

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE BIOLOGIA ANIMAL



**Development of a mobile application for georeferenced data
collection in the field**

Catarina Isabel Rodrigues Pereira da Silva

Mestrado em Bioinformática e Biologia Computacional

Dissertação orientada por:
Professor Doutor Octávio S. Paulo

2019

“Yes, we can.”

Barack Obama

**To my family, which I love from the bottom of my heart.
Thank you for your support and faith.**

Acknowledgments

To Professor Doctor Octávio Paulo, my advisor that proposed me this challenge and guided me throughout its course.

To the CoBig2 Group, for all guidance along the design and architecture of the intended information system.

To my colleagues Doctor Francisco Pina Martins and Master Pedro David for all the contributions and route guidance, without them I would not have been able to overcome some of my limitations in this area.

To the Faculdade de Ciências da Universidade de Lisboa and its Professors, for providing me with access to the knowledge that today is an integral part of me.

To Manuel Quintãos for all his help.

To my colleagues, friends and family for company and conviviality, for the good times and support along this route. In order to avoid the risk of not enumerating someone I will not identify anyone, those to whom these thanks are addressed will know it.

To my boyfriend, for the absences in the small things, and for his motivation and inexhaustible source of patience in bad times.

To my best friends, Célia Costa, Débora Silva, Inês Catita, Inês Vieira and Joana Vital, with whom I have always counted.

To Junior, for being my companion at all times and keeping my mind healthy.

My father, who in spite of a few words, always gave me his support and accompanied me at all times of my life.

To my sister and my brother for the encouragement, unconditional support and above all, complicity in all my journeys.

To my mother for the strong and systematic motivation she has always given me, for the help and unreserved attention..., for the love and lap I always have at my disposal, and above all, for the model of courage and determination that is reflected in my personality today.

To all, my thanks.

This dissertation was part of the FCT Keep Pace Project: Seleção de árvores capazes de acompanhar as rápidas alterações ambientais, base para a gestão de montados sustentáveis do século XXI PTDC/ASP-SIL/29263/2017



FCT Fundação
para a Ciência
e a Tecnologia

Resumo

O século XX foi um período que se notabilizou por um conjunto de conquistas tecnológicas no campo da aquisição, processamento e distribuição da informação, com grande destaque à escala mundial na instalação das redes de telefonia, na invenção do rádio e da televisão e no crescimento ímpar da indústria informática, lançamento de satélites de comunicação e descoberta da Internet.

O século XXI, tem sido marcado com todo este desenvolvimento tecnológico. O crescimento da nossa capacidade de recolher, processar e distribuir informações tem vindo a potenciar formas cada vez mais sofisticadas de processar a informação.

Entre as principais mudanças do novo século está a evolução da computação móvel. Esta área tecnológica visa estudar sistemas de computadores nos quais há total mobilidade do utilizador, tornando aspetos como a dimensão um aspeto central no desenvolvimento de sistemas móveis. Desta forma, o utilizador tem total liberdade para obter e manipular informação em qualquer sítio e em qualquer lugar.

O avanço na computação móvel tem potenciado um aumento significativo da utilização de dispositivos móveis para a resolução de tarefas diárias e, em particular, no acesso à informação. Com este aumento na utilização de dispositivos móveis, surgiram os *smartphones*, que permitem integrar as funcionalidades típicas de um telemóvel num conjunto de aplicações. O aumento destas funcionalidades, em conjunto com a tão aguardada convergência entre telefones e a Internet, tem vindo a potenciar o desenvolvimento de aplicações nas mais diversas áreas. Estas aplicações visam fornecer aos utilizadores uma função específica, que pode ser uma ferramenta de entretenimento ou trabalho, permitindo novas formas de trabalhar e novos estilos de vida.

O grupo CoBiG2, uma das equipas do Centro de Ecologia, Evolução e Mudanças Ambientais (cE3c) da Faculdade de Ciências da Universidade de Lisboa, tem como objetivo estudar a diversificação evolutiva e ecológica de espécies em ambientes naturais e o processo genómico de adaptação de organismos e populações aos seus *habitats*. O objetivo geral deste grupo é entender a genética e a genómica das mudanças ambientais e as suas consequências para a biodiversidade. De forma a atingir os seus objetivos, os investigadores têm a necessidade de recolher dados descritivos de amostras de diferentes espécies no seu *habitat* natural.

Neste momento, todos os dados recolhidos em campo, bem como o seu armazenamento em formato de dados, são feitos manualmente. Este método mostrou-se ineficaz, pois, dada a quantidade de trabalho a ser feito e dados a serem tratados, afeta fortemente a produtividade do grupo.

Assim, para contornar a ineficiência na recolha e armazenamento de dados, este projeto tem como objetivo automatizar todo o processo de recolha e desenvolver uma solução baseada no uso do processamento de dados.

Pretende-se com esta dissertação identificar os desafios no desenvolvimento de aplicações móveis para o sistema operacional Android que atendam aos seguintes requisitos: Suporte à navegação em mapas digitais; Determinação da localização geográfica do utilizador em tempo real, usando tecnologias de localização para telemóveis; Suporte no armazenamento local de conteúdo, para lidar com o uso *offline* da aplicação móvel; Comunicar com um servidor remoto para consultar e atualizar dados numa base de dados.

Os objetivos desta dissertação são os seguintes: Conceber e implementar uma base de dados para suportar o armazenamento e manipulação dos dados recolhidos pela aplicação móvel no campo; Identificar e analisar os desafios inerentes ao desenvolvimento de aplicações Android com serviços de localização e recursos de armazenamento local; Conceber e implementar um protótipo de uma aplicação móvel que atenda aos requisitos mencionados acima; Conceber e desenvolver um *Website* para que os utilizadores possam visualizar e gerir os dados recolhidos pela aplicação; Avaliar os resultados dos protótipos em cenários controlados.

Como resultado esperado está a construção de um sistema informático que visa automatizar os processos de recolha, armazenamento, visualização e gestão de dados descritivos e georreferenciados de amostras biológicas recolhidas em campo. É ainda esperado que o sistema permita que o processo de recolha de dados se realize mesmo quando o utilizador não tem conexão à Internet, uma vez que, geralmente, os locais de recolha deste tipo de dados possuem pouca cobertura de rede.

Assim, este projeto teve como objetivo desenvolver uma aplicação móvel como ferramenta de trabalho para investigadores de um grupo, cujo objetivo é auxiliar na recolha de dados georreferenciados em campo. A escolha do desenvolvimento de uma aplicação móvel deve-se à forte expansão global do mercado dos *smartphones* e à subsequente disponibilidade de aplicações móveis. Devido ao tamanho reduzido dos *smartphones*, facilidade de transporte e alta complexidade tecnológica, tanto em termos de capacidade de processamento quanto dos inúmeros recursos que oferecem, foi estabelecido que esta seria a escolha mais favorável para atender às necessidades em campo dos membros do grupo CoBiG2.

A aplicação toma o nome de “C2MC – CoBiG2 Mobile Collector”.

A existência de várias plataformas para desenvolvimento móvel, que utilizam diferentes linguagens e ferramentas de programação, dificulta a produção de uma aplicação que cubra todos os sistemas operativos. No entanto, a aplicação foi desenvolvida para o sistema operativo Android, uma vez que, é a plataforma móvel com maior crescimento nos últimos anos, fornece o *kit* de desenvolvimento de aplicações (disponível para a comunidade de *developers* a custo zero) e integra as APIs da Google, como as Google Maps APIs.

Como suporte à aplicação móvel, foi desenvolvida uma base de dados para armazenar os dados recolhidos de forma consistente, não redundante e com integridade lógica. Desde o início da computação que as operações de armazenamento e recuperação de informações andam de mãos dadas. Estas operações, por mais simples que pareçam, têm as suas dificuldades e determinam em grande parte a fiabilidade e eficiência do sistema envolvente, sendo assim necessário aplicar uma abordagem inteligente e eficaz. Desta forma, de maneira a retirar da aplicação cliente, a responsabilidade de gerir o acesso, a persistência, a manipulação e a organização dos dados, foi utilizado o Sistema de Gestão de Bases de Dados Relacionais MySQL.

Também como suporte, foi desenvolvido um *Website*, de forma a permitir aos utilizadores visualizar e gerir os dados recolhidos pela aplicação móvel na base de dados.

Um dos desafios inerentes ao desenvolvimento de uma aplicação para auxílio no trabalho de campo, é a necessidade de haver comunicação entre a aplicação e a base de dados, e por isso, necessidade de conexão à Internet. Para o efeito, a aplicação foi desenvolvida para que funcionasse tanto offline como online.

À medida que o sistema foi desenvolvido, todos seus componentes foram alvo de testes, de forma a verificar se os requisitos necessários foram implementados corretamente. A primeira fase do processo de testes ocorreu durante o desenvolvimento das componentes do sistema em ambiente local usando ferramentas de trabalho para programadores. A segunda fase consistiu em realizar testes em ambiente real, ou seja, utilizar a aplicação móvel no campo, recolher e enviar dados ao servidor e, de seguida, visualizá-los e manipulá-los por meio da aplicação Web.

Todos os objetivos inicialmente propostos, foram concretizados com sucesso.

Sem dúvida que os próximos anos terão um enorme impacto sobre como as novas tecnologias da informação interagem com áreas do conhecimento científico. O futuro dos sistemas de informação parece promissor quando se trata de automatizar processos e desenvolver soluções baseadas no uso de processamento de dados, que visam aumentar a eficiência de investigadores e, conseqüentemente, a sua produtividade no trabalho.

Como sugestão para trabalho futuro, propõe-se desenvolver a aplicação móvel para os sistemas operativos mais utilizados, a fim de integrar uma maior comunidade de utilizadores.

É também proposta a implementação de funcionalidades que permitam integrar imagens e arquivos de som aos dados adicionados na aplicação móvel.

Poderá também ser interessante desenvolver aplicações semelhantes, embora adaptadas, para outras equipas do cE3c.

Palavras-chave: Aplicação Móvel, Aplicação Web, Android, MySQL, Georreferenciação.

Abstract

The advance in mobile computing has boosted a significant increase in the use of mobile devices for solving daily tasks and, in particular, to access information. With this increase in the use of mobile devices, smartphones have emerged. The increase of these features, together with the long-awaited convergence between phones and the Internet, has been driving the development of applications in the most diverse areas.

This project aimed to develop a mobile application as a working tool for researchers of the CoBiG2 group. The main objective was to assist in the collection of georeferenced data in the field. For this purpose, and due to the strong global expansion of the smartphone market and subsequent availability of mobile applications, it was decided to develop a mobile application that would meet the proposed requirements. The application was developed for the Android operating system, and integrates the Google Maps APIs. The application's name is 'C2MC - CoBiG2 Mobile Collector'.

To support the mobile application, a database was developed in order to store the collected data consistently, non-redundantly and with logical integrity. Also, as a support, a Website was developed to allow users to visualize and manage the data in the database. Regarding the implemented database, the proposed and implemented database structure was developed in the Relational Database Management System MySQL.

As the system was developed, its components were thoroughly tested to verify if the required requirements were implemented correctly. The first phase of the testing process occurred during the development of the system's components in a local environment, using developers working tools. The second phase consisted of performing tests in a real environment, i.e., using the mobile application in the field, collecting data and sending it to the server, and then visualizing and manipulating it through the Web application.

Keywords: Mobile Application, Android, MySQL, Web Application, Georeferencing.

TABLE OF CONTENTS

Acknowledgments	IV
Resumo	VI
Abstract.....	IX
Chapter 1 - Introduction.....	1
1.1 Theoretical Framework	1
1.2 Problem Description.....	2
1.3 Goals.....	2
1.4 Expected Results	2
1.5 Document Structure.....	3
Chapter 2 - Literature Review	4
2.1 Databases.....	4
2.1.1 Database Management System	4
2.1.2 Relational Databases and SQL	5
2.1.3 Integrity Constraints	5
2.1.4 Database Creation.....	6
2.1.5 MySQL.....	6
2.1.6 Database Security	7
2.1.7 SQL and Other Languages.....	7
2.2 Android Operating System.....	7
2.2.1 Android Platform Architecture	8
2.2.2 Programming Languages.....	10
2.2.3 Android Fundamentals.....	11
2.2.4 Android Studio	14
2.2.5 Network Operations in Android	16
2.3 Website Fundamentals	17
2.3.1 Front-end Web development	17
2.3.2 Back-end Web development.....	18
2.4 Computer Applications Architecture.....	20
2.5 Client-Server Communication.....	21
2.6 Maps for Computer Applications	21
2.6.1 Mobile Applications Maps	22
2.6.2 Web Applications Maps	22
2.6.3 Mobile Location Technologies	22
2.7 Similar Applications.....	23
Chapter 3 - Methodology and Implementation	24
3.1 System Requirements.....	24
3.2 System Architecture	24
3.3 Database Development and Implementation.....	25

3.3.1	Planning	25
3.3.2	Analysis	25
3.3.3	Implementation	26
3.4	Android Application Development and Implementation	27
3.4.1	Requirements	28
3.4.2	Application Architecture	28
3.4.3	Android Client-Server Communication.....	31
3.4.4	User Interface design and modelling	33
3.4.5	Passing data between Android UI components	46
3.5	Web Application Development and Implementation	47
3.5.1	Requirements	47
3.5.2	Front end Development	47
3.5.3	Back end Development.....	52
Chapter 4 - Tests		56
Chapter 5 - Conclusions and Future Work		58
5.1	Conclusions	58
5.2	Future Work	59
Chapter 6 – References.....		60
Annexes		65
	Annex 1 – Example of data contained in the database tables	65

LIST OF FIGURES

Figure 2.1. Mobile Operating System Market Share Worldwide between 2009 to 2019	8
Figure 2.2. The Android software stack	10
Figure 2.3. Simplified illustration of the activity lifecycle.	12
Figure 2.4. Android Project Structure.	15
Figure 2.5. Android Emulator.	16
Figure 2.6. Representation of the 3-tier client-server model.....	20
Figure 3.1. System Architecture.....	24
Figure 3.2. E-R diagram - Database schema.	26
Figure 3.3. Jetpack components.	28
Figure 3.4. Android Application Architecture.	29
Figure 3.5. Login Activity interface.....	33
Figure 3.6. Main Activity interface..	34
Figure 3.7. NavGraph.....	35
Figure 3.8. Bottom Navigation bar.....	36
Figure 3.9. User Fragment interface.....	38
Figure 3.10. Maps Fragment interface.	39
Figure 3.11. Data Fragment interface.....	43
Figure 3.12. History Fragment interface.	45
Figure 3.13. Sign In Web page interface.....	48
Figure 3.14. Sign Up Web page interface.	48
Figure 3.15. Administrator user's header.....	49
Figure 3.16. Maps Web page interface.....	49
Figure 3.17. Marker's data Web page interface.....	50
Figure 3.18. Search Web page interface.....	50
Figure 3.19. Input Web page interface.	51
Figure 3.20. Administrator Web page interface.	51

LIST OF ABBREVIATIONS AND ACRONYMS

Ajax: Asynchronous JavaScript and XML
API: Application Programming Interface
APK: Android Package
ART: Android runtime
AVD: Android Virtual Device
C2MC: CoBiG2 Mobile Collector
C3ec: Center for Ecology, Evolution and Environmental Changes
CRUD: Create, Read, Update, Delete
CSS: Cascading Style Sheets
DAO: Data Access Object
DB: Database
DBMS: Database Management System
DCL: Data Control Language
DDL: Data Definition Language
DML: Data Manipulation Language
E-R: Entity-Relationship
GPS: Global Positioning System
GUI: Graphical User Interface
HAL: Hardware Abstraction Layer
HTML: HyperText Markup Language
HTTP: Hypertext Transfer Protocol
ID: Unique Identifier
IDE: Integrated Development Environment
JSON: JavaScript Object Notation
MVVM: Model–view–viewmodel
OS: Operating System
PHP: Hypertext Preprocessor
POJO: Plain Old Java Objects
RDBMS: Relational Database Management System
SDK: Software Development Kit
SQL: Structured Query Language
UI: User Interface
URL: Uniform Resource Locator
WPS: Wireless Positioning System
XML: eXtensible Markup Language

Part I

Chapter 1 - Introduction

1.1 Theoretical Framework

The twentieth century was a period that was notable for a number of technological achievements in the field of acquisition, processing and distribution of information, highlighted in the growth of the computer industry, launch of communication satellites and Internet (Tanenbaum & Wetherall, 2018).

The 21st century has been marked with all this technological development. The growth of the ability to collect, process and distribute information has been fostering increasingly sophisticated ways of processing information.

Among the major new century's changes is the evolution of mobile computing. This technological area aims to study computer systems in which there is total user mobility, making aspects such as dimension, a central aspect in the development of mobile systems. Thus, the user has complete freedom to obtain and manipulate information anywhere, anytime.

The advance in mobile computing has boosted a significant increase in the use of mobile devices for solving daily tasks and, in particular, access to information. With this increase in the use of mobile devices, smartphones have emerged, which allow to integrate the typical features of a mobile phone in a set of applications. The increase of these features, together with the long-awaited convergence between phones and the Internet, has been driving the development of applications in the most diverse areas. These applications aim to provide users with a specific function, which can be a work or entertainment tool, enabling new ways of working and new lifestyles.

Thus, this project aimed to develop a mobile application as a working tool for researchers from a group of scientists, whose objective is to assist in the collection of georeferenced data in the field. The choice of developing a mobile application is due to the strong global spread of the smartphone market and subsequent availability of mobile applications. Due to smartphone's small size, ease of transport and high technological complexity, both in terms of processing capacity and the numerous features it offers, it was established that this was the most favourable choice for meeting the needs of the CoBiG2 researchers in the field.

The existence of several mobile development platforms, which use different programming languages and tools, make it difficult to produce an application that covers all operating systems (Android operating system (OS), Apple iOS, Blackberry OS, Nokia's Symbian, Microsoft's Windows Phone OS, among others). However, the application was developed for the Android OS, as it is the fastest growing mobile platform in recent years, provides the applications' development kit (available to the developer community at zero cost), and integrates the popular Google Application Programming Interfaces (APIs), such as the Google Maps APIs. The application's name is 'C2MC – CoBiG2 Mobile Collector'.

To support this application, a database has been developed to store the collected data consistently, non-redundantly and with logical integrity. Since the dawn of computing, the operations of storing and retrieving information have gone hand in hand. These operations, as simple as they may seem, have their difficulties and largely determine the reliability and efficiency of the surrounding system. As such, a smart and effective approach needs to be applied. Thus, in order to remove from the client application the responsibility of managing data access, persistence, manipulation and organization, a database management system was used (Costa, 2019).

Also as a support, a Website has been developed to allow users to view and manage the data collected by the application in the database.

One of the challenges inherent in developing a fieldwork application that needs to communicate with a database, and therefore, needs an Internet connection, is that many of the collection locations have poor network coverage. Thus, it was intended to develop an application for data collection that works both *offline* and *online*, which, after user intention, would communicate with the database and efficiently stores the data.

1.2 Problem Description

The CoBiG2 group, one of the teams of the Center for Ecology, Evolution and Environmental Changes (cE3c), aims to study the evolutionary and ecological diversification of species in natural environments and the genomic process of organism adaptation and populations to their environments. The overall objective of this research group is to understand the genetics and genomics of environmental changes and their consequences for biodiversity. In order to achieve their objectives, researchers need to collect descriptive data of different species in their natural environment.

At this time, all data collected in the field, as well as its storage in data format, is done manually. This method has proved ineffective since, given the amount of work to be done and data to treat, it strongly affects the group's productivity.

Thus, in order to circumvent the ineffective collection and storage of data, this project aims to automate the entire collection process and develop a solution based on the use of data processing.

1.3 Goals

It is intended with this dissertation to identify the challenges in the development of mobile applications for the Android OS that meet the following requirements:

- Support navigation on digital maps;
- Determine the user's real-time geographic location, using location technologies for mobiles;
- Support local storage of contents, in order to deal with *offline* app usage;
- Communicate with a remote server to query and update data in a database.

The objectives of this dissertation are as follows:

- Design and implement a database to support the storage and manipulate of the data collected by the mobile application in the field;
- Identify and analyse the challenges inherent in developing Android apps with location services and local storage features;
- Design and implement a prototype of a mobile application that meets the above requirements;
- Design and develop a Website so the users can visualize and manage the data collected by the application;
- Evaluate the prototypes' results in controlled scenarios.

1.4 Expected Results

The expected results are the construction of an informatic system that aims to automate the processes of collection, storage, visualization and management of georeferenced descriptive data of biological samples collected in the field, as well as allow the data collection process to be carried out if there is no Internet connection, as collection locations often have poor network coverage.

1.5 Document Structure

The present work is structured in three parts, which in turn are divided into six chapters, being carefully separated as follows:

Part I - Chapter 1 (Introduction) - describes the nature and scope of the research, the relevance of the study and its methodological framework.

Part II - Chapters 2 (Literature Review) - In these chapter the research's theoretical framework is described. This chapter consists of seven sections, whose intention is to provide background information and requirement concepts that will help understand what comes further on.

Part III - Chapters 3, 4, 5 and 6 - Chapter 3 (Methodology) - This chapter describes the system architecture, the process of database creation, mobile application and Web application development.

Chapter 4 (Tests) - In this chapter the tests performed on the system's components are described. These tests aim to ensure the correct functioning of components when manipulating data, and the correct operation and visualization of mobile application interfaces on various devices, and Web application in various browsers. **Chapter 5 (Conclusions and Future Work)** presents the main conclusions of the study, as well as the limitations that arose during the project development, and future work. **Chapter 6** consists of the **Bibliography**, followed by **Annexes**, that underlie the development of the research.

Part II

Chapter 2 - Literature Review

This chapter consists of seven sections and is intended to provide background information and requirement concepts that will help understand what comes further on.

2.1 Databases

The origin of databases, prior to the invention of computers, goes back to libraries, government, commercial and medical records. Once people realized that they needed the means to store data and keep their files for later retrieval, they made several attempts to find the best way to store, index, and retrieve data (Berg, Seymour & Goel, 2013). The data were stored directly in files, which implemented some weaknesses in data storage, such as redundancies and inconsistencies, difficulty in accessing data, lack of logical integrity, lack of transaction atomicity, and insecurity (Castelano, 2015).

In order to solve the weaknesses mentioned above, a set of organized files, whose main purpose was to remove from client applications the responsibility for managing data's access, persistence, manipulation and organization, appeared and is known as Database Management Systems (DBMS) (Costa, 2019).

Although the model adopted by DBMS solved many of the problems mentioned above, if the internal structure of the data were changed, the programs that used the same data had to be modified and restructured. Thus, in 1970, the concept of relational model emerged, based on the theory in which a database is seen as a collection of relationships, which in turn are sets of tuples, which are groups of attributes (Costa, 2019).

Therefore, the notion of database is nothing more than a collection of structured, organized and persistently stored data. (Damas, 2017; Molina, Ullman & Widom, 2008). Databases are essential to every business and are used to preserve internal records, present data to employees and customers on the World-Wide-Web, and support many other business processes. "Databases are likewise found at the core of many scientific investigations, representing the data gathered by astronomers, by investigators of the human genome, and by biochemist exploring the medicinal properties of proteins, along with many other scientists." (Molina, Ullman & Widom, 2008).

