POST-GRADUATE APPLIED ARTIFICIAL INTELLIGENCE



**Intelligent Interfaces: Taboo Project**

Guessing game based on natural language processing

*Wouter Parys, Wissam Nasser, Charlotte Waltregny*

Lecturer: Ms. Catherine Middag

Academic Year 2019-2020

ERASMUSHOGESCHOOL BRUSSEL – CAMPUS KAAI NIJVERHEIDSKAAI 170, 1070 ANDERLECHT

TEL: 02 520 18 10

1. Contents



[1. Contents 1](#_Toc11422890)

[2. Preface and acknowledgments 3](#_Toc11422891)

[3. Abstract 4](#_Toc11422892)

[4. Legend 5](#_Toc11422893)

[5. Introduction 6](#_Toc11422894)

[6. Chapter 1 - Problem Definition 7](#_Toc11422895)

[6.1 Definition 7](#_Toc11422896)

[6.2 Elaboration 7](#_Toc11422897)

[6.2.1 Food Transportation 8](#_Toc11422898)

[6.2.2 Plant-Based Food Production 9](#_Toc11422899)

[6.3 Requirements 11](#_Toc11422900)

[6.3.1 Functional Requirements 11](#_Toc11422901)

[6.3.2 Non-Functional Requirements 11](#_Toc11422902)

[7. Chapter 2 – Solution Architecture 13](#_Toc11422903)

[7.1 Actors 13](#_Toc11422904)

[7.1.1 Knowledge Providers 13](#_Toc11422905)

[7.1.2 End Users 14](#_Toc11422906)

[7.2 System 14](#_Toc11422907)

[7.2.1 Scope 14](#_Toc11422908)

[7.3 Knowledge Base 14](#_Toc11422909)

[7.4 AI Diagnosis Engine 14](#_Toc11422910)

[7.4.1 Plant Deficiency Dataset 15](#_Toc11422911)

[7.4.2 Actor Model 15](#_Toc11422912)

[7.5 Database 16](#_Toc11422913)

[7.5.1 Relational (SQL) 16](#_Toc11422914)

[7.5.2 Non-Relational (NoSQL) 16](#_Toc11422915)

[7.5.3 Firebase Storage & Realtime Database 16](#_Toc11422916)

[7.6 Interaction Platform 17](#_Toc11422917)

[7.7 Conclusion 17](#_Toc11422918)

[8. Chapter 3 – Technical Implementation 18](#_Toc11422919)

[8.1 Software Design 18](#_Toc11422920)

[8.1.1 Domain Model 18](#_Toc11422921)

[8.1.2 Implementation Designs 20](#_Toc11422922)

[8.2 Android Implementation 20](#_Toc11422923)

[8.2.1 Design 20](#_Toc11422924)

[8.2.2 MVC 20](#_Toc11422925)

[8.2.3 Firebase SDK 21](#_Toc11422926)

[8.2.4 Glide 22](#_Toc11422927)

[8.2.5 Image Picker 22](#_Toc11422928)

[8.3 Diagnosis Engine Implementation 22](#_Toc11422929)

[8.3.1 Design 22](#_Toc11422930)

[8.3.1 Firebase Admin / Pyrebase 22](#_Toc11422931)

[8.3.2 Actor Model 22](#_Toc11422932)

[8.4 AI Implementation 24](#_Toc11422933)

[8.4.1 Anaconda Distribution 24](#_Toc11422934)

[8.4.1 AI Model 24](#_Toc11422935)

[8.4.2 Training & Testing 24](#_Toc11422936)

[8.5 Version Control & Development Tools 26](#_Toc11422937)

[8.5.1 GitHub 26](#_Toc11422938)

[8.5.2 PyCharm 26](#_Toc11422939)

[8.5.3 Android Studio 26](#_Toc11422940)

[9. Conclusion 27](#_Toc11422941)

[9.1 Future 27](#_Toc11422942)

[9.2 Multi-purpose Architecture 27](#_Toc11422943)

[9.2.1 Step 1: AI 27](#_Toc11422944)

[9.2.2 Step 2: Database 28](#_Toc11422945)

[9.2.3 Step 3: Interaction platform 28](#_Toc11422946)

[9.3 Resulting impact 28](#_Toc11422947)

[10. Critical Reflection 29](#_Toc11422948)

[11. Bibliography 30](#_Toc11422949)

[12. Appendix 33](#_Toc11422950)

[12.1 Android UI 33](#_Toc11422951)

1. Preface and acknowledgments



1. Abstract



For centuries, people have been planting and caring for indoor and outdoor plants in and around their homes. With often very little gardening and plant caring skills, the passion for succeeding in growing plants, even in urban areas, has never faded.2

Yet, the biggest challenge remains in bridging the gap in gardening and plant caring skills to successfully and enjoyably grow plants. While identifying visual changes and recognizing the illness or deficiency is a very difficult task for most people, finding the suitable remedy in a timely manner before the plant is beyond saving, is the second biggest challenge.

The goal in this bachelor thesis is to help people in their quest to successfully grow indoor and outdoor plants. This will be achieved by delivering an App to support the identification of deficiencies and/or illnesses that occur in house/garden plants that are kept in certain conditions. Typically, a plant could present one or multiple symptoms that relate to one or multiple deficiencies and/or diseases. The main problem addressed by this project, is the classification of such symptoms and diseases, which is often a very challenging task for the unexperienced eye of the average plant caretaker. To conquer this quest, this project provides a diagnosis engine, based on computer vision- a branch of AI - to analyze requests from the user and return up to three classifications in the top 3 likeliness of illnesses that it could detect. This is meant to indicate what relevant remedy is advisable, to prevent the plant in question from wilting, dying or spreading disease.

this functionality is provided to the End User in the form of a comprehensive mobile application called "PlantCare”. The approach to solve this problem consists of two essential parts, researching architectural requirements and possibilities on one hand and implementing these architectural choices on the other. This resulted in a solution architecture consisting of 3 main entities. An interaction platform in the form of an Android mobile app, an external diagnosis engine in the form of a Python webserver, and a computer-vision AI implementation that utilizes a pre-existing visual recognition model called AlexNet. This AI implementation is part of the diagnosis engine.

The main technical challenge facing this thesis lies in the integration between a mobile application and a diagnosis engine. This involves keeping data consistent across systems, communicating between the main architecture entities, achieving a sufficiently performant, and fault tolerant design.

The true idea behind this project is to provide a platform with multiple functionalities supporting vertical farming. However, the project scope is limited to the delivery of a technical base with the following scope limitations:

* The diagnosis engine supports 38 classifications spread over 14 plant types
* the app development will be limited to Android.

