technology Review

3.1 Back-End: Technologie Research and Decision

Jose I. Retamal

We start with some research about the best technologies that can be used to develop the system. The technologies that we need are:

- Backend programming language.
- Communication interface.
- Databases for each service.
- How to store images
- Client technologies.
- How and where run the services.

We have used the criteria described in 2.1.2 for deciding the technologies.

3.1.1 Backend Programing Language

Jose I. Retamal

We want a simple, with excellent performance, fast and straightforward to learn programing language. Go is a modern programming language that was specially designed by Google to develop scalable software systems. After reading some forums where developers gave opinions about how it is to work with go, we have extracted the advantages and disadvantages of it.

https://builtin.com/software-engineering-perspectives/golang-advantages
https://www.pluralsight.com/blog/software-development/golang-get-started advantages

- Go was specially designed to make software designed easier.
- General-purpose back end language.
- Simple to understand can be pick up quickly by new developers.
- Go is designed to be simple, avoiding having many features and flexibility to be simple and very practical.
- Enforce a coding style.
- Simple concurrency primitives.
- Very fast compiling time and small usage of memory.
- Very mature libraries for gRPC.
- Static typing that speed up developing time.
- SDK for all popular cloud providers.
- Built-in testing and benchmarking.
- Fast publish of libraries through Github.
- Scalability, go was designed by Google with scalability in mind.
- Go was designed by experienced developers having in mind the industry requirements and taking into consideration the developer's experience.

Disadvantages:

- Relative young programing language.
- Code with no-frills means that you write more code than other modern programming languages like Ruby.
- Go has no classes, so there is a need to think about the design differently. Go is not an object-oriented programing language.

Most of the disadvantages have a positive aspect, as well. Writing more code sometimes makes it more transparent and easier to debug. Also, having to design the program in a not object-oriented way make the design more simple and straight forward. Goland fits all the requirements we are looking for the backend programing language, and we think it will be excellent to learn and based on other experiences that seem like it is excellent for backend, and we found that many big companies used it as primary backend programing language.

3.1.2 Communication Protocol

Jose I. Retamal

We need a protocol for the microservices to communicate. We have considering using REST or gRPC.

We need many services to communicate in a fast and reliable way, and they would all share the same network that is the internet.

Some advantages of gRPC over REST are:We have compared gRPC with rest and found advantages of gRPC over REST for our system:

- CLear defined Interfaces.
- Bidirectional streaming.
- Much faster and compact.
- East to understand and implement.
- Lossy coupled between client/server.

The interface is defined using Protobuffers, which is programming language independent. The same code is compiled for the different programming languages, it has libraries in most programming languages, and it does have with Go and Java that we will use in the application.

So gRPC is a modern implementation of RPC that has been developed by Google to meets requirements of high traffic distributed systems. It uses HTTP and is fast to learn and easy to implement.

3.1.3 Ahutentication Service Database

Jose I. Retamal

We need a database on where the data can be accessed quickly and reliably by index. We are also looking for a database that can be replicated. A relational SQL database fits the requirements. From MariaDB, MySQL, PostgreSQL, and oracle. MySQL and PostgresSQL have been selected for further analysis.

MySql

MySQL 5.6 [4]

Pros

- Oracle has brought more investment into MySQL, meaning there is a future with it.
- It is a solid product. MySQL 5.6 is a reliable product with all the features fully tested.
- There are many teams from Oracle working in MySQL.

Cons

- Not so mature as other relations databases like PostgreSQL. This means that it has fewer features than the other more mature database systems.
- Not fully open source anymore, in theory, is open source, but in practice, Oracle has taken over.
- Many have replaced MySQL for MariaDB. Since oracle takes over MySQL, many big names like RedHat Enterprise Linux, Fedora, have moved to MariaDB.

PostgreSQL [5]

Pros

- Reach libraries for transactions. Fully documented with comments made it easy to know what part of the code does what and how it is done.
- Many adjustable parameters make the system easier to personalize.
- Easy to extend, if extra features are needed is possible to add the feature. Secure and reliable, also security can be personalized and extendible.

Cons

- Open source, not owned for any organization that takes care of it.
- Slow performance, there have been reported issues with performance and backup recovery.

Conclusion There is not a real need for a feature-rich database; the database that we need to implement is simple, and the main aspect to consider is the replication—also, some of the cons of MySQL that be considered as advantages. Because MySQL is robust and has all the functionality needed, it has been chosen as the best option.

3.1.4 Profiles Service Database

Jose I. Retamal

We need a database that store names and description of the users, cities, and places. Also, we need to access places that are in a city, and users need to be able to mark any city or place as visited for, then get info and make comments about the places that they have been.

It makes sense to have some relationship between the users and the places/cities to connect them. Also, a relationship between the place and city can be used.

We Considered three types of databases: relational, document, and graph database. If we chose the relational database, we would need to have a table where each user has entries for the cities and places they visit. To get the data, we will need to perform expensive joins. If in the future we want to add more relations like friendships, for example, it will get more complicated, and queries will be more expensive.

If we chose a document database, we could have like a list of places and a list of cities for each user, this would be easy, and queries would be not so expensive but there would it would use much more storage.

For a graph database relationship are natural, it has flexibility and would be much easier to add extra relationships and also more fields on each node if necessary. The performance would be good, and the relations will not produce the use of extra storage as with a document-based database [6].

Considering this, we have chosen a graph database to store the profiles, the key advantages this type of database will give are:

- Performance, relationships are natural for a graph database; queries would be no expensive.
- Flexibility, more fields, or relationships can be added in a relatively easy way.

Considerations

Some accessible graph databases are TigerGraph, Neo4j, and DataStax. They all have great performance and offer more or less the same functionality [7]. We have chosen Neo4j because it has a more significant community and is more popular, meaning that there is more online documentation and resources.

Database access driver

After we have decided what database would be used, we need to choose the programing language and the driver to access it. Given the main pro-

graming language of the system is going, we have considered this programming language. We also research java for the driver.

Drivers considered:

- Go -https://github.com/johnnadratowski/golang-neo4j-bolt-driver
- Java- https://github.com/neo4j/neo4j-java-driver

Decision

Neo4jdatabase with the official java driver.

3.1.5 Store Images

Jose I. Retamal

We have considered a few ways to store the images.

- Store image in MongoDB using Gridsfs https://docs.mongodb.com/manual/core/gridfs/
- Store images in MySQL database using binary blobs.
- Store in the file system and store URL in a database.

The best way to store the images is to use the file system and store the URL in another database, because:

- Fewer data need to be transferred. If we store the image in a database, it would need to be retrieved from the database and then sent to the user. So the service will need to deal with a huge amount of data. If we just store the URL and allow the user to access the file system, the service will need to deal with a much lower amount of data. Resulting in better performance.
- Much easier to escalate. More buckets can be easily added, and the amount of data store in the database is meager and can be stored in a very simple way, just as single tables without the need to join data.
- Internet services providers(Google Cloud, Azure, and AWS) supply ready to use file systems that can have public access: buckets. They are cheap, have high performance, and are scalable.

To store the URL, we have decided to use a relational database because we need to access the data using keys.

3.1.6 Post Database

Jose I. Retamal

For store post, we need a database that:

- Can store a huge amount of data, because posts are the things that occur more in the application because cities and places can have "any" amount of post each we expect that they would more posts that anything in the application.
- The post would need to be stored by a city or place, and they would be for one particular city or place. Some sort of indexing would help performance.

In resume, we need a highly scalable database with little structure, on this description fits perfect a document-based database.

We have considered three different document base databases: InterSystems cache, MongoDB, and Apache CouchDB(). We have chosen MongoDB because:

- Simple to use.
- It easy to install in the Linux environment.
- Native replication is included.
- Even if it has fewer features than the others, it is good because we don't really need the extra features.
- It has a go driver to access the database.

3.1.7 running the services

Jose I. Retamal

When it comes to research, the best way to run a service, it looks like there is only one path to follow, the docker container.

Docker is an open-source project that aims to automate the development, deployment, and running of the applications. Applications run in independent containers that use just the needed amount of resources.

It allows packing the full application for then deployed and run it where needed.

Advantages of using docker [2]:

- Fast deployment, no need to set up the environment,we only install docker in the VM, and we are good to go.

- The management of containers is easy. We can manage each service and updated it in a fast way.
- Proper Usage of resources, since the application runs in a container, do not use extra resources.
- Easy and fast start of the containers, the app is running in a few seconds with just one command.

3.2 Front-End

Elena Makarenko

3.2.1 React Native

I assume the biggest challenge in mobile development nowadays is the heterogeneity of mobile operating systems. One way of solving this problem is to write code for each platform in its own language. The other solution to this could be cross-platform development, which allows to develop an application for different platforms at once.

React Native is an open-source javascript mobile application framework created by Facebook in 2015 for developing mobile applications for iOS and/or Android. These 2 platforms cover more than 90% of market share, which makes their choice clear.

It is a native scripting framework which allows to create cross-platform phone/tablet apps with one code base. Facebook call this approach "Learn once, write everywhere". Moreover, most of the code can be used for both platforms, and adding platform-specific code is quite simple. In our project we have implemented only android version due to time constraints and lack of IOS device. Probably technology evaluation might seem to be incomplete as we did not have experience with development of code for both platforms, still a research was conducted: information about RN was gathered, analyzed and evaluated. Thus, we believe we can give an unbiased assessment and draw some conclusions about this framework.

Most of the principles and concepts are derived from the web framework ReactJS. Because of these similarities web developers can quickly learn how to write mobile applications without prior knowledge in the native languages such as Java and Objective C/Swift respectively.

Features

Native scripting applications. There are different ways of developing cross-platform applications and each of them has its own pros and cons [**?**]: 1. Mobile web applications. An application that is accessed in a web

browser. Back in the days the limitations of these apps included requirement of internet connection and inability to access the mobile device software and hardware such as camera, and GPS sensors. Nevertheless, nowadays Progressive Web Apps (PWA) gain more and more popularity leaving those limitations behind.

2. Hybrid applications. Simply speaking, it is a combination of the web App and the native App but has a lower performance compared to native applications (e.g. Ionic).

3. Cross-compiled applications.

A cross-compiled application is an application written in a non-native language which can be compiled into a fully native application using a cross-compiler(e.g. Xamarin).

4. Native scripting/Interpreted applications. Separate tools and programming language is used for each certain mobile platform. Since interpreter is used to call native APIs everything that is possible in a native application is possible through native scripting. However this will result in a loss of performance compared to calling the native environment directly.

React Native uses native scripting approach.

Virtual DOM. There are 3 main scenarios how the page is rendered when changes on the page occur:

1. A new HTML request is sent to the server in order to re-render the entire page.

2. Client-side HTML templating is used that re-renders parts of the page.

3. Imperative HTML Document Object Model (DOM further in the text) changes (considered to be the most efficient one).

All these approaches have their own merits and flaws.

For example, the last two are quite expensive in terms of processing and time as good chunks of DOM or even the whole DOM might need to be re-rendered. Nevertheless, it allows to achieve code readability and maintainability.

ReactJS attempts to use the advantages of the aforementioned approaches and work around the disadvantages by using a virtual DOM. When changes occur the virtual DOM creates a queue/patch of imperative DOM operations that is asynchronously added to the Browser DOM [**?**]. React Native uses the same approach as ReactJS but instead of a virtual DOM it uses a virtual application hierarchy.

JSX and Components Another distinctive feature of React is JavaScript syntax extension, that looks similar to XML, called JSX. This code is compiled into regular JavaScript code. Consequently, JSX is just a syntactic sugar and anything written in JSX can be written in plain vanilla javascript

but still it is highly recommended to use as it allows to make code more readable and easy to write.

The main idea of React is using components like building blocks in order to create an app. They can be swapped with each other, they are are reusable, interactive and stateful. States are used in order to merge data into components and they are the core of the React interactivity. SetState method is used in order to pass the data to a component which is then re-rendered.

Threads React Native runs on two threads as well as with additional dispatch queues where the the layout is handled. The main thread, where the native UI rendering is performed, and a JavaScript thread that runs the JavaScript code of the React Native application. Every native component also uses a Grand Central Dispatch Queue15 (iOS) or a dedicated MessageQueue16 (Android) to handle threads.

Native Modules In our project we use React Native CLI as we needed to build native code. Unlike Expo CLI it requires Xcode or Android Studio to get started. Although, it takes a while to install and configure them, you have more control when writing code. In our case it was a necessity as we were using GRPC for establishing communication between client and server that required to a platform API that React Native doesn't have a corresponding module for yet.

"If React Native doesn't support a native feature that you need, you should be able to build it yourself." Although, it is a bit frustrating at the beginning to wire the things up, it gives an access to the full power of the platform.

The main disadvantage is writing code in 2 separate IDEs: Java code in Android Studio and javascript in Webstorm as intelli-sense for java was not working in Webstorm and js in Android Studio respectively, which was a bit annoying. According to the research conducted the solution to this would be to use Intellij Idea Ultimate Edition with required plugins.

Redux Just to make things clear, redux is a js library for managing application state. It can be used with any platform like Angular or Deku, it just happened so that it is most commonly used with React. It can probably be compared to global storage, which manages passing state between components. There are 3 ways of passing states between components: from parent to children by using props (level down), from a child to a parent by using a callback and states (level down) or between siblings by combining the above 2 methods.

In our project we used the above mentioned techniques and global variables which is not the best practice and gets tricky as soon as your app is

growing bigger but as our application is not very complex, it did its job.

Hot Reloading 'Fast Refresh' is a feature that was introduced in September, 2019. It allows code to compile automatically whenever changes in the code are made. Yet sometimes it was noticed that it does not pick up the changes and requires a reboot which is quite annoying.

States & Props The main difference is that props(short for properties) are immutable and are used to pass data down to Child Components. If draw a parallel, state is equivalent to local variables in a function, while props are equivalent to function parameters. State can only be used inside the component, props on the other hand allow components receive data from the parent component in the form of props.

Components The main idea of React is the usage of components: independent and reusable bits of code. There are 2 types of component in React: functional and class components. Originally states could be used only in class components(in simple words these are dynamic and interactive components). But things changed after introduction of hooks in React 16.8. Consequently, now they can be used interchangeable.

Packages There are 2 main package managers: npm and yarn. In the project npm was used.

To check if you have Node.js and npm installed, run these commands in your terminal:

```
node -v
npm -v
```

In order install a library with native dependencies and link it run the following commands.

```
npm install <library-with-native-dependencies> --save

npx react-native link
```

Packages need to be imported in each components they are used in. A list of some packages that were used in the project: 1. React Native Elements 2. React Native Vector Icons 3. NativeBase 4. React Navigation

A complete list of packages that were used in the project can be found in package.json file in dependencies.