2.1.1 Database Management System

When creating a database, it is necessary to guarantee that the data have some kind of meaning and organization. A DBMS is a versatile and powerful tool that enables to create and manage large amounts of data, and provides the interface between data that is physically stored in the database and the user (an individual or computer application). All the information is efficiently and persistently stored for large periods of time and can be transferred from one place to another more easily than any traditional information storage method. Using a DBMS allows the user to stop worrying about how data is stored, searched or sorted, becoming the responsibility of the system manager to perform these tasks. The functionalities provided by a DBMS can range from creating new databases and specify their *schema* (logical structure of data), give users the ability to query and modify the data using an appropriate language, support safely storage of large amounts of data persistently, and control the simultaneous access to data by different users at once without the actions of one user affecting the action of another. As such, the DBMS provides the complete set of data access services. In order for the user to be able to

query and operate on the database, it is necessary to use some kind of an appropriate language. This is where the Structured Query Language (SQL) emerges (Damas, 2017; Molina et al., 2008).

2.1.2 Relational Databases and SQL

The SQL language is not used in all database management systems. However, those who use it have adopted the so-called relational model and are known as Relational Database Management Systems (RDBMS). Thus, SQL is a specific and standardized language used to access and manipulate relational databases (Damas, 2017).

The concept of relational database system first appeared in 1970, originally proposed by E. F. Codd, where the database system should present the user with a view of data organized as tables called *relations*, consisting of rows and columns, where columns represent the *attributes*, i.e., values describing properties of an *entity* (a concrete object in its reality), and rows represent the records or instances of an entity. Connections between entities are called *relationships* (Badiru, 2014; Biscobing, 2014; Damas, 2017; Molina et al., 2008).

As such, each table contains one or more data attributes as columns. Each attribute has an associated data type according to the type of information it will store (Badiru, 2014; Biscobing, 2014; Damas, 2017).

Each row, also referred to as record or tuple, represents a singular instance of data for the categories outlined by the columns (Badiru, 2014; Biscobing, 2014).

Each table contains a unique primary key, which consists of an attribute (or combination of attributes) that uniquely identify each row in a table. Primary keys cannot contain NULL values or be repeated (Badiru, 2014; Biscobing, 2014; Damas, 2017).

Relationships across tables in a relational database are established by using foreign keys – attribute (or combination of attributes) of a table that links to the primary key of another table (Badiru, 2014; Biscobing, 2014).

2.1.3 Integrity Constraints

In order to ensure data consistency, there is a set of rules, called integrity constraints, that must be guaranteed by the RDBMS. Integrity constraints are one of the primary purposes of an RDBMS and, as it deals with the validations of data before storing it, allows data from a database to be considered healthy (always in accordance with the rules defined in the database and consistent with each other). Data integrity simultaneously includes the concepts of consistency, accuracy and correctness of data stored in a database. There are three types of integrity (Damas, 2017; MySQL, n.d.):

- **Entity Integrity:** Each row of a table must be uniquely identifiable, i.e. it prevents two tuples from having the same value for the primary key.
- **Domain Integrity:** Data items in a column must conform to the valid data type and value constraints allowed for that column.
- **Referential Integrity:** Foreign key value of a table must match the value present in the primary key of the table they refer to, or at most, these fields can be NULL. In order to maintain referential integrity, if the referenced value (children) is updated or deleted from the parent table, strategies must be defined in order to redefine the value of the foreign key. These strategies may be:
 - NO_ACTION: rejects updating/deleting a parent when there are children;
 - CASCADE: updates/deletes the children if the parent is updated/deleted;
 - SET_NULL: redefines the children as NULL if the parent is updated/deleted;

- SET_DEFAULT: redefines the children with their default value if the parent is updated/deleted.

2.1.4 Database Creation

The logical design of a database, using analysis tools, is the first step in creating a database.

The initial phase of this process consists of elaborating a visual scheme that represents the structure to be implemented as a database. To elaborate the scheme, usually the entity-relationship model (E-R model) is used. At this stage, entities are identified, and their relationships established, as well as identifying the characteristics/attributes that are part of each one (Damas, 2017).

After the data model design is completed, the physical implementation phase of the database begins. This phase involves considering the structure specified in the E-R model and the respective data model. For this purpose, it is necessary to select one of the commercially available RDBMS (e.g. MySQL) and then create a database that supports the previously defined structure. Next, the tables, that will be used to store the data, are created, as well as the columns that will contain and the data type of each one (Damas, 2017).

The implementation phase uses certain SQL commands in order to carry out the required tasks, which are mainly categorized into three categories/sublanguages (Damas, 2017):

- Database *schema* is defined using an appropriated language, usually called Data Definition Language (DDL). "DDL statements create, modify and remove database objects, such as tables, indexes and users" (TechTarget, 2009). The most common DDL statements are CREATE to create a database and its objects (such as tables), ALTER to modifying a database structure and DROP for delete a database or its objects (Damas, 2017; TechTarget, 2009; Varshni, n.d.).
- Security control of the database is done using Data Control Language (DCL). DCL includes commands that deal with the rights, permissions and other controls of the database system, such as GRANT (give user's access privileges to database), REVOKE (remove user's access privileges), among others (Damas, 2017; Varshni, n.d.).
- After the implementation, the manipulation of data inside a relational database is done using an appropriated language, usually called Data Manipulation Language (DML). DML statements insert, delete and update data in a database. DML statements are INSERT for insert new data into a table, DELETE for removing rows from a table and UPDATE for modifying existing data in a table. Although these statements are frequently used, DML also includes the SELECT statement, which is the most common interaction with a database. It allows users to fetch data from database tables and display only the data that meets specific criteria in the form of a result table (Damas, 2017).

"SQL is a high-level language because it only needs to tell the system what to do, not how and where to do it" (Damas, 2017).

2.1.5 MySQL

MySQL is one of the most popular open-source RDBMS developed, distributed, and supported by Oracle Corporation (Banerjee, n.d.; DB-Engines, 2019; Oracle, n.d.) that uses the SQL language as its interface. Due to its proven performance, reliability, scalability, easy-of-use, security, uptime, among others, MySQL has become the premier database choice for small and large Web-based applications (DuBois, 2005; Oracle, n.d.).

2.1.6 Database Security

Security is one of the most important features when it comes to databases, as such ensuring data confidentiality is essential. Data should only be available to those who need it and are authorized to access it (Damas, 2017).

The simplest method for controlling access to database information is to only allow it using a Login/Password set that will be validated by the RDBMS every time a connection to the database is made (Damas, 2017).

Also as a security measure, it is important to restrict the set of actions that each user can perform on the information they have access to, such as restricting the set of users who can execute the DROP TABLE command (Damas, 2017).

2.1.7 SQL and Other Languages

Although SQL allows to do anything in a database, it lacks some common mechanisms that are available in most programming languages, such as *if-then-else*, *while*, *for*, functions, and so on (Damas, 2017).

In order to overcome this limitation, the SQL language is used in conjunction with another language that contains the normal functionality of a programming language (Damas, 2017).

Therefore, it is common for applications' interface to be implemented in a given language that is responsible for handling communications with the database where the data is stored via SQL, e.g. Hypertext Preprocessor (PHP) (Damas, 2017).

PHP is a server-side scripting language, which means that the scripts are executed on the server, that is particularly strong in its ability to interact with databases (php, n.d.-b; Tanenbaum & al, 2018; Valade, 2004; TutorialsPoint, 2019). MySQL is the most commonly database system used with PHP, allowing users to connect and manipulate databases (Banerjee, n.d.).

2.2 Android Operating System

Android is a mobile OS based on Linux kernel and other open source software, designed primarily for touch screen mobile devices such as smartphones and tablets. Android is developed by a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices, known as the Open Handset Alliance, with the main contributor and commercial marketer being Google (Farkade & Kaware, 2015).

This OS is mainly used to run on mobile phones and tablets. However, due to its open and customizable features, its usability is not limited to mobile phones and tablets, being used in a wide array of devices and form factors, like laptops, smart televisions, cameras, headphones, wristwatches, game consoles and many more (Android Developers, n.d.-j; Poudel, 2013).

Android is the operating system with the highest growth in the past ten years, currently accounting for about 76% of the active mobile phones in the world (Figure 2.1) (statcounter, 2019).

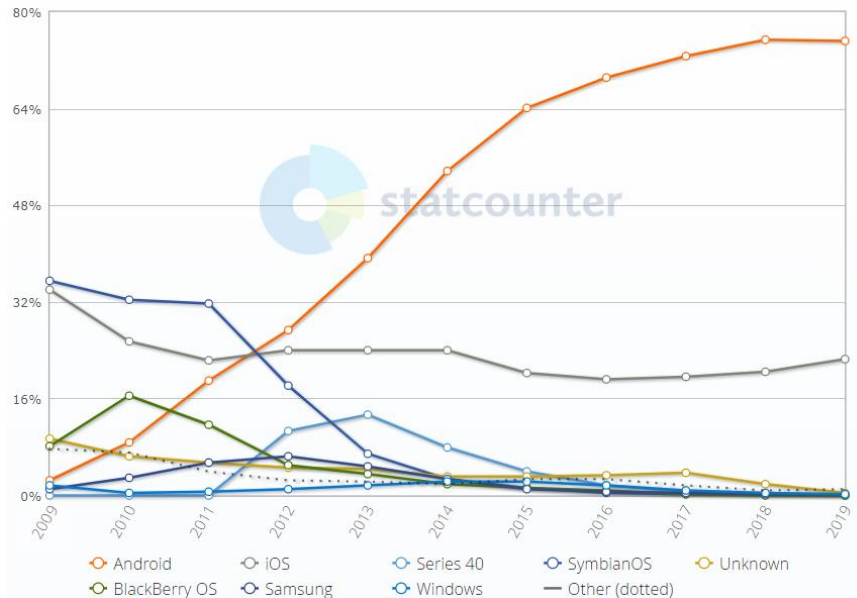


Figure 2.1. Mobile Operating System Market Share Worldwide between 2009 to 2019 (statcounter, 2019).

Unlike other operating systems like Apple iOS (Apple Inc. Products), Blackberry OS (Blackberry), Windows OS (Windows Phone), among others, this operating system is hardware independent and runs on devices from different vendors (Poudel, 2013).

With the evolution of mobile computing and mobile applications’ development, this platform provides several tools that help develop applications for the Android system. The most complete tool is the Android Software Development Kit (SKD), which contains an environment with the features and specifications of the Android system. Android APIs are also available to developers, which are rich set of system services that provide easy access to several features like location, browser, Wi-Fi, camera, among others. Application developers can easily access the huge stack of system services, available tools, and libraries to use in their applications, if required, for free (Android Developers, n.d.-d; Android Developers, n.d.-b; Poudel, 2013).

2.2.1 Android Platform Architecture

Android provides a rich development architecture. The diagram in Figure 2.2 shows the major components of Android.

Android is built on top of Linux. The Linux Kernel corresponds to the base of the platform, which deals with core functionalities such as threading, low-level memory management, and key security features, and allows manufacturers to develop hardware drivers for a well-known kernel (Android Developers, n.d.-j; Google Developers Training Team, 2018).

The Hardware Abstraction Layer consists in several library modules, which implements standard interfaces that allow the Java API framework to communicate with hardware-specific components, such as the camera or Bluetooth. The Android system loads the library module for a hardware component when a framework API makes a call to access that device hardware (Android Developers, n.d.-j; Google Developers Training Team, 2018).

The Android Runtime layer contains a set of core libraries that provide most of the functionality of the Java language and is where the Android runtime (ART) also resides. ART is the managed runtime used by applications and some system services on Android that performs the translation of the application’s bytecode into native instructions for the hardware on the device itself (Frumusanu, 2014; Android Developers, n.d.-j; Google Developers Training Team, 2018).

Native Libraries are written in C and C++ and are required by the native code used to build system components and services. These native libraries are available to apps through the Java API framework. SQLite is one of the many important native libraries available, which is the database engine that provides a relational database management system (Android Developers, n.d.-j; Google Developers Training Team, 2018, Poudel, 2013).

The Java API framework provides, through APIs, the entire feature-set of Android (user interface (UI) components, resource management, and lifecycle management). These APIs form the building blocks needed to create Android apps. This architecture is designed to simplify code reuse, modular system components and services, like display custom alerts in the status bar, build the app's UI, manage app's lifecycle and enable apps to access data from other apps, such as the Contacts app, or to share their own data (Android Developers, n.d.-j; Google Developers Training Team, 2018).

The System Apps corresponds to the set of core apps of the Android OS, such as, messaging, calendars, contacts, Google Maps, internet browsing and others. These apps work both as apps for users to use and to provide features that developers can access from their own app. This means that there is no need for the developer to create a feature from scratch, just needs to invoke an application that has that feature built in. For example, if the application under development needs the photo capture feature, is just needed to invoke the default Android camera app (Android Developers, n.d.-j; Google Developers Training Team, 2018).

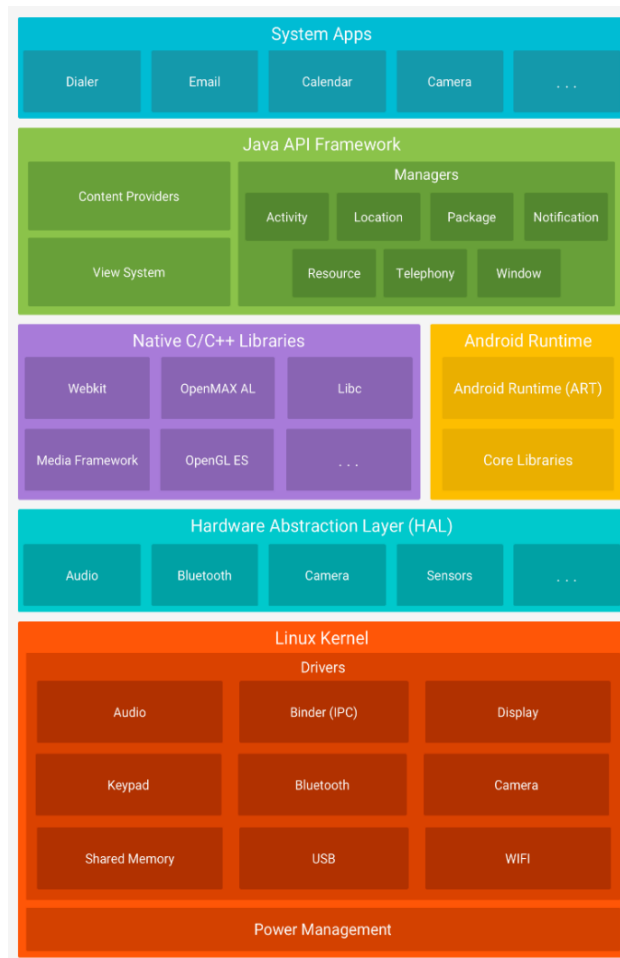


Figure 2.2. The Android software stack (Android Developers, n.d.-j).

2.2.2 Programming Languages

Android is the most used operating system in the world, and Android applications can be written in several different languages, such as Kotlin, Java and C++. To develop apps using the software development kit (SDK), the Java programming language may be used for developing the app, and eXtensible Markup Language (XML) for describing data resources (Google Developers Training Team, 2018; Android Developers, n.d.-d).

2.2.2.1 Java

Java is general purpose, simple, portable, distributed, robust, secure, dynamic, architecture neutral, class-based computer programming language. Basic core Java can be used in order to develop the source code of an Android application. The main reasons for choosing Java as a native programming language for Android applications are because Java is easy to understand and learn, is platform-independent and secure, and object-oriented (Board of Intermediate Education Andhra Pradesh, 2017; Google Developer Training Team, 2018; McKenzie, 2019).

2.2.2.2 XML

XML is a simple, scalable, and very flexible text format that is both human- and machine-readable. It defines a set of rules for encoding hierarchically organized documents that emphasize simplicity, consistency, generality and straightforward usability over the Internet. In Android, XML is used for designing UI layouts and other elements, and since it is a lightweight language, doesn't make layouts heavy ("Linked Heritage Glossary - XML", n.d.; Quin, 2016; W3C, 2008; Saini, n.d.)

2.2.3 **Android Fundamentals**

The structure in which new projects/applications are built in Android allows the separation of the work into small conceptual units, so the developer can work on them independently and then assemble them as a complete package. Following are described some of the most relevant components used to build Android applications.

2.2.3.1 Activity

The entry point for interacting with the user is designated as Activity. An activity is nothing but a Java class in Android (`Activity` class) that provides the window in which the app draws its user interface (UI), and has some pre-defined functions that are triggered at different app states, which can be overridden to perform the desired. It is also where the developer controls all UI elements in order to interact with the user, e.g. by defining the behaviour of a button or a link (Android Developers, n.d.-d).

An application consists in one or more activities, and each activity is independent of the others, enabling multiple entry points for an application. All classes that inherit from the `Activity` class must implement the method `onCreate(Bundle)`. This method is where an activity is initialized and where the method `setContentView(int)` is called, whose parameter is a layout resource id that defines the activity's UI. It is also where the method `findViewById(int)` is called to retrieve the elements in that UI with which the developer needs to interact programmatically in order to have a specific behaviour. All the activities must be declared with the tag `<activity>` in the `AndroidManifest.xml` file so the system can handle requests to start activities (Android Developers, n.d.-a; Android Developers, n.d.-d).

As the user navigates through (between activities), out of, and back to the app, or even when device configurations change (device rotations from portrait to landscape), each activity moves between states in a lifecycle. An activity's lifecycle is a set of states to which an activity migrates during its entire lifetime, starting when it is first created and ending when it is destroyed. The diagram in Figure 2.3 summarizes an activity's lifecycle, in which the coloured ovals are the major states an Activity can be in and the square rectangles represent callback methods that can be implemented to perform operations when the Activity moves between states (Android Developers, n.d.-n; Google Developers Training Team, 2018).

Each stage in an activity's lifecycle has its corresponding callback method (`onCreate()`, `onStart()`, `onPause()`, and so on – please see Figure 2.3), and, whenever an activity changes state, the associated callback method is invoked. It is possible to change the default behaviour of an activity in response to different users or system actions by overriding any of the lifecycle callback methods (Android Developers, n.d.-n; Google Developers Training Team, 2018).

All the activities that the user interacts with, when performing a certain task, are arranged in a stack (the “back stack”), in the order in which each activity is opened (Android Developers, n.d.-m).

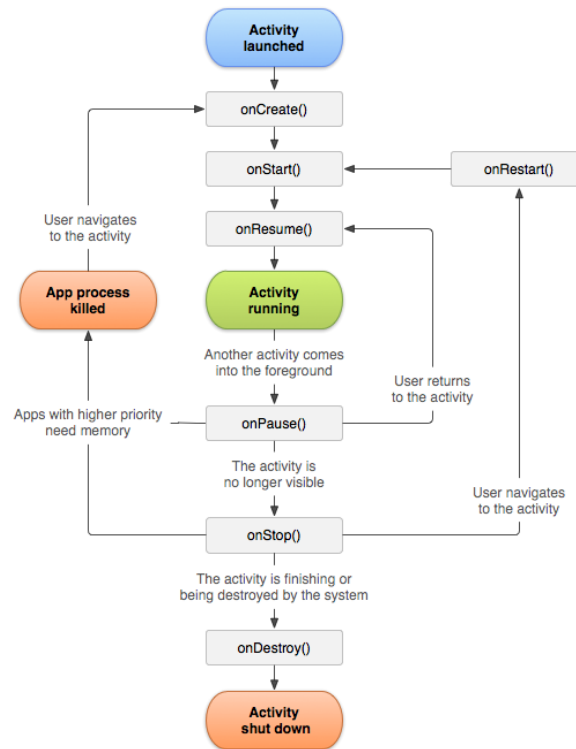


Figure 2.3. Simplified illustration of the activity lifecycle (Android Developers, n.d.-n).

2.2.3.2 Layout, View and ViewGroup

The visual structure for a UI is defined by a layout. All elements in a layout are built according to a hierarchy of `View` and `ViewGroup` objects (Android Developers, n.d.-g).

A `View` occupies a rectangle area on the screen that holds the UI elements that the user can see and interact with (buttons, links, text, etc). `View` objects are commonly referred to as *widgets* and include several classes such as buttons (`Button` class), editable text boxes (`EditText` class), images (`ImageView` class), among others (Android Developers, n.d.-o).

A `ViewGroup` is an invisible container that defines the layout structure of a `View` and other `ViewGroup` objects. `ViewGroup` objects are commonly referred to as *layouts* and include several types that provide different layout structures, such as `LinearLayout` or `ConstraintLayout` (Android Developers, n.d.-p).

There are two ways to declare a layout: 1) Declaring UI elements in an XML file, using a straightforward XML vocabulary that corresponds to widgets and layouts. It is also possible to declare UI elements with a Layout Editor to build the XML layout using a drag-and-drop interface; 2) Instantiate layout elements at runtime, by creating and manipulate widgets and layouts programmatically (Android Developers, n.d.-g).

2.2.3.3 Intent

The `Intent` class represents message objects that make a request to the Android runtime to start an Activity, triggering actions or events. To start activities, it is necessary to build the intent and call the `startActivity()` method (Google Developers Training Team, 2018).

In addition to starting activities, intents are also used to pass data between activities. The Activity that creates the `Intent` can pass data under the form of key/value to other Activity using the `putExtra()` method, and the other Activity just needs to call `getIntent().getExtras()` method to access the data (Google Developers Training Team, 2018; Silva, 2013).

2.2.3.4 Fragment

A `Fragment` (`Fragment` class) represents a portion of UI, i.e., it is a modular section of an Activity, which has its own lifecycle, receives its own input data, and can be added or removed while the Activity is running (Android Developers, n.d.-f).

Fragments are a combination of an XML layout file and a Java class much like an Activity. When a `Fragment` is added as part of an Activity's layout, it lives in a `ViewGroup` inside the Activity's view hierarchy. The `Fragment` itself defines its own view layout just like an activity, with buttons, editTexts, imageViews, etc. (Android Developers, n.d.-f).

Fragments must always be hosted in an Activity and even though they have their own lifecycle, they are also affected by the host Activity's lifecycle, e.g. if the Activity is destroyed, so are all fragments. Every time a `Fragment` is called, it is added to the host Activity, and if another `Fragment` is called, that first `Fragment` is removed and replaced with the new `Fragment` and so on (Android Developers, n.d.-f).

Usually, fragments are created when there are multiple UI with common items. If there is the need to "copy" multiple items over different user interfaces, it is desirable to create only one Activity that holds all duplicated elements and simply combine multiple fragments in that activity, with their own and different user interfaces.

2.2.3.5 Manifest File

The manifest file (`AndroidManifest.xml`) is the foundation of any Android application, so, it must be at the root of the app project directory. All the app's components, such as all activities, must be declared in this file so that the system knows that this component exists before it can be started (Android Developers, n.d.-a; Android Developers, n.d.-c; Android Developers, n.d.-d).