Therefore, this implementation will provide the end-to-end technical basis for future releases and functionality extensions to enhance the app and provide further support for the quest of successful urban and vertical farming for everyone.

1. Introduction



The basis for this thesis originally stemmed from my passion for the following three domains: Cutting edge technology, data science and ecology.

As a citizen in a large and densely populated city like Brussels, I have experienced the impact of mass transportation and consumerism first-hand. It’s hard to find a street that doesn’t have large amounts of plastic packaging or other insoluble waste lying around.

“By 2050, 68% of the world population will live in urban areas while in 2016 this was just 54.5%”28 As a large part of the world moves further into urbanization, the delivery of different goods to these densely populated areas starts to increase in unprecedented speeds. The underlying consequence of this trend is an ecological inefficiency. In such situations, victuals are often shipped from farms or other production facilities to far destinations before they are consumed. This raises the need for substantial amounts of plastic and other preserving materials or chemicals.

A solution to this problem is urban- and vertical farming. Both methods allow for plant-based products to be sourced locally in urban areas, eliminating the need for plastics, long polluting shipments and chemical preservatives. But neither methods have reached their full potential yet. In 2018, Klaus De Geyter, a student of Environmental Management at the University College of Ghent presented a thesis with the title “A comparison of the environmental impact of vertical farming, greenhouses, and food import” which is a case study for the Norwegian vegetable market in collaboration with BySpire, a vertical farm in Norway. This case study clearly defines the ecological differences between vertical farming, represented by BySpire and more traditional methods of agriculture.10

To support this urban- and vertical farming evolution, in this thesis I aim to provide a solution that helps the process of maintaining healthy indoor or outdoor plants and making this process increasingly convenient and efficient. The key benefit is putting this tremendous amount of pathological data and knowledge, usually only available to agricultural professionals, between the hands of the average plant caretaker. This intuitive medium could support not only the agricultural professional and the gardening passionate, but also the rest of the population, and encourage them in their road to botanical bliss. Not needing to be an expert of diseases and deficiencies in multiple plants allows for the streamlining of the maintenance process which is useful in an urban- or vertical farming setting.

1. Chapter 1 - Problem Definition



## Definition

As described in the introduction, the goal of this thesis is to streamline the maintenance process of indoor or outdoor plants. Originally this process requires a person with

## Elaboration

To elaborate on the problem definition, I want to explain some of the aspects that this problem encompasses, which might not be clear at this point. These aspects are not so clear because I am packaging this solution in a PoC, enabling me to make this project.

Figure 5: Water depletion in m³ equivalents for the three production methods 10

Figure 9: Terrestrial acidification in mole of H+ equivalents for the three production methods 10

## Requirements

After understanding the problem on a theoretical level, the technical challenges within this problem are identified on a technical level. They are then translated to requirements that must be considered in order to deliver a quality product. A distinction remains between functional and non-functional requirements.

### Functional Requirements

This represents a method that gets an input and provides an output or solution. For example, a beverage dispenser that takes money and a typed input number to provide the desired drink as output. It essentially describes the method to achieve a user goal for a target user-base in a certain usage environment.7

For this project, the first functional requirement is to provide the correct illness or deficiency classification based on the user’s provided information. The target user-base is home users that own a plant. The operational environment requires a user to operate an Android smartphone and have an internet connection. A second functional requirement would be to provide correct remedy advice based on the illness or deficiency classification. As described in section 6.3, this second requirement is not part of the scope of this project.7

#### Recognition

The difficulty here lies in the classification of diseases and deficiencies that appear differently based on the plant type or have similar symptoms. Identification could be possible for a phytopathologist but would take a lot of time and require detailed images. This approach would also be very expensive as a single expert can only handle a certain amount of load. So, to support growing demand more experts would be needed. On top of that, these experts are expensive to employ for this purpose. In conclusion, this process is not efficient.

To drastically improve this process, a streamlined approach is needed. The field that has proven to enable a streamlined approach in such cases is computer vision, the study of analyzing images. Computer vision has two distinct approaches, one uses hand crafted algorithms and the other uses machine learning, these approaches can also be combined. In section 7.4 more information will be provided on the choice and reasoning.

### Non-Functional Requirements

This type of requirement includes all requirements that functional ones don’t cover, usually involving reliability, performance and many more. These requirements are often specified in a measurable way. The important non-functional requirements for this project are described in the following subsections. 7

#### Reliability

For a product to be truly useful, it must be dependable, provide consistent results, and maintain a safe and operational status. This is reflected in the following traits. 7

#### Accuracy

The results provided by the solution must be accurate enough to be relevant. In other words, if the results are only rarely accurate it will lead to more problems than answers. This renders the tool almost useless. For example, even if an AI implementation would provide a larger than 70% accuracy in its detection of plant diseases, a method must be implemented to handle this known inaccuracy of the remaining 30% of cases. 7

#### Security

This trait plays an important role, as the resulting solution provided by this project is intended to be used by many people. Thus, it will need an authentication system. Besides that, users will be uploading personal data. This data needs to be kept private. 7

#### Fault Tolerance

This quality is essential to the continuity of the provided solution or service. In a complex system, there are many ways in which exceptions or failures of sub-components might occur. These subcomponents together provide the solution or service a user desires. This raises the need to make sure that this system can keep operating even if such exceptions occur. Downtime can be prevented by handling these exceptions in code, by optimizing the backend for fault tolerant behavior and by implementing a fault tolerant software architecture or design pattern. 7

#### Performance

This plays a key role in the success of this project. If a solution takes a whole day to solve the problem at hand, the user might try to solve it themselves. In the on-demand age, it is important to provide quick answers. This has become a standard across the IT industry. Swiftness is a generally desirable trait, no matter the application. 7

#### Concurrency

As the resulting service from this project is intended to support thousands of simultaneous users. It is critical for performance to make sure that as many things as possible, are executed in parallel. Just like “fault tolerance”, good parallelism requires a concurrent backend implementation or design pattern to be utilized. 7

#### Scalability

In a fast pace industry, a lot depends on adaptability. Great agility helps in adapting based on demand. Services that suddenly experience major growth, and thus a huge increase in traffic or load, might lose potential customers due to a lack of robust scalability. This lack of scalability results in a decrease in reliability or even downtime. 7

#### Supportability

In the last paragraph we mentioned the IT industry as being fast paced. This characteristic means that the demands and needs of an end-user are shifting quickly and thus, expandability needs to be built into the solution architecture. The design must be flexible enough to allow changes to be implemented with relative ease. 7

#### Continuous Delivery

In the case of this project, the area’s that could be expanded are mainly recognition and platform support. Recognition can be expanded by supporting new diseases or plants. Additionally, platform support can be expanded by supporting new interfaces like web, iOS, Android, OSX and/or Windows. The design must allow incremental expansion to take place. 7

#### Usability

Lastly a consideration needs to be made for an appropriate user-friendly design. Usability has a paramount effect on the user experience and satisfaction. The solution needs to be tailored to the circumstances of the end-user to increase user acceptance ratings. 7

1. Chapter 2 – Solution Architecture



To meet the requirements of the overall solution described in the previous chapter, an architecture was visualized that could provide the functionality that is needed, e.g. the solution architecture. This visual architecture model acted as a guide in the developing and researching process and was revised multiple times when a flaw or a more efficient design was discovered. It consists of 2 high-level parts, the actors and the systems. The systems can be further divided into 3 main entities within the project scope that operate independently:

* The AI Diagnosis Engine (AIDE)
* An interaction platform represented by the PlantCare mobile app
* The database that acts as communications channel between the other two

In this chapter I will detail each individual component of this architecture.