Google API

2 Google APIs were used in the project. These are:

1. Geolocation In order to identify user's current location and display it in profile page.

2. Places API AutoComplete It is used in the search bar, eliminates errors due to users misspelling of a city. Consequently, the same city (the right one and the one that has a spelling mistake in it) cannot be created twice.

Pros and Cons of RN

One of the advantages of this framework is that it has a large community and solutions to most of the problems a developer can encounter can be found online (github or stackoverflow). There is also a large amount of open source custom components which makes up quite a big library. The downside of it though is the fact that the quality of some packages can vary which leads to a more detailed analysis of the plugins available.

Moreover, RN documentation is brilliant: very comprehensive and easy to follow. Apart from that an impressive list of video and blog tutorials is available online as well.

Nevertheless, while working with React Native I have encountered some difficulties and I would like to enumerate the following disadvantages:

1. First issue with React Native (and all hibernate platforms in general) even though they promise that code can be written once and then used for both platforms, it is not as easy as it seems from the first glance. What happens is you are still writing native code (which contradict to its other advantage that programmers are required to know only one language). Sooner or later a javaScript programmer will have to figure out how swift or java works if there is no React Native API available. In the long run, it might happen so that you will write the same code twice: for IOS and Android. So the question is: "Are we actually killing two birds with one stone?"

2. Second quite noticeable drawback is performance, to be more precise, the time it takes React to load at first start of the app. Launch time leaves much to be desired. Although, RN claims near-native performance, still it cannot compete with native apps.

The size of React Native apps is usually larger than native ones and it increases with each added third-party library. This also has a negative impact on the app speed. Code of the platform adds another layer of complexity, which makes it more intricate and gives an impression that you cannot have the full control and should rely on the platform itself. Or I could be wrong and it just requires a deep knowledge of the targeted platform.

3. Another disadvantage, is dependency on third party modules. Sometimes these libraries are not well maintained as in most cases it is an open source software. Consequently, a developer should be more careful when installing a certain package: pay attention to the number of downloads, check the last updates (is it still maintained) and read issues section and Readme

carefully. Apart from that there can be issues with versioning and library compatibility.

4. It's relatively new, thus codebase is not stable. It is still at its infancy stage, at least compared to mature native technologies, which were tested by time. React Native is getting better but still there is room for improvement.

5. Debugging is quite tricky. Some error messages do not help in understanding the issue, even figuring out which component causes the issue and needs to be fixed can be a problem. As a result, you spend a lot of time on fixing even small errors. Especially, when it comes to debugging native code. What makes it even more complicated is the fact that live reloading does not work in this case and code needs to be recompiled. If your laptop is not powerful enough it can take up to 5-7 minutes of you time, which cannot be called a 'smooth developer experience'.

3.2.2 gRPC

Even though REST is probably the most preferred architectural style nowadays gRPC is a communication mechanism that gains popularity. A number of different organizations have adopted gRPC, such as Square, Netflix, CoreOS, Docker, CockroachDB, Cisco, Juniper Networks.

gRPC (gRPC Remote Procedure Calls) is an open source remote procedure call (RPC) framework initially developed at Google in 2015[**?**].

It uses Google Protocol Buffers (protobuf or proto) as a mechanism for serialization or deserialization structured data. Compared to text-file formats such as json, XML or yaml proto is smaller, simpler and faster.

Here is an example of a proto file:

Among other advantages are: 1. Language agnostic. which means that it does not depend on a language.

2. Machine Readable Protos are binary or machine readable but not human readable.

3. Provides Generators to serialize or deserialize Protobuf can be easily compiled to source code with protobuf compiler which makes the process of marshaling and unmarshaling easier. At the time this paper was written it supported 14 different languages and platforms (such as JAVA, Kotlin, Objective c, python, c#, ruby, go, c++, etc).

4. Supports types and Validations Field types and validations can be specified in the .proto file.

5. Lesser Boilerplate code It generates stubs in the chosen language.

Technology is relatively new, that's why there are not much open source projects and documentation on it.

```
syntax = "proto3";

package wcity;

option java_multiple_files = true;
option java_package = "io.grpc.wcity.postservice";
option java_outer_classname = "PostProto";
option objc_class_prefix = "RPC";

service PostsService {
    rpc CreateCityPost (CityPost) returns (CreatePostResponse);
    rpc CreatePlacePost (PlacePost) returns (CreatePostResponse);
    rpc GetPlacePosts (PostsRequest) returns (PlacePostsResponse);
    rpc GetCityPosts (PostsRequest) returns (CityPostsResponse);
    rpc UpdatePlacePost (UpdatePostRequest) returns (UpdatePostResponse);
    rpc UpdateCityPost (UpdatePostRequest) returns (UpdatePostResponse);
}
message CityPost {
    int32 indexId = 1;
    string creatorEmail = 2;
    string cityName = 3;
    string cityCountry = 4;
    string title = 5;
    string body = 6;
    string timeStamp = 7;
    repeated string likes = 8;
    string mongoId = 9;
}
```

Figure 3.1: Protobuff

Chapter 4

System Design

4.1 Ahuthentication Service

Jose I. Retamal

This service provides user authentication. It is composed of three components: the hash service, the database, and the main service (Figure 4.1).

The password is store securely, is hashed and salted using the hash service and then the hash and salt is stored in the database.

The database is replicated using the master follower topology. Create, update, and delete operations are always performed in the master, read operations are performed in followers.

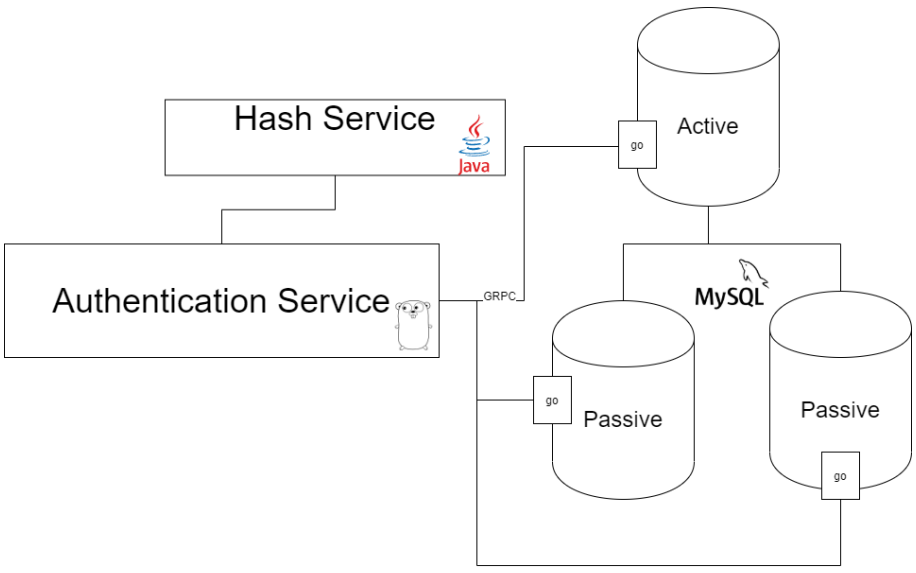


Figure 4.1: Authentication Service- UML.

4.1.1 Database Replication

Replication it has been set up using MySQL 5.7 (See Appendix B.1.4), we have set up a master and a replicate running in different Virtual Machine using Azure services. Using the instruction in Appendix B.1.4 is possible to add more followers, for do tables in master need to be locked for a few minutes.

The load balancing is done using Round Robin, and standards go grpc librarie. helps in performance because the most used operation is to get user data or token to check authentication.

Is implemented as client-side replication, the client chooses a server one at the time to make the calls, the client in this situation is the authentication service, and the server is the authentication dba. The load balancing is implemented in the authentication service(Figure 4.3).

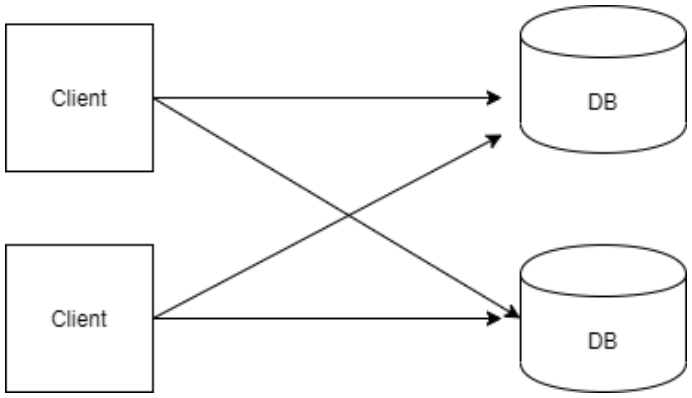


Figure 4.2: Authentication Service- Create User Sequence Diagram.

4.1.2 Endpoints

Create User

Users create an account, and when this happens, a new entry is created in the authentication database. Also, a new entry is created in the profiles database (Figure 4.3). To create a user, this service needs to communicate with the profiles service.

Login user

To login a user, we perform the followings steps (Figure 4.4):

- Get user data from the authentication database.
- Check the password using the hash service.

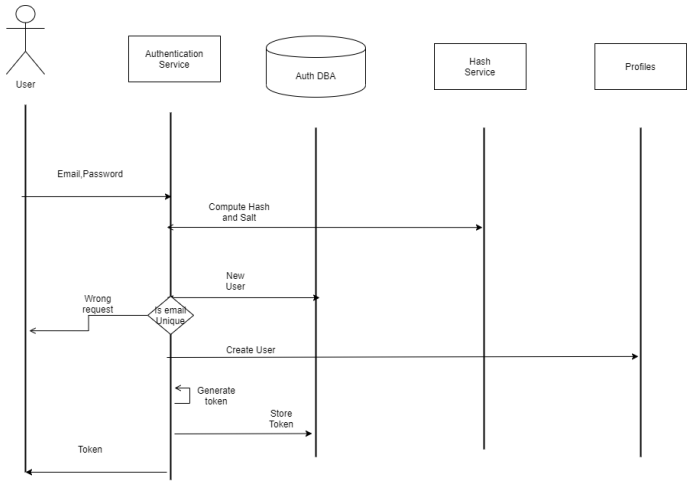


Figure 4.3: Authentication Service- Create User Sequence Diagram.

- If the password is valid will generate a unique token, it stores the token in the database and sends the response to the client.

Check Token

The most used endpoint, here is where replication plays a role for fast checking tokens. This is used in the most requests to all services to ensure security across the application (Figure 4.5).

Log out

The request includes the token, and that token is removed from the sessions table in the authentication database.

4.1.3 Authentication DBA

This program provides access to the authentications database. Is written in go and runs in a docker container, it connects to a MySQL database running in localhost. The application communicates with the main authentication service through a grpc interface.

Database

The authentication database store the necessary user data for authentication and login.

It is composed of two tables: the authentication table and the sessions table. The authentication table contains the user name, the password hash, and the password salt(Figure 4.6).

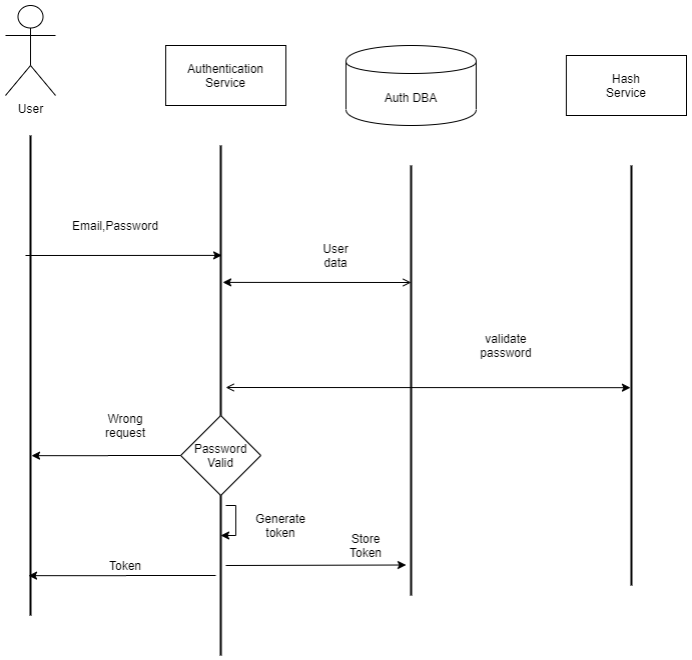


Figure 4.4: Authentication Service- Login Sequence Diagram.

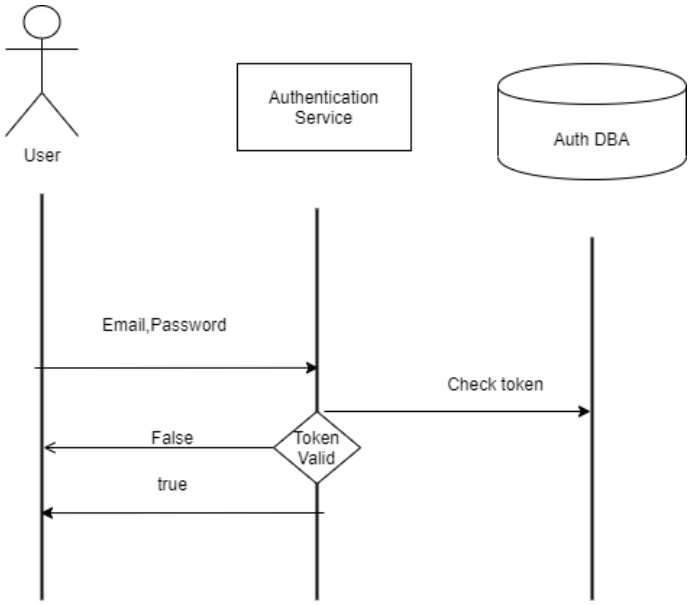


Figure 4.5: Authentication Service- Check Token Sequence Diagram.

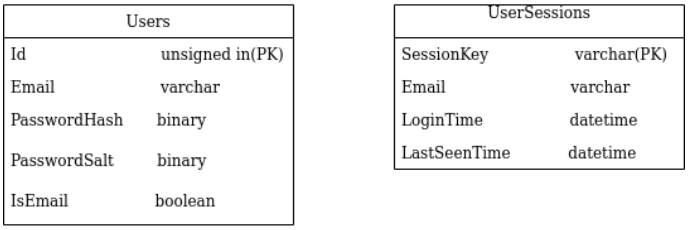


Figure 4.6: Authentication DBA- Authentication Database.

The user name is the email and is the unique identifier for all the systems, is not the primary key of the database, but is indexed for quick access. For this database, an extra integer field is added as the primary key. The password hash and salt are 32 bytes binaries strings. The sessions table uses a unique session key as a primary key for a quick check if a session exists.