Also, Android apps must request permission to access sensitive user data (such as contacts and messages), as well as system features (such as Internet and Global Positioning System (GPS)), in order to protect the privacy of the users. So, any user permission required by the application must also be declared in this file, as well as the minimum API level required by the app, based on which APIs the app uses. Any hardware and software features used or required by the app, such as a camera or Bluetooth services, as well as API libraries to which the app needs to be linked against, such as Google Maps libraries, must also be declared in the manifest file (Android Developers, n.d.-c; Android Developers, n.d.-i; Android Developers, n.d.-d).

The `AndroidManifest.xml` file provides essential information about the application to the Android system. The system must have this information before executing any application code (Android Developers, n.d.-c; Android Developers, n.d.-d).

2.2.3.6 Application Resources

Defining resources to access in an app is an essential part of Android development. These resources are separate from the source code and may include images, colours, layouts, menus, string values, and more (Android Developers, n.d.-d).

One important aspect of providing resources separate from the source code is the ability to provide alternative resources for different device configurations, such as device language (translations), screen sizes or orientation (Android Developers, n.d.-d).

2.2.4 **Android Studio**

Android Studio is the official integrated development environment (IDE) for Google's Android OS, designed specifically for developing Android software (Google Developers Training Team, 2018; Walter, 2018).

The Android Studio provides a Gradle-based build system, software libraries of prewritten code, a debugger, a device emulator, code templates and tutorials to support application development within the Android operating system. Developers also have at their disposal a code editor, which assists them with writing code and offering code completion, refraction, and analysis (Google Developers Training Team, 2018; Walter, 2018).

Every project in Android Studio contains everything that defines the workspace for an app, from source code and features, to test code and build settings. When a new project starts, its files are shown in the following groups (Figure 2.4) (Android Developers, n.d.-k):

- manifest: Contains the AndroidManifest.xml file, which describes the nature of the application and each of its components.
- java: Contains the Java source code files.
- res: Contains all non-code features such as drawable, XML layouts, UI strings, and bitmap images.

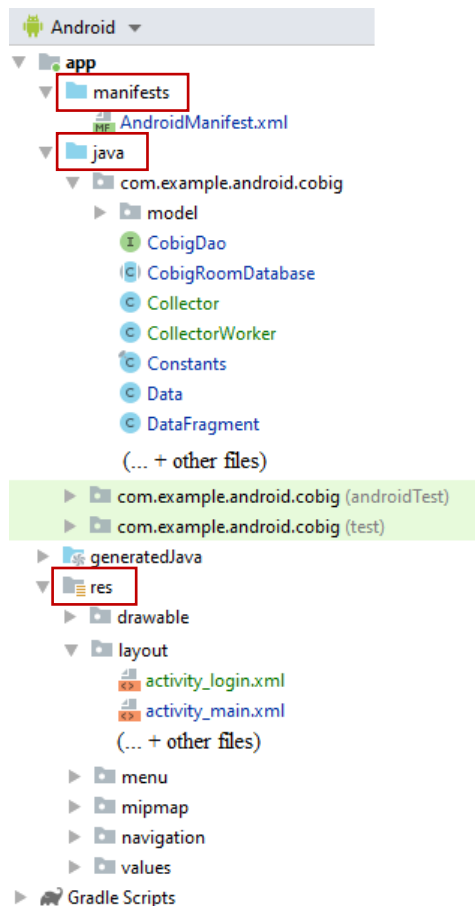


Figure 2.4. Android Project Structure.

2.2.4.1 Android SDK

The Android SDK comes along with Android Studio and provides rich tools and APIs needed to build, test and debug applications for Google’s Android platform. This kit includes required libraries to build applications, a debugger, an emulator, relevant documentation for Android APIs, sample projects with source code and tutorials for the Android OS (Google Developers Training Team, 2018; Techopedia, n.d.)

2.2.4.2 Android Emulator

An Android emulator (Figure 2.5) is a virtual Android device running on the computer that simulates all the hardware and software features of a typical mobile device. It is used by developers so they can test and debug Android applications on a variety of devices and Android API levels without using each physical devices. Along with the Android emulator, the Android Virtual Device (AVD) is a configuration which defines the characteristics of the device that will be simulated in the emulator. Such characteristics include the Android platform that will run on the emulator, as well as the hardware options and emulator skin files to be used. The AVD Manager is a graphical interface in which a developer can model and manage different AVDs that are required by the Android emulator (Android Developers, n.d.-l; Android Developers, n.d.-e; Linuxtopia, n.d.).

The emulator provides almost all of the capacities of a real Android device, such as simulate incoming phone calls and text messages, access the network, play audio and video, store and retrieve data, specify

the device's location, simulate rotation, access the Google Play Store, and much more (Android Developers, n.d.-l; Android Developers, n.d.-e; Linuxtopia, n.d.).

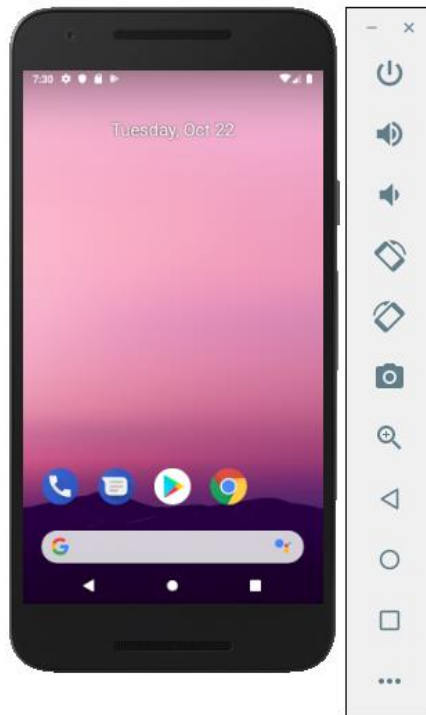


Figure 2.5. Android Emulator.

2.2.4.3 Android Package (APK)

APK is the file format for applications built for the Android operating system. Once an app is built, the Android SDK tools compile all the code along with any data and resource files into an APK. This file will be used by Android-powered devices in order to install the app (TechTarget, 2018; Android Developers, n.d.-d).

To release an application on Google Play Store, the official marketplace for Android applications, developers must compile their application into the APK format prior to uploading it. Android applications can also be distributed directly to other Android users by installing the APK files on their devices. Android users may wish to install an APK directly if they are beta testing an unreleased version of an app, or due to a device restriction that makes impossible to download the app from the Google Play, or even when it is not intended to make an app public for anyone to install and use. For this purpose, Android users need to grant permission to their device to install unknown apps if they wish to access APK files from another source and install them directly (TechTarget, 2018; Android Developers, n.d.-d).

2.2.5 **Network Operations in Android**

Most network-connected Android apps use the Hypertext Transfer Protocol (HTTP) to send and receive data to/from a Web server. The Android platform provides the Volley library, which is an HTTP library that makes networking easier and faster for Android apps (Android Developers, n.d.-q).

HTTP requests can be sent via PHP using Volley, in order to fetch results from a database and returned them to the app, or send data from the app to the server.

To store and exchange structure data over a network connection, applications use a format known as JavaScript Object Notation (JSON) (Poudel, 2013).

2.2.5.1 JSON

JSON is a lightweight data-interchange format, whose syntax is derived from JavaScript object notation to represent data. It is easy for an application developer to read and write, since it uses conventions that are familiar to programmers of the C-family of languages (including C, Java, and many others), and easy to Android devices to parse and generate.

JSON is built on two structures: a collection of name/value pairs, and an ordered list of values (array), (`{“name”:“value”}`). It is a text format usually used to interchange data between a server and a client (Crockford, 2018; Duckett, 2014).

2.3 Website Fundamentals

Websites often represent a connecting point for multiple modes of communication: one to many (text, images and sounds posted on the Website); one to one (emails or newsletters), and many to many (discussion forums or blogs) (Gurâu & Duquesnois, 2011).

A Website is a collection of different Web pages which are running under one domain name. There are two types of Website, static and dynamic. A static Website is the simplest form of a Website. It has a fixed number of pages and its content is delivered consistently to all end users (does not change in response to user interactivity, since it cannot gather information from the user or serve content dependent on user actions). Static Websites are usually developed using HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and simple JavaScript driven features. “Compared to static Websites, which are purely informational, a dynamic Website is more functional” (WP Amelia Staff, 2019) (McMahon, Seaman & Buckingham, 2011; WP Amelia Staff, 2019). Dynamic Websites allow users to interact with the content listed on the page, which changes depending on the viewer, the time of the day, the viewer’s native language, among other factors. While static Websites only use client-side HTML and CSS code, dynamic Websites rely on both client-side and server-side scripting languages, such as JavaScript and PHP, to provide advanced interactivity, and usually use a database to deliver the content for individual pages. Client-side scripting refers to code that is executed by the browser, whereas server-side scripting refers to code that is executed by the server (McMahon et al., 2011; WP Amelia Staff, 2019).

2.3.1 Front-end Web development

The first layer a user encounters and interacts directly when accessing a Website is termed as the frontend (Citrus7, n.d.; GeeksforGeeks, n.d.). “It is the user interaction interface, the way content is presented, the structure of information and the application of design to a Web page” (Edit, 2019). Front-end is developed using HTML, CSS and JavaScript. These three languages are essential when it comes to front-end Web development, in which HTML is used to define the content and structure of Web pages, CSS is used to define a Web page’s appearance and JavaScript is used to program the functionality/behaviour of Web pages (GeeksforGeeks, n.d.).

2.3.1.1 HTML

“HTML is the most basic building block of the Web” and is used to create Web pages, defining the meaning and structure of their contents (MDN Web docs, n.d.).

HTML code consists of characters that are placed inside angled brackets, called HTML elements. Elements are usually made up of two tags: an opening tag (<) and a closing tag with an extra forward slash in it (>). The content that sits between an HTML element’s opening and closing tag is viewed through a browser that reads and translates this information into a visible form (Duckett, 2011).

2.3.1.2 CSS

CSS is used to control the display format of an HTML document, describing how the HTML elements are to be displayed on the screen. It also defines a Web page style, including the design (colours, fonts, font sizes, etc.), layout and variations in display for different devices and screen sizes (Duckett, 2014; W3School, n.d.-a, W3School, n.d.-b).

2.3.1.3 JavaScript

JavaScript is used to make Web pages more interactive, interesting, and user-friendly by accessing and modifying the Web page’s content while it is being viewed in the browser, and responding to what the user does. With JavaScript is possible to access content - access any element, attribute or text from an HTML page-, modify content - add or remove elements, attributes and text to a Web page-, and react to events - specify that a script should run when a specific event has occurred (Duckett, 2014).

2.3.1.4 Ajax

Ajax stands for Asynchronous JavaScript and XML and is a client-side script whose function is to request information from a Web server, loading the data to a part of a page, without the need for a post back or a complete page refresh (Segue Technologies, 2013).

When using Ajax, the browser doesn’t wait for the server to respond. Once the server finished responding to the request (with data – usually HTML, XML, or JSON), the browser fires an event. This event can be used to trigger JavaScript functions that will process the content and incorporate it into one part of the page without affecting the rest of the page (Duckett, 2014)

2.3.1.5 jQuery

The jQuery library is a JavaScript file that can be included in Web pages and used to find elements using CSS-style selectors. jQuery methods are applied to manipulate an element and do something with it. jQuery offers a simple way to perform common JavaScript tasks quickly and consistently. It also simplifies the creation of Ajax requests and the processing of data returned by the server (Duckett, 2014).

2.3.2 **Back-end Web development**

A well-designed front end is much more useful if it has the proper backend support. (Arsenault, 2017). The back end of a Website comprises a server, an application, and a database. For these components to communicate with each other, server-side languages are used to build the application,

and tools, like MySQL, are used to find, store or change data and serve it back to the user in front-end code. In order to ensure a consistent performance, all of the calculations and database interactions are handled by the backend of an application (Arsenault, 2017; Wales, 2014).

2.3.2.1 PHP

Web developers use PHP to create dynamic content that interacts with databases, making Web pages more interactive. PHP code runs on the server and can be integrated with HTML, CSS or JavaScript code to be executed on the client. PHP offers many features designed for Websites, including the following (Valade, 2004; php, 2019):

- Communicate with databases: PHP can interact with databases to store information from the user and use it to query the database, (e.g. the user submits data to be stored in the database, through an HTML form) or retrieve information that is displayed to the user.
- Generate secure Webpages: PHP allows to create secure Web pages that require users to enter a valid username and password before seeing the Web page content.

2.3.2.2 Apache

Apache is an open-source and free Web server software that allows Website owners to serve content on the Web. It is one of the oldest and most reliable Web servers (Gediminas, 2019; Valade, 2009).

The main goal of a Web server is to serve Websites on the Internet. To achieve this, a Web server acts as a middleman between the server and client machines, pulling data from the server on each user request and delivers it to the Web (Gediminas, 2019).

Web servers support the processing files written in different programming languages, such as PHP, Java, and others, turning them into static HTML files and serving them in the Web users' browser. Basically, a Web server can be seen as the tool responsible for the proper server-client communication (Gediminas, 2019).

Apache powers around 44% of Websites around the world because of its reliable and stable software, security, flexibility, easy to configure, beginner-friendly, cross-platform (runs on a wide variety of operating systems), huge community and easily available support and, as already mentioned, because it is open-source and free (Gediminas, 2019; Valade, 2009; W3Techs, 2019).

2.3.2.3 Docker

Docker is a tool designed to simplify the creation, deployment and execution of applications using containers (OpenSource, n.d.).

Containers allow a developer to package up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. As a result, the developer ensures that the application will run on any other Linux machine, despite of any customized settings the machine might have that could differ from the machine used to write and test the code (OpenSource, n.d.; Docker, n.d.).

A Docker container image is a software package that includes everything needed to run an application: code, runtime, system tools, system libraries and settings (Docker, n.d.).

In part, Docker, resembles a virtual machine. However, instead of creating an entire virtual operating system, Docker allows applications to use the same Linux kernel as the system that they are running on, and only requires application to be shipped with things that are not already running on the host computer (OpenSource, n.d.; Docker, n.d.).

2.4 Computer Applications Architecture

Applications architecture refers to the design of how multiple software processes cooperate to carry out their tasks (Damas, 2017; Jenkov, 2014-b).

There are several types of architectures, however, some architectural patterns occur more commonly than others. Some of the common software architecture patterns are single process, client/server (two processes collaborating) and 3 Tier systems (three processes collaborating in chains) (Damas, 2017; Jenkov, 2014-b).

A **single process** or a **1-tier architecture** is a software system that consists of a single running process. Single processes applications are also referred to as standalone applications - command line programs are a common example of this type of systems (Jenkov, 2014-b).

When a user is browsing the Internet, is typically using a Web browser software such as Google Chrome or Mozilla Firefox. The computer, which is running a browser, is called the *client*, whereas the machine, which provides the Web pages, is called the *server*. When the Web server is connected to one or more clients is known as a **client-server** or **2 tier architecture model** (Damas, 2017; Jenkov, 2014-b; Sheffield Hallam University, n.d.).

Computer applications that consists in three different and distinct components are based on a **3-tier architecture**, which is a very common architecture. This architecture is typically divided into a presentation or graphical user interface (GUI) tier, an application logic tier, and a data tier. The presentation or GUI tier contains the UI of the application and just forwards the user's actions to the application logic tier. The application logic tier is responsible for all the application decisions and for read and store the data in the data tier. The data tier stores the data used in the application and consists of a DBMS that is responsible for securely storing data, performing transactions and quickly searching through large amounts of data, etc. The diagram in Figure 2.6 illustrates a 3-tier architecture (Jenkov, 2014-b; Damas, 2017; Sheffield Hallam University, n.d.).

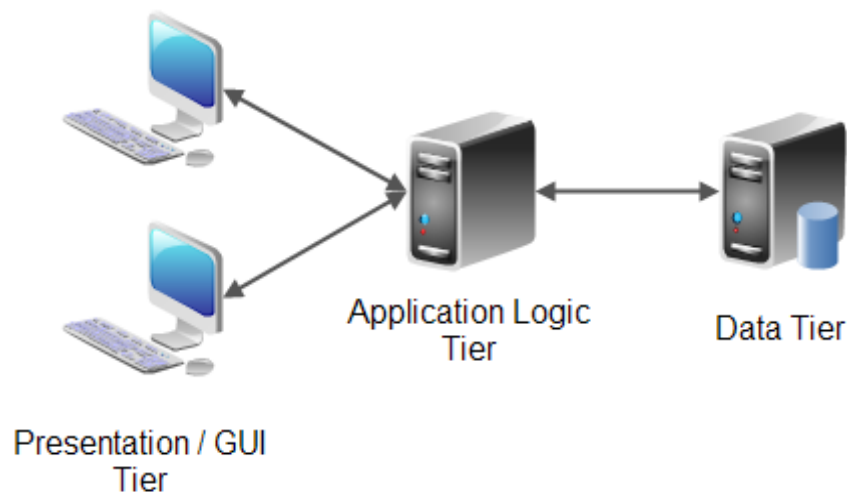


Figure 2.6. Representation of the 3-tier client-server model (Jenkov, 2014-a).

Web applications usually adopt the 3-tier architecture, in which the presentation tier consist of HTML, CSS and JavaScript; the application logic tier runs on a Web server in the form of PHP; and the data tier consists of a database, e.g. MySQL. Mobile applications that are not standalone applications follow the same principle, since a mobile application that connects to a server typically connects to a Web server and sends and receives data (Jenkov, 2014-b).

2.5 Client-Server Communication

The basic protocol for encoding and transporting information across the Internet between a client and Web server is the HTTP (NGINX, n.d.).

HTTP works as a request-response protocol and the information is exchanged between clients and servers in the form of Hypertext documents - the client sends an HTTP request to the server, and the server returns a response to the client, which contains relevant status information about the request, as well as the requested content (if requested) (NGINX, n.d.; W3Schools, n.d.-c).

The most common HTTP methods used for a client to communicate with the server are GET and POST (TutorialRepublic, n.d.-a, W3Schools, n.d.-c).

The GET method is used to request data from a specified resource. The data is sent as uniform resource locator's (URL) parameters in the form of name/value pairs separated by ampersands (&) – example: <http://www.example.com/action.php?name=john&age=24>. In PHP, the super global variable `$_GET` gives access to all the information (specified in the URL) sent through the request (TutorialRepublic, n.d.-a; W3Schools, n.d.-c).

The POST method is used to send data to a server in order to create or update a resource, and unlike the GET method, the data sent through POST method is not visible in the URL. In PHP, just like the variable `$_GET`, there is also the super global variable `$_POST`, which gives access to all the information sent by the request (TutorialRepublic, n.d.-a; W3Schools, n.d.-c).

To communicate with the server, an application can use PHP in order to perform HTTP requests. Data returned by the HTTP request comes either as JSON or XML for parsing and/or processing by the application (Adawale, 2018).

2.6 Maps for Computer Applications

A map is a graphical representation or scale model of a geographical area, which contains geographic information such as orientation and geographic coordinates. Maps are an important source of information, easily understood and appreciated by most people regardless of their language or culture, and allow to develop various human activities based on the data they provide (Conceito de, 2012; Silva, 2013).

Nowadays, with the technological advancement and the society leading to a technological modernization, there is a growing bet on the development of computer applications that support maps in electronic format.

Digital maps are available for both mobile and Web applications in order to aid navigation, being able to take advantage of the user's current location, or visualization of geospatial data.

There are several available platforms that provide APIs for custom designed maps for display on computer applications. However, the Google Maps API has been the ideal choice for most of developers for years because of Google Maps' expansive database of geographical features, small businesses, and street images across the globe. Google Maps is the Google's official geographic information and map services, and is a favourite for its geolocation services worldwide (Bush, 2018).

Google Maps APIs are available for a wide variety of use cases. Google APIs usually come along with complimentary usage limits, although, there are some that are completely free, such as the Google Maps API for Android. In 2018, Google severely hiked the rates for their maps API, so developers had to find more sustainable alternatives in order to integrate map services into their applications (Google Cloud, n.d.).

In the following subsections, map services for both Android and Web, and mobile location technologies are described.

2.6.1 Mobile Applications Maps

The Google Maps API for Android is one of the few that has no usage limits. With this API, it is possible to add maps based on Google Maps to an application. Access to Google Maps servers is handled automatically by the API, as well as data downloading, map display, and response to map gestures. It is also possible to use API calls to add markers to the map (Android Developers, n.d.-h).

A much-requested feature in mobile map applications is the use of *offline* maps when there is no Internet connection. To deal with *offline* maps, an application that uses the API can access *offline* maps previously downloaded from the Google Maps system app.

2.6.2 Web Applications Maps

The Google Maps API for Websites is the Maps JavaScript API. This is one of the APIs that has free traffic limits, and which, after redesigning the pay-per-use pricing plan, has become untenable for most small developers to use due to excessive costs. Alternatively, there are several platforms available to provide custom APIs for displaying maps on Web pages at a lower or even free cost, such as OpenLayers, TomTom, Mapbox, HERE, or Mapfit (Bush, 2018).

2.6.3 Mobile Location Technologies

Generally, map applications come along with user's location. Thus, in order to serve the objectives of this dissertation, the main location technologies present in most Smartphones were analysed.

2.6.3.1 GPS

The GPS is an earth-orbiting-satellite based navigation system that provides geolocation and time information to GPS receivers (e.g. smartphones with GPS). It is the most referenced and used system in location-based services. As it is more accurate and more precise as compared to other location technologies, GPS technology is preferably used for the majority of mobile applications for smartphones for calculating the user's geographical position (Silva, 2013; Dana, 1997).

The limitations of GPS, when compared to other alternative technologies, are the high battery consumption and signal loss in indoor or urban environments with poor satellite signal (Silva, 2013).

2.6.3.2 Wi-fi (WPS)

Wireless Positioning System (WPS) is the technology most frequently used as a location alternative when GPS is not available. This system takes advantage of public Wi-fi hotspots and is used in environments where GPS signal loss is frequent, such as indoors or urban tall buildings. This technology has a lower battery consumption than GPS (Silva, 2013).

2.6.3.3 Cell and Cell Tower

The Cell and Cell Tower uses the smartphones' antenna and the cellular networks of mobile operators to locate the smartphone's position. This technology is better at battery efficiency than WPS (Silva, 2013).

2.7 Similar Applications

There are mobile applications available in the market that similarly serve some of the objectives of the application developed in this project. However, these applications do not have the degree of specificity or meet the needs required by the CoBiG2 group, and so they were not even considered. Following are briefly described some of the applications available on the market with features similar to those developed in this project:

- **Qfield** is an application that works with GPS tracking, with all *offline* features, synchronization capability and cartography visualization (QField, n.d.; GooglePlay, n.d.-b);
- Regarding georeferencing, **GoogleMaps** works with location technologies and allows to show user's current location, as well as to place markers at that location or other places of interest;
- **Epicollect5** is a mobile and Web application for data collection hosted by Imperial College London. It provides the generation of forms (questionnaires) and freely hosted project Websites for data collection (including GPS and media) multiple devices. All data can be viewed on a central server (EpiCollect5, n.d.; GooglePlay, n.d.-a).