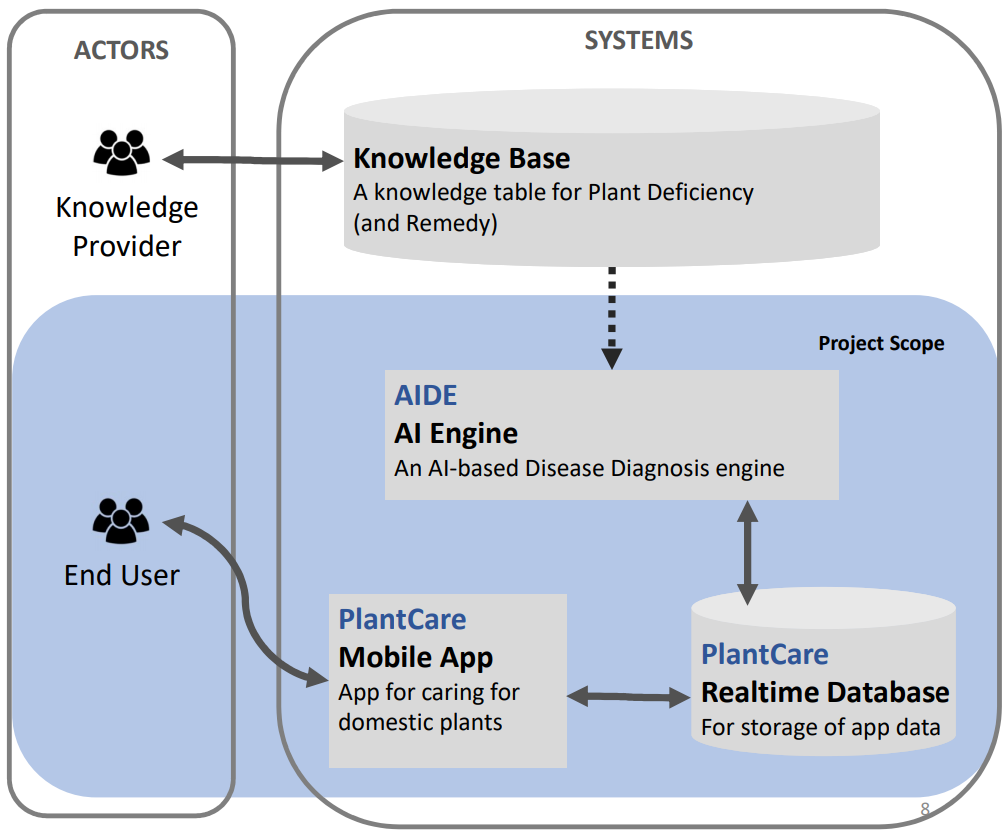


Figure 10: Solution Architecture

## Actors

The first high-level part of the architecture represents external entities that interact with a system and may or may not have a goal in this interaction. In my architecture, they are divided in 2 different roles, the “Knowledge Providers” and “End users”. The following 2 paragraphs will describe these roles in further detail. 7

### Knowledge Providers

A secondary actor, this type of actor aids the system so it can fulfil the user goal of a primary actor e.g. the end user. In this case the KP is an agency, company or expert that provides the remedying advice and knowledge that the user often lacks, to the system. This happens in close cooperation with a system admin who inserts this data into the database. In turn the system can now present the appropriate information to the end user, in a case where this information is required. For example, when a user detects a disease on his plant and needs the appropriate remedying advice. Finally, a feedback system can be utilized to improve on individual advices.

### End Users

A primary actor, they want to achieve a user goal by acting upon the system(s). In this case the end user is a person with a smartphone, who wants to find out what disease or deficiency his/her plant has, by operating the system that was created to do this, e.g. the mobile application resulting from this thesis.

## System

The system encompasses all the functional parts that together, supply a service to the primary actor. The system can interact with all kinds of actors or can be observed by actors, through a medium. 7

### Scope

As seen on figure 10, the blue part of the solution architecture represents the project scope. This part of the architecture contains the functionality of the system that I will implement for my bachelor thesis. I excluded the knowledge provider and knowledge base from this scope, because I don’t have the necessary expertise or “knowledge” to provide accurate remedying information, as this is not my field of study. However, it is very important that a trusted source is used. Supported by scientific testing and verification. If a system like this, is to be used for large vertical farming use-cases, then it becomes necessary to regulate these knowledge providers to avoid wasted product and damage to the plant.

## Knowledge Base

This is the first functional component of the system that I will cover. It’s a dictionary containing all the diseases of plants the system supports, and their remedying advice. It is used to supply the system with relevant data so it can be presented by the system to the end-user in case that specific information is needed. Furthermore, as stated before this functional component will be supplied by credible knowledge providers. This dictionary will be stored in the main database as a separate collection. To demonstrate my project, it will be supplied by generic data, as this functional component is not part of my project scope. (see previous paragraph)

## AI Diagnosis Engine

This functional component is a high-level representation of multiple smaller components. It resembles the AI component of this project. This component is responsible for analyzing pictures of plants that are sent by the end-user. These messages will be treated as analysis requests from the end-user in their interaction with AIDE. This engine interacts with the rest of the system by communicating through the database using REST API calls and reading the database for new analysis requests. Next, these requests are processed, and a log is made of the detected classifications. These classifications each have their own probability, indicating how likely AIDE estimates their accuracy. The precision in this process strongly depends on how well the computer vision AI used by this system, is trained. It is based on the amount and quality of data that is used to train it. In the next few paragraphs, I will further clarify these individual sub-components.26