When a user login a session is created and stored in this table. The user then can log in to the application using that session. When the user logs out, the session is deleted from the database.

Endpoints

- AddUser() : Create a new user in the database. When the user creates the account, it will create the profile automatically in the profiles database.
- GetUser(): Returns the user data used to authenticate the user.
- UpdateUser(): Update the user data, is used for changing the password.
- CreateSeassion(): Create a new session in the database. Used when the user login using the password.
- GetSeassion() : Return a user session if exist. Used to check if the session exists so the user can log in without the password.
- DeleteSession : Delete user session if exits. Used when the user logs out from the device.

4.1.4 Hash Service

This service creates a password salted password hash using a randomly generated salt. It has been adapted from a Distributed Systems project from semester 7. It checks the password in fix amount of time for security reasons. Attackers can guess passwords guess by comparing the time it takes to validate a password.

4.2 Profiles Service

Jose I. Retamal Jose I. Retamal

The profile service manages information about users, cities, and places. The data contained is editable by the users and provide information about them.

The service is composed of the main service that connects to a Neo4j database using a Java DBA(Figure 4.7). The main service is design to be stateless; therefore, many instances of the service can be created for load balancing and scaling the system.

There are relations between the different types of profiles that allow us to get data quickly and avoid the use of complicated queries.

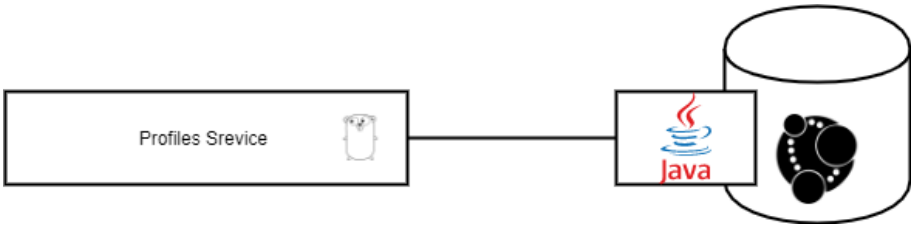


Figure 4.7: Profiles Service - Main UML.

4.2.1 Request Sequence

Each request contains a token that is checked with the authentication service if the token is valid, then the service access the database to serve the request(Figure 4.8).

4.2.2 Endpoints

Users

- Create User. This method is only called by the authentication service because users create an account using that service. The authentication service then sends the user data to the profiles service.
- Get User. Get all data about the user.
- Update a User. Updated the user.

City's Users create the cities; the user who creates a city can then update the data. A city can only be created once.

- Create Cty. Create the city and a relation between the user who creates the city and the city

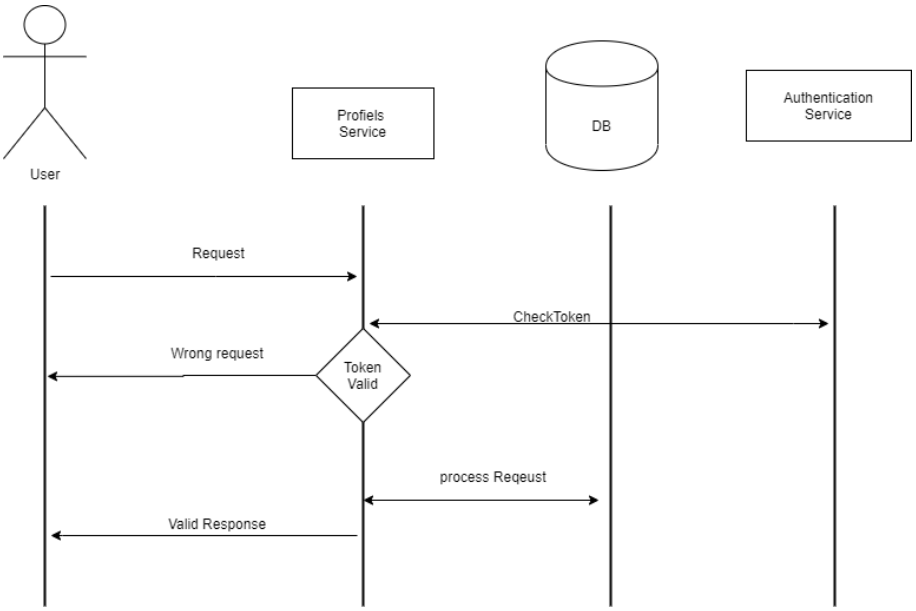


Figure 4.8: Profiles Service- Request Sequence.

- Visit City. Create a relation between the city and the user Update City.
- Get City
- Get all Cities

Places Users create places; places belong to a city.

- Create Place. Create the place and a relation between the user who creates the place and the place. Also, create a relation between the place and the city that belong.
- Visit Place. Create a relation between the place and the user/ Update Place.
- Get Place.
- Get All Place

4.2.3 The Database

The database is composed of three types of nodes: User, City, and Place(Figure 4.9). Each node has a unique integer id. Users are also identified by the email, which is the unique id al over the system. A city can

User	
Id	int (PK)
Email	string
Name	string
Description	string

Place	
Id	int (PK)
Name	string
City	String
Country	string
Lat	float
Lon	float
Description	string

City	
Id	int (PK)
Name	string
Country	string
Lat	float
Lon	float
Description	string

Figure 4.9: Profiles Service- Neo4j DB Classes.

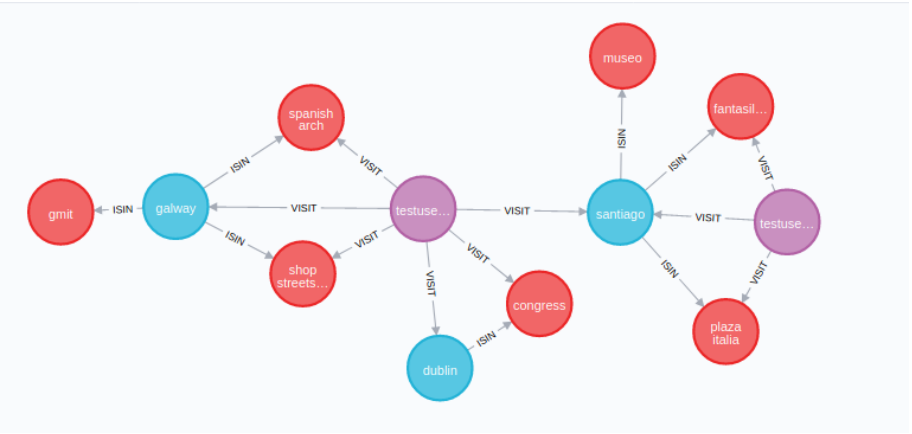


Figure 4.10: Profiles Service- Neo4j DB Dodes.

also be identified by the name and the country, place by name, the city that belongs, and the country(Figure 4.10).

Relations

- Places are in a City. A relation between the city and the place, this relations allows us to retrieve quickly the places that belong to a city.
- Users visit cities and places. Create a relation between the user and the city/place. As users can mark several cities/places as visited, having this as a relation simplified the queries and give quick access to all cities/places that the user has visited.

4.3 Photo service

Jose I. Retamal Jose I. Retamal

The photo-service provides image storage and access to them. It is composed of the main service, the database, and storage buckets. The binary image is store as a public object in the bucket, and the URL to the image is stored in a MySQL database. Images can be accessed from the client using the public URL(Figure 4.11).

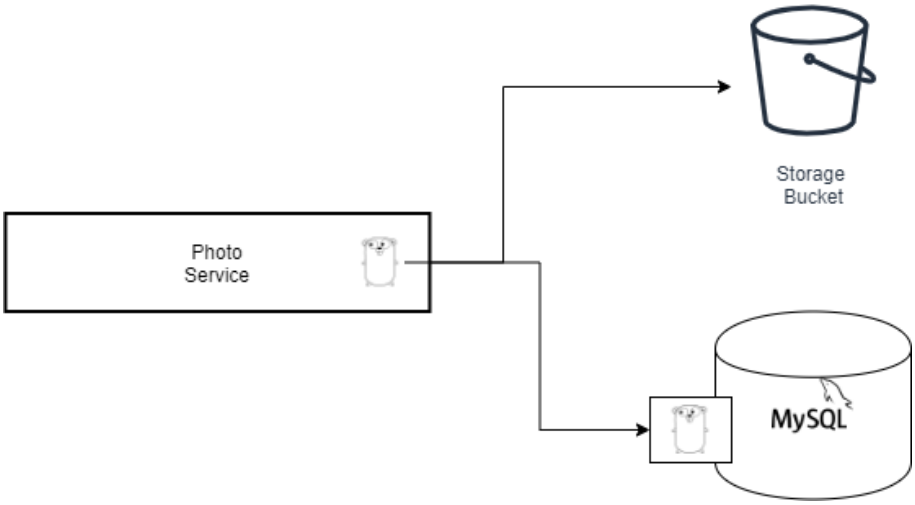


Figure 4.11: Authentication Service- UML.

4.3.1 Request Sequence

Upload Image

When the main service receives a request to upload an image, it generates a random URL, the image is upload to the bucket using that URL, and the URL is stored in the database. Then the URL is sent to the client for access to the image(figure 4.12).

Get Image

When the client requests an image, the client is first authenticated then the image URL is retrieved from the database. The URL is sent to the user who accesses the image directly from the bucket 4.13).

4.3.2 The Database

The database store the URL to the image in the bucket. Images for City, Place, Posts, and User Profile are stored, there is a table for each of them(Figure 4.14).

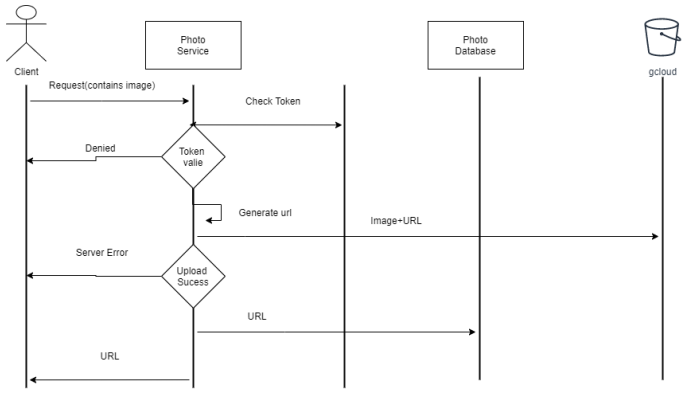


Figure 4.12: Photo Service- upload Image Sequence Diagram.

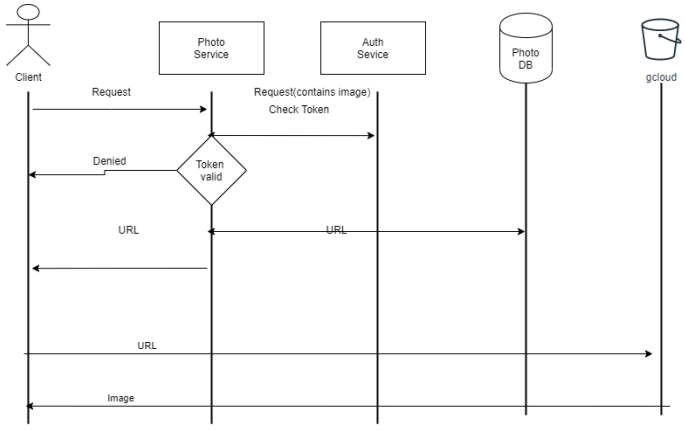


Figure 4.13: Photo Service- Get Image Request Sequence Diagram.

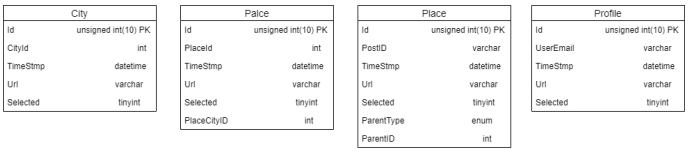


Figure 4.14: Photo Service-URLS Database Entity Diagram.

All tables have an autogenerated integer primary key and each, and all images are also identified depending on the type. Bellow, we explain how they are linked using id from different databases.

- City Images

  The unique id is the CityId, which is the Neo4j unique id.

- Place Images

  The unique id is the PlaceId, which is the Neo4j unique id.

  The PlaceCityId is the id of the city of which the place belongs. It is used for getting all the images of the places in a city.

- PostImages

  The postId is the id of the post in MongoDB.

  There are two types of Posts, city and place post. They are stored in the same table, and an enum is used to distinguish.

- Profile images

  The UserEmail is the unique id, which is the PK in the auth database.

4.3.3 The bucket

Images are store in jpg format with a medium compression for maximum storage capacity without losing notable quality(Figure 4.15). Inside the bucket, images are public objects, and they can be accessed for anyone who has the URL. Folders organize objects (Figure 4.16), they are numbered, and more folders and buckets can be added. The URL is generated randomly by the main service, which is composed of two integers for a fast generation of them.

4.4 Post Service

Jose I. Retamal Jose I. Retamal

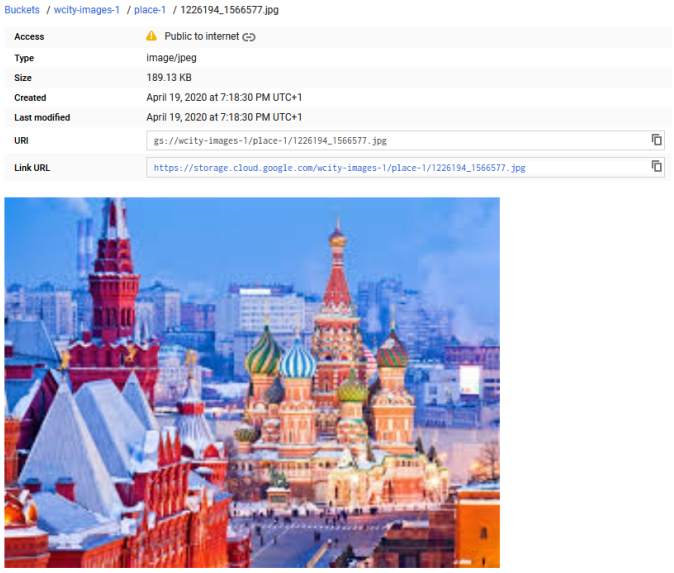


Figure 4.15: Photo Service- Image in Bucker.

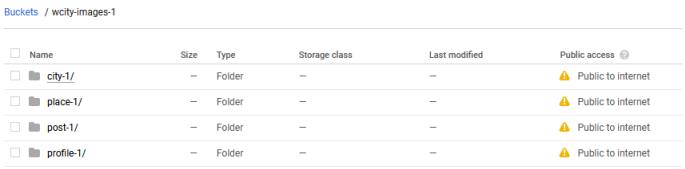


Figure 4.16: Photo Service-Folders Structur in a Bucket.