The main problem with applications like EpiCollect5 is that they use a general server to store the data and they turn out to be too general (in questionnaires) in order to serve a wider community.

Part III

Chapter 3 - Methodology and Implementation

This chapter describes the system architecture, the database creation process, the mobile application and the Web application development.

3.1 System Requirements

The developed system must meet a set of requirements and have certain functionalities.

The first general requirement of the system allows the user to mark locations on a map, and collect data for those locations, using a mobile device.

The second requirement is to provide the user with a platform where they can visualize all existing collection locations on a map, and information about them.

The third requirement is to create a set of features that allow the user to add and edit information.

The fourth requirement is to restrict system accessibility and control access to applications.

Applications-specific requirements are described in more detail in the following sections.

3.2 System Architecture

Figure 3.1 shows the architecture of the developed system consisting of 3 main components: the mobile and Web applications (presentation tier), PHP scripting language (logic tier) and the database (data tier).

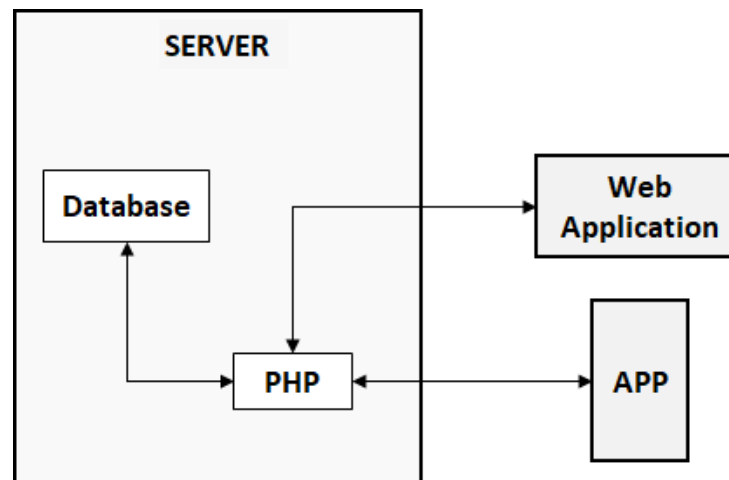


Figure 3.1. System Architecture.

PHP is responsible for all the applications' decisions and for read and store the data in the database.

The mobile application starts with the request of information essential to its correct functioning. After the information is requested from the server, a map is provided to which the user can add places and later collect information about those places. After the user intends to send the data to be stored in the database, a request is made to the server for the data to be effectively stored in the database. The Web application requests the server to retrieve location data and displays it on a map using markers. By selecting a marker the user can view the data collected (using the mobile app) for that marker. The Web application will contain features that allow users to manage information from both markers and markers'

collected data. Whenever data is changed in the Web application, the information is sent to the server and the data is updated in the database.

3.3 Database Development and Implementation

3.3.1 Planning

The structure in which the data collected in the field is organized is essential for achieving the project objectives.

The planning phase involved the end users (CoBiG2 members), so that they could suggest ideas, concepts and functionalities, as well as describe the data contained in the database, define entities (tables), relationships between entities and data constraints (Damas, 2017).

It was defined that the construction of the database aims to store georeferenced descriptive species' biological data of samples collected in the field, by the end users.

These data, concerning data collection locations, are linked to the different research projects carried out by the CoBiG2 group. Each project may have associated several locations and respective geolocalization variables (latitude, longitude, altitude, country, region, location), where data are collected for different species. The nature of the data collected has underlying variables such as species, species' sex, number of individuals, collector, any sample observations, among others.

Considering the above needs, the respective entities were defined: User, Project, Marker (locations), Data and Species, where users are assigned to projects, projects have markers, and markers and species have associated different data, although a single data must have associated only one marker and one species.

It was established that the access to mobile and Web applications is restricted, i.e. only desirable users will be able to take advantage of system features. For this purpose, an authentication system was created, assuming user's registration to access applications, followed by system administrators' validation. Regarding access, different user profiles were defined, namely the normal user profile and the administrator profile. Normal users can only add, remove and update data collected in the field. The administrator, in addition to the normal access profile, also has privileges to manage information about projects, species, users and projects, as well as validate new users' registrations. For this purpose, the Validation entity was defined.

3.3.2 Analysis

In the analysis phase, the problems detected in the planning phase were studied in more detail in order to define the exact scope that the system should embark on and to analyse and define the requirements to be implemented (Damas, 2017).

This phase was initially devoted to dialogue with end users, with the aim to meet their needs and expectations, identify processes involved and potential areas of application. Additionally, an evaluation of the available software and hardware was made to understand how they can be used and integrated in the system (Damas, 2017).

The analysis phase also included the creation of the logical design of the system, or database *schema*, which gives an overall description of the database. The database *schema* can be represented using an E-R diagram, which represents how data should be structure and related (Damas, 2017).

At a later stage, the final and detailed database *schema* was completed regarding all the information gathered above and by specifying primary keys, foreign keys and other data constraints, considering a specific DBMS. Any specifications for hardware, software, involved operating systems and network

and communications were also considered (Damas, 2017). The E-R diagram is illustrated in Figure 3.2. The chosen database management system was MySQL, because of its proven performance, reliability, and easy-of-use.

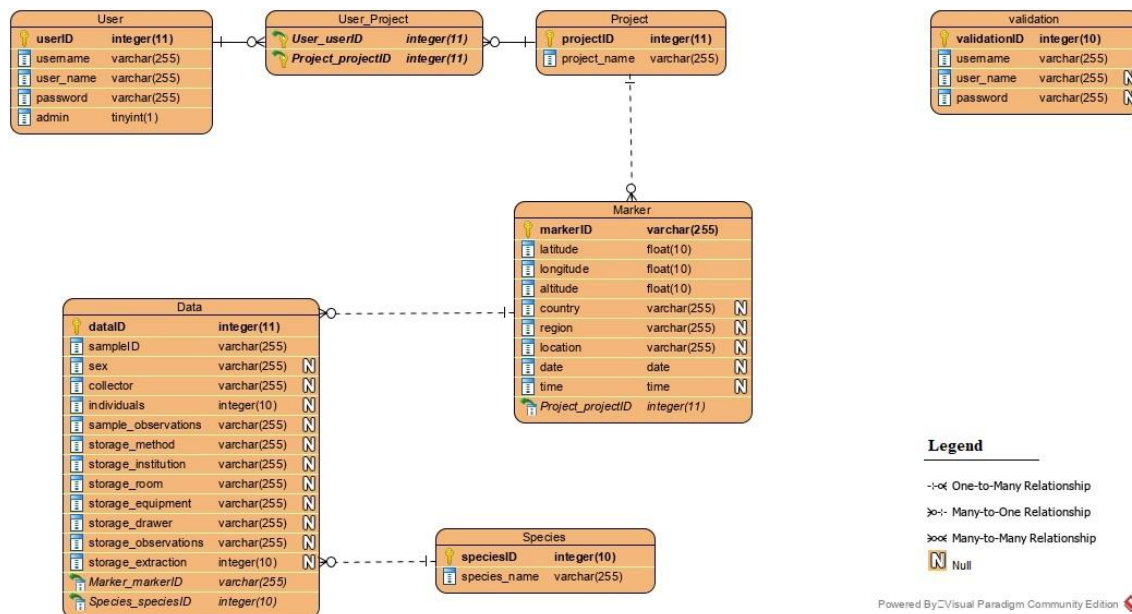


Figure 3.2. E-R diagram - Database schema.

The database consists of 6 entities with specific functions:

- **User table**: contains registered users' details such as unique identifier (ID), name, username, and hashed and salted password.
- **Project table**: contains the ID and name of research projects, in which is possible to associate research collection locations.
- **User/Project table**: this table refers to projects to which users are assigned. A user can be part of multiple projects and one project can be assigned to multiple users.
- **Marker table**: contains the locations where data were collected in the context of a research project. A project can have multiple markers. However, a marker is unique to a project.
- **Species table**: contains the name of different species that may be the study object of a research project.
- **Data table**: contains data added to collection locations. Users can add multiple data to a marker. However, each data can only belong to a single marker. Data generally refer to individuals of a species that were collected at such collection locations. Each data concerns only one species.
- **Validation table**: holds new users' registrations for confirmation. If a registration is accepted, the user's information is copied to the User table and erased from this table. If a registration is denied, it is erased from this table.

The user_project table was created because of the many-to-many relationship between the user entity and the project entity. This table contains information about which projects users are assigned to.

Attached are examples of data that each table can contain.

3.3.3 Implementation

After the planning and analysis phases, a prototype was created and implemented to give end users a rough idea of what the system will look like and its key features. This way, end users were able to

validate the system and features to implement and review the options taken (Damas, 2017). The prototype was implemented using XAMPP, which allowed to create a local Web server for testing purposes. XAMPP is a free and open-source package, developed by Apache Friends, that consists of a MariaDB database, Apache Web server, PHP and PERL (Apache Friends, n.d.). The use of a prototype provided end users with an overview of the system that was later implemented and allowed to verify if the system was useful to the organization (Damas, 2017).

During the implementation phase, the hardware and software selected for the system development were installed and configured (Damas, 2017).

The previous stage was followed by the code writing component to implement the features designed and specified in the analysis phase. At this stage, code writing, testing and debugging, and error corrections were performed (Damas, 2017).

MySQL was installed on the server and the data model designed in the previous phases was physically created using SQL language, more precisely, using DDL commands. To deal with referential integrity, the CASCADE strategy was adopted. For this purpose, phpMyAdmin was used. PhpMyAdmin is a software tool intended to handle the administration of MySQL over the Internet (phpMyAdmin, n.d.)

Also, at this stage, in order to deal with database security, sets of Login/Password were defined for all the database users, which is validated by RDBMS whenever a connection to the database is made. Their permissions were also defined using DCL commands: the set of users that can execute the DROP TABLE command, which, as previously mentioned, allows deleting complete tables from the system, was restricted.

The system was subjected to tests to verify the implementation's correctness, robustness, security, performance, etc., in order to verify if the requested features were implemented correctly or not (Damas, 2017).

3.4 Android Application Development and Implementation

The mobile application allows the user to collect georeferenced data in the field. It interacts with the other components of the system in two situations: to obtain information that is essential for the correct operation of the application on the Android terminal, and to send information collected by the user to be stored in the server database.

The 'C2MC' application was developed using Android Studio 3.4 and written in the Java programming language. Although, this application is heavily coded on Java, it profoundly relies on a huge stack of native libraries written in C and C++. XML was also used to design UI layouts and other elements, and JSON to store and exchange data over a network connection.

The minimum API level of the application was established to 22 (Android 5.2, Lollipop), which means that the application will run approximately in 80,2% of Android devices.

The application's development process involved the following steps (Google Developers Training Team, 2018):

- Define the idea of what the app should do and its requirements;
- Prototype the user interface;
- Develop and test the app:
 - Create the layout: Place UI elements on the screen in a layout, and assign string resources and menu items, using XML;
 - Write the Java code: Create source code for components and tests, and use testing and debugging tools.
 - Register activities: Declare the activities in the manifest file.

- Publish the app: Assemble the final APK and distribute it.

The application uses Android Jetpack, which is a set of components, tools, and guidelines recommended by Google that ensure developers build high quality and robust applications. Android Jetpack components help developers to follow best practices, eliminate boilerplate code, and simplify complex tasks (Android Developers, n.d.-r; Android Developers; n.d.-s).

Android Jetpack components are arranged into four categories: architecture, UI, behaviour and foundation (Figure 3.3). Some of these components such as Room, Navigation, ViewModel, among others, were used for the application's development.



Figure 3.3. Jetpack components. Adapted from (Android Developers Blog, 2018).

3.4.1 Requirements

The 'C2MC' application must meet the following main requirements:

- Authentication, to limit the access to the application by desirable users only;
- Import and update via Internet database content necessary for the correct behaviour of the application on the Android terminal;
- Detect the user's current geographical position using location technologies, and display it on the device screen;
- Provide information about collection locations depending on the research project;
- Send information entered by the user to the database on the server;
- Ensure local storage of information essential to the correct functioning of the application so that it can be used when the user is *offline*;
- Ensure that data entered by the user, when *offline*, is stored locally so that it can then be sent to the server when there is Internet connection;

3.4.2 Application Architecture

The development of an application must consider some common architectural principles. Model-View-ViewModel (MVVM) is an architectural pattern for building applications that aims to provide a clear separation of concerns, keeping the UI logic separated from the backend logic (Gupta, 2019; Microsoft, 2012).

The Model-View-ViewModel architecture consists of 3 parts (Gupta, 2019; Microsoft, 2012):

- **Model** is the data access layer of an application and it abstracts the data source.
- **View** corresponds to the structure, layout and appearance of what the user seen on the screen, i.e. it is the layer that contains the application's UI. The ViewModel is responsible for receiving the user's interaction with the UI, and forward the handling of these to the ViewModel. It also displays results received from the ViewModel. It is most often implemented as an Activity or a Fragment.
- **ViewModel** acts as a bridge between the View and Model. It works with the Model in order to get and save data. The View observes and reacts to the data changes exposed by the ViewModel.

The application architecture developed in this project follows the MVVM architecture, which is the one recommended by the Android platform. JetPack architecture components take this model into account and so they were used to structure the app. The following diagram (Figure 3.4) shows a basic form of this architecture and how Architecture Components interact with one another.

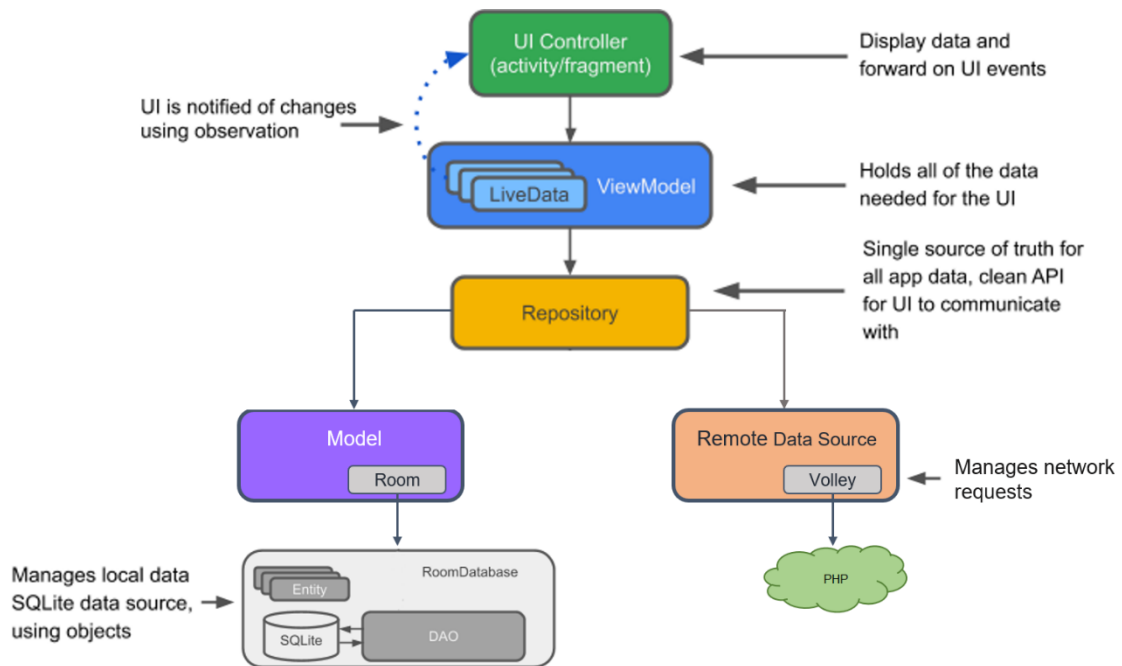


Figure 3.4. Android Application Architecture. Adapted from (Android Developers, n.d.-s; Google Codelabs, 2019-a).

Each component depends only on the component one level below it. Activities and fragments (view layer) depend only on a view model, which holds the data needed for the UI. The repository depends on a persistent data model and a remote backend data source, being the only class that depends on multiple other classes (Android Developers, n.d.-t).

This design creates a consistent and pleasant user experience. Regardless of whether the user closes the app and then comes back, they can still have access to information that the app persists locally. If this data is obsolete, the repository module starts updating the data in the background (Android Developers, n.d.-t).

Below is a brief introduction to the Architecture Components used to achieve the MVVM architecture.

The **ViewModel**'s role is to provide data to the UI as well as hold app's UI data in a lifecycle-conscious way. It acts as a communication center between the Repository and the UI, completing separating them. It is important to separate application's UI data from Activity and Fragment classes in order to better follow the single responsibility principle: activities and fragments are responsible for

drawing data to the screen, while the `ViewModel` holds and processes all data needed for the user interface (Google Codelabs, 2019-a).

The `ViewModel` uses **LiveData** for changeable data that will be used or displayed by the UI. Using `LiveData`, it is possible to place an observer on the data and only update the UI when this data actually changes. Whenever the data changes, the `onChanged()` callback method is invoked, performing what is desired (Google Codelabs, 2019-a).

There are no database calls from the `ViewModel`, which makes the code more testable. To implement a **ViewModel**, the `ViewModel` class was used in order to store and manage UI-related data in a lifecycle conscious way (Google Codelabs, 2019-a).

The **Repository** is not a Jetpack's architecture component, however using it is considered a best practice for code separation and architecture. A `Repository` class handles data operations and abstracts access to multiple data sources. A `Repository` was used in order to manage query threads and decide whether to fetch data from the remote server or use results cached in the local database (Google Codelabs, 2019-a).

In summary, the `ViewModel` hides everything about the backend from the UI layer and so `Views`, `Activities` and `Fragments` only interact with the data through the `ViewModel`, which provides methods for accessing the data layer. As such, it does not matter where the data comes from. The data comes from the `Repository`, and the `ViewModel` does not need to know what the `Repository` interacts with. "It just needs to know how to interact with the `Repository`, which is through the methods exposed by the `Repository`" (Google Codelabs, 2019-a).

In order to deal with *offline* usage, **Room** was used, which is a local database that provides an abstraction layer over `SQLite`¹. There are three major components in `Room`: **Database**, **Entity** and **Dao** (Data Access Object) (Google Codelabs, 2019-a). These components are nothing but Java classes, or an interface in the case of `Dao`. Annotations are used in order to mark components and other methods, making them meaningful to the `Room` database.

- The `Database` is a class annotated with `@Database` that extends the `RoomDatabase`. This class defines the list of entities and data access objects in the database (Google Codelabs, 2019-a; Android Developers, n.d.-u).
- An `Entity` is a Plain Old Java Object (POJO) class annotated with `@Entity`, which is going to be transformed as a table in the `Room` database. For each entity a set of attributes that makes up the table's columns is defined and are provided getter and setter methods to access these attributes. At least one attribute must be defined as a primary key with the `@PrimaryKey` annotation. Adding `@PrimaryKey(autogenerate=true)` allows `SQLite` to generate automatically a primary key when a new record is inserted into a table (Google Codelabs, 2019-a; Android Developers, n.d.-u).
- `Dao` is an interface annotated with `@Dao`. This interface is responsible for defining CRUD (create, read, update and delete) operations that access the database data. SQL queries are specified and associated with method calls. The compiler checks the SQL and generates queries from convenience annotations for common queries, such as `@Insert`, `@Delete`, `@Update` or `@Query`. `Insert`, `delete` and `update` annotations are used to insert, delete and update rows. The `@Query` annotation is used to perform custom queries by providing SQL, such as `SELECT` statements to read content from a table. When inserting data, it is possible to provide a conflict strategy to inform the behaviour the database should take when inserting a new record that already exists in the table, e.g. `@Insert(onConflict=OnConflictStrategy.REPLACE)` to replace a row. (Google Codelabs, 2019-a; Android Developers, n.d.-u).

¹ `SQLite` is a relational database management system.

Room does not allow issuing database queries on the main thread in order to avoid poor UI performance. As such, tasks are performed asynchronously using the `AsyncTask` class², meaning that they run on the background thread and whose results are published on the UI thread. `LiveData` also applies this rule by automatically executing the query asynchronously on a background thread, when needed (Google Codelabs, 2019-a).

Data fetched from the remote server, as well as new data added by the user, are stored locally in the Room database. This way, when the device cannot access the network, the user can still browse that content, and save information to be later sent to the server so it could be stored in the server database. Any user content changes are then synced to the server after the device is back *online*.

3.4.3 Android Client-Server Communication

To interact with the server database, to retrieve and submit data to the server, the Volley library was used.

To use Volley³, and therefore be able to connect to the network, the `android.permission.Internet` permission was added to the manifest file.

This project involved the use of two Volley request types, `StringRequest`⁴ and `JSONArrayRequest`⁵:

- String data HTTP POST request using Volley, in order to **send** data to the server, is performed using a `StringRequest` object. For this purpose, the HTTP method as POST is passed to the constructor, the URL is specified, and the request's `getParams()` method is overridden to add post data. A raw string is received in response (usually simple messages whether informing the user if the post was successful or not).
- Json data HTTP GET requests using Volley, in order to **request** data from the server, is performed using a `JSONArrayRequest` object. A `JSONArrayRequest` is a request for retrieving a JSONArray response body at a given URL, to which is possible to add parameters as query parameter values.

To perform Volley requests, a `RequestQueue`⁶ was create, to which Request objects are passed. This queue manages worker threads for running the network operations, reading from and writing to the cache, and parsing objects. Requests do the parsing of raw responses and Volley dispatches the parse response back to the main thread for delivery. Since this app makes use of the network in many moments, a single instance of `RequestQueue` was set up, and lasts the lifetime of the app. To achieve this, a singleton class that encapsulates `RequestQueue` was implemented. To set up a `RequestQueue`, the `getRequestQueue()` method was implemented and calls the `Volley.newRequestQueue()` method.

Once built, a request can be sent by adding it to the `RequestQueue`. To achieve this, the `addToRequestQueue(Request req)` method was created in the singleton class, and the request can be added to the queue by calling `getRequestQueue().add(req)`. Once the request moves through the pipeline, gets serviced, and has its raw response parsed and delivered.

The URL specified in the request, points to the PHP file to be reached in the server. For communications between PHP and MySQL, the `MySQLi`⁷ extension was used. This extension allows to

² <https://developer.android.com/reference/android/os/AsyncTask>

³ <https://developer.android.com/training/volley>

⁴ <https://developer.android.com/training/volley/simple.html>

⁵ <https://developer.android.com/training/volley/request>

⁶ <https://developer.android.com/training/volley/requestqueue>

⁷ <https://www.php.net/manual/en/book.mysqli.php>

open a database connection using credentials and then perform CRUD operations using SQL prepared statements. A prepared statement is a SQL query containing placeholders (?) rather than the actual parameter values (TutorialRepublic, n.d.-b).

Prepared statements execution consists of two stages: preparation and execution (php, n.d.-a; Marcus, 2017):

1. Preparation: An SQL statement template is sent to the database. The SQL statement is prepared with unspecified parameter's values as placeholders. E.g.: `INSERT INTO user values (?, ?, ?);`
2. Execution: During execute, the client binds variables or values to the placeholders by stating each variable along with its type, and sends them to the server. The server creates a statement from the statement template and the bound placeholders' values to execute it.