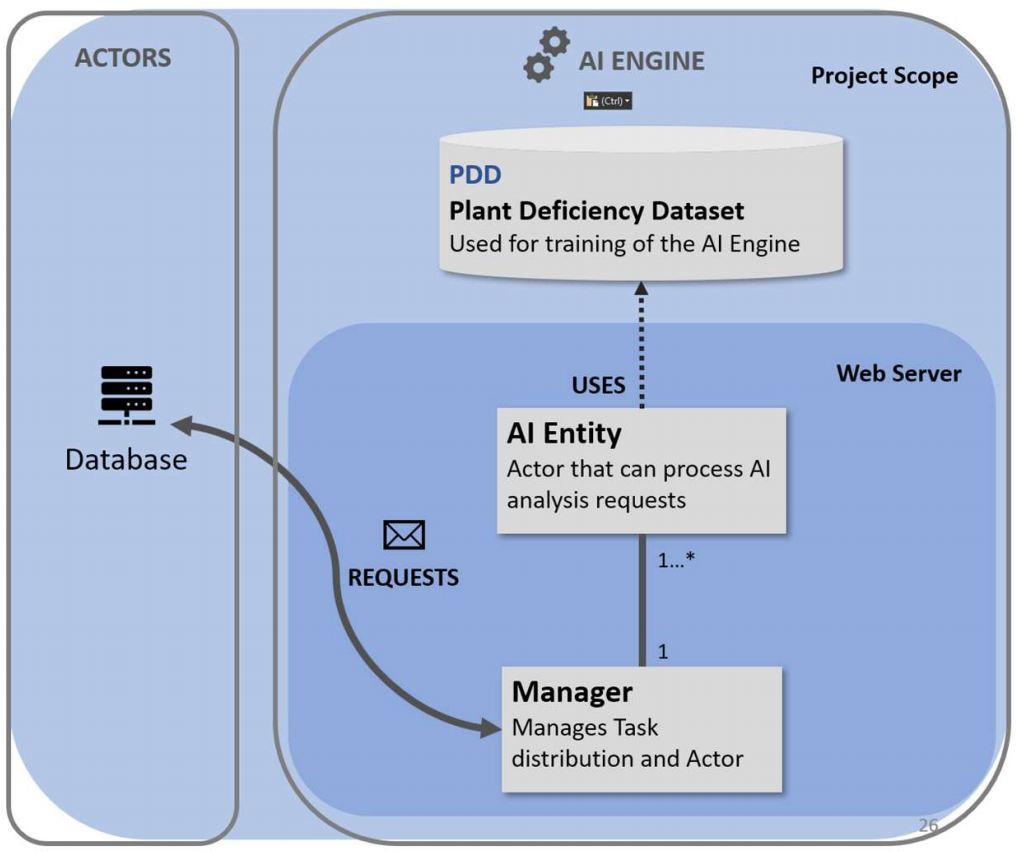


Figure 11: AI Engine Architecture

### Plant Deficiency Dataset

The first sub-component of AIDE that I will cover is also the most critical part. The PDD (Plant deficiency dataset) is a dataset with labelled images representing different disease-plant combinations. The quality of the PDD is heavily reliant on the quality and quantity of labelled pictures it contains. This dataset is eventually used to train the AI model that will be used in the following sub-component to process analysis requests from the end-user. 26

### Actor Model

This sub-component represents the engine behind AIDE. It is an architectural model used for concurrent computation. This pattern has been used by Microsoft and Twitter for fault tolerant computing and scalability. The way this pattern works, is by having multiple independent computational entities, e.g. actors, controlled by a single managing entity. The manager does not control these actors directly but through messaging and task distribution. Each of these actors have their own private state which can be observed by the manager. These actors can create more actors, make local changes, send messages and decide how to answer the following received message. This model is async-, concurrent- and scalable by design. It also has fault tolerant capabilities if implemented correctly. This is because the manager can observe if an entity is taking longer than normal or has crashed while computing. It can then react by restarting the actor or taking other actions. In my project these actors represent my AI analysis functionality, that is trained by the PDD. Whenever an analysis request is found in the database by the manager, it distributes these tasks among the AI actor entities. 25, 16

As you can see in figure 11 on the previous page, this model will run in a web server environment. It is also possible to do this natively, but I found it interesting to do this externally so that it can be offered as a service. On top of that, if I improve the AI functionality by adding a new plant for example, the user doesn’t have to update their app to enjoy these improvements.

## Database

In this project the database plays a key role in the communication between the user interface of the system and the external analysis engine. As the user uses the system, it will save the data objects that the user creates. These data objects might contain “analysis requests” which are meant to be processed by the AI Diagnosis Engine. All these interactions represent a read, delete, insert and/or update operations on the database. In the next three paragraphs, I will go over the type of databases available to me and which one I eventually picked for this project.

### Relational (SQL)

Examples - MySQL, Microsoft SQL Server, SQLite, Oracle SQL, ...

This is the most known type of database. This type of database takes longer to fully configure and deploy compared to non-relational. However, once configured and deployed it is a powerful tool that can handle complex queries and complex data structures can be used as a result. The reason for this long configuration time is because the user needs to define a database schema (database structure) that the data must adhere to. This requires the developer to design a robust and accurate design from the start as it is time consuming to make many minor changes.14

### Non-Relational (NoSQL)

Examples - Firebase, MongoDB, Redis, ...

This type of database is not as well-known as the previous type. But opposite to relational databases, it allows the developer to rapidly configure a working database system with little design or upfront work. The drawbacks of this kind of database are related to the non-relational nature of the query language. Where data is saved as a tree of key-pair values, in graph stores, documents, or wide column stores. This storage system is document based, where RDBMS is based on a table with indexed rows. Those differences make NoSQL better in storing hierarchical data. Designing a robust hierarchical structure makes it easy to manipulate big data. Yet, you don’t need to do this from the start. This type of database does not have a database schema, this means that it is highly adaptable and ideal for rapid development. (Google Inc., n.d.)14, 20

### Firebase Storage & Realtime Database

This NoSQL database developed and provided by Google Inc. is the chosen database for this project. The reason this database is uniquely suited for this project, is because of the extensive support across many platforms, seamless cross-platform synchronization, cloud access, file storage, authorization service, remote configuration, offline sync. These features are ideal for my mobile application and satisfy many technical requirements.12

The remote configuration function allows for making changes to the app and add functionality in the backend without needing to publish a new update. For example: If the AI model would support new plants or diseases. A plant type could be added to the database. This would make this option available in the mobile app, for users to select when adding a new plant to their collection. The high adaptability of Firebase enables you to adjust the data architecture many times with very little configuration changes. This is highly advantageous when developing a software architecture from scratch. 12

Firebase also satisfies nearly all technical requirements, allowing for scalability in the form of rapidly scalable services, for security in the form of an internal authentication service based on oAuth 2.0 and server-side access rules, for supportability in the form of remote-configuration capabilities and for fault tolerance in the form of server-side validation. 12

All these factors combined with the fact that the data architecture for this project is not too complex, were convincing enough to use firebase database for this project.