Post Service manage posts, there are posts for cities and places. The service is composed of two paint parts: The main service and the database(Fugure 4.17).

The main service connects to the client and provides the main endpoints for creating view and update posts. It connects to the Mongo database and checks requests on the authentication service.

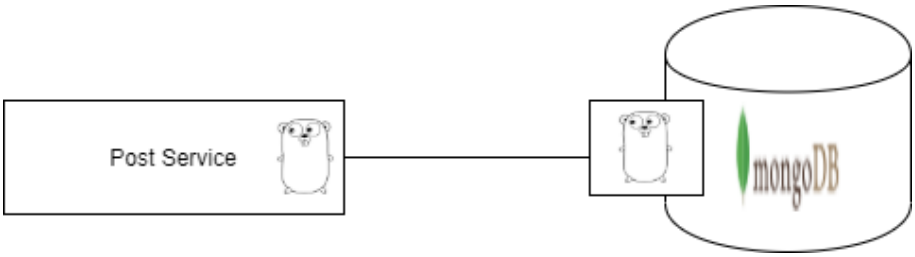


Figure 4.17: Post Service- UML.

4.4.1 Request Sequence

- Create Post

  When a user creates a post, the request contains the post data and the authentication token. The token is check in the authentication service, and then the post is stored in the database.

4.4.2 The database

Post are grouped by place and cities using the Neo4j unique id for each city and place. The database is composed of two tables, one for cities and another for places(Figure 4.18). All data is stored in one collection. The data is indexed by the unique id (index id) therefore, the performance is not affected by the amount of data stored.

4.5 Frontend

4.5.1 Navigation

For navigation the following libraries were used:

```
"react-navigation": "^4.0.10",
"react-navigation-drawer": "^2.3.3",
"react-navigation-stack": "^1.10.3",
"react-navigation-tabs": "^2.7.0",
```

As you can see in the code provided below, app starts with a splash screen (which was implemented by the following tutorial at:

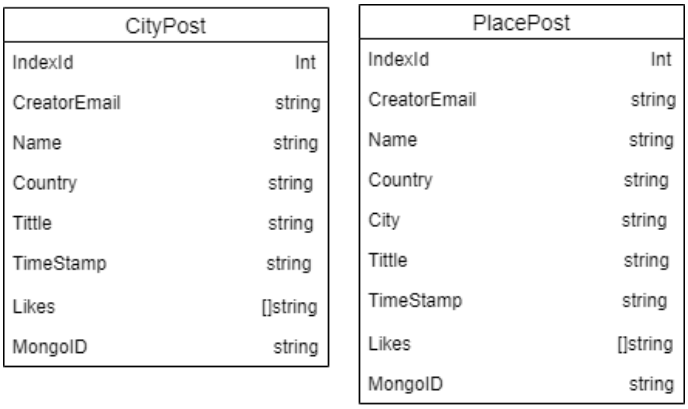


Figure 4.18: Post Service- Database Entity Diagram.

```
https://medium.com/handlebar-labs/how-to-add-a-splash-screen-to-a-react-native-a
const MyApp = createSwitchNavigator({
        loading: {
            screen: SplashScreen
        },
        app: appDrawer,
        auth: authStack
    },
    {
        initialRouteName: 'loading'
    })

const AppNavigation = createAppContainer(MyApp);
```

In SplashScreen component the saved email and token from the local storage are sent to the server, which compares it with its data and sends a response back. Depending on the response (valid/invalid) it navigates either to authentication or application route correspondingly.

There are 3 stacks in navigation: Authentication, Drawer and Tabs Stacks. Authentication consists of 2 components: Login and Register.

Drawer or ”hamburger” menu has Profile, Settings and Log out options.

There are 2 bottom tabs: ’Feed’ and ’My City’.

Main components and navigation between them is shown in the diagram below Fig. 4.20

Authentication Stack

Consists of 2 components:

1. Login

2. Register

In order to use the app user has to register first. After logging in (or first registration) server assigns a token to the user. Next time if the user has not logged out (just closed the app) then there is no necessity to log in again. Token and email get saved in local storage of a phone, after launching the app they are sent to the server, which checks if they are valid or not (if such email and token exist). If a user logs out then 'token-email pair' get erased from the local storage and the server and user has to log in again.

Screenshots of register and login pages are shown below in Fig. 5.7 and Fig. 5.8

App Stack

1. Drawer

As illustrated in Fig. 5.9 drawer contains user profile information, settings and an option to log out alongside with user's avatar and user's current geolocation. In more detail about each component further below.

•*Profile*

Profile contains user's avatar, geolocation, email, name and description. Apart from that, there are two carousels of visited cities and visited places. A user can add a city or a place to 'visited list' by pressing a check-box icon at the top right corner in City details or Place details page.

There is an option to edit profile, which navigates to settings component, described further.

•*Settings*

Allows to modify name and description. Avatar can be changed either in drawer or in profile by clicking on the image and uploading a new one.

•*Logout*

As mentioned before calls logout method in authentication server, which clears token assigned to email when logging in (as well as clears it from local storage).

2. Tabs

Bottom tab navigation contains Feed (with all the cities' info) and My City (city, identified by the geolocation).

•*Feed* Displays all cities created, posts, places and posts about the places.

•*MyCity* Displays a city information, a user is currently in and posts about that city.

4.5.2 Client-Server Communication

As described in Methodology chapter client 'communicates' with the Server using GRPC.

There are 4 services:

1. Authentication - required for login and register.
2. Post Service - for retrieving, creating and updating posts.
3. Profiles Service - for retrieving, creating and updating cities, places and users.
4. Photos Service - responsible for all the images.

In order to establish communication between client and server as shown in Fig. 4.19 a 'bridge' between Android native and JavascriptCore was required to be built.

2 classes were written per each service:

Client.java - a class that calls stub methods.

Module.java - a class which calls methods of Client.java. IP address and port number are specified here as well.

Note: Each module needs to be added to a package. All packages have to be added to the List¡ReactPackage¿ in MainApplication.java

In javascript code a NativeModules library has to be imported.

```
import { NativeModules} from 'react-native';
```

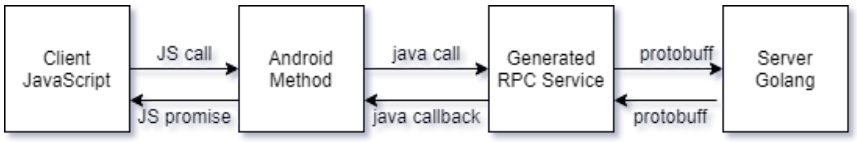


Figure 4.19: Client-Server Communication Diagram

4.5.3 Prototypes

Fig. 4.21 - 4.24 represent some of the prototype pages, final UI (screenshots) can be found in the next chapter.

As many pages as needed.

- Architecture, UML etc. An overview of the different components of the system. Diagrams etc... Screen shots etc.

Column 1	Column 2
Rows 2.1	Row 2.2

Table 4.1: A table.

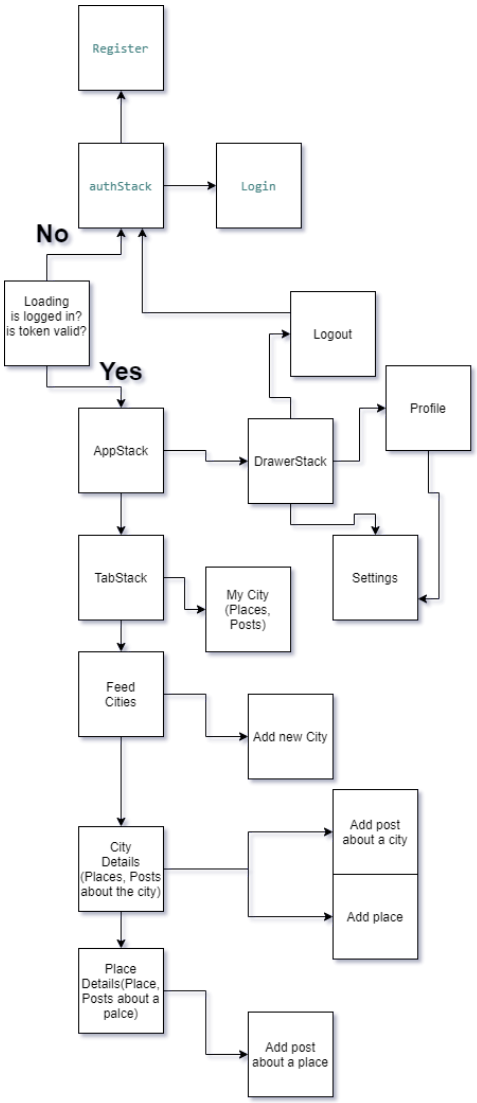


Figure 4.20: Navigation Diagram

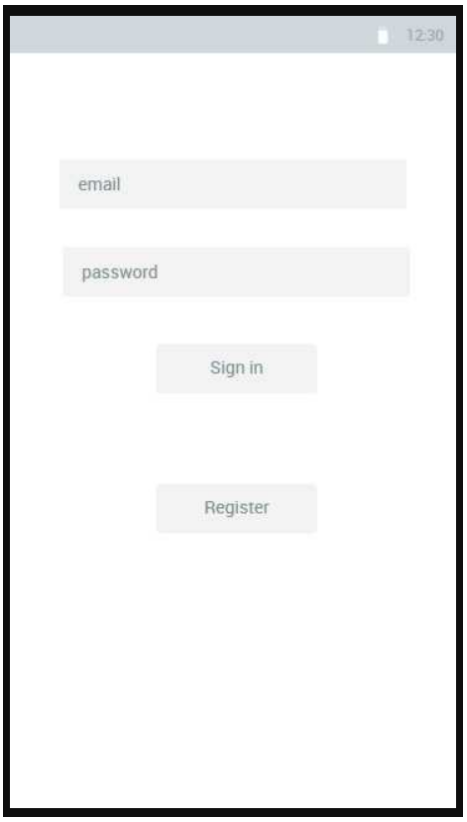


Figure 4.21: Login Prototype

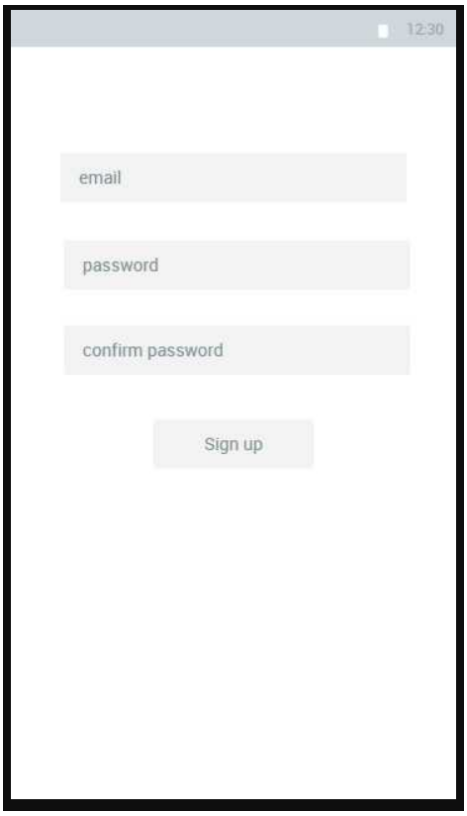


Figure 4.22: Register Prototype

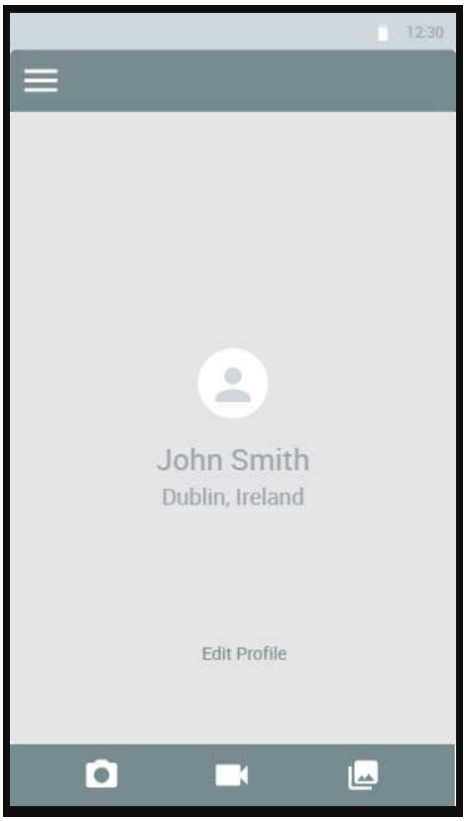


Figure 4.23: Profile Prototype

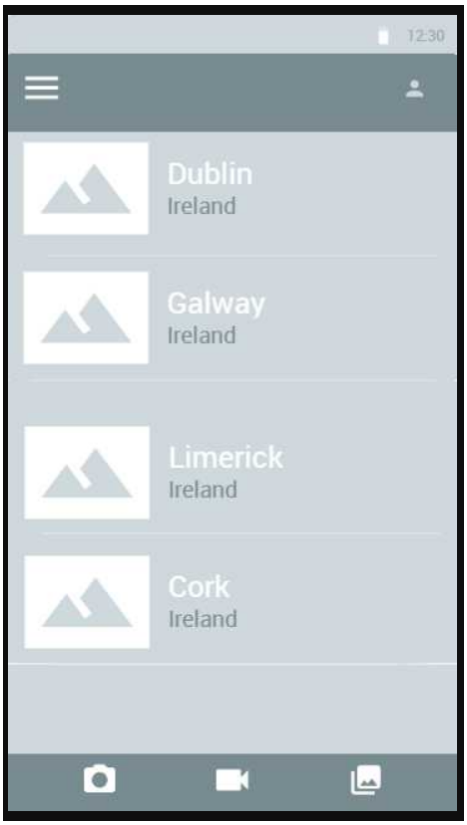


Figure 4.24: Home Page Prototype

Chapter 5

System Evaluation

5.1 Back-End

Jose I. Retamal

5.1.1 Implementation

The system is implemented using 6 Virtual machines(2 in Azure, 1 in AWS and 3 in gCloud), one file bucket(gCloud), and the React Native app Accessible through Google Play Store(Figure 5.1).

The databases

All databases are static, and they can be replicated or partitioned for scalability if we create multiple instances of one service, each instance access the same database or a replica of it. We have test replication in the authentication MySQL database.

Profiles, posts and Photo databases

For convenience and limitation of resources, the profiles neo4j database, post mongo database, and photos MySQL database run all in the same Virtual Machine.