Prepared statements provide superior protection against SQL injection attacks, if properly implemented, because they create a separation between the submitted data and the SQL query itself, ensuring that the data cannot be misinterpreted as the SQL query. This way, escaping is not necessary since values will be treat as literals (Marcus, 2017). Escaping strings consist of not treating certain characters that are being used as part of the coding language, but understand them as part of the value of the string.

To deal with tasks that are not required to run immediately and required to run reliably even if the app exits or the device restarts, the `WorkManager` API was used. `WorkManager` is one of the Android Architecture Components and part of Android Jetpack that runs background work while taking care of compatibility issues and best practices for battery and system health. With `WorkManager` is also possible to indicate when the work should run, by adding constraints to it. The task will run only when all the constraints are met (Maggi, 2018).

`WorkManager`⁸⁹ was used to make Volley HTTP requests to a remote server on the background (i.e. that do not require immediate execution), only if the device has network connection, like when data is sent to the server.

To create a background task that runs synchronously on a background thread provided by `WorkManager`, a class that extends the `Worker` class was created, and the `doWork()` method overridden.

To configure how and when a work should be run, a `WorkRequest` was used. It is also possible to include additional information to a request¹⁰, such as the constrains under which the task should run, input to work, a delay, and back off policy for retrying the work.

In the case of adding a network constraint, a `Constraint` object was instantiated, and the network constraint was added by calling the `setRequiredNetworkType(NetworkType.CONNECTED)` method.

In order to pass data to the work as input parameter (such as the URL for the Volley request), the input values were stored as key-value pairs in a `Work's Data` object and added to the work using `setInputData()`. To access the input arguments in the `Worker` class, the `getInputData()` method was called.

Once defined, the `WorkRequest` is scheduled with `WorkManager` using the `enqueue()` method. The exact time the work is executed depends on the previously defined constraints in the `WorkRequest` and on system optimizations.

`WorkRequests` can be initialized in the UI by calling it from the `ViewModel`.

⁸ <https://codelabs.developers.google.com/codelabs/android-workmanager/>

⁹ <https://developer.android.com/topic/libraries/architecture/workmanager/basics.html>

¹⁰ <https://developer.android.com/topic/libraries/architecture/workmanager/how-to/define-work>

3.4.4 User Interface design and modelling

An important aspect to consider is how the application's graphical interface will be designed. It was decided from the outset that it should be as simple as possible.

'C2MC' contains a series of activities and fragments that serve different purposes of the application. The first step was defining the activities and fragments layouts, by placing UI elements on the screen in layouts, and assign string resources and menu items, using XML and the drag-and-drop interface.

Events, like search, edit, display or browsing, are then declared in their corresponding activities and fragments. Each event, when triggered, invokes a series of actions, results of which are then displayed on the user's screen.

The requirements of this project are met in two activities and four fragments. Following, the features of each user UI are presented.

3.4.4.1 LoginActivity

This is the first activity the user encounters (Figure 3.5), in which they are asked for their login details so they can access the app's content.

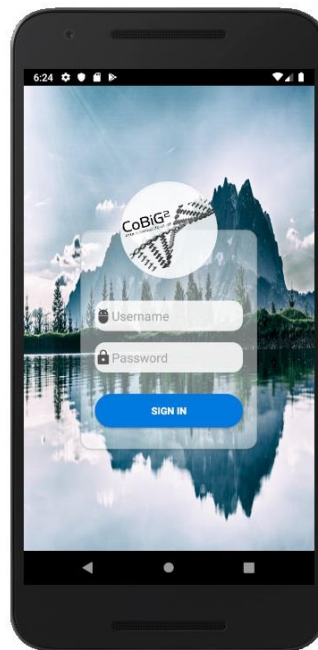


Figure 3.5. Login Activity interface.

In this activity, the user comes across a login form in which enter their username and password. This information is sent to the server via Volley POST request and the server response is returned to the application informing whether the user is allowed to access the app or not. If the login is successful, the MainActivity starts and the user can start taking advantage of all the features the app offers. If the server returns an authentication failure due to submission of incorrect login details, the user is provided with informative feedback of the failure.

The passage from one activity to another, or the initialization of another activity, is made by instantiating an Intent¹¹, whose constructor receives two parameters, a Context and a Class. The

¹¹ <https://developer.android.com/reference/android/content/Intent>

Context class allows the access to settings and resources shared between the various app screens (activities). Each activity has its own context, so the Context parameter is the activity's context where the user is in. The Class parameter is to where the system delivers the Intent, in this case, the activity to be started.

This Intent is then used as argument of the `startActivity()` method that starts an instance of MainActivity, as specified in the Intent.

The background image used in the Login Activity was taken from: <https://www.pexels.com/photo/daylight-environment-forest-idyllic-459225/>.

3.4.4.2 MainActivity

This activity (Figure 3.6) has two main purposes: 1) allocate a Bottom Navigation bar that allow users to navigate in the app, and 2) trigger the process of retrieving and locally storing content from the remote server.

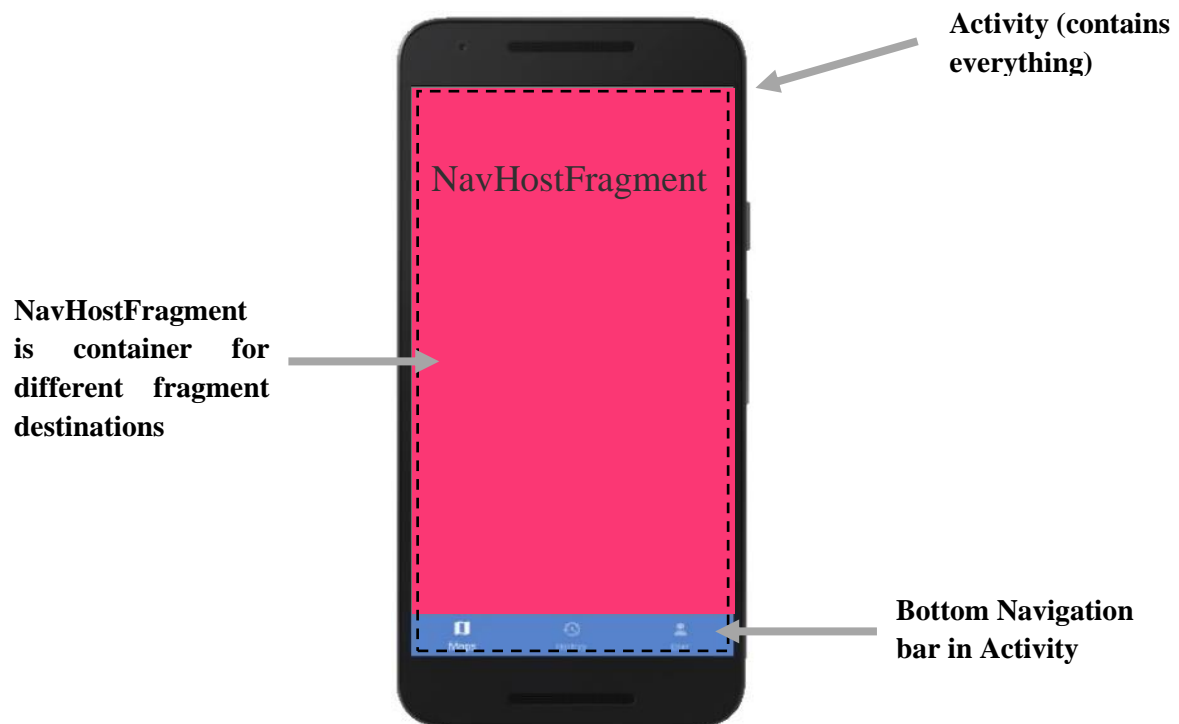


Figure 3.6. Main Activity interface. Adapted from (Google Codelabs, 2019-b).

1) Navigation through the app¹²¹³¹⁴

Navigation refers to interactions that allow users to browse across, into, and back out to different parts of the app's content. To ensure a consistent and predictable user experience regarding navigation through the app using a bottom navigation bar, the Android Jetpack's Navigation component was implemented (Android Developers, n.d.-z).

Navigation component makes the development easy by handling fragment transaction and attaching UI components to a navigation graph, and reducing boilerplate code written to wire up user actions with UI navigation (Viradiya, 2018).

¹² <https://developer.android.com/guide/navigation>

¹³ <https://developer.android.com/guide/navigation/navigation-getting-started>

¹⁴ <https://codelabs.developers.google.com/codelabs/kotlin-bootcamp-introduction/index.html?index=.%2F..index#0>

There are three main key parts that make up the Jetpack's Navigation component: navigation graph, NavHost and NavController (Android Developers, n.d.-b, Android Developers, n.d.-x; Google Codelabs, 2019-z).

The **navigation graph** (Figure 3.7) is an XML resource file that shows visually all the navigation-related information, such as all of the individual content areas within the app (activities or fragments), called *destinations*, as well as the possible paths a user can take through the app. The paths that users can take are represented via *actions*, which are logical connections between the *destinations* (connecting one destination to another). *Actions* are represented in the navigation graph as arrows. (Android Developers, n.d.-b, Android Developers, n.d.-x; Google Codelabs, 2019-z).

In order to implement the NavGraph, the <navigation> element was added to the XML file, which is the root element of a navigation graph. This element can include the app:startDestination attribute to establish the first screen users see when opening the app, and the last screen users see when exiting the app.

To add destinations and connecting actions to the graph, the corresponding <destination> and <action> elements were added as child <navigation> elements.

An <action> element is placed as a child <destination> element of the destination where an action originates and the app:destination attribute defines the destination to which a user should be taken.

The development of this application involved the creation of four *destinations* (UserFragment, MapsFragment, DataFragment and HistoryFragment) and two *actions* (between MapsFrangment and DataFragment, and HistoryFragment and DataFragment).

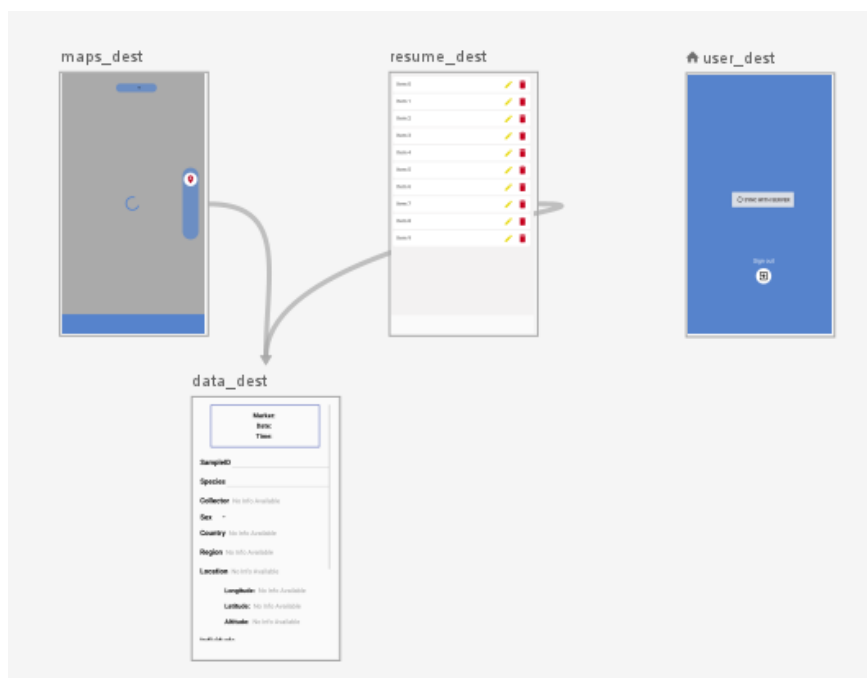


Figure 3.7. NavGraph.

The **NavHost** is an empty container where destinations are swapped in and out as a user navigates through the app. The Navigation component contains a default NavHost implementation, NavHostFragment, that handles swapping fragment destinations (Android Developers, n.d.-x, Android Developers, n.d.-z).

To implement the NavHost, a NavHostFragment was added to the MainActivity via XML (the same way a button or a text view is added to an activity or fragment), and the values of android:name,

`app:navGraph` and `app:defaultNavHost` attributes were defined (Android Developers, n.d.-x, Google Codelabs, 2019-b):

- The `android:name` attribute contains the `NavHostFragment` class name.
- The `app:navGraph` attribute associates the `NavHostFragment` with the navigation graph, specifying all of the destinations in the `NavHostFragment` to which users can navigate.
- The `app:defaultNavHost` attribute ensures that the `NavHostFragment` intercepts the Back-Button system, when its value is set to `“true”`. The Back-Button system is used to navigate, in reverse chronological order, through the history of interfaces the user has recently worked with. When the user presses the back button, the current destination where the user is in is popped of the top of the back stack, and the user then navigates to the previous destination.

The **NavController** is an object that manages app navigation within a `NavHost`, orchestrating the swapping of destination content in the `NavHost` as users move throughout the app (Android Developers, n.d.-x, Android Developers, n.d.-z, Google Codelabs, 2019-b). In order to retrieve the `NavController`, the `Navigation.findNavController(Activity, @IdRes int viewId)` method was used, where `Activity` parameter is the activity that hosts the `NavHost` (`MainActivity`), and the `viewId` parameter is the ID of the `NavHost`.

Regarding the navigation bar (Figure 3.8) implementation, the `NavigationUI` class included in the Navigation Architecture Component, was used, which contains static methods that manage navigation with a bottom navigation bar.



Figure 3.8. Bottom Navigation bar.

Bottom navigation bars should be used when an application has three to five top-level destinations, making it easier for users to explore and switch between top-level views in a single tap.

The bar contents are populated by specifying a menu resource file. Each menu item's title, icon and enabled state is used for displaying bottom navigation bar items. Menu items are also programmed to display which destination is currently active by changing the colour of its corresponding item. To this end, the `android:state_checked` element's value was set to `“true”` in order to set the colour of the currently active destination to white.

The `Maps`, `History`, and `User` menu items allow the user to navigate to the `MapsFragment`, `HistoryFragment`, and `UserFragment` destinations, respectively. Only the `DataFragment` destination does not have an associated menu item as it can only be accessed through the `MapsFragment` and `HistoryFragment` destinations.

2) Retrieve and locally store server's contents

To guarantee a correct behaviour of the application on the Android terminal both *online* and *offline*, it is necessary to import and update, via Internet, database contents to the app. This includes projects in which the user is involved, as well as user details (user ID and full name), markers to be displayed on the map and species' details necessary when data is added to markers.

In order to achieve this purpose, the first step to be considered was setting up the Room database and its tables, and implementing the methods required to perform CRUD operations on the database.

Four tables were created for this purpose: `User` table, `Project` table, `Marker` table and `Species` table.

The `User` table was created to store user's details as their ID, username and full name.

The `Project` table was created to store the ID and name of projects to which the user is assigned.

The Marker table stores data about collection locations that were retrieved from the database, as well as new locations added by the user at a later stage. The stored information contains the marker's ID, the project of which it is part, its geographical coordinates such as latitude, longitude and altitude, the corresponding address for its coordinates such as country, region and location, and the date and time in which the marker was added.

The Species table stores information about the ID and name of a wide range of species that users will need when adding data to markers. There was a need to create this table not only to maintain data's uniformity in the server database, but also to facilitate the process of gathering data, and increase user's productivity since the form field where the species is specified has a autocomplete text feature, making it unnecessary for the user to enter the full species' name.

Before populating the database, the required information was requested from the server using Volley `JSONArrayRequests`, in which are specified the URLs that point to the PHP files that retrieve the information requested.

The arrays returned as a response to the requests are then converted into Java objects of the several Entity classes using POJO classes, and later inserted into the database by calling the corresponding table's `insert()` method of the `ViewModel`.

3.4.4.3 UserFragment

The UserFragment (Figure 3.9) is the first screen users see after a successful authentication, and the last screen users see when exiting the app. This fragment contains a `TextView` object¹⁵ with a message to welcome the user, a `Button`¹⁶ to send data to the server, and a `Button` to perform log out of the app. When the user logs out, local database contents are deleted.

Data is sent to the server using the Volley library, which manages the processing and caching of this requests over network. The requests were scheduled using the `WorkManager` library, which agenda the requests to run when there is network connection, and in the background.

Data sent to the server refers to collection locations added to the map, as well as data added to those locations.

¹⁵ <https://developer.android.com/reference/android/widget/TextView>

¹⁶ <https://developer.android.com/reference/android/widget/Button>

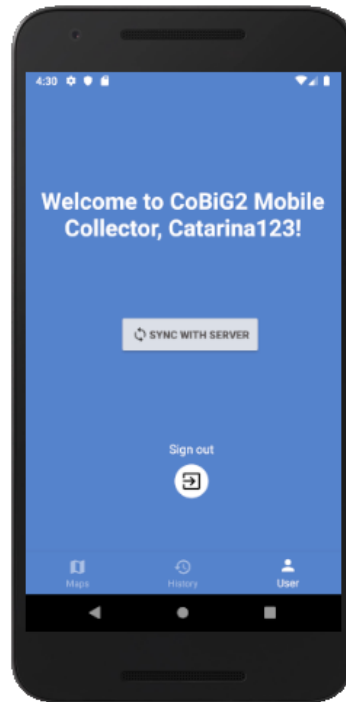


Figure 3.9. User Fragment interface.

3.4.4.4 MapsFragment

The MapsFragment (Figure 3.10) handles a map in which users can visualize and add markers according to a research project of their choice.

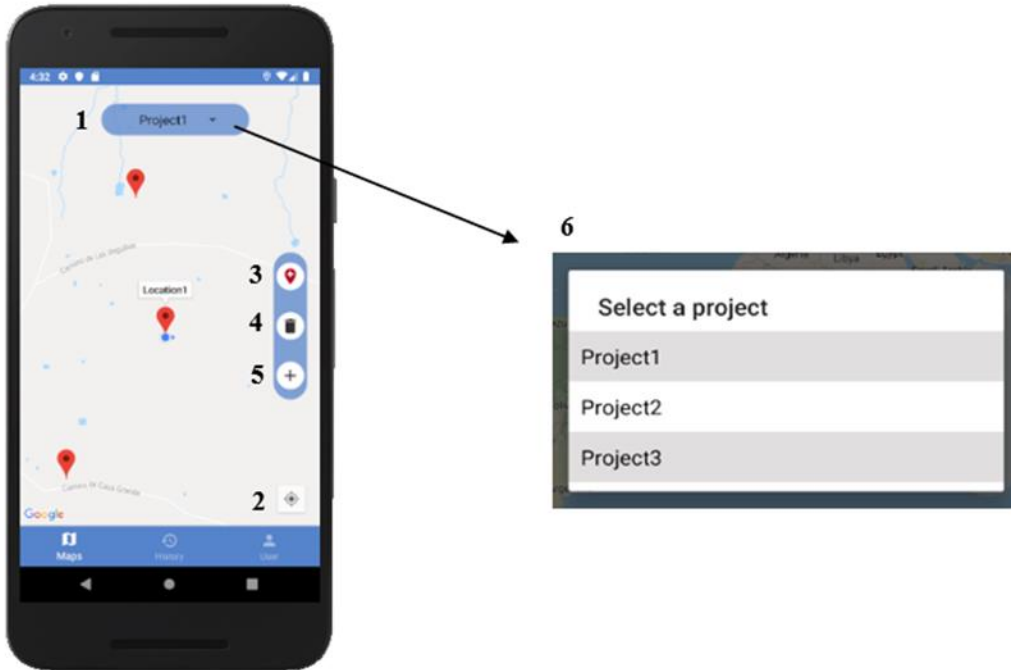


Figure 3.10. Maps Fragment interface. Legend: 1- project dropdown; 2 - user's location button; 3 - add marker to current location; 4 - delete marker; 5 - add data to marker;

The map was created using a `MapView`¹⁷, a subclass of the Android View class, which allows to display a map in a View with data obtained from the Google Maps services. The `MapView` acts as a container for the map, exposing core map functionality through a `GoogleMap` object¹⁸.

To ensure that users only see and add markers to a particular project, since different markers are part of different projects, a `Spinner` object¹⁹ was created (Figure 3.10 - 1) containing all the projects stored in the local database in which the user is involved. A spinner provides a quick way to select one value from a set and it shows its currently selected value. When the user taps the spinner, a dropdown menu is displayed containing the projects (Figure 3.10 - 6), from which the user can select a new one. After selecting a project, the user can visualize and add markers to that project.

This activity also contains four `Button` objects on the right side of the view, which serve the following purposes: show user's current position on the map (Figure 3.10 - 2), add a marker to user's current position (Figure 3.10 - 3), delete marker (Figure 3.10 - 4), and add data to a marker (Figure 3.10 - 5).

Adding new markers can be done by long-clicking anywhere on the map or by clicking on button 3, which will place a marker in user's exact position at the moment. When there's no connection to the Internet, users can still add new markers that will be stored into the local database, and later synchronized with the server when the user is back *online*.

It is also possible to delete a marker from the map, and therefore, from the server database if it does not have any associated data. So, since markers' removal requires communication between the app and the server, the user must be connected to Internet in order to perform this task.

Button 5 allows the user to navigate to the `DataFragment` to add data to a specific marker, sending the marker's name as argument.

Both buttons 4 and 5 are invisible to the user, and only appear when the user selects a marker.

¹⁷ <https://developers.google.com/android/reference/com/google/android/gms/maps/MapView>

¹⁸ <https://developers.google.com/maps/documentation/android-sdk/map>

¹⁹ <https://developer.android.com/guide/topics/ui/controls/spinner>

3.4.4.4.1 Map's Implementation

The first steps taken to implement the map were to get the Google Maps API, obtain the key and add the required attributes to the Manifest file.

In the fragment's class file, the `onMapReadyCallback` interface²⁰ was implemented, and the instance of the callback was set on the `MapView` object²¹. The `MapView` automatically initializes the maps system and the view.

The `onMapReady (GoogleMap)` callback method was used to get a handle to the `GoogleMap` object²² and is triggered when the map is ready to use. The `GoogleMap` object provides several methods used to set the view options for the map, such as the map type, or to add markers, for example. To set the type of the map, the `GoogleMap` object's `setMapType ()` method was called and the type constant `"MAP_TYPE_NORMAL"` was passed as its parameter. The normal map type is a typical road map that shows roads, some features built by humans, and important natural features like rivers. Road and feature labels are also visible.

In the case the user needs to use the app when offline and take advantage of the map, they must download it from the Google Maps system app.

3.4.4.4.2 Location feature

To provide user with a location feature, the Android offers the My Location Layer, which can be used to display a device's location on the map, and the Google Play services Location API, which requests location data.

The first step to allow the user to take advantage of the location feature is to give to the app access to user's location, by declaring the location permission (`android.permission.ACCESS_FINE_LOCATION`) in the Manifest file:

This permission allows an app to determine as precise a location from the available location providers, including GPS, as well as Wi-fi and mobile cell data. It is considered a "dangerous" permission and so the user has to explicitly grant it to the app. Upon installation, the app prompts the user to grant permission at runtime and until the user approves it, the app does not provide the location functionality. To request the permission the `requestPermission ()` method was used.