## Interaction Platform

Coming back to the solution architecture, the last component is the “Interaction Platform”. This functional component represents the medium through which the End-user will interact with the system. My architecture is made in a way that any digital medium should be able to interact with the system. In my implementation this medium is a mobile application called “PlantCare”, which is developed during the implementation phase of my thesis. This medium interacts with the rest of the system in the same way as AIDE, through REST API calls to the database. Whenever a user wants to use the system, all they need to do is provide a picture of the plant to the medium and the medium will make an analysis request entry in the database. After which AIDE will read this request and send results back. Following this response, the medium will present these results to the user.

## Conclusion

This concludes the solution architecture. It is meant to satisfy all technical requirements to an acceptable extent for this proof of concept. In the following chapter the technical implementation of this solution architecture will be explained. This technical implementation is built to represent the solution architecture. It exists of 3 critical parts. The Android -, Diagnosis Engine - and AI implementations.

1. Chapter 3 – Technical Implementation



A good idea is just that without a good implementation. To achieve a representative solution implementation, an iterative process is employed. This iterative process allows for reflection, redesign, and adjustment along each step. Thus, directional changes can be made before technical dept reaches critical mass. Agile was used as best practice in the development phase, but not consistently followed due to the nature of a thesis overlapping with other courses. During each iteration, time was spent on each functional level, depending on the stage the project was in.7

## Software Design

Before going into the specific solution implementations, the domain objects involved and how they interact were visualized in a domain model. These domain objects are the necessary elements involved in tackling the problem. They consist of users, plants, plant types, diseases, logs, classifications. Figure 14 below shows the visualization of these elements and the interactions between them as a domain model. However, this model was altered throughout the project and does not represent the final version.7

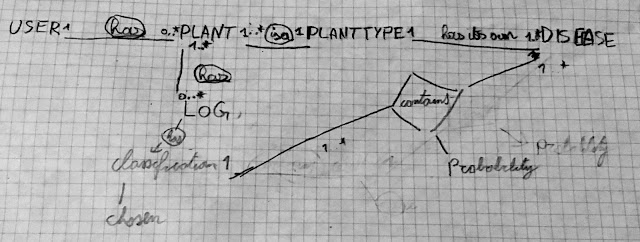


Figure 12: Sketch of problem domain model

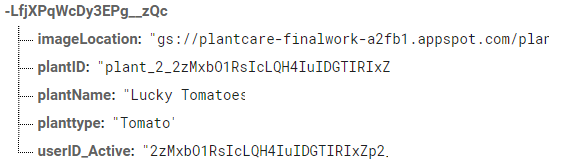
### Domain Model

It is the base for the data structure and design of the implementation. Since this model has such an important role, it is necessary to think this model through. Making a change in this step of software development can cost a lot of time as the whole system is built on top of it. This means that most of the time a redesign of this model influences many functional levels of the software. Therefore, a lot of code must be written or changed to support this redesign.7

#### User

The first domain object that is part of many problem domains is the user entity. This entity represents the end-user and their related information. It also represents the input provider of the system, trying to achieve a user goal based on interactions in the problem domain. This user has one relational link with the PLANT domain object. As the user can own none or more plants in this implementation.

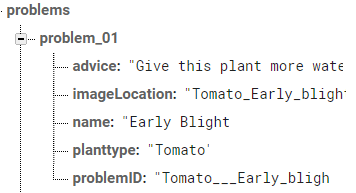
#### Plant

The next entity is the center piece of the model. It represents a plant that is owned by a user, as seen in the previous paragraph. It also has relational links with the “plant type” and “log” entities. The link with plant type implies that one or more plants represent a single plant type. Furthermore, the link with log indicates that one or more plants can have none or more logs. The plant entity includes information that describes it, consisting of a name and image URL.

#### Plant Type

This entity is self-explanatory. It represents the type of plant a plant can be. For example, a plant can be a “Tomato plant” or a “Grape plant”. It also has a relational link other than the one with a plant entity, explained in the next paragraph.

#### Problem

An entity that represents the disease or deficiency a plant can have. However, in the final model it is related differently. Here, one or more problems are related to a single plant type. For example, a “tomato plant” type can have the disease “early blight” or “spider mites”. The other relational link of a problem is described in the next paragraph. Furthermore, a problem contains data that describes it, consisting of a name, images location, and remedying advice. The name of this domain object is not perfect as it can resemble both diseases/deficiencies and healthy plants, not only problems.

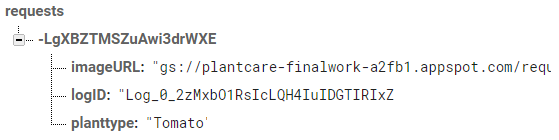
#### Classification

An entity that represents a prediction of a problem. For example, a classification is an instance of the “early blight” problem. It’s relational link with a log is that up to three classifications can be part of a log. This log entity is explained further in the next paragraph. In short, a single classification object is used as a link between a log and a problem.

#### Log

This entity represents a time-based snapshot of a plant’s state. It contains up to three classifications representing problems or the lack there of. The user that created the log for a plant compares and chooses the correct classification based on similarities of symptoms and problem information once these are available. This information can be gathered through the classifications as they link a log with a problem. After choosing the classification, it becomes the “chosen classification” for this log. For example, a user creates a log for his tomato plant. The log is processed and contains three references to classifications. The user compares these classifications with the symptoms he sees on his plant and picks the one with the most similarities, “Septoria leafspot”. The log now has a “chosen classification” that references to the problem “Septoria leafspot”.

#### Request

This final entity is used to pass a picture URL and a log reference to the AI diagnosis engine described in section 6.5. This is used as a request to process a new log and provide it with up to three classifications for the user to compare and pick from. As soon as this request has been processed by the AIDE it is deleted. How this works exactly will be explained in Chapter 3’s “AIDE Implementation” sub-chapter.

### Implementation Designs

Both implementation that were created for this bachelor thesis use parts of the domain model as reference to interact with each other. These implementations have their own internal software design patterns as they fulfil different tasks within the solution architecture. These individual designs are covered in their related chapters.

## Android Implementation

In the second chapter, the last of the architectural components was the “Interaction Platform”. This component is represented in my thesis as an Android mobile app called “PlantCare”. In the following paragraphs the composition of this mobile app will be described in further detail.

### Design

To develop a good-looking app that is also intuitive to use and user-friendly, it is important to spend some time on a good use case model.6 In the case of this implementation, there are not that many different use-cases. Instead, a UI flow was created. This allows some time to be saved on details, as the design will change and expand throughout the development process. This initial implementation design is not critical as it is highly adaptable and is modified or expanded on a function by function basis during implementation. In figure 15 on the next page, you can see the initial diagram.