We implement them in a Google Cloud small size VM:

Ram: 1.7 GiB
VCPU: 1
Estimate cost: 14.04 p/month
OS: Ubuntu Server 18.04 LTS Minimal
Disk: Standard HHD

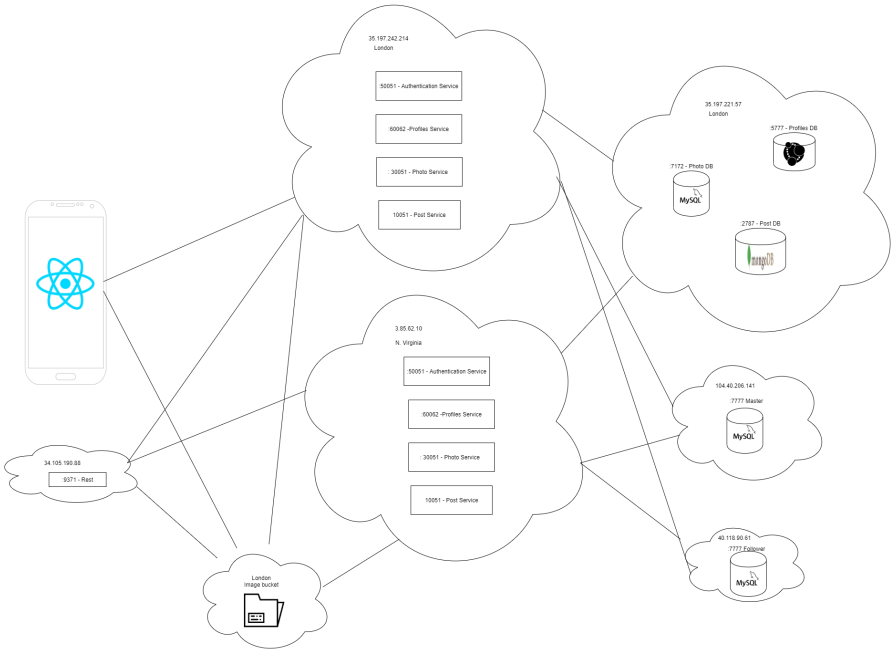


Figure 5.1: System Implementation.

Location: London

The Memory usage is show below (Figure 5.2):

```
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
1ee37dbb141e        neo4jdba            0.09%               198.4MiB / 1.646GiB   11.77%              0B / 0B             206MB / 81.9kB      113
a643d5edaae3        neo4j               0.72%               452.8MiB / 1.646GiB   26.87%              5.46MB / 6.94MB     470MB / 39.1MB      57
35c72dc4de01        photodba            0.00%               3.141MiB / 1.646GiB   0.19%               0B / 0B             109MB / 0B          6
682877f75c19        postdba             14.88%              77.73MiB / 1.646GiB   4.61%               0B / 0B             174MB / 0B          7
```

Figure 5.2: System Implementation: Databases Memory Usage.

Auth database We have implemented one master and one follower, and each database runs in different virtual machines from Azure with same specifications, the VM speficcations are :

Ram: 1 GiB
VCPU: 1
Estimate cost: 6.96 p/month
OS: Ubuntu Server 16.04 LTS
Disk: Standard HHD
Location: Western Europe

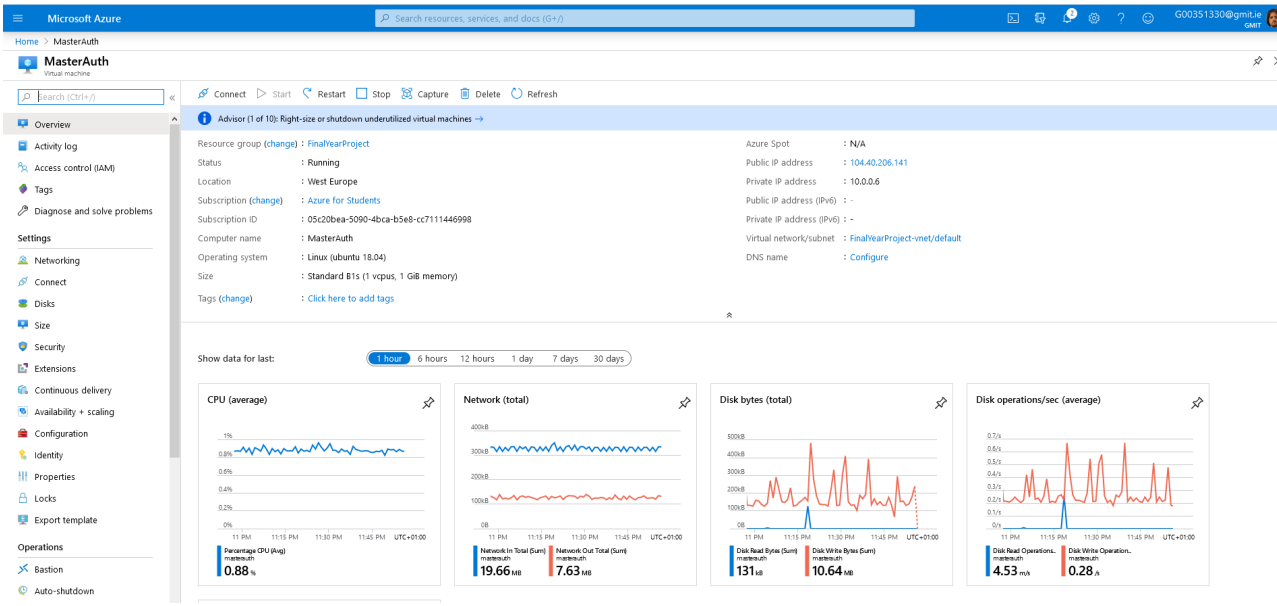


Figure 5.3: System Implementation-Master VM.

The memory usage for the master(Figure **??**) :

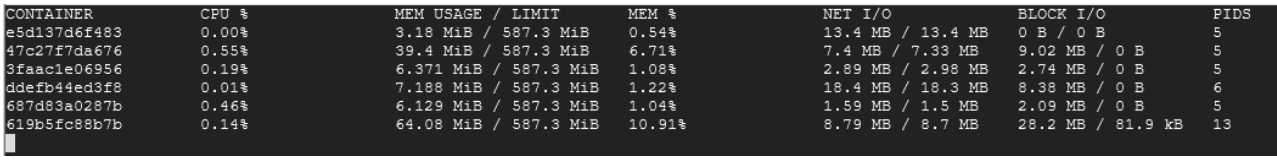
```
pepe@MasterAuth:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:            897         607          95           1         193         142
```

Figure 5.4: System Implementation- Master Memory Usage.

The Services Services need a small number of resources to run. The go service has a very low memory usage, and we were able to run all of them in a small virtual machine. We run on this way for testing and because we do not have many resources available to run the project. We run two instances of each service, one in google cloud located in London(Figure **??**) and another in AWS in North Virginia(Figure **??**).

Gcloud Ram: 0.6 GiB
VCPU: 1
Estimate cost: 5.39 p/month
OS: Container Optimazed OS 69-10895.385.0 stable

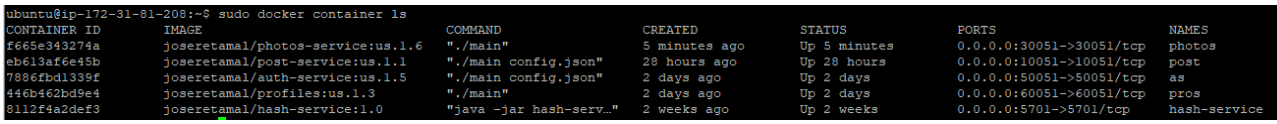
kernel :ChromiumOS-4.14.145 Kubernetes: 1.11.8 Docker: 17.03.2 Family: cos-69-lts
Disk: SSD
Location: Western Europe



CONTAINER	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
e5d137d6f483	0.00%	3.18 MiB / 587.3 MiB	0.54%	13.4 MB / 13.4 MB	0 B / 0 B	5
47c27f7da676	0.55%	39.4 MiB / 587.3 MiB	6.71%	7.4 MB / 7.33 MB	9.02 MB / 0 B	5
3faac1e06956	0.19%	6.371 MiB / 587.3 MiB	1.08%	2.89 MB / 2.98 MB	2.74 MB / 0 B	5
ddefb44ed3f8	0.01%	7.188 MiB / 587.3 MiB	1.22%	18.4 MB / 18.3 MB	8.38 MB / 0 B	6
687d83a0287b	0.46%	6.129 MiB / 587.3 MiB	1.04%	1.59 MB / 1.5 MB	2.09 MB / 0 B	5
619b5fc88b7b	0.14%	64.08 MiB / 587.3 MiB	10.91%	8.79 MB / 8.7 MB	28.2 MB / 81.9 kB	13

Figure 5.5: System Implementation- GCloun Services.

AWS
Ram : 1 Gib
VCPU : 1
Esitmate Cost 3.97 p/month
OS : ubuntu-bionic-18.04-amd64-server-20200112
Disk : Standard HHD
Location: N. Virginia



```
ubuntu@ip-172-31-81-208:~$ sudo docker container ls
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS                     NAMES
f665e343274a        joseretamal/photos-service:us.1.6    "./main"                 5 minutes ago       Up 5 minutes        0.0.0.0:30051->30051/tcp  photos
eb613af6e45b        joseretamal/post-service:us.1.1      "./main config.json"     28 hours ago        Up 28 hours         0.0.0.0:10051->10051/tcp  post
7886fbd1339f        joseretamal/auth-service:us.1.5      "./main config.json"     2 days ago          Up 2 days           0.0.0.0:50051->50051/tcp  as
446b462bd9e4        joseretamal/profiles:us.1.3          "./main"                 2 days ago          Up 2 days           0.0.0.0:60051->60051/tcp  pros
8112f4a2def3        joseretamal/hash-service:1.0         "java -jar hash-serv…"   2 weeks ago         Up 2 weeks          0.0.0.0:5701->5701/tcp    hash-service
```

Figure 5.6: System Implementation- AWS Services.

Rest service We run the rest service in a small VM on google cloud.

Gcloud Ram: 0.6 GiB
VCPU: 1
Estimate cost: 5.39 p/month
OS: Container Optimazed OS 69-10895.385.0 stable
kernel :ChromiumOS-4.14.145 Kubernetes: 1.11.8 Docker: 17.03.2 Family: cos-69-lts
Disk: SSD
Location: Western Europe

5.2 Frontend

5.2.1 Bugs & 'Features'

Application was tested on 2 devices: Samsung A70 and Samsung S7. While testing there were a few 'bugs' and 'features' identified in the application that a user must know.

The goal to implement an application which can retrieve, insert and update data was achieved. Nevertheless, the quality of the application leaves much to be desired.

1. The most noticeable drawback is performance. To be more precise app start and loading of components is a bit slow.

Every time state changes it affects performance and has an adverse effect on the speed of the app, especially on old devices, which does not lead to good user experience. In my opinion, the fact that I was struggling to understand the React hooks, which could possibly help in components rendering faster has a huge impact.

Before, react native performance was criticized when it came to complex animations and lots of views being rendered at once `https://medium.com/braus-blog/airbnb-is-dropping-react-js-should-you-too-dcbff36def5c`. And it has improved a lot since then. Common sources of performance problems and solutions to them can be found at `https://reactnative.dev/docs/performance`

I can't say that our app is doing something computationally expensive but still it lags a bit. For example, when it tries to call Google Geolocation API it takes a couple of seconds to identify the geolocation of the user.

I assume that both frameworks' flaws and my inexperience resulted in a satisfactory still not perfect application performance.

2. State is not updated when a new city is added. Adding a new town/city does not appear in the list of towns/cities straight away. When a user adds a new city, he/she doesn't see his/her city/town, the changes are not displayed on the Home page immediately. User will see the changes only when he/she reopens (closes and opens again) the app. Attempt to fix this bug by using hook componentDidUpdate(), which updates components based on comparison of previous and current state (in our case list of cities) resulted in infinite loop and crash of the whole application. In the long run a decision to leave this 'feature' was made.

3. App crashes when uploading images/photos on older android versions. Attempt to import java.util.Base64 instead of android.util.Base64 as advised on forums was not successful. Still works fine on newer versions.

Minimum requirement: Android 4.1 (API 16)

4. Only one image per page can be added. A user can add only one image of a city, place or post.

5. Poor UI design. More work has to be done in order to make UI look better and more professional.

5.2.2 User Interface

Although we had a few prototypes and initial ideas of UI design still some changes were made. For example, instead of creating separate pages for adding new cities, places and posts, it was decided to use Modals (pop up windows), which resulted in easier navigation and better user experience.

In order to avoid typos and duplication when a user creates a new city, he/she needs to search it first. If city is not already created and user decides to add that city, fields city name and country name get populated automatically.

As an obligatory condition for adding a city, place or a post is to provide description and a photo, a number of notifications such as in Fig. 5.12 were added in order to improve user experience.