It is important to consider that users can revoke permissions from any app at any time. So even if the app used the user's current location yesterday, it can't assume it still has that permission today. Thus, every time the `MapsFragment` is launched, it is checked whether the app has the location permission or not, by calling the `ContextCompat.checkSelfPermission ()` method.

If the app has the permission, the method returns `PERMISSION_GRANTED`, and the app is allowed to access the user's current location and display it on the screen. When the method returns `PERMISSION_DENIED`, the app asks again for the location permission by calling `requestPermission ()` method.

Another aspect to consider before requesting location information is to verify if location services are enabled in user's device settings. To check if location is enabled, the method `isLocationEnabled ()` was implemented, which asks to the `LocationManager`²³, a class that provides access to the system location services, if location providers are available.

²⁰ <https://developers.google.com/android/reference/com/google/android/gms/maps/OnMapReadyCallback>

²¹ <https://developers.google.com/android/reference/com/google/android/gms/maps/MapView>

²² <https://developers.google.com/maps/documentation/android-sdk/map>

²³ <https://developer.android.com/reference/kotlin/android/location/LocationManager>

In the case where the user has location turned off, a dialog box appears in the screen warning for this problem and if the user agrees with turn it on, is redirected to the Location Settings (by initializing an Intent).

After location permissions are granted and location services are available, the app is allowed to request user's current location.

The approach taken was to request user's location considering battery efficiency, as this process is very costly in terms of battery. Another important aspect to consider is which location provider is best to use (GPS, network / Wi-Fi or cells). Since there is no perfect provider, either because it only works outdoors or because it is not very accurate, it is important to find a solution that all complement each other, without additional battery costs.

The solution found to deal with these issues was to use the `FusedLocationProviderClient`²⁴. The fused location provider is one of the location APIs in Google Play services that combines different signals/location sources (providers), provided by multiple sensors in the device, to determine device location. With fused provider it is possible to specify requirements at a high level, like high accuracy or low power. It also optimizes the device's use of battery power. To use a fused location provider, a new instance of `FusedLocationProviderClient` was created by calling the `LocationServices.getFusedLocationProviderClient()` method. To get user's location updates from the fused provider when the user is using the map, the `requestLocationUpdates(LocationRequest, LocationCallBack, Looper)` method was called:

- **LocationRequest:** the location request for the updates. `LocationRequest` objects are used to request a quality of service for location updates from the `FusedLocationProviderClient`. In this case, the app wants high accuracy location, so a location request with `setPriority()` set to `PRIORITY_HIGH_ACCURACY` was created.
- **LocationCallback** is the callback for the location updates. It is used for receiving notifications from the `FusedLocationProviderClient` when the device location has changed or can no longer be determined.
- **Looper:** the `Looper` object whose message queue will be used to implement the callback mechanism.

After user's current location is known, and in order to show it on the map, the My Location layer and My Location button (Figure 3.10 - 2) was used. The My Location layer is enabled by calling the `setMyLocationEnabled(true)` method, and therefore, the My Location button appears in the bottom right corner of the map. When the button is clicked and the current location of the device is known, the camera centers the map on that location. The location is indicated on the map with a small blue dot if the device is stationary, or as a chevron if the device is moving.

Regarding battery efficiency, even though `FusedLocationProviderClient` tries to get location in a battery-efficient way, if the priority is set to high accuracy, there is still some significant battery drain as GPS is used (to be so accurate). So, in order to get around this limitation, whenever the user is not using the map, i.e., when `MapsFragment` is in the background, and therefore, there is no need to know their current location, the `onPause()`, `onStop()` and `onDestroy()` methods were overridden in order to remove location updates by calling the `removeLocationUpdates()` method.

²⁴ <https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient>

3.4.4.4.3 Adding markers

As mentioned earlier, users can add markers anywhere on the map, by long-clicking on that map's specific position, or to their current location, by clicking on button 3. Markers stored locally are also added to the map when a project is selected.

To add a marker to the map²⁵ the `GoogleMap` object is used and the `addMarker()` method is called. This method receives as argument a `MarkerOption` object²⁶, which creates a new set of marker options used to customize the marker. Using the `MarkerOption` object, it is possible to define the marker's title and position where it will be placed. To set the title's marker the `setTitle()` is used, which receives as argument a `String`. To define the marker's position, the `setPosition()` method is used. This method receives as argument a `LatLng` object²⁷ that represents the pair of latitude and longitude coordinates of where the marker will be placed.

The `String` that is going to be used to set the marker's title is obtained by presenting users with a dialog box (before the marker is added to the map) with an `AutoCompleteTextView` object²⁸ included in which they fill in the title they want to give it. As the users add the name, this `AutoCompleteTextView` displays all markers in the local database that have a similar title. This feature has been implemented so that markers' titles are not repeated since it is a unique attribute.

The coordinates used to define the marker's position are obtained differently according to the strategy chosen by the user: 1) add the marker by **long-clicking on the map** or 2) place a marker in their **current position**.

- 1) To place a marker by long-clicking on the map, the `OnMapLongClick(LatLng point)` method was overridden and its behaviour defined to add a marker where the user has long clicked. This method is called when the user makes a long-press gesture on the map. Its parameter is a `LatLng` object, which is the point on the ground (projected from the screen point) that was pressed. The location coordinates of this point are used to set the marker's position.
- 2) To get user's current location coordinates, the `onLocationResult()` method has been overridden to get the location resulting from the location request, and to use that location's geographical coordinates (latitude and longitude) to create a marker, whenever the user clicks on button 3.

After adding a marker to the map, a `Marker` object is created, its ID (title), project to which it belongs, latitude, longitude and altitude are set and then it is inserted in the Room local database by calling `ViewModel`'s corresponding `insert()` method. To get the project to which the marker belongs, the selected project in the spinner is retrieved by calling the `onItemSelected()` method. When setting the marker's project ID, the selected project's `getProjectID()` method is called and the value returned is used.

In the case of the user uses the long-click strategy to add a marker, the marker's altitude is not obtainable and therefore is set to 0. This information can later be changed in the Web application, also developed under this project.

In case users want to view all markers for a certain project, the project selected in the spinner is first obtained as mentioned before. It is then requested to the local database all markers whose associated project is the one selected in spinner. All of these markers are added to a list which is later iterated, and for each marker in this list a marker is created and added to the map. The marker's title corresponds to

²⁵ <https://developers.google.com/maps/documentation/android-sdk/map>

²⁶ <https://developers.google.com/android/reference/com/google/android/gms/maps/model/MarkerOptions>

²⁷ <https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLng>

²⁸ <https://developer.android.com/reference/android/widget/AutoCompleteTextView>

the markerID column's value, and the Latitude and Longitude columns' values are used to instantiate a new `LatLng` object that is used to define the marker's position.

3.4.4.5 DataFragment

The DataFragment (Figure 3.11) is responsible for providing users with a set of editable fields where they add information/data to markers. User can reach this fragment by selecting a marker on the MapsFragment and click on button 4.

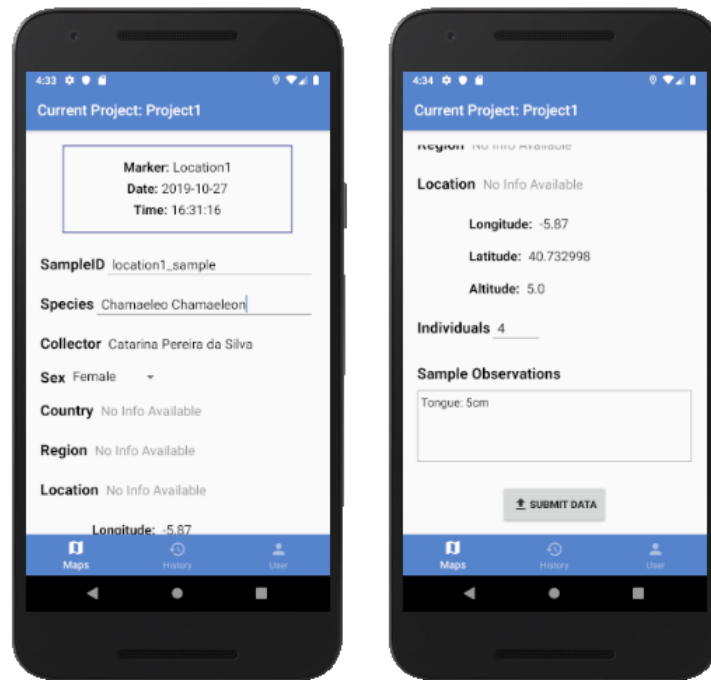


Figure 3.11. Data Fragment interface.

This fragment contains a `ScrollView`, a view group that allows the view hierarchy placed within it to be scrolled.

User interface elements contained in this view are `TextView`, `EditText`, `AutoCompleteTextView` and `Spinner` objects.

`TextView` elements are not editable and are used to display text to the user, such as Marker, Date, Time, Collector (who adds the data), Country, Region, Location, Latitude, Longitude and Altitude. These elements serve the purpose of showing user all the information about the selected marker. To get this information, the database was asked to provide all the data stored in the database for this marker. Collector's default name is the logged in user. However, this information can later be changed using the Web application²⁹.

When the user adds new markers, the address's information for the marker's geographical position is not stored or even searched. The `Geocoder` class³⁰, which transforms a coordinate into an address, only works when users are connected to network, and since users are supposed to be *offline* when using the application, this information is only obtained at the stage users are connected to the Internet. Thus, when users are *online* and click the button to synchronize information with the server (in `UserFragment`),

²⁹ <https://developer.android.com/reference/android/widget/TextView>

³⁰ <https://developer.android.com/reference/android/location/Geocoder>

the marker's address is searched and updated in the local database and the information is subsequently sent to the database on the server.

`EditText` elements are used for entering and modifying text. `EditText` objects must have specified the `android:inputType` attribute, which configures the keyboard type that is shown, acceptable characters, and appearance of the edit text. All the `EditText` objects, excepting the `Individuals` and `Observations` elements, have `inputText` attribute specified as `"textNamePerson|textCapSentences"`. Even if they are not meant to accept a person's name, the `"textNamePerson"` value is used so the user cannot insert new lines (the enter button is not showed). The `"textCapSentences"` value is used to request capitalization of the first character of every sentence. The `Individuals` element's input type is specified to `"number"`, which means it only accept numbers and triggers a number keypad with the numbers 0 to 9. The `Observations` element's input type is specified to `"textMultiLine|textCapSentences"`. The `"textMultiLine"` value allows multi-line text³¹.

`AutoCompleteTextView` elements show completion suggestions automatically while the user is typing. "The list of suggestions is displayed in a dropdown menu from which the user can choose an item to replace the content of the edit box with" (Android Developers, n.d.-v) The `Species` element contains a list of suggestions with all the species that were fetch from the server database and are now stored in the local database³².

After the user inputs all the data they want to add to the marker, they can click on the `Submit Data` button placed in the bottom of the view. This button will perform the action of adding this data to the local database. For this purpose, a `Data` entity class was created (columns: ID, sample, marker's ID, species' name, collector, sex, individuals, sample's observations and status) and a set of getter and setter methods were defined in order to access and modify its content. Every time the user adds a new data to the marker, an instance of the `Data` entity class is created, and its attributes are set with the values inserted in the user interface elements.

The data is stored in the local database either when the user is *online* or *offline*, in order to overcome the limitation of having the same features in both situations. For the data to be effectively sent to the server (if the user has internet) it is necessary that they express their intention to perform this action by clicking on the `sync` button available in the `UserFragment`.

If data is successfully stored in the server database, it is deleted from the local database. The `status` attribute was created to show the user the data's current status (if it not yet sent to server or if an error occurred while uploading it).

3.4.4.6 HistoryFragment

The `HistoryFragment` (Figure 3.12) contains a `RecyclerView`^{33,34}, a flexible view for providing a limited window into a large data set, which displays the history of data added to markers by the user.

³¹ <https://developer.android.com/reference/android/widget/EditText>

³² <https://developer.android.com/reference/android/widget/AutoCompleteTextView>

³³ <https://developer.android.com/guide/topics/ui/layout/recyclerview>

³⁴ <https://developer.android.com/reference/android/support/v7/widget/RecyclerView>

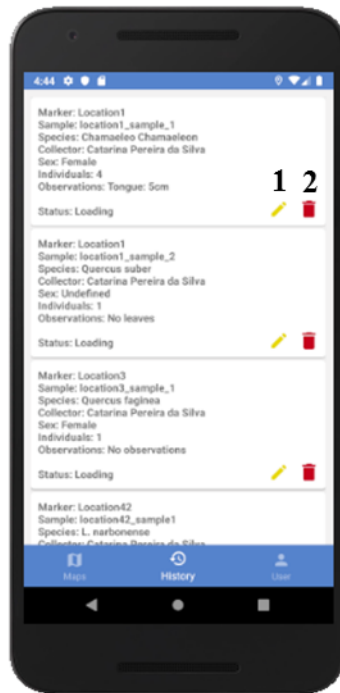


Figure 3.12. History Fragment interface. Legend: 1 - update data button; 2 – delete data button;

This interface contains an edit button (1) that can be used to edit data. If the user clicks in this button, it is redirected to the DataFragment where they can change the data in question. To facilitate this process, all current data's information is automatically displayed in the UI elements, which can then be modified. If the user submits the new information, the data's information is updated and stored in the local database.

It also contains a delete button that can be used to delete permanently data from the local database, which means that the deleted data is not going to be stored in the server database.

The RecyclerView fills itself with views provided by a layout manager. For this purpose, a LinearLayoutManager was used. The individual items are represented by view holder objects, which are instances of a class that was defined by extending a RecyclerView.ViewHolder. There are created as many view holders as are needed to display the on-screen portion of the dynamic content and, each one is in charge of displaying a single item, and has its own view. As the user scrolls through the list, the RecyclerView creates new view holders as necessary, and also saves the view holders which have scrolled off-screen and links them back to the data that is scrolling on the screen, so they can be reused. View holder objects are managed by an adapter, which was created by implementing a class that extends the RecyclerView.Adapter abstract class. The adapter creates view holders as needed and binds them to their data. To accomplish this, the view holder is assigned to a position, and the adapter's onBindViewHolder() method is called. This method uses the view holder's position to determine what the contents should be, based on its list position. When the displayed items change, the adapter is notified by calling the notifyDataSetChanged() method³⁵³⁶.

All the data collected and stored in the local database are cached in the RecyclerViewAdapter as a List of data. This list of data automatically updates and redisplay when the data in the database change. The automatic update is possible because LiveData³⁷ was used. In the HistoryFragment, there

³⁵ <https://github.com/fjoglar/android-dev-challenge/blob/master/articles/lesson-04-recyclerview.md>

³⁶ <https://developer.android.com/guide/topics/ui/layout/recyclerview>

³⁷ <https://developer.android.com/topic/libraries/architecture/livedata>

is an Observer that observes the collected data `LiveData` from the database and is notified when they change. Whenever the data changes, and the fragment is in the background, the `onChange()` callback method is invoked, which calls the adapter's `setData()` method to update the adapter's cached data and refresh the displayed list.

3.4.5 Passing data between Android UI components

3.4.5.1 Between activities

As mentioned before, when an app creates an `Intent` object to use in `startActivity()`, as in starting the `MainActivity` from the `LoginActivity`, the app can pass in parameters using the `putExtra()` method. The data is passed under the form of key/value. To access the data passed through the `Intent`, the `getIntent().getExtras()` method was called in the `MainActivity`'s `onCreate()` method.

3.4.5.2 Navigation operations

Navigation also allows to attach data to a navigation operation by defining arguments for a destination. To pass data between destinations, the `<argument>` element is added as a child of the destination's element that receives the argument and the name, argument's type and default value attributes are set as intended. Arguments are automatically added to the corresponding action(s) that takes the user to this destination.

To pass data securely during navigation operations, `SafeArgs`³⁸ was used: A Navigation Architecture Component plugin that produces simple object and builder classes for type-safe access to arguments specified for destinations and actions. According to Google Android, this is the preferred way to pass data when using navigation, since it ensures type-safety.

Code generated by `SafeArgs` contains safe-type classes and methods for each action along with the sending and receiving destinations. Three classes are created, as described below (Android Developers, n.d.-aa):

- A class is created for each destination where an action originates, and it has a method for each action defined in the originating destination;
- An inner class is created for each action used to pass the argument;
- A class is created for the receiving destination. The class's `fromBundle()` method is used to retrieve the arguments.

In the destination where an action originates, the action's argument is set and passed to the receiving destination by calling the method `navigate()` that takes as parameter the customized action.

To retrieve the argument, the `fromBundle()` method is called in the receiving direction and takes as parameter the `getArguments()` method, which retrieves the bundle and use its contents.

³⁸ <https://developer.android.com/guide/navigation/navigation-pass-data>

3.5 Web Application Development and Implementation

The Web application comes with the intention of giving users a tool to view and manage database content in an interactive way.

When a user accesses the Web application's URL in a Web browser, the server returns an HTML document that is interpreted by the Web browser and allows users to visualize the system interface.

The website interface development involved the use of HTML for defining the website's structure and content, CSS for defining the HTML tags representation through a set of applied rules according to a predefined syntax, making Web pages more attractive and organized, and JavaScript for accessing and modifying content, and allowing user interaction, making the Website dynamic.

However, when users access the interface through the browser, they can only see the server-provided HTML pages, styled with CSS, and client-side scripts in the form of JavaScript. All the communication between the client and the server is done in the backend, which is not visible to users. To solicitate new data or to transmit user-created data to the backend, HTTP request are sent to the server using PHP. The server response, which may come in the form of XML, JSON or HTML, is interpreted by the frontend and displayed in the website.

3.5.1 Requirements

The Web application must meet the following main requirements:

- Authentication, in order to the access to the Website's content by desirable users only;
- Import and update database content necessary for the correct behaviour of the Web application;
- Send information entered by the user to the database on the server;
- Query the database and retrieve the results;

3.5.2 Front end Development

The website interface was built using HTML version 5, CSS and JavaScript.

All HTML documents that make up the website begin with a document type declaration, `<!DOCTYPE html>`.

Documents start with the `<html>` tag and end with the `</html>` tag, which indicates that anything between them is HTML code.

HTML document's information is placed between `<head>` and `</head>` tags. This information typically defines the document's title, styles (CSS file), scripts, among others. Everything placed between these `<head>` tags is not displayed on the website.

Everything placed between the `<body>` and `</body>` tags is shown inside the main browser window. Bootstrap framework templates were used to design some of the site's content, such as buttons, dropdowns, tables and forms.

The style and layout of the different Web pages are specified and compiled into a single CSS file, separate from the HTML document, which makes it easy to organize from a programmer's point of view. This file specifies the styles and layouts for different screen sizes and resolutions in order to ensure that the site comes up correctly in different browsers, different operating systems and different devices.

Client-side scripts, such as JavaScript, are defined inside the `<script>` and `</script>` tags.

Following are brief descriptions of each of the Website interfaces.

3.5.2.1 Sign up Interface

Sign-up interface (Figure 3.13) is the Web page where users can log- in in order to access site content. Only registered users can view and manage Website’s content.

This Web page also contains a “here” link, which redirects to a sign-up page (Figure 3.14) with a form that allows new users to register. Registration must be validated by an administrator in order for the user to access the content of the Website.

The background image used in the Sign up interface was taken from: <https://www.pexels.com/photo/daylight-environment-forest-idyllic-459225/>.

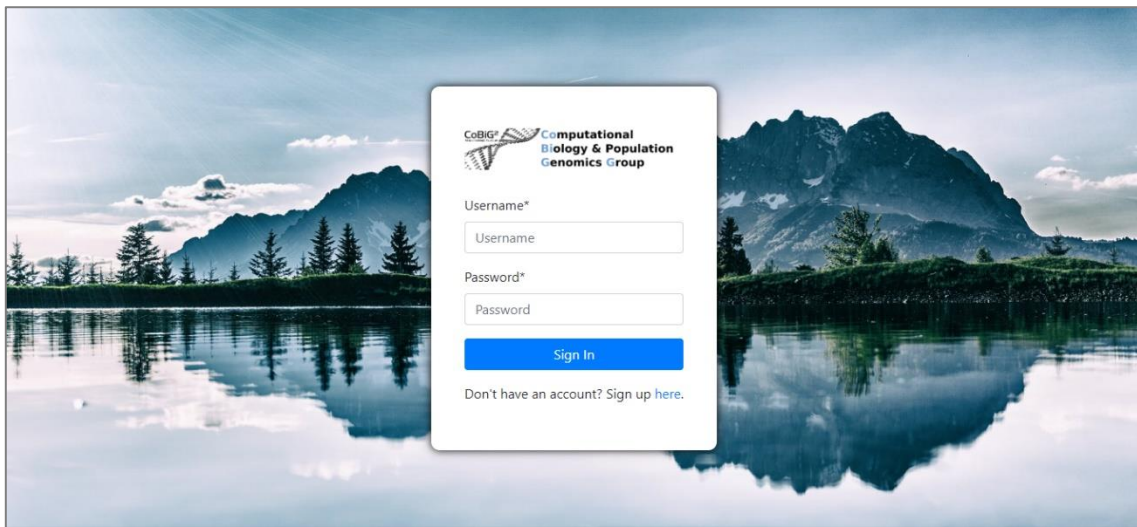


Figure 3.13. Sign In Web page interface.

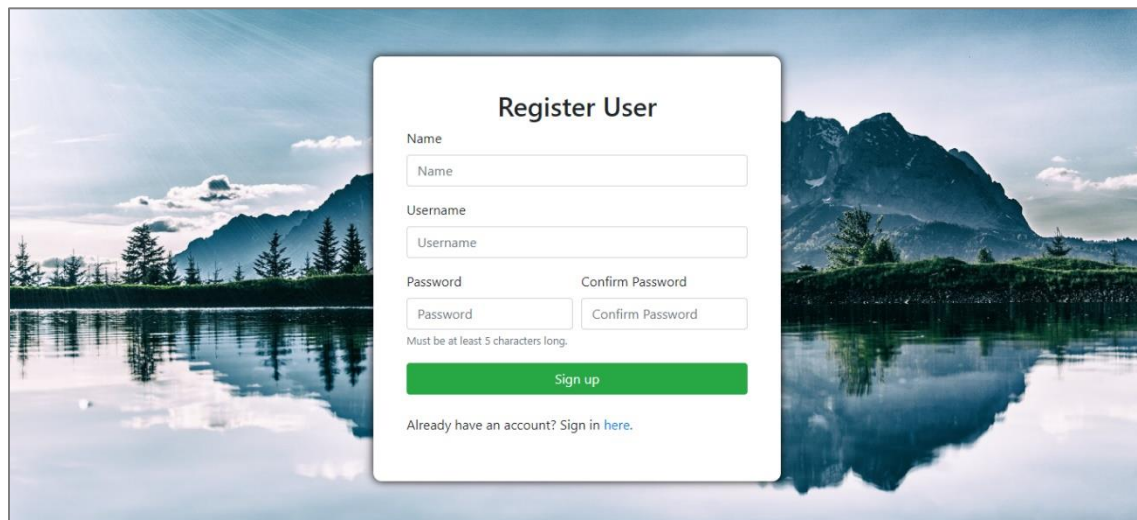


Figure 3.14. Sign Up Web page interface.