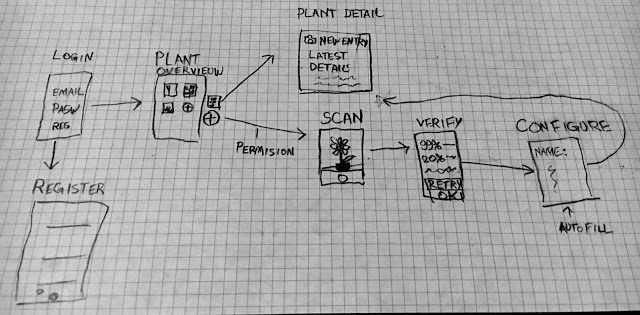


Figure 13: Sketch of UI flow for PlantCare app

### MVC

The software design pattern that is most used in android development, is the Model View Controller pattern. This pattern allows the developer to split the software into three functional layers that can only talk to the layer below them. It will be the main pattern for this android implementation. 21

#### View

The top layer of this pattern is the “view” layer. This layer consists of all the layout elements and dictates their positioning and appearance. Most of the front-end work happens in this functional layer. In android this layer is comprised of xml layout files. The elements in this layer are available for manipulation or observation by the layer described in the next paragraph.21

#### Controller

The controller is the central layer. It is responsible for the wiring of the layout elements, dictating how they are connected and behave. It also turns input on the view layer into data manipulations in the model layer below it. This layer represents the brain of your application and is written in Java using the Android Software development kit. This kit provides ways to map functionality to supported android devices, based on the target version that will be developed for. 21

#### Model

The data layer is an implemented representation of the domain model that was covered in section 7.1. This layer defines the domain objects, their variables, and relations with each other. It also handles network or database API. This keeps the interactions in the layers above structured and coherent. In layer consist of Java Classes, each representing one domain object. 21

### Firebase SDK

As described in section 6.6, firebase is the chosen database, storage, authentication and remote configuration tool for this project. In android development, this is made possible using the Firebase Software development kit for android. This kit provides abstraction to easily implement communication with firebase services.12

#### FirebaseUI

FirebaseUI is an open source framework that provides another layer of abstraction above standard firebase sdk to make certain functionalities even easier to implement. For instance, an Android RecyclerView with special FirebaseRecyclerAdapter that makes loading firebase data into a RecyclerView much easier to implement.24

#### Authentication

On top of fast and reliable database services, firebase offers authentication services. The authentication service can be used to securely log-in and register users as soon as they want to use the app. There are a few different forms of authentication, including email & password -, federated identity provider -, phone number -, anonymous and custom authentication. For this project, email and password authentication was chosen. This also stores the personal information of the user in a secure way and satisfies the security requirement described in 5.3.2. 12

#### Storage

Even more useful for my implementation is the “storage” service, firebase offers. This allows me to easily upload and download files that are stored in the Google cloud storage system. The syntax for this is almost identical to firebase database. In this project I use this service to pass images between architectural components and store them.12

### Glide

This popular opensource android framework is a media management and image loading utility that handles media decoding, memory and disk caching, and resource pooling. It makes it easy to download and load images into Android ImageViews and is used in this implementation for that purpose.17

### Image Picker

This library makes it easy to select images from the gallery and camera. It is used to pick images from local device storage or take new images using the device camera, that need to be uploaded and stored using firebase storage.11

## Diagnosis Engine Implementation

This implementation is another critical part of this project. It is implemented as a Python server. This python server uses an AI implementation to process analysis requests, made by users of the Android app “PlantCare”. Python was chosen as programming language because most AI implementations are written in python. This allows for easy integration with an AI implementation. The composition of this python server will be described in the following paragraphs.

### Design

In contrary to the Android implementation, this implementation has no need to be intuitive or user-friendly. It operates in the background and is never directly interacted with. A system admin can turn it off or on and a developer can expand its capabilities. This implementation requires scalability, concurrency and fault tolerance (section 5.3). These requirements are achieved using the actor model explained in section 6.5.2.

### Firebase Admin / Pyrebase

This is the Firebase library for server-side python applications. Just like the regular Firebase SDK described in 8.2.4, this library provides abstraction in python to easily implement communication with Firebase services. In this implementation, Firebase is used to send results back to the database and check for new analysis requests.

Pyrebase was first used to implement this functionality. However, during development this library was deprecated. It had a flaw that would cause this implementation to stop listening for new analysis request provided by the end-user after running for 30 minutes. This was caused by the lack of a re-authentication system as the services required the user, e.g. diagnosis engine, to send a refresh token after the 30-minute mark. Hence, the switch was made to the new official Firebase Admin SDK provided by Google. The new framework did not require many adjustments to be made and solved the problem.6, 12

### Actor Model

To implement this model in Python, a library called Pykka is used. It was inspired by the Scala framework “Akka”, which is a powerful implementation of the actor pattern. Pykka however, does not support as many features as Akka. It is not meant to be a python port of this framework but is capable enough for this implementation.16, 25

The actor model has a few basic rules in Pykka: 16

* An actor is an execution unit that executes concurrently with other actors.
* It does not share state with anybody else, but it can have its own state.
* It can only communicate with other actors by sending and receiving messages, if it has their address.
* When it receives a message, it may take actions like:
  + Altering its own state, e.g. so that it can react differently to a future message
  + Sending messages to other actors
  + Starting new actors
* It only processes one message at a time. Thus, a single actor does not give you any concurrency, and it does not need to use locks internally to protect its own state.

My implementation using this framework resulted in two important classes, which I will describe below.

#### Threading Actor

This implementation by the Pykka API is a class that provides the basic actor capabilities. This actor is not the same as the actor described in section 7.2. It refers to a class which, when initialized, is executed by its own thread i.e. a Python threading.Thread. Moreover, when it is asked for a response on a message as part of its implementation, it provides a “Future” object which is a thin wrapper around a Python queue.Queue. This object represents a value that will be available in the future. The value of this “Future” is calculated as soon as its “get” method is called. My two classes expanded on this base actor class to include the custom functionality necessary for this project.16

#### Aid Actor

The first custom actor class represents the worker. Its goal is to process analysis requests parallel to other workers as they execute on their own threads. To do this, it includes the AI implementation explained in section 8.4. When it successfully identifies up to three diseases that match the same plant type as the request indicates. The diseases that are also part of the top 3 results in its analysis are returned encapsulated in a Future.