Below you can find a few screenshots of User Interface. Fig. 5.10-Fig. 5.21

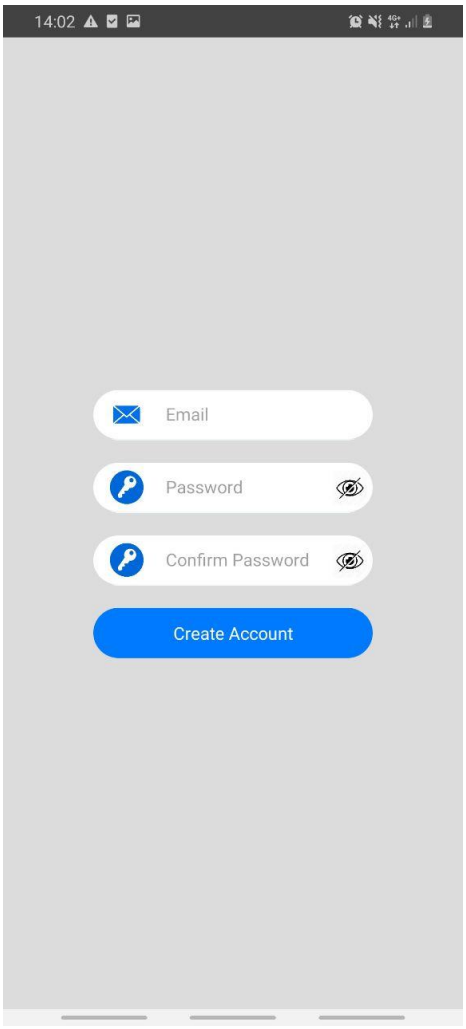


Figure 5.7: Register Page

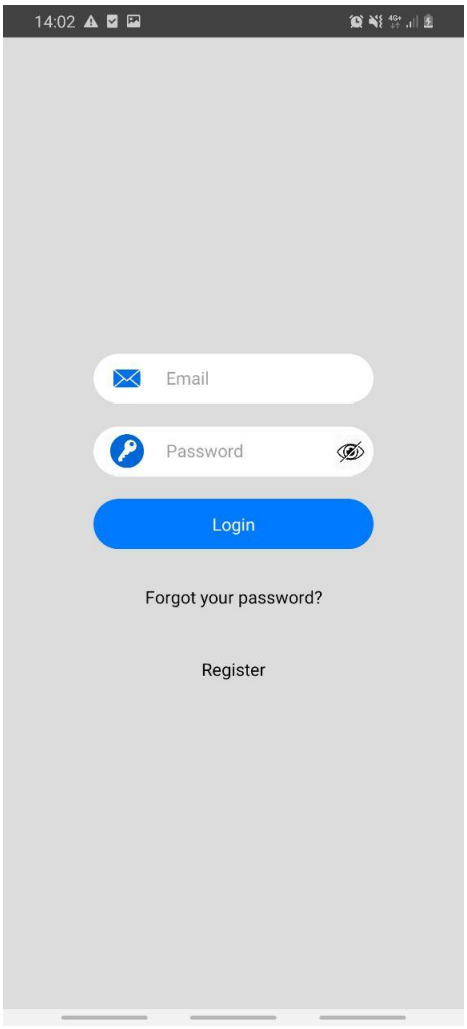


Figure 5.8: Login Page

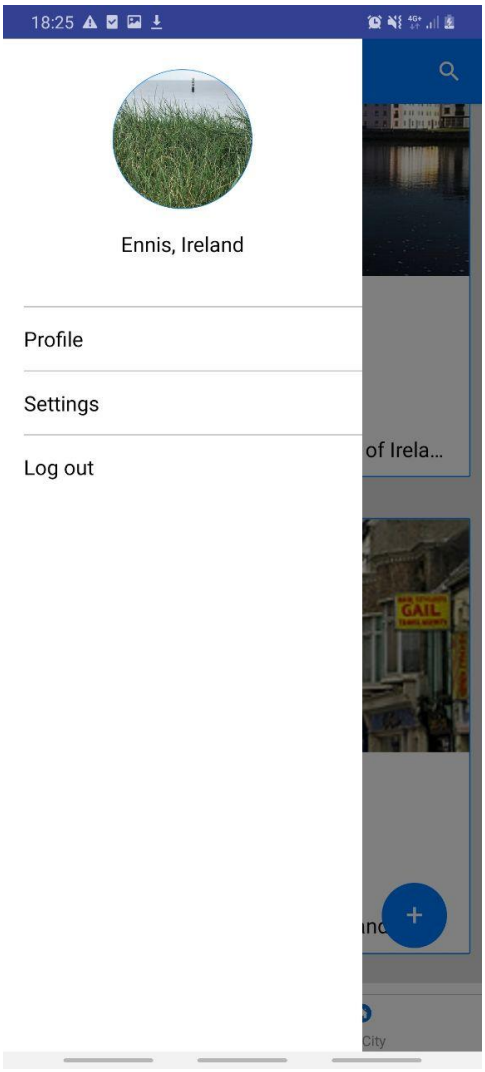


Figure 5.9: Drawer

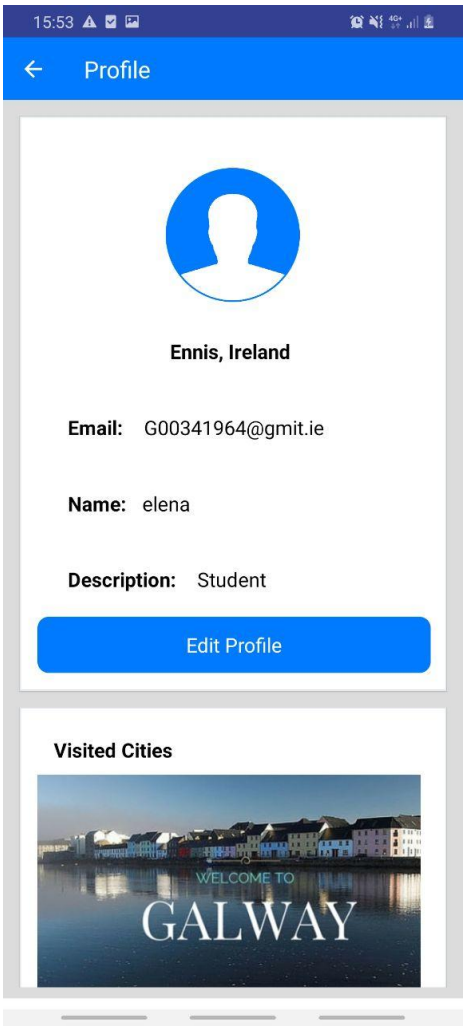


Figure 5.10: Profile

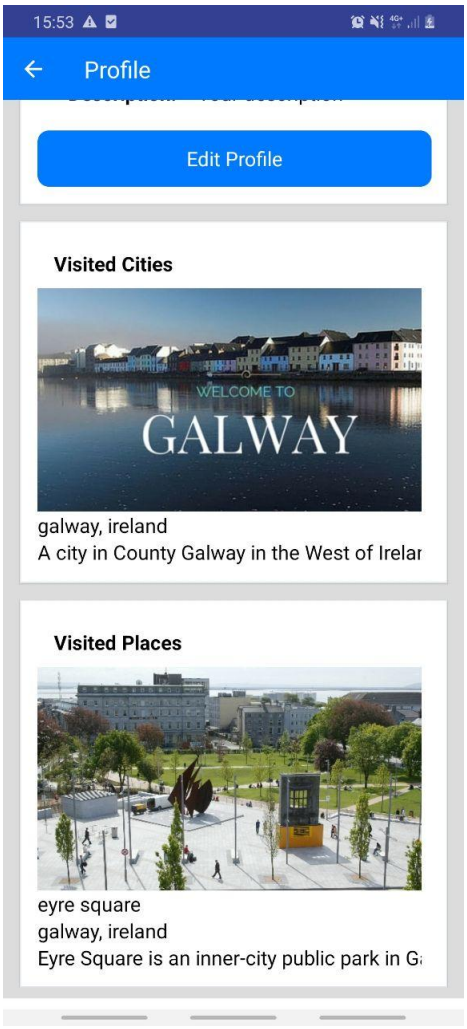


Figure 5.11: Profile cont.