3.5.2.2 Header

All Web pages, except the Sign In and Sign Up pages, contain a header that allows users to navigate through the Website. The header contains the Maps, Search and Input links, which redirect to the map, information search and new data entry interfaces, respectively.

It also contains an Account dropdown, which has the Download APK and logout links that allow users to download the mobile application's APK and perform the log out of the Website, respectively.

If the user is identified in the database as administrator, the dropdown also contains the Administrator link, which redirects to the Webpage which allows the administrator to administer database data.

The Figure 3.15 shows the header of an administrator user.

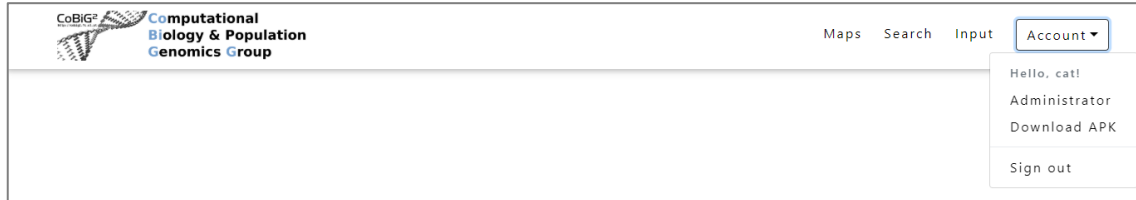


Figure 3.15. Administrator user's header.

3.5.2.3 Map Interface

The Map interface (Figure 3.16) is the first Web page accessible to registered users. On this Web page is presented a map and a dropdown list with the name of all research projects within the database. When the user selects one of the dropdown menu's options, all markers (or locations in the field) where data was collected, appear on the map. If the user clicks in the marker, a popup with the marker's name and other details, as well as links to delete and update the marker, appears along with the marker. Users can delete markers from the database by clicking on the delete link, also available in the popup. Only markers that do not have any data associated can be deleted. If the user clicks on the update link, a new Web page where it is possible to change the marker's values, such as name, latitude, longitude, country, region, among others, is rendered.

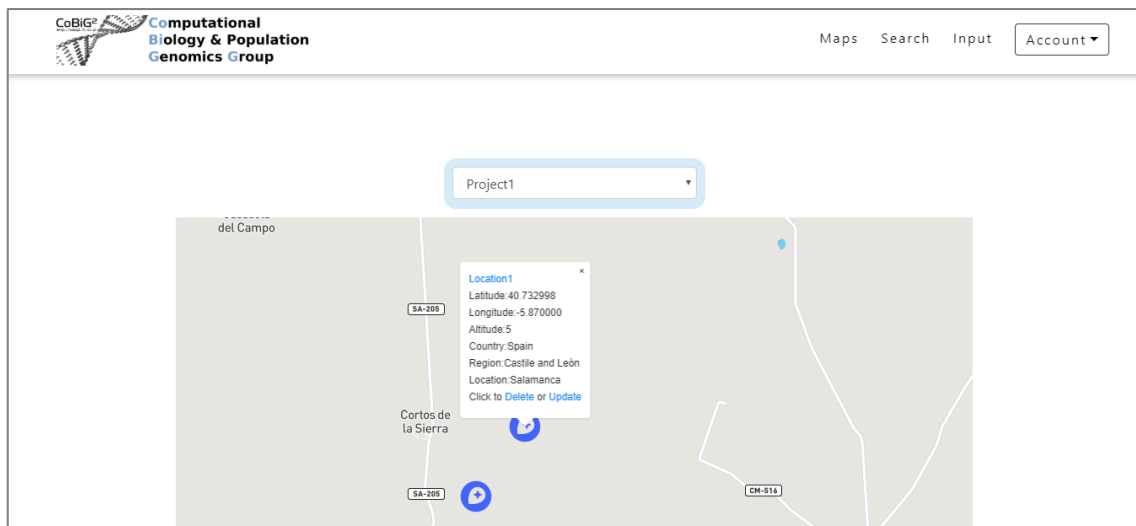


Figure 3.16. Maps Web page interface.

Whenever a user clicks on the marker's name, a Web page (Figure 3.17) with a table that contains the collected data for that location is presented.

It is possible to export the table's data into .csv format and visualize it in Excel or another tool that supports .csv files.

Each data in the table can be deleted by clicking the Delete link, or its values changed by clicking the Update link. If the user wants to delete the data, a confirmation box will appear to confirm the

deletion. If the user proceeds with deletion, the data is deleted from the table and permanently from the database.

In case the user wants to change the values of any available data, and consequently, clicks on the update link, is going to be redirect to another Web page where is possible to fill only the fields whose values the user wants to change. Fields sent with empty/null values will not change the value in the database, remaining the same before submission.

Delete	Edit	Marker	sampleID	Species	Sex(F/M)	Country	Region	Location	Latitude	Longitude	Altitude(m)	Date	Time	Collector	Indiv.	Observations	Storage Method	Storage Institution	Storage Room
Delete	Update	Location1	location1_sample_1	Chamaeleo Chamaeleon	Female	Spain	Castile and León	Salamanca	40.732998	-5.87	5	2019-10-27	16:31:16	Catarina Pereira da Silva	4	Tongue: 5cm			
Delete	Update	Location1	location1_sample_2	Quercus suber	Undefined	Spain	Castile and León	Salamanca	40.732998	-5.87	5	2019-10-27	16:31:16	Catarina Pereira da Silva	1	No leaves			

Figure 3.17. Marker's data Web page interface.

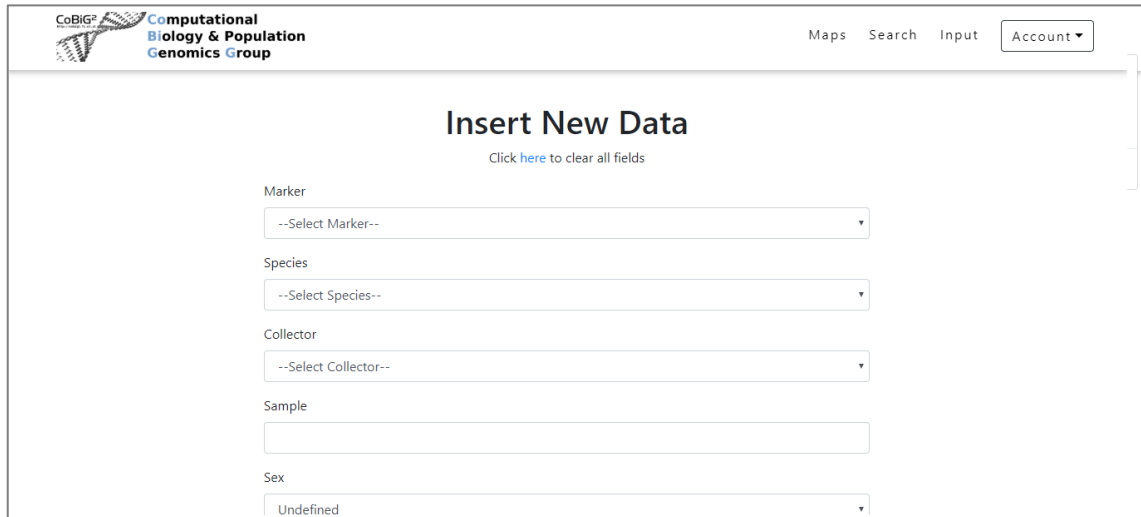
3.5.2.4 Search Interface

The Search Interface (Figure 3.18) contains a search bar that can be used by users in order to search for data without using the map. This alternative allows users to search for data by project, marker, local (country, region or location), collector or species.

Figure 3.18. Search Web page interface.

3.5.2.5 Input Interface

The Input interface (Figure 3.19) is used to add new data to a marker. Users will be presented with a form similar to the one in the Android app, however with some extra form fields.

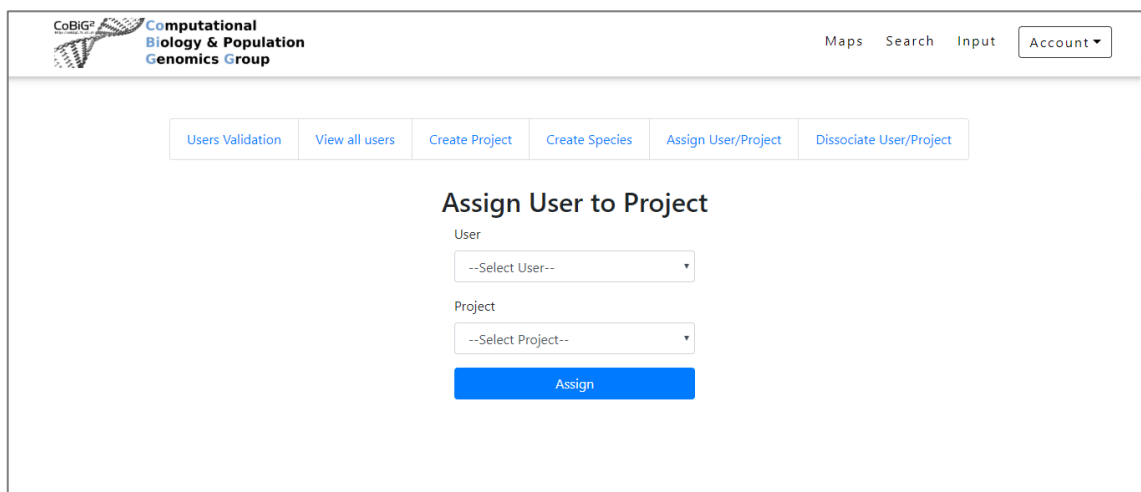


The screenshot shows the 'Insert New Data' web page interface. At the top left is the CoBIG logo and the text 'Computational Biology & Population Genomics Group'. At the top right are links for 'Maps', 'Search', 'Input', and an 'Account' dropdown menu. The main heading is 'Insert New Data' with a sub-link 'Click here to clear all fields'. Below this are five form fields: 'Marker' (dropdown menu with '--Select Marker--'), 'Species' (dropdown menu with '--Select Species--'), 'Collector' (dropdown menu with '--Select Collector--'), 'Sample' (text input field), and 'Sex' (dropdown menu with 'Undefined').

Figure 3.19. Input Web page interface.

3.5.2.6 Administrator Interface

The Administrator interface (Figure 2.20) is only available for administrators. It can be reached by clicking in the Administrator link of the Account dropdown placed in the header. Administrators are presented with a set of tabs in order to perform certain actions. Administrators can validate new users, view/delete registered users, add new projects and species, and assign/dissociate users to projects.



The screenshot shows the Administrator web page interface. At the top left is the CoBIG logo and the text 'Computational Biology & Population Genomics Group'. At the top right are links for 'Maps', 'Search', 'Input', and an 'Account' dropdown menu. Below the header is a row of six tabs: 'Users Validation', 'View all users', 'Create Project', 'Create Species', 'Assign User/Project', and 'Dissociate User/Project'. The 'Assign User/Project' tab is selected. The main heading is 'Assign User to Project'. Below this are two form fields: 'User' (dropdown menu with '--Select User--') and 'Project' (dropdown menu with '--Select Project--'). At the bottom is a blue 'Assign' button.

Figure 3.20. Administrator Web page interface.

3.5.3 Back end Development

The backend is the part of the Web application that handles the processes that happen on the server and is not directly visible to the user. “It receives requests and prepares data which is transmitted back to the user’s browser” (Supalov, n.d.).

Both backend and frontend work together to meet a single goal – so a user can access them. Interaction between backend and frontend can be visible when (Supalov, n.d.):

- The user accesses any of the Web application’s URLs in the browser – the browser sends one or more requests to the server, which returns a response that is interpreted by the Web browser and used to render a part of the Website;
- The user interacts with the Web page, causing more requests to be send in order to get more data and display new information, and so on.

The browser communicates with the backend over HTTP. HTTP requests arrive from the browser at the backend and may contain data in the HTTP headers or request body. The purpose of these requests may be to solicit new data or to transmit user-created data to the backend (Supalov, n.d.).

For each request, the server returns a response to the user’s browser by loading information into HTTP headers and request body (Supalov, n.d.).

A browser triggers a request to the backend when (Supalov, n.d.):

- The user enters a URL, which makes the browser to request it;
- While loading a Website - the browser reads the incoming HTML and realizes that there is a resource that needs to be loaded (JavaScript file, image or CSS file);
- The user clicks on a link – the browser knows that the user needs to navigate to a new Web page so requests the corresponding URL, making a new Webpage to be loaded and rendered;
- JavaScript code is executed on the site and wants to load data in the background, using Ajax asynchronous calls.

These HTTP requests can be GET or POST requests, as mentioned earlier. A GET request uses the GET method and has all of the parameters and corresponding values that it will passed to the backend in the URL itself. When the call is made, it makes a get request to the endpoint of the backend (PHP scripts that are in server) and passes the parameters and corresponding values specifying them in the URL. The PHP code to be executed within the request can access the parameters and its values, passed via URL, through the variable `$_GET`.

POST requests use the POST method and have all the values that will be passed to the backend encoded in the request body. In this case, the browser invokes the URL, but the values are encoded in the request body itself. This happens whenever a user fills a form. In which case the request body is the content that the user inputs in the form’s fields. After the user clicks the “submit” button, these values are accessed in PHP using the variable `$_POST`.

Both `$_GET` and `$_POST` values are then used to perform the queries against the database in order to retrieve/send the required data.

The requests made from the frontend to the backend can be synchronous or asynchronous. Synchronous typically means that the user makes a request and waits for the response, while asynchronous means that the request is made in the background, and the page (or particular elements of the page) are updated when the response from the request is received.

After the request is made, PHP communicates with the MySQL database and the backend returns the data in XML, JSON or HTML.

When making a synchronous request, the returned data is rendered, meaning a whole page is shown, like when the user logs in or the data is submitted in a form (the user receives a response from the server saying if the submission was successful or not).

When making an asynchronous request, the frontend may take those result and render them to the page in a particular place (like when the markers are placed in the map) or it can do anything as a result or using the response from the backend. Ajax is the traditional way to make asynchronous HTTP requests. Data can be sent using the HTTP's POST method and received using the HTTP's GET method. Since writing Ajax code can be complicated, because different browsers require different syntax for Ajax implementation, jQuery is used to simplify the creation of Ajax requests and the processing of data returned by the server.

So, the \$.ajax method is used to perform this kind of requests. What happens is that the JavaScript loaded in the browser sends a HTTP request and the data is loaded directly into a selected HTML element of the Web page.

\$.ajax³⁹ method takes many parameters, some of which are required and other optional, and contains two callback options success and error to handle a function to run if the request succeeds or fails, correspondingly. Some of the \$.ajax method's parameters are: **method**, in which is specified the HTTP method do be performed; **url**, in which is specified the URL endpoint to which the request is sent (the default is the current page) and **data**, in which is specified the data to be sent to the server.

In the case the user wants to delete data from a table, as in Figure 3.17, an ajax call is also made. Before the data is actually deleted, the user is presented with an alert box where a message is displayed to confirm if that is actually the action they want to perform. If the user confirms the action, the information to be deleted is specified and sent to the database on the server, the action is performed and the data disappears from the table (only the table is updated, the whole page is not reloaded again).

If the user wants to request data to view, or send data to the server in a form or using the search bar, the requests are synchronized. A new page is displayed and information coming from the server is displayed on that page. Normally, to perform these actions, both buttons and links are directly linked to URL endpoints to be reached. In forms, the URL is specified in the `action` attribute, which informs to where the form-data is sent when the form is submitted. Also, in a form, the HTTP method is also specified in the `method` attribute. The server's response is then retrieved by the server and can be a simple informational message that is displayed to the user, or it can be embedded into HTML (like when data is displayed in a table).

Regarding PHP, all the PHP code developed is enclosed in the documents between start and end processing instructions `<?php` and `?>`.

For communications between PHP and MySQL, the same approach as in the Android app was used (using the MySQLi⁴⁰ extension, which allows to open a database connection using credentials and then select, update and delete data using SQL prepared statements).

3.5.3.1 User's registration and sign in

User's registration can be reached in the Sign In interface. By clicking in the link on the bottom of the Sign In form, the user is redirect to a page that contains a form, in which the user can insert their name, username and password. Besides these fields, there is also a password's confirmation field, in which the user is asked to type the previous password again, in order to confirm if they match .

After the form is submitted, the database informs if already exists a user with that username in the database. If that username does not exist, the registration process begins and waits for validation. All registration requests are stored in the Validation table, waiting for confirmation. Registrations' validations are made by Administrators and are available in the Administrator Page, tab User's

³⁹ <https://api.jquery.com/jquery.ajax/>

⁴⁰ <https://www.php.net/manual/en/book.mysqli.php>

validation. If the administrator accepts the registration's request, the information inserted in the registration form is stored in the User table and the user is successfully registered.

Passwords submitted while registering are encrypted and salted before they are stored in the database. To encrypt and salt passwords, the PHP method `password_hash()`⁴¹ is used. This method uses a brycpt algorithm that is designed to change over time as new and stronger algorithms are added to PHP. A random salt is generated by this method for each password hashed.

Regarding the login in, every time a user logs in, the entered password's hash is compared to the one stored in the database. If both hashes are compatible, the login is successful, and the user is now able to navigate in the Website.

In order to store user information to be used across multiple pages, a session is started (`session_start()`⁴²) by the time the user performs a successful login, and the information is saved in session's variables. By default the session's variables holding user information expire when the user closes the browser, however the cookie session is set to be valid for over one year, meaning that a user just needs to perform the next login within one year, if he does not log out of the Website (both session and session variables are erased when the user performs a log out). User's username and information about whether a user is an administrator or not are saved in these session variables.

The value for the administration attribute is used to decide which header is going to be available for the user. In the case the user is an administrator, the corresponding header has a link that redirects to an area for administration purposes.

3.5.3.2 Maps and Markers

The map available on the Map interface is provided by Mapbox. All map-related code development was done using the Mapbox platform documentation and code template⁴³.

To use the Mapbox's APIs, it was necessary to get a Mapbox access token.

To get the map, the `mapboxgl.Map` class was instantiated and associated with var `map`, according to the following code:

```
var map = new mapboxgl.Map({
    container: 'map',
    style: 'mapbox://styles/mapbox/streets-v11',
    center: [-9.156681, 38.756191],
    zoom: 5
});
```

The class takes as its arguments the map element, which corresponds to the container that will host the map, and a JSON object, where the initial map location is defined (`center` - in this case, corresponds to Portugal), the map scale (`zoom`) and the map style (`styles`). The next step is the creation of the markers according to the project selected in the dropdown. For this, the selected project is obtained and a request to the server to get all database markers for that project is made. In response to the request, a JSON object is obtained which contains the information of all returned markers. A cycle that traverses all objects contained in the JSON object is created. For each object, the values corresponding to the name, country, region, etc. of the markers are used to create the popup that is available when the user clicks in the marker. The latitude and longitude values are used to define where the marker will be placed.

⁴¹ <https://www.php.net/manual/en/function.password-hash.php>

⁴² https://www.php.net/manual/pt_BR/function.session-start.php

⁴³ <https://docs.mapbox.com/help/tutorials/custom-markers-gl-js/>

The popup is created using a new instance of the `mapboxgl.Popup()` class, which defines the content that will be available in the popup:

```
var popup = new mapboxgl.Popup()
    .setHTML(' <h8>Country: '+key["country"]...'</h8');
```

The marker is created later as follows:

```
var marker = new mapboxgl.Marker()
    .setLngLat([key["longitude"], key["latitude"]])
    .setPopup(popup)
    .addTo(map);
```

The `var marker` variable, which corresponds to an instance of the new `mapboxgl.Marker` class, is created, the marker's position and popup are set, and the marker is added to the map.

Chapter 4 - Tests

In this chapter are described the tests performed on the applications that make up the system. These tests aim to ensure the correct functioning of applications when manipulating data, and the correct operation and visualization of mobile application interfaces on various devices, and Web application in various browsers.

Testing an application (mobile or Web) is an integral part of the development process. By running tests against an application consistently, it is possible to verify application's correctness, functional behaviour, and usability before releasing it publicly.

As the system was developed, it underwent extensive testing. All system components, database, android application and Web application were tested to verify that the requested features were implemented correctly or not. The testing process took place in two phases. The first phase occurred during the development of the applications that make up the system, in which tests were performed in local environment using developer working tools. These tests are intended to analyse application source code and search for errors or possible causes of errors in the program. The second phase consisted of performing tests in real environment, i.e. using the mobile application in the field, collecting data and sending it to the server, and then viewing and manipulating it through the Web application.

A database, a Website, and PHP files that allow to communicate with the server, should never be uploaded to the “live” server until they have been thoroughly tested. In the first phase, the tests were performed on a local pc using the Apache HTTP server component of the XAMPP package which allows the pc to act as a Web server. XAMPP includes everything needed to set up a Web server - server application (Apache), database (MariaDB), and scripting language (PHP).

A database was created on the local server, whose physical design was the proposed in the planning and analysis phase of the actual database construction. This database served to test whether the data collected in the Android application was actually stored correctly in the database, and whether the Web application correctly provided and manipulated the data stored in the database.

Consequently, the tests and the evaluation of the state of the database were being performed as the applications that manipulate the data stored in the RDBMS, were developed and tested.

PHP scripts developed in the project that allow to communicate with the database, were also uploaded to the XAMPP local server.

To test and debug the Android application, in order to verify:

- consistency of different Android app interfaces;
- obtaining prior information from the server for the correct functioning of the application;
- application behaviour when the device is *offline* (data storage in local device database), since one of the limitations of fieldwork is that there is not always network coverage;
- sending and later storing the collected data in the local server database,

the Emulator provided by Android Studio was used, which allows to replicate the environment of different android mobile phones, and as a communication bridge between the local server database and the application, the previously provided PHP scripts, mostly to test Volley requests, were used.

Regarding the Web application tests, the XAMPP platform also has the ability to serve Web pages on the World-Wide-Web. Thus, the pages that make up the site, for viewing and managing the data collected by the mobile application, could be made available on the Web. Thus, it was possible to test and evaluate:

- consistency of different Web pages interfaces;
- communication between the Website and the database on the local server, in order to obtain personalized information to be displayed, manipulate data, submit and store new data.

Additionally, Chrome DevTools⁴⁴ were also used for functional website verification and business logic. Chrome DevTools is a set of Web debugging tools built into Google Chrome for Web developers to iterate, debug, and profile websites.

In the second phase of the tests, the system was tested in a real environment. An infrastructure similar to the one used in the first phase was set up. This was based on docker-compose⁴⁵ (which can be used to define and run multi-container Docker applications), running two container images from docker-hub: one, based on the image "php:7.1.20-apache", running the Webservice, and the other, based upon "mysql:5.7", running the database. The setup is based upon <https://github.com/sprintcube/docker-compose-lamp> (as of commit 3eee248380). The HTTPS certificates were deployed on the host system using EFF's Certbot⁴⁶. The database and the PHP scripts that comprise the website and the mobile application are deployed from host system volumes mounted on each docker container respectively. A new "test" user was then created in the database in order to test all features of both the Web server and the mobile application.