#### Actor Manager

The second custom actor focuses on creating and managing these aid actors. It distributes incoming requests for analysis among its initialized aid actors. When one of its aid actors returns a request result, it shapes these results into up to three classifications and places these on the database. The log related to that request is then updated by this actor, with references to up to three classifications. The user can compare these classifications on the PlantCare app with their own observation and pick the one that matches best as “chosen classification” for that log. However, if the aid actor returns no values at all, then the log is updated to trigger the retry system. The user can then choose to take a new picture and retry the analysis process.

## AI Implementation

This component represents the core of the diagnosis engine. Together with the diagnosis engine implementation it forms the AI Diagnosis Engine or AIDE. It includes the pre-trained AI model. This model allows the diagnosis engine from the previous section to calculate the disease prediction rankings. It also handles the image pre-processing of incoming requests. The following components are involved in this process.

### Anaconda Distribution

This tool is a python package manager and provides the ability to easily manage, test and deploy development environments.1

#### Pytorch

One of those packages is a popular open-source deep learning platform for Python. one of the most popular among AI researchers/developers. It allows for the development, training and testing of AI models using dedicated graphical processing units (GPU) or central processing units (CPU). This framework was used to train a pre-existing model well known for image classification called AlexNet. This pre-trained model is then implemented into a class that is used in the diagnosis engine implementation.27

### AI Model

The class that resulted from this implementation was created as part of the diagnosis engine to encompass the AI implementation. It includes the pre-trained model described in the previous paragraph.

#### AlexNet

This convolutional neural network designed by Alex Krizhevsky and published with Ilya Sutskever and Krizhevsky’s PhD advisor Geoffrey Hinton, is a powerful implementation for large scale visual recognition. It was used as model for the AI resulting from this implementation. By using this general model, time was saved in the designing of a new model from scratch. Instead, this pre-existing model was trained with the relevant data to recognize and classify plant diseases using leaf images.9, 18

The reason for the implementation of the retry system described in section 8.3.2 – “Actor Manager” is related to this component. This model is not perfect, depending on the data it learned from, it can have biases for multiple visual markers. Thus, a failsafe must be put in place. In this implementation there are three such fault tolerant systems to tackle this uncertainty. The first is a result filter that only allows matching plant types to pass through. The second measure loops the request through another AI analysis process if the expected plant type was not found in the top 3 results the first time it was processed. The last failsafe is activated if this previous loop resulted in another plant type mismatch among the top 3 results. It resembles the retry system described earlier. 9, 18

### Training & Testing

To achieve this, a separate open-source Pytorch project was discovered that does exactly that. It represents the implementation of the chapter “Deep Learning for Plant Diseases: Detection and Saliency Map Visualization” in the book “Human and Machine Learning, 2018”. It takes different existing models and trains these to create an accuracy over time chart representing the different results achieved by different models in classifying plant diseases. As shown in the figure below, AlexNet achieved good results in this implementation while needing little time to train. On top of that, it was easy to work with for developers new to the field. Hence, AlexNet was used for this project. The code of the original project was altered to only train AlexNet using a dataset provided by PlantVillage and output this trained model. This trained model is then used as part of the AI Model class in the diagnosis engine implementation in order to provide the AI functionality. 3, 9, 15, 18

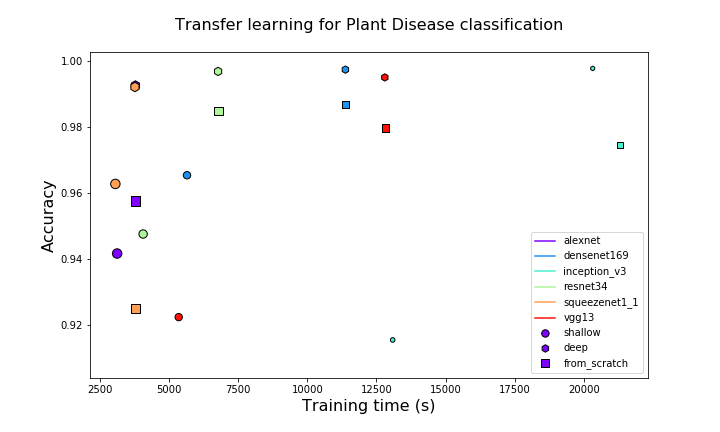


Figure 14: plant disease classification scores3

|  |  |
| --- | --- |
| Plant type | # of distinct classifications |
| Tomato | **10** |
| Strawberry | **2** |
| Squash | **1** |
| Soybean | **1** |
| Raspberry | **1** |
| Potato | **3** |
| Bell Pepper | **2** |
| Peach | **2** |
| Orange | **1** |
| Grape | **4** |
| Corn | **4** |
| Cherry | **2** |
| Blueberry | **1** |
| Apple | **4** |

#### PlantVillage Dataset

The open-source dataset used to train AlexNet is made available by this R&D unit of Penn State University. it includes 38 classifications of diseased and healthy plants. A background classification was added from Stanford’s open dataset of background images in order to make a distinction between leaves and their image background. The table on the right described the supported plant types in this implementation and the number of distinct classifications these individually represent. 8, 22

Each classification included more than 1000 images for training, of which 20% was used purely for validation. This testing resulted in an accuracy of approximately 91%, which was good enough as PoC for this project. Further improvements could be achieved by increasing the size of the dataset. This can be done by using the data the user provides while using this implementation.19

## Version Control & Development Tools

To make the previous implementations and the cheer scope of this project possible and manageable for a single student, a VCS was leveraged to manage progress in development. Furthermore, two important IDEs where utilized to code for Java in the Android implementation, and Python in the Diagnosis Engine and AI implementations.

### GitHub

This free hosting website was used for version control. It uses the git open-source VCS enabling a combination of both local and centralized version control. This allows for easy and efficient management of progress in development.

### PyCharm

This IDE made by the software company JetBrains, is used for Python development. It integrates nicely with Git version control and provides robust code completion and assistance.

### Android Studio

This is the official IDE for Google’s Android operating system. It is built on JetBrains’ IntelliJ IDEA and was designed specifically for Android development. Just like PyCharm it integrates nicely with Git and provides robust code completion and assistance. On top of that, it enables android App simulation for testing.

1. Conclusion



Recent developments in computer vision and artificial intelligence have made it possible to achieve classifications of complex patterns in images. These developments have made the successful implementation of this solution architecture achievable. My implementation does not represent a production-ready product as it is a PoC. However, it does prove that a product resulting from this solution is achievable. With this implementation it is safe to say that all technical requirements have been satisfied to an acceptable degree for this thesis.9

## Future

To achieve a production ready product a few improvements must be implemented.