Figure 5.12: Feed

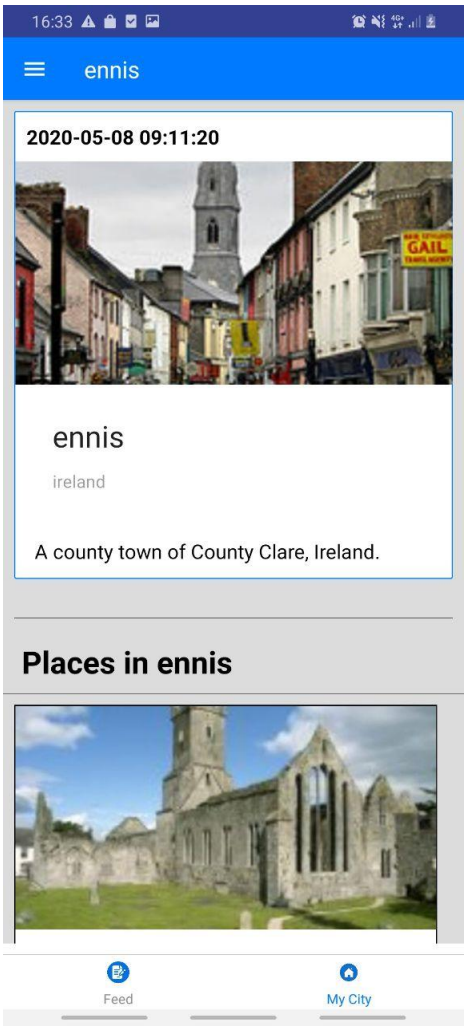


Figure 5.13: My City

Figure 5.14: Create City Step 1

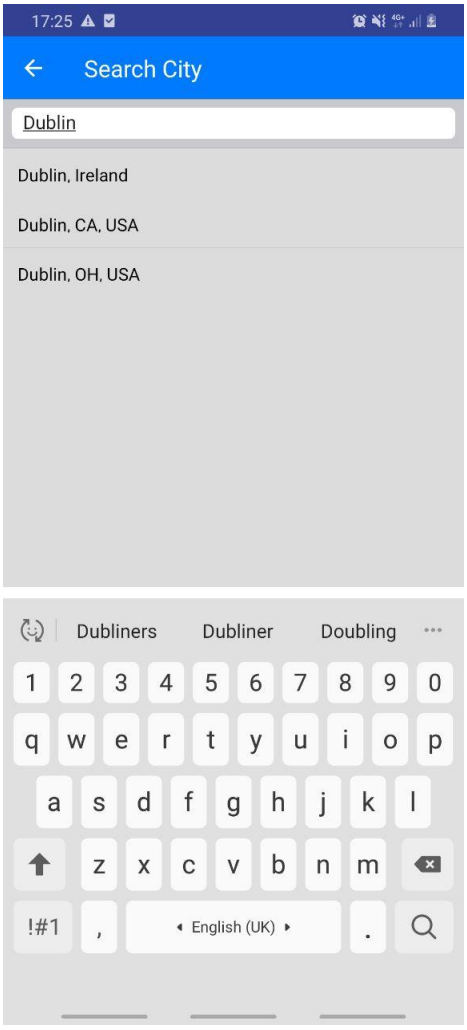


Figure 5.15: Create City Step 2

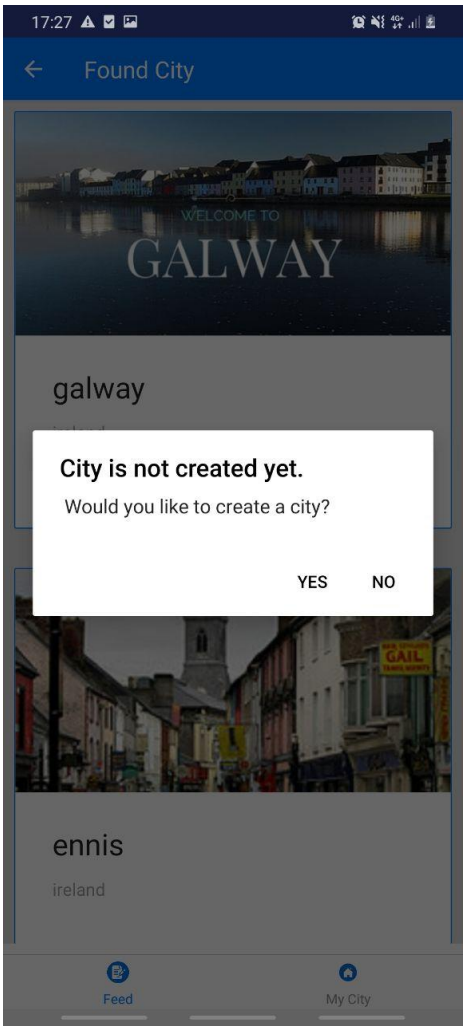


Figure 5.16: fig:Create City Step 3

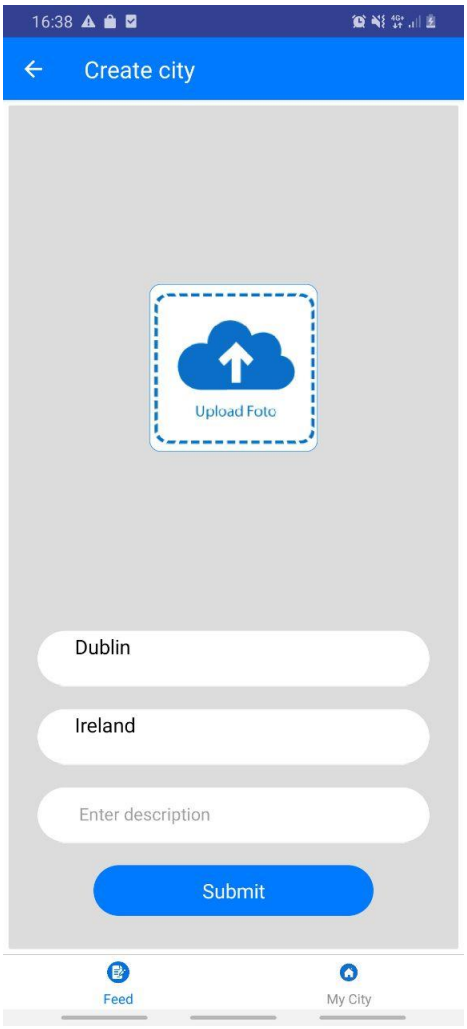


Figure 5.17: Create City Step 4

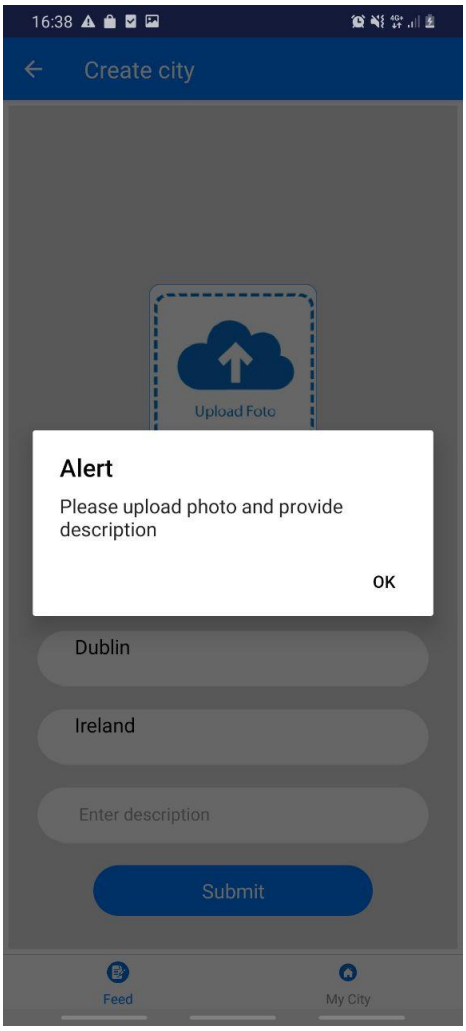


Figure 5.18: Create City Step 5

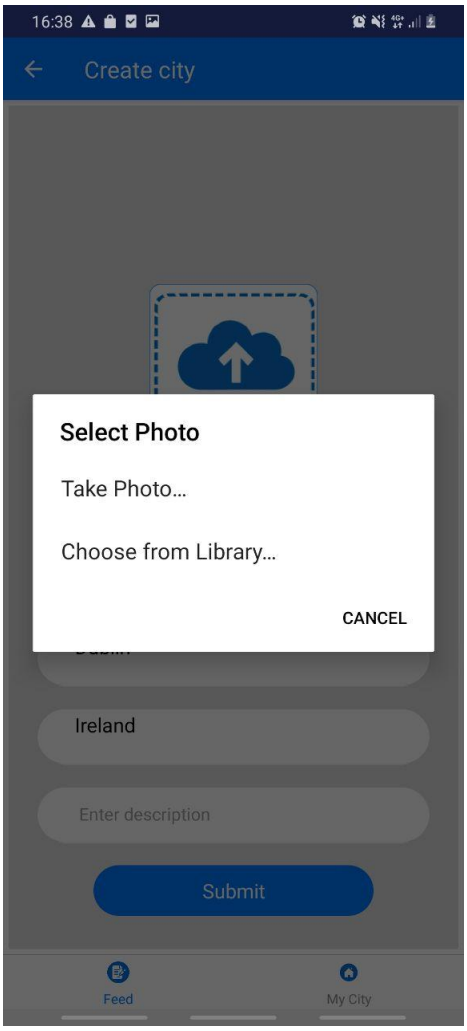


Figure 5.19: Create City Step 6

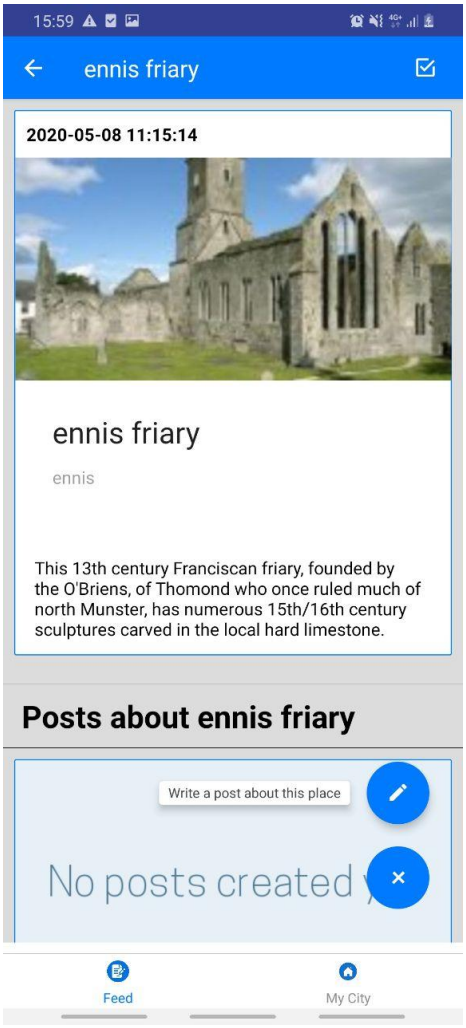


Figure 5.20: Place Post

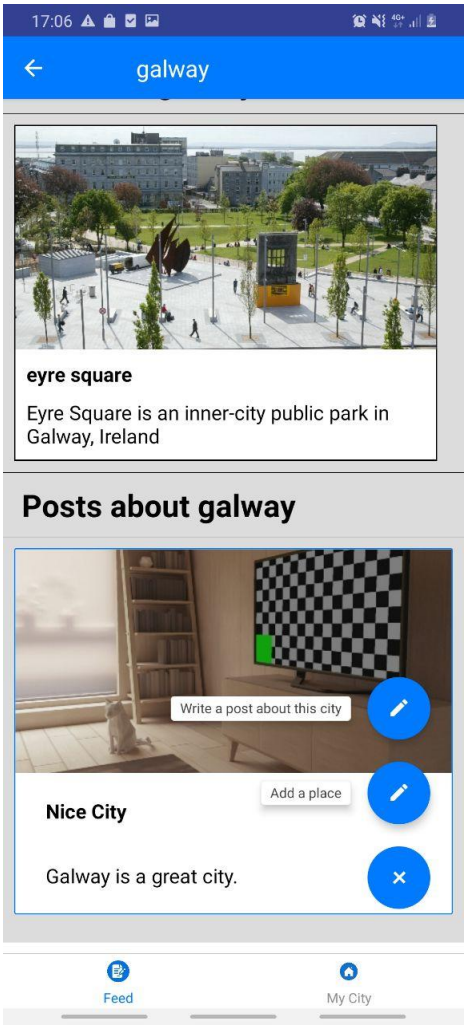


Figure 5.21: Post added

Chapter 6

Conclusion

We wanted to learn as much as we can by doing this project, improve our software hard and soft skills to be ready to afront the software industry as professional software developers in the best possible way.

We managed to build a working application using technologies that were new to us. We work in an adaptative way, and we adapt to changes that were totally out of our control.

This project helped us not only to learn new languages, frameworks, and tools but also to improve our soft skills: project management, critical thinking, problem-solving.

As it is a team project, it helped us to improve our communication skills, taught us how to negotiate, cooperate, and find compromises. We believe that not only the ability to work individually but the ability to work in a team is a valuable skill that employers are looking for in their potential employees.

We have summarized our achieved goals:

- We create a microservices application that can be easily scalable, composed of four encapsulated microservices, each with his own database.
- Work in a big project using Agile. We work in an iterative process, building the application block by block and constantly reviewing the features, design, and even the process. Following the adapted agile principles, we were able to have a working application.
- We learn and apply new technologies. Docker, Go, ReactNative, and gRPC were new to us. We manage to learn them as we go and develop the application.
- We work as a team, review our and other member's works. Analyze the application and found improvements and defects. We managed our time and were able to finish the project in time, with the need for some trade-off at several stages.

- We implement a secure authentication system, which stores passwords using hash and salt following the security measures that establish at the start.

During this project, there were many challenges encounter, principally because we chose to work with new technologies and learn them in the way. Some of the big challenges encounter where:

- Using one Github repository for the whole project, all the project was developed using one repository to facilitate the grading. The project was composed of several applications, and it was challenging to keep all of them together. We could not implement Docker's continuous integration and delivery because it requires a repository for each application.
- React Native complexity. React native is a huge and complex framework. There is a large number of libraries and components that need to be researched and implemented. Package management became hard, and it is very easy to made mistakes. Learning it without the guidance of an experienced developer in the framework was not an easy challenge.

Opportunities and possible future work

- Kubernetes. After dockerizing the services, the next step is to use Kubernetes to manage the system images and the load balancing of the application. It was mainly not possible to implement because of the need for an independent VM for each component of the application that makes the price of the system outside of the student credit we got.
- Use machine learning to classify and filter images. More services can be easily added to the system. And of course, the use of convolutional neuron networks for recognizing and classify images is a great thing that can be implemented.
- Design Language system. A collection of reusable functional elements (button, form, header, etc.), which provides a consistent and cohesive experience. It is about conventions and standards. It is not about strict rules but recommendations and guidelines.
- Server-Side Rendering. The client reads the instruction and renders whatever the server tells it to do. It is server-driven: all changes are done on the server-side, so you don't have to ship a brand new app version each time.

- Web Progressive App. PWA is a type of application software delivered through the web, built using common web technologies, including HTML, CSS, and JavaScript. There is a chance that we will end up with mobile apps and switch to a mobile web app.

Bibliography

[1] A. S. Tanenbaum, *Distributed Systems*. 2017.

[2] S. C. and S. A., "Advantages of using docker for microservices." `https://rubygarage.org/blog/advantages-of-using-docker-for-microservices`. [Online; accessed 29-Novemer-2019].

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development."

[4] L. Beatty, "The pros and cons of mysql." `https://www.smartfile.com/blog/the-pros-and-cons-of-mysql/`. [Online; accessed 12-Novemer-2019].

[5] S. Dhruv, "Pros and cons of using postgresql for application development." `https://www.aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/`. [Online; accessed 12-Novemer-2019].

[6] M. Wu, "What are the major advantages of using a graph database?." `https://dzone.com/articles/what-are-the-pros-and-cons-of-using-a-graph-databa`. [Online; accessed 29-Novemer-2019].

[7] solid IT, "System properties comparison datastax enterprise vs. neo4j vs. tigergraph." `https://db-engines.com/en/system/Datastax+Enterprise%3BNeo4j%3BTigerGraph`. [Online; accessed 29-Novemer-2019].

Appendices

Appendix A

Docker

Jose I. Retamal

A.1 Install Docker in Ubuntu Using Command Line

A.1.1 Setup repository

- Update packages

```
$ sudo apt-get update
```

- Install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

- Add Docker's official GPG key :

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg
↪  | sudo apt-key add -software-properties-common
```

- Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

```
$ sudo apt-key fingerprint 0EBFCD88
```

- set up the stable repository:

```
$ sudo add-apt-repository \
  "deb [arch=amd64]
  ↪ https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

A.1.2 Install Docker Community

- Install the latest version of Docker Engine - Community and container, or go to the next step to install a specific version:

```
  $ sudo apt-get install docker-ce docker-ce-cli
  ↪ containerd.io
```

- Verify that Docker Engine - Community is installed correctly by running the hello-world image.

```
  $ sudo docker run hello-world
```

A.2 Run Image Using Docker Hub

- Create repository in Docker Hub.

  https://docs.docker.com/docker-hub/repos/

- Build Image (Local machine):

```
$ sudo docker image build -t
↪ docker-hub-user-name/image-name:version-tag .
```

  Example:

```
$ sudo docker image build -t joseretamal/hash-service:1.0
↪ .
```

- Push Image (Local machine):

```
$ sudo docker push
↪ docker-hub-user-name/image-name:version-tag
```

  Example:

```
$ sudo docker push  joseretamal/hash-service:1.0
```

- Pull Image (Remote machine):

```
$ sudo docker pull
↪ docker-hub-user-name/image-name:version-tag
```

Example:

```
$ sudo docker pull joseretamal/hash-service:1.0
```

- Run image (Remote machine):

  Opening a port and restart on crash or reboot:

```
$ sudo docker run -d -p internal-port:open-port --restart
↪ always --name instance-name
↪ user-name/image-name:version-tag
```

Example:

```
$ sudo docker run -d -p 5151:5151 --restart always --name
↪ hash-service joseretamal/hash-service:1.0
```

  Allowing instance to full network acess (allows acess to local host):

```
$ sudo docker run -d -p  --network="host" --restart
↪ always --name instance-name
↪ user-name/image-name:version-tag
```

Example:

```
$ sudo docker run -d -p  --network="host" --restart
↪ always --name hash-service
↪ joseretamal/hash-service:1.0
```

- Stop instance: (Remote machine):

```
$ sudo docker rm --force instance-name
```

Example:

```
$ sudo docker rm --force hash-service
```

- Check logs: (Remote machine):

```
$ sudo docker logs instance-name
```

Example:

```
$ sudo docker logs hash-service
```

- Bash into the container: (Remote machine):

```
$ sudo docker exec -it instance-name bash
```

Example:

```
$ sudo docker exec -it hash-service bash
```

Appendix B

MySql

Jose I. Retamal

B.1 Install Mysql in Linux Using Command Line

B.1.1 Install MySQL-shell

- Make sure you do not skip the step for updating package information for the MySQL APT repository:

```
$ sudo apt-get update
```

- Install MySQL Shell with this command:

```
$ sudo apt-get install mysql-shell
```

B.1.2 Install MySql server

```
$ sudo apt-get install mysql-server
```

B.1.3 Uninstall MySql server

```
$ sudo apt-get remove --purge mysql*
$ sudo apt-get purge mysql*
$ sudo apt-get autoremove
$ sudo apt-get autoclean
$  sudo apt-get remove dbconfig-mysql
$ sudo apt-get dist-upgrade
```

B.1.4 Setup Replication

https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql

setup master

- Edit the mysql config file,for open the file using vi:

```
$sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

- make the followings changes to the the file, if the field are missing they must be added or if they are commented un commented:

```
server-id       = 1
log_bin                 = /var/log/mysql/mysql-bin.log
binlog_do_db  = replica1
sudo mysql_secure_installation
```

- Restart MySQL:

```
$ sudo service mysql restart
```

- Create user for replication and give permissions:

```
$ sudo mysql -u root
mysql>GRANT REPLICATION SLAVE ON *.* TO
↪  'slave_user'@'%' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

- Get master status, after select the database in one MySQL seasiion :

```
 mysql>FLUSH TABLES WITH READ LOCK;
```

- then open another MySQL seasion(keep the other open):
- Get master status, after select the database in one MySQL seasiion :

```
mysql>SHOW MASTER STATUS;
| File             | Position | Binlog_Do_DB |
+------------------+----------+--------------+--
| mysql-bin.0001580 |      154 | user_login   |
+------------------+----------+--------------+----
```

Note the file (mysql-bin-0001580) and the position.