To test the mobile application in a real environment, the application's APK was installed on a Samsung J7 2016 device, and data was collected near Campo Grande, with the help of CoBiG2 group members. Markers were added to the map at the time the tests were being performed and fictional data was collected. The addition of markers as well as data collection were performed both *online* and *offline*. Later, with Internet available, the collected information was sent to the server to be stored in the database.

After the real environment testing of the mobile application, the Website was tested on a computer. The website was accessed via its URL, and it was verified if the information collected by the mobile application was correctly sent and stored in the database and whether it could be viewed and manipulated correctly using the website features.

After verifying the proper functioning of the whole system, the mobile application's APK was uploaded to the website so that it could be downloaded and installed by next system users.

⁴⁴ <https://developers.google.com/web/tools/chrome-devtools>

⁴⁵ <https://docs.docker.com/compose/>

⁴⁶ <https://certbot.eff.org/>

Chapter 5 - Conclusions and Future Work

5.1 Conclusions

From the study carried out and the development of the project that gave rise to this thesis, there were many challenges faced during the development of the components that make up the system. One of the challenges was due to the need to acquire knowledge in the areas of mobile and Web application development, as the researcher's knowledge in these areas was limited.

As for the project developed, all the objectives outlined were achieved. The aim was to develop a computerized system that would allow the collection, storage, visualization and management of georeferenced and descriptive data of biological samples. To this end, a web and Android applications were developed, and a database built to support the collection of information from ongoing investigations.

The Android application was successfully completed as it met the requirements proposed by the CoBiG2 group. This was efficient in collecting georeferenced data in the field. Its assumptions include security in access, the need to obtain the current location of the user and make it available on a map, obtaining data from the server in order to allow the correct operation of the application, sending data to be stored in the server database, and finally, the operation of the application when the device is *offline*.

For the first assumption, an authentication system was implemented so that the application could only be used by registered users. This measurement was successfully performed as intended.

Regarding the second assumption, the features provided by Google Maps APIs, which allow the development of mobile applications based on location services and map navigation, facilitated the process of obtaining the user's location as well as displaying it on a map. Android implements location techniques using GPS and data networks such as Wi-fi and cell towers. According to the literature review, the most accurate location technology is GPS. However, its main limitations are energy consumption and lack of coverage in indoor environments, or urban environments with high infrastructure. To circumvent the problems inherent in using GPS to obtain user location, sensors fusion was implemented, which allows GPS to be used in conjunction with other location techniques in a battery-efficient way, and location updates were disabled when the user is not using the map. Sensors fusion reduced the number of GPS readings, increased the accuracy and precision of the user's position, as technologies complement each other, and made the application more energy efficient.

Regarding obtaining data, that allows the application to function properly, as well as sending data to be stored in the server database, the use of the HTTP library Volley was quite efficient. This library, in addition to being effective in both importing information and sending data to the server, also made network requests more manageable and faster in terms of overall application performance.

The main obstacle to mobile application has been the use of mobile applications in remote locations with limited Internet access. With this limitation, both map's visualization, as content's browsing or storage of new information to be later sent to the server, would be impossible to accomplish.

In order for the user to have access to the map when *offline*, it was necessary to integrate the Google Maps APIs. With this integration, users will be able to use *offline* maps as long as they are careful to download it in the Google Maps system application before using the app.

The approach taken to overcome the limitation of content browsing or saving information entered by the user when the device is *offline*, was to store all information locally either *online* or *offline*, and to synchronize it with the server only when the user expresses intention to perform this action. For this purpose, a local database was implemented using the Room library. This approach proved effective in overcoming this condition.

Regarding the proposed application's layout, it proved to be simple and easy to use. The use of a bottom navigation bar ensured a consistent and predictable user experience as it facilitates user interaction with different parts of the app's content.

The web application developed in the project met the users' needs, as it allows to effectively visualize and manage the data collected by the mobile application. Regarding security in access, administrator validation of new users has been implemented upon registration, which has efficiently allowed only desirable users to access the website's functionalities (manipulate and visualize data). Also the web pages' interface that make up the website, proved to be consistent, predictable and easy to use.

Regarding the implemented database, its robustness was characterized by evidence of consistent data. This robustness is due to the proposed database structure and implemented in the RDMBS MySQL.

Undoubtedly, the coming years will have a huge impact on how new information technologies interact with areas of scientific knowledge. The future of information systems looks promising when it comes to automating processes and developing solutions based on the use of data processing, which aim to increase researchers' efficiency and, consequently, their productivity at work.

5.2 Future Work

As a suggestion for future work, it is proposed to develop the mobile application for the most used operating systems in order to integrate a larger user community.

It is also proposed to implement features that allow integrating images and sound files with the data added in the mobile application.

It may also be interesting to develop similar, however adapted, applications for other cE3c teams.

Chapter 6 – References

- . Linked Heritage Glossary - XML. (n.d.) Retrieved 2019.08.26, from <https://elearning.unipd.it/sba/mod/glossary/print.php?id=5&mode=letter&hook=X&sortkey&sortorder=asc&offset=0&pagelimit=10>
- Adewale, A. (2018). Understanding The Basics of Restful APIs Retrieved 2019.10.05, from <https://pusher.com/tutorials/understanding-rest-api>
- Android Developers. (n.d.-a). Activity, 2019.09.02, from <https://developer.android.com/reference/android/app/Activity>
- Android Developers. (n.d.-b). API reference, 2019.09.02, from <https://developer.android.com/reference>
- Android Developers. (n.d.-c). App Manifest Overview, 2019.09.07, from <https://developer.android.com/guide/topics/manifest/manifest-intro>
- Android Developers. (n.d.-d). Application Fundamentals, 2019.09.02, from <https://developer.android.com/guide/components/fundamentals>
- Android Developers. (n.d.-e). Create and manage virtual devices, 2019.09.02, from <https://developer.android.com/studio/run/managing-avds>
- Android Developers. (n.d.-f). Fragments, 2019.09.02, from <https://developer.android.com/guide/components/fragments>
- Android Developers. (n.d.-g). Layouts, 2019.09.02, from <https://developer.android.com/guide/topics/ui/declaring-layout>
- Android Developers. (n.d.-h). Maps SDK for Android Overview, 2019.09.02, from <https://developers.google.com/maps/documentation/android-sdk/intro>
- Android Developers. (n.d.-i). Permissions overview, 2019.09.03, from <https://developer.android.com/guide/topics/permissions/overview>
- Android Developers. (n.d.-j). Platform Architecture, 2019.08.28, from <https://developer.android.com/guide/platform>
- Android Developers. (n.d.-k). Projects overview, 2019.09.03, from <https://developer.android.com/studio/projects>
- Android Developers. (n.d.-l). Run apps on the Android Emulator, 2019.09.02, from <https://developer.android.com/studio/run/emulator>
- Android Developers. (n.d.-m). Understand Tasks and Back Stack, 2019.09.02, from <https://developer.android.com/guide/components/activities/tasks-and-back-stack>
- Android Developers. (n.d.-n). Understand the Activity Lifecycle, 2019.09.02, from <https://developer.android.com/guide/components/activities/activity-lifecycle>
- Android Developers. (n.d.-o). View, 2019.09.02, from <https://developer.android.com/reference/android/view/View.html>
- Android Developers. (n.d.-p). ViewGroup, 2019.09.02, from <https://developer.android.com/reference/android/view/ViewGroup.html>
- Android Developers. (n.d.-q). Volley overview, 2019.09.02, from <https://developer.android.com/training/volley>
- Android Developers. (n.d.-r). Android Jetpack, 2019.09.09, from <https://developer.android.com/jetpack>
- Android Developers. (n.d.-s). AndroidX Overview, 2019.09.09, from <https://developer.android.com/jetpack/androidx>
- Android Developers. (n.d.-t). Guide to app architecture, 2019.09.13, from <https://developer.android.com/jetpack/docs/guide>
- Android Developers. (n.d.-u). android.arch.persistence.room, 2019.09.03, from <https://developer.android.com/reference/android/arch/persistence/room/package-summary>
- Android Developers. (n.d.-v). AutoCompleteTextView, 2019.09.13, from <https://developer.android.com/reference/android/widget/AutoCompleteTextView>
- Android Developers. (n.d.-x). Get started with the Navigation component, 2019.09.02, from <https://developer.android.com/guide/navigation/navigation-getting-started>

- Android Developers. (n.d.-z). Navigation, 2019.09.02, from <https://developer.android.com/guide/navigation>
- Android Developers. (n.d.-aa). Pass data between destinations, 2019.09.16, from <https://developer.android.com/guide/navigation/navigation-pass-data>
- Apache Friends. (n.d.). What is XAMPP? Retrieved 2019.10.10, from <https://www.apachefriends.org/index.html>
- Arsenault, C. (2017). Development - Exploring Both Sides - Back End vs Front End Retrieved 2019.08.10, from <https://www.keycdn.com/blog/back-end-vs-front-end>
- Badiru, A. B. (2014). *Industrial and Systems Engineering*: CRC Press, Taylor & Francis Group.
- Banerjee, S. (n.d.). PHP - MySQL Database Introduction, 2019.08.22, from <https://www.geeksforgeeks.org/php-mysql-database-introduction/>
- Berg, K. L., Seymour, Tom, Goel, Richa. (2013). History of Databases. *International Journal of Management & Information Systems*, 17(1).
- Biscobing, J. (2014). How to step up performance with Oracle in-memory database options- relational database, 2019.07.20, from <https://searchdatamanagement.techtarget.com/definition/relational-database>
- Board of Intermediate Education Andhra Pradesh. (2017). Object Oriented programming and Java Retrieved from <http://bieap.gov.in/Pdf/CSCPaperIYR2.pdf>
- Bush, T. (2018). 5 Powerful Alternatives to Google Maps API Retrieved 2019.08.21, from <https://nordicapis.com/5-powerful-alternatives-to-google-maps-api/>
- Castelano, C. R. (2015). História dos Bancos de Dados, 2019.08.16, from <http://castelano.com.br/site/aulas/bd/Aula%2001%20-%20Introdu%C3%A7%C3%A3o.pdf>
- Citrus7. (n.d.). O que é front-end e back-end? Retrieved 2019.07.20, from <https://citrus7.com.br/artigo/o-que-e-front-end-e-back-end/>
- Conceito de. (2012). Conceito de mapa Retrieved 2019.08.29, from <https://conceito.de/mapa>
- Costa, G. (2019). A História das Bases de Dados - O Início, 2019.08.16, from <https://pplware.sapo.pt/software/a-historia-das-bases-de-dados-o-inicio/>
- Crockford, D. (2018). *How JavaScript Works*: Virgule-Solidus.
- Damas, L. (2017). *SQL* (14 ed.). Lisboa: Lidel - Edições Técnicas, Lda.
- Dana, P. H. (1997). Global Positioning System (GPS) Time Dissemination for Real-Time Applications. *12*(1), 9-40. Retrieved from doi: <https://doi.org/10.1023/A:1007906014916>
- DB-Engines. (2019). DB-Engines Ranking of Relational DBMS, 2019.08.22, from <https://db-engines.com/en/ranking/relational+dbms>
- DuBois, P. (2005). *MySQL™ - The definitive guide to using, programming, and administering MySQL 4.1 and 5.0* (Third ed.): Sams.
- Duckett, J. (2011). *HTML&CSS - design and build Websites*. Indianapolis: John Wiley & Sons, Inc.
- Duckett, J. (2014). *JavaScript&jQuery - Interactive front-end Web development*. Indianapolis: John Wiley & Sons, Inc.
- Edit. (n.d.). Front-End: Conceito, Fases e Linguagens Retrieved 2019.07.20, from <https://edit.com.pt/blog/front-end-conceito-fases-e-linguagens/>
- EpiCollect5. (n.d.). Epicollect5 Data Collection User Guide Retrieved from <https://epicollect5.gitbooks.io/epicollect5-user-guide/content/>
- Farkade, A. M., Kaware, Sneha. R. (2015). The Android - A Widely Growing Mobile Operating System With its Mobile based Applications. *International Journal of Computer Science and Mobile Applications*, 3(1), 39-45.
- Frumusanu, A. (2014). A Closer Look at Android RunTime (ART) in Android L - Introduction and Architecture, 2019.08.08, from <https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l>
- Gediminas, B. (2019). What is Apache? An In-Depth Overview of Apache Web Server, 2019.08.11, from <https://www.hostinger.com/tutorials/what-is-apache>
- GeeksforGeeks. (n.d.). Front End vs Back End Retrieved 2019.09.21, from <https://www.geeksforgeeks.org/frontend-vs-backend/>
- Google Cloud. (n.d.). Google Maps Platform Pricing, 2019.09.02, from <https://cloud.google.com/maps-platform/pricing/?hl=pt-br>

- Google Codelabs. (2019-a). Android Room with a View – Java Retrieved 2019.08.16, from <https://codelabs.developers.google.com/codelabs/android-room-with-a-view/#0>
- Google Codelabs. (2019-b). Jetpack Navigation - Retrieved 2019.08.17 Java, from <https://codelabs.developers.google.com/codelabs/kotlin-bootcamp-introduction/index.html?index=.%2F..index#0>
- Google Developers Training Team. (2018). Android Developer Fundamentals Retrieved from <https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/index.html>
- GooglePlay. (n.d.-a). Epicollect5 Data Collection Retrieved 2019.08.15, from https://play.google.com/store/apps/details?id=uk.ac.imperial.epicollect.five&hl=pt_PT
- GooglePlay. (n.d.-b). QField for QGIS Retrieved 2019.08.15, from https://play.google.com/store/apps/details?id=ch.opengis.qfield&hl=pt_PT
- Gupta, A. (2019). Building an Android App using Android Architecture Components: Room, ViewModel, and LiveData Retrieved 2019.09.15, from <https://proandroiddev.com/building-an-android-app-using-android-architecture-components-room-viewmodel-and-livedata-702a0af899ae>
- Gurău, C., & Duquesnois, F. (2011). The Website as an Integrated Marketing Tool: An Exploratory Study of French Wine Producers. *Journal of Small Business & Entrepreneurship*, 24(1), 17-28. doi: 10.1080/08276331.2011.10593523
- Hostadvice. (2019). Global Web Server Market Share October 2019 Retrieved 2019.08.11, from <https://hostadvice.com/marketshare/server/>
- Jenkov, J. (2014-a). N Tier Architecture Retrieved 2019.08.11, from <http://tutorials.jenkov.com/software-architecture/n-tier-architecture.html>
- Jenkov, J. (2014-b). Software Architecture Retrieved 2019.08.11, from <http://tutorials.jenkov.com/software-architecture/index.html>
- Linuxtopia. (n.d.). Android Emulator Vol. 2019.09.02. Retrieved from https://www.linuxtopia.org/online_books/android/devguide/guide/developing/tools/emulator.html
- Maggi, P. (2018). Introducing WorkManager Retrieved 2019.08.16, from <https://medium.com/androiddevelopers/introducing-workmanager-2083bcfc4712>
- Marcus, D. (2017). PHP MySQLi Prepared Statements Tutorial to Prevent SQL Injection Retrieved 2019.10.08, from <https://Websitebeaver.com/prepared-statements-in-php-mysqli-to-prevent-sql-injection>
- McKenzie, C. (2019). Java, 2019.08.30, from <https://www.theserverside.com/definition/Java>
- McMahon, D., Seaman, Samuel, Buckingham, John. (2011). Non Profit Adoption of Websites and Websites Types. *Journal of Marketing Development and Competitiveness*, 5(6).
- MDN Web docs. (2019). HTML:Hypertext Markup Language Retrieved 2019.09.20
- Microsoft. (2012). The MVVM Pattern Retrieved 2019.09.15, from [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246\(v=pandp.10\)](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10))
- Molina, H. G., Ullman, Jeffrey D., Widom, Jennifer (2008). *Database Systems: The Complete Book* (United States Ed of 2nd Revised Ed ed.). United States: Pearson Education (US).
- MySQL. (n.d.). My SQL8.0 Reference Manual - Using Foreign Key Constrains Retrieved 2019.07.21, from <https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html>
- NGINX. (n.d.). What Is HTTP? Retrieved 2019.08.12, from <https://www.nginx.com/resources/glossary/http/>
- OpenSource. (n.d.). What is Docker? Retrieved 2019.08.28, from <https://opensource.com/resources/what-docker>
- Oracle. (n.d). MySQL, 2019.08.22, from <https://www.oracle.com/database/technologies/mysql.html>
- Pathan, I. (2017). What is the difference between Android SDK and Android Studio? Retrieved 2019.08.17, from <https://www.quora.com/What-is-the-difference-between-Android-SDK-and-Android-Studio>
- php. (n.d.-a). Prepared Statements Retrieved 2019.08.10, from <https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php>
- php. (n.d.-b). What is PHP? Retrieved 2019.08.10, from <https://www.php.net/manual/en/intro-whatis.php>

- Poudel, A. (2013). *Mobile Application Development for Android Operating System – Case: NepGuide Mobile*. Bachelor, Turku University of Applied Sciences. Retrieved from https://www.theseus.fi/bitstream/handle/10024/64719/Poudel_Amrit.pdf
- Qfield. (n.d.). Qfield Documentation Retrieved from <https://qfield.org/docs/>
- Quin, L. (2016). Extensible Markup Language (XML) Retrieved 2019.08.30, from <https://www.w3.org/XML/>
- Saini, A. (n.d.). XML in Android: Basics And Different XML Files Used In Android Retrieved 2019.09.05, from <https://abhiandroid.com/ui/xml>
- Segue Technologies. (2013). What is Ajax and Where is it Used in Technology? Retrieved 2019.08.26, from <https://www.seguetech.com/ajax-technology/>
- Sells, C., Poiesz, Benjamin, Ng, Karen (2018). Use Android Jetpack to Accelerate Your App Development Retrieved 2019.09.18, from <https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html>
- Sheffield Hallam University. (n.d.). Architecture of Web-based systems - Client Server and 2 tier Web Architectures Retrieved 2019.08.12, from https://teaching.shu.ac.uk/aces/ict/de/Web_based_systems_architectures_1_tutorial.htm
- Silva, R. (2013). *mtAndroid*. Master, Instituto Superior Técnico. Retrieved from <https://fenix.tecnico.ulisboa.pt/downloadFile/395145407945/disserta%C3%A7%C3%A3o.pdf>
- statcounter. (2019). Mobile Operating System Market Share Worldwide: 2009 - 2019, 2019.09.27, from <https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2019>
- Supalov, V. (n.d.). How Does the Frontend Communicate with the Backend? Retrieved 2019.09.15, from <https://vsupalov.com/how-backend-and-frontend-communicate/>
- Tanenbaum, A. S., Wetherall, David (2018). *Redes de Computadores*
- Segue Technologies, S. (2013). What is Ajax and Where is it Used in Technology? Retrieved 2019.07.21, from <https://www.seguetech.com/ajax-technology/>
- Techopedia. (n.d.). Android SDK Retrieved 2019.09.04, from <https://www.techopedia.com/definition/4220/android-sdk>
- TechTarget. (2009). Data Definition Language (DDL), 2019.08.19, from <https://whatis.techtarget.com/definition/Data-Definition-Language-DDL>
- TechTarget. (n.d.). APK file (Android Package Kit file format) Retrieved 2019.09.06, from <https://whatis.techtarget.com/definition/APK-file-Android-Package-Kit-file-format>
- TutorialRepublic. (n.d.-a). PHP GET and POST Retrieved 2019.08.12, from <https://www.tutorialrepublic.com/php-tutorial/php-get-and-post.php>
- TutorialRepublic. (n.d.-b). PHP MySQL Prepared Statements Retrieved 2019.09.12, from <https://www.tutorialrepublic.com/php-tutorial/php-mysql-prepared-statements.php>
- TutorialsPoint. (n.d.). PHP Tutorial Retrieved 2019.08.08, from <https://www.tutorialspoint.com/php/index.htm>
- Valade, J. (2004). *PHP 5 For Dummies*.
- Valade, J. (2009). *Php And Mysql For Dummies*.
- Varshni Dimpay. (n.d.). SQL, DDL, DQL, DML, DCL and TCL Commands
- Viradiya, S. (2018). Android Jetpack - NavigationUI, 2019.09.02, from <https://proandroiddev.com/android-jetpack-navigationui-a7c9f17c510e>
- W3C. (2008). Extensible Markup Language (XML) 1.0 Retrieved from <https://www.w3.org/TR/xml/>
- W3Schools. (n.d.-a). CSS Introduction Retrieved 2019.08.12, from https://www.w3schools.com/css/css_intro.asp
- W3Schools. (n.d.-b). CSS Tutorial Retrieved 2019.08.08, from <https://www.w3schools.com/css/>
- W3Schools. (n.d.-c). HTTP Request Methods Retrieved 2019.08.06, from https://www.w3schools.com/tags/ref_httpmethods.asp
- W3Techs. (2019). Usage statistics of Apache Retrieved 2019.08.28, from <https://w3techs.com/technologies/details/ws-apache/all/all>
- Wales, M. (2014). 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack Retrieved 2019.09.04, from <https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-Web-developers.html>

Walter, D. (2018). Android Studio Retrieved 2019.09.03, from
<https://searchmobilecomputing.techtarget.com/definition/Android-Studio>
WP Amelia Staff. (2019). Static vs. Dynamic Website: What Is the Difference? Retrieved 2019.07.20,
from <https://wpamelia.com/static-vs-dynamic-Website/>

Annexes

Annex 1 – Example of data contained in the database tables

Table 1. Example of data contained in the User table.

userID	username	user_name	password	admin
1	catarina123	Catarina Pereira da Silva	\$2y\$10\$ZZvRf1stX0Te...	1
2	maria123	Maria Pereira	\$2y\$10\$W0WeXRj4Ew...	0

Table 2. Example of data contained in the Project table.

projectID	projectName
1	Project1
2	Project2
3	Project3

Table 3. Example of data contained in the User_Project table.

User userID	Project projectID
1	1
1	2
2	3

Table 4. Example of data contained in the Species table.

speciesID	speciesName
2	Chamaeleo Chamaeleon
3	L. narbonense
4	L. vulgare
5	L. brasiliense

Table 5. Example of data contained in the Marker table

dataID	sample	marker_ID	species_ID	sex	collector	individuals	sample_observations	storage_method	storage_institution	storage_room	storage_equipment	storage_drawer	storage_observations	storage_extraction
56	location1_sample_1	Location1	2	Female	Catarina Pereira da Silva	4	Tongue: 5cm							
59	Location4_sample5	Location4	5	Undefined	Catarina Pereira da Silva	2	No observations	Dry	FCUL	Bioterio	Cabinet 1	Drawer 2		Yes

Table 6. Example of data contained in the Data table.

markerID	Project_projectID	latitude	longitude	altitude	country	region	location	date	time
Location1	1	40.732998	-5.870000	5	Spain	Castile and León	Salamanca	2019-10-27	16:31:16
Location4	2	38.833740	-9.224537	0	Portugal	Lisboa	Loures	2019-10-28	18:53:50