- After take note of file name and position tables can be unlocked :

```
mysql>UNLOCK TABLES;
```

setup slave

- Edit slave config file :

```
$ sudo vi /etc/mysql/my.cnf
```

Make the following modifications:

```
server-id               = 2
relay-log               =
↪  /var/log/mysql/mysql-relay-bin.log
log_bin                 = /var/log/mysql/mysql-bin.log
binlog_do_db            = newdatabase
```

- Restart MySQL service :

```
$ sudo service mysql restart

```

- Config slave in mysql shell:

```
mysql> CHANGE MASTER TO
MASTER_HOST='104.40.206.141',
MASTER_USER='repl',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='mysql-bin.000160',
MASTER_LOG_POS= 2439;
```

- Start slave

```
mysql> START SLAVE;
```

- Check status

```
mysql> SHOW SLAVE STATUS\G
```

Appendix C

Neo4J

Jose I. Retamal

C.1 Neo4j With Docker

C.1.1 Install

- Pull the latest image from docker hub(`https://hub.docker.com/_/neo4j/`):
- Run Neo4j:

```
$ sudo docker run \
--name neo4j \
-p7474:7474 -p7687:7687 \
-d \
-v $HOME/neo4j/data:/data \
-v $HOME/neo4j/logs:/logs \
-v $HOME/neo4j/import:/var/lib/neo4j/import \
-v $HOME/neo4j/plugins:/plugins \
--env NEO4J_AUTH=neo4j/test \
neo4j:latest
```

C.1.2 Access bash console:

- Access image bash:

```
$ sudo docker exec -it neo4j bash
```

- Access neo4j bash:

```
1 $ cypher-shell -u neo4j -p test
```

Appendix D

Google Cloud Storage

Jose I. Retamal

D.1 Upload Images To Bucket

https://cloud.google.com/storage/docs/reference/libraries#command-line

- Create the Service Account account, [NAME] is thenew name:

```
$ gcloud iam service-accounts create [NAME]
```

- Grant permissions:

```
$ gcloud projects add-iam-policy-binding [PROJECT_ID]
↪ --member
↪ "serviceAccount:[NAME]@[PROJECT_ID].iam.gserviceaccount.com"
↪ --role "roles/owner"
```

- Generate the key file, [FILENAME] is the name of the new file:

```
$ gcloud iam service-accounts keys create
↪ [FILE_NAME].json --iam-account
↪ [NAME]@[PROJECT_ID].iam.gserviceaccount.com
```

- Provides authentication to the application by setting credentias in the paht(Linux):

```
$ sudo export GOOGLE_APPLICATION_CREDENTIALS="[PATH]"
```

- Set path variablle in docker image file:

```
$ ENV GOOGLE_APPLICATION_CREDENTIALS="[PATH]"
```

- Go code to upload a file :

```
$ // Sample storage-quickstart creates a Google Cloud Storage bucket.
package main

import (
"context"
"fmt"
"log"
"time"

"cloud.google.com/go/storage"
)

func main() {
  ctx := context.Background()

  // Sets your Google Cloud Platform project ID.
  projectID := "YOUR_PROJECT_ID"

  // Creates a client.
  client, err := storage.NewClient(ctx)
  if err != nil {
    log.Fatalf("Failed to create client: %v", err)
  }

  // Sets the name for the new bucket.
  bucketName := "my-new-bucket"

  // Creates a Bucket instance.
  bucket := client.Bucket(bucketName)

  // Creates the new bucket.
  ctx, cancel := context.WithTimeout(ctx, time.Second*10)
  defer cancel()
  if err := bucket.Create(ctx, projectID, nil); err != nil {
    log.Fatalf("Failed to create bucket: %v", err)
  }

  fmt.Printf("Bucket %v created.\n", bucketName)
}
```

40

Appendix E

Goland

E.1 Create and Publish Modules

- Create module

```
$ go mod init <module-path>
```

- Install or update dependencies

```
$ go get
```

- Remove unused dependencies

```
$ go mod tidy
```

- Publish to GitHub

```
$ git add .
$ git commit -m "the commit"
$ git tag v1.0.0
$ git push origin v1.0.0
```

E.2 Install Go in Linux

```
$ sudo snap install go --classic
```

E.3 Go Docker Image

```
FROM golang:latest
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
```

```
4     EXPOSE <PORT-NUMBER>
5     ENTRYPOINT ["./<EXECUTABLE>", "<COMMAND-LINE-PARAMETERS>"]
```

E.4 Protocol Buffer

- Install

```
1     $ sudo snap install protobuf --clasic
```

- Compile using GRPC plugin

```
1     $ protoc --go_out=plugins=grpc:<OUT-FOLDER>
      ↪  <FILE-NAME(.proto)>
```

Appendix F

How to run

In order to run the project on android, you need to navigate to the project folder and type in the command as shown below:

```
cd ProjectName
npx react-native run-android
```

This (in theory) will automatically start the Metro server in another command line console as in Fig. F.1.

The first time you run the code, the development server (metro) might not start, in that case run the code again.

From time to time when a program doesn't compile and origin of the thrown errors cannot be identified the following command helps:

```
gradlew clean
```

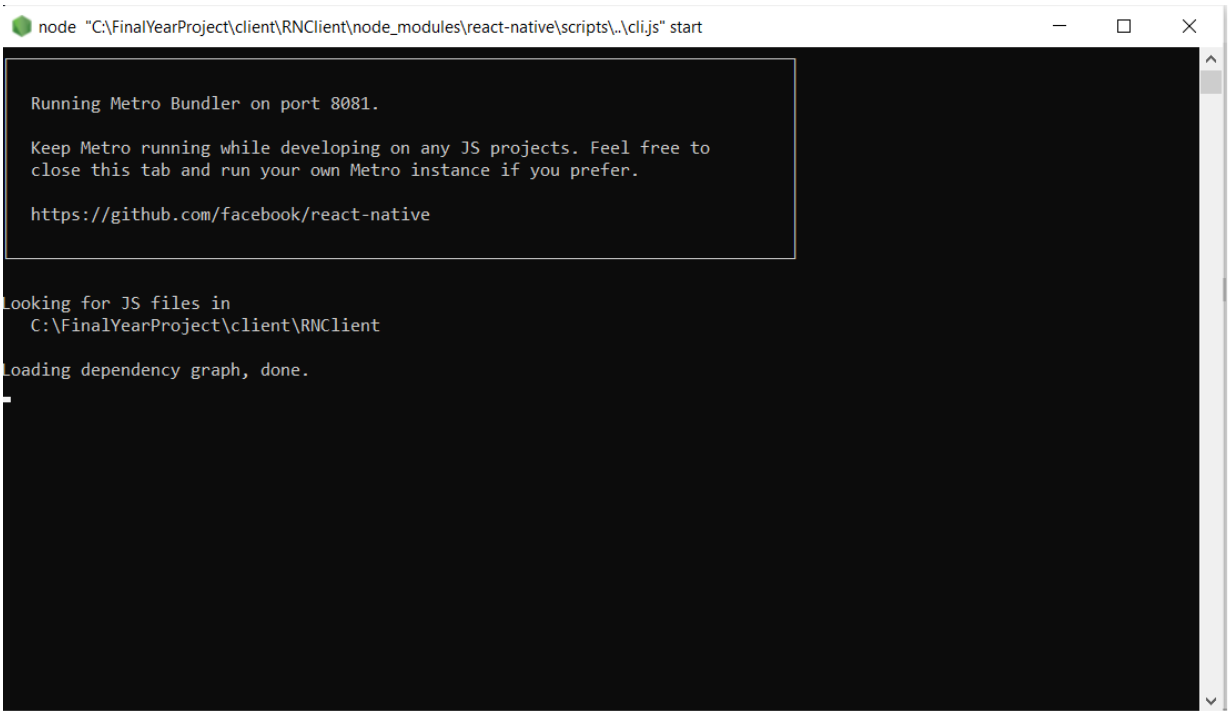


Figure F.1: Metro Server

Appendix G

Installation & User Guide

In order to install the app

1. enable access developer options and enable USB debugging Instructions how to do that on Android device can be found here:
`https://www.howtogeek.com/129728/how-to-access-the-developer-options-menu-and-en`
`2/`
2. download APK at the following page: `https://github.com/JoseIgnacioRetamalThomsen/FinalYearProject/tree/master/APK`
3. navigate to 'installation files' folder on your phone as illustrated in Fig. G.1
4. change security settings by sliding (or checking) 'Unknown sources'
5. install the app by clicking 'install' button

All the steps described above are illustrated in Fig. G.1-Fig. G.6

Video explaining how to use the app can be found at
`https://github.com/JoseIgnacioRetamalThomsen/FinalYearProject/tree/master/video`

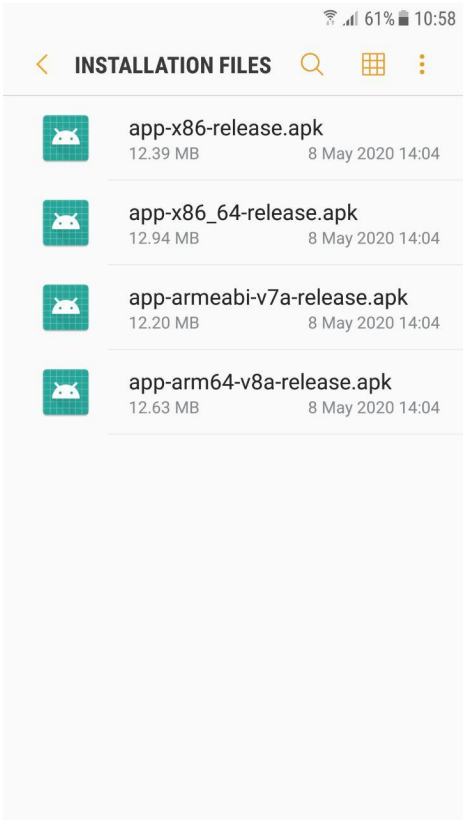


Figure G.1: APK1

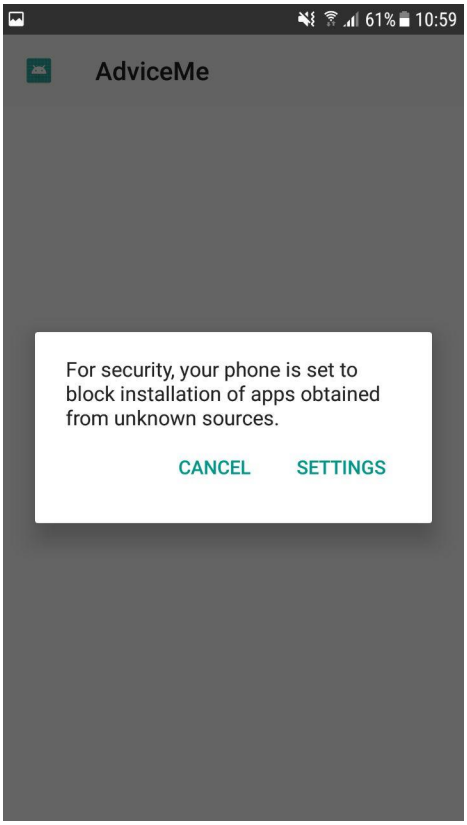


Figure G.2: APK2

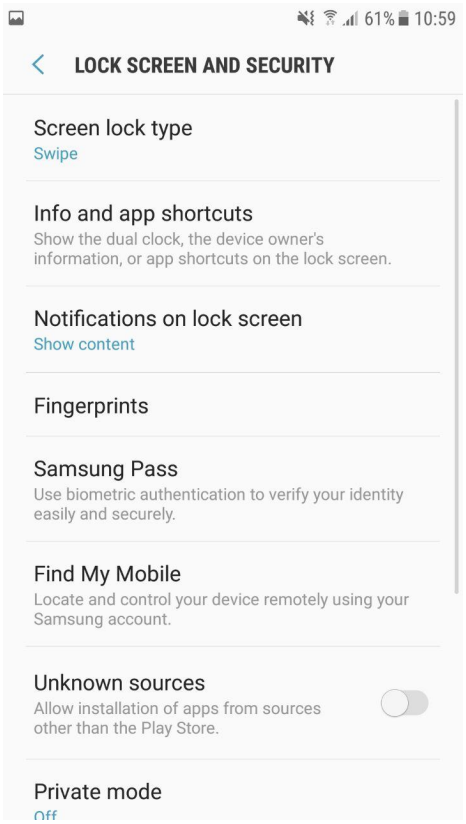


Figure G.3: APK3

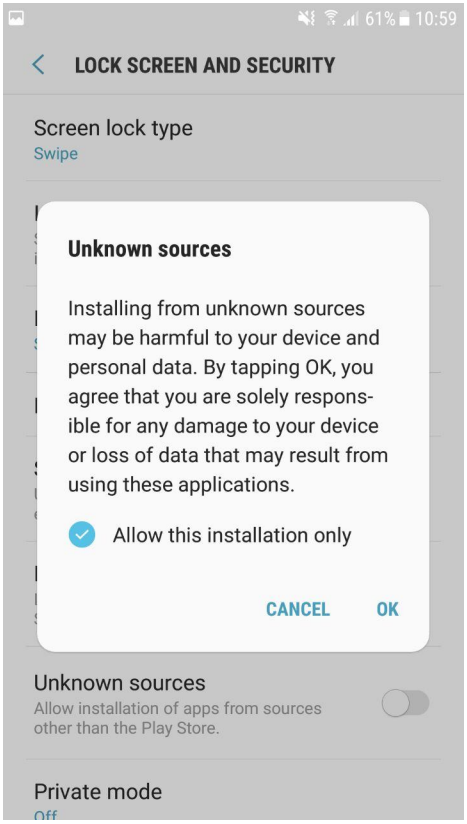


Figure G.4: APK4

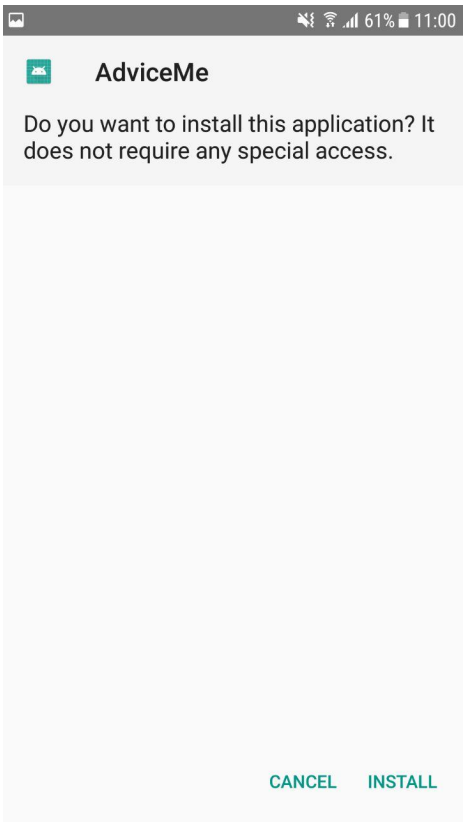


Figure G.5: APK5

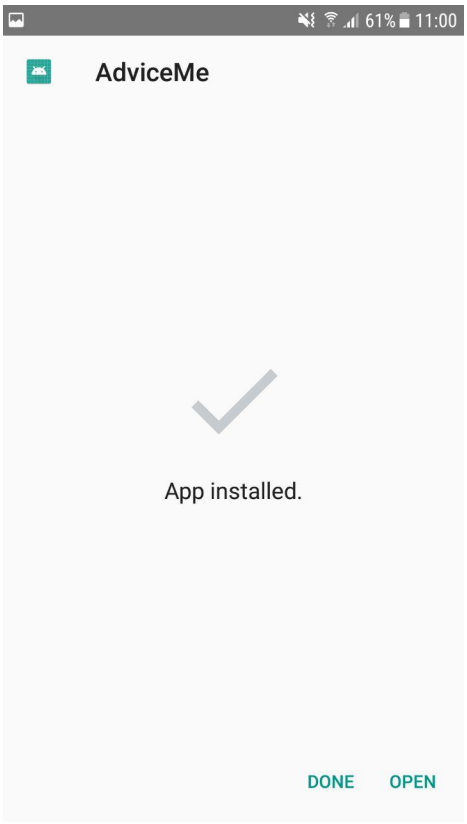


Figure G.6: APK6