1. Legal Compliance in general and specifically with regards to privacy/integrity (GDPR) and Data Storage laws must be researched and implemented.
2. A feature must be implemented in the AI implementation that allows for live training to support continuous improvement of the AI. This live training will lead to ever increasing accuracy.
3. The number of supported plants must be increased to be viable for more users.
4. Automatic identification of plant type can be implemented; this functionality is already available in many Apple App Store and Google Play Store apps.
5. The external python server can be containerized and deployed in different regions for low latency. Load balancing can then be applied to stabilize usage surges. Lastly and most importantly, accurate remedying advice must be provided from a reliable source as this is outside the scope of this thesis.

If these requirements are met, then this implementation could be considered ready for public use. After deployment many more expansions and improvements can be made to increase the user-base even further. Supporting additional interaction platforms like web, iOS, Mac, and Windows will surely enable this increase. The increasing trend of urbanization, urban farming and vertical farming will also have a positive influence in the need for this type of solution.

Due to the loosely coupled diagnosis engine implementation, it is also possible to treat it as a microservice. 23 If a licensing system is implemented it is possible to license the use of this engine to third parties.

## Multi-purpose Architecture

A diversification can also be made in Use-case as the solution architecture is made to not only support plant disease classification, but any kind of functional classification. It does this in a scalable and flexible way. A few minor modifications must be made to make this switch possible in the current implementation. However, this can be avoided by increasingly generalizing this implementation. The following steps must be taken to switch the use-case of this architecture:

### Step 1: AI

First a new AI model must be trained. Therefore, new data needs to be collected or retrieved in order to train the model for a new goal. When this is done, 20% of the collected data needs to be reserved for validation. Before training can start, an application needs to be created using TensorFlow, Pytorch or other equivalent AI frameworks. In this application the data must first be imported, the number of classifiers, the training length and hyper parameters must be set. Finally, the data must be prepped for training. When a sufficiently precise accuracy is reached, the trained model is exported as “.pt” file and imported in the diagnosis engine. If an existing service like plant disease recognition is already present in this engine, the engine is easily modified to make a distinction between the type of requests. This allows it to send requests to the relevant AI implementation for processing.

### Step 2: Database

Next, the necessary related data must be added to the database. For example, in this implementation this included all the distinct problem information related to supported classifications. In the case of animal recognition, it would be animal information.

Furthermore, if the data format for analysis requests is not generalized or generalizable over all implementations, then for SQL the database schema would need to be updated. This is because a new type of request format must be added to its database schema. For NoSQL this is not required, as it has a dynamic database schema.

### Step 3: Interaction platform

Lastly a new or generalized interaction platform must be created/used based on the use-case. In this implementation this represents the “PlantCare” app. This app is case specific and cannot be used for other use-cases. Thus, a new use-case requires a new platform. However, it is possible to create a generalized platform that can display all kinds of data and support multiple request and/or classification types.

## Resulting impact

A conclusion cannot be made on the impact of this implementation on the case described in the introduction of this thesis, as it has not yet released for public use. Thus, the impact of the implementation cannot be researched. Neither is it possible to conclude if this solution would truly help most people to care for their indoor and/or outdoor plants. To make such conclusions, the app must be tested extensively, and the user experience must be researched over an acceptable period.

1. Critical Reflection



As mentioned in the preface, a thesis project is a learning process and a lifetime opportunity to get out of one’s comfort zone and boost one’s ability to learn and grasp new fields in a very short time. One’s capability to independently learn, discover, think out of the box and innovate tremendously increases. With this precious capability, one is ready to face and proceed in the professional (and academical) world. What a person does with this capability is up to their ambition, personal interest and motivation.

I purposefully treaded out of my comfort zone during the fulfilment of this thesis, I used many different technologies I had never used before, because it was often the best suitable one or the only way to complete certain implementations. For example, I had never truly used Python before this thesis. However, for my AI implementation this was necessary, hence, I did not hesitate to use it. By stepping out of my comfort zone I exposed myself to new risks, some I could mitigate from the very beginning of the project but others I couldn’t, hence I acknowledged them, put a mitigation plan and tackled the problems when they occurred. This is because I wanted to get the most out of this experience and use it to boost my knowledge in this ever-growing field of applied computer science. If there is one thing that I have learned during my experience in this field, it is that the best way to learn is to venture, explore and just try out!

This approach surely resulted in numerous risks, but with each risk I discovered, I learned, and I mitigated. However, there are some problems that were left unsolved as these would have taken too much time to solve at that point in development. For instance, as I was still new to Firebase, I had not learned how to properly use key, value pair data structures. Thus, I did not properly implement to use the key as identifier for a data object. This made it so that certain data objects in my implementation would cause a problem if they were deleted. This is because their identifier value was based on the amount of data objects in a list. In the end I found out how to solve this, but at that point in time it was too late in the project timeline to implement the fix.

As we all know, the design of the user interface is a key factor in the success of any mobile app. Many app projects and design specialists invest ample time in the most important aspects of good user interfaces: Intuitive and consistent design, clarity, high responsivity and attractiveness. When delivering this first version of the app I laid great effort to make the best user interface, yet, due to the time limitations for this version (project), I had to compromise and limit the efforts invested for this part and skip preparing wireframes, conduct UX workshops to collect user feedback and iteratively improve the user interface.

Another challenge involved the ever-changing development library landscape. It occurred to me when the library I was using to develop Firebase in Python, e.g. Pyrebase, was deprecated during development. I was required to make the switch to “Firebase Admin” to avoid a major flaw.

In the end, I am truly and deeply satisfied with what I have achieved in the context of this thesis. Proving my architectural insight, project management, creative, initiative taking, and critical thinking skills. I am glad that the topic I chose was very interesting and challenging to me throughout the whole project. This kept me invested in the project without needing much self-motivation. I found what I was doing wildly fascinating and enjoyed each step of the way no matter how tough it became.

1. Bibliography



1. Anaconda Inc. (2019). Distribution. From anaconda.com: https://www.anaconda.com/distribution/
2. André, V., Katrin, B., & Joe, H. (2005). Continuous productive urban landscapes: designing urban agriculture for sustainable cities. Oxford: Architectural Press.
3. Arsenovic, M. (2018). Deep Learning Plant Diseases. From GitHub: https://github.com/MarkoArsenovic/DeepLearning\_PlantDiseases
4. Boye, J. I., & Arcand, Y. (2012). Green technologies in food production and processing. New York: Springer.
5. Busschers, J. (2018, November 3). Computer vision based approach for analyzing fluids inside transparent vessels. From scribd: https://www.scribd.com/document/392262597/Research
6. Childs-Maidment, J. (2017, January 7). Pyrebase. From GitHub: https://github.com/thisbejim/Pyrebase
7. Craig, L. (2005). Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development (3rd ed.). Upper Saddle River, New Jersey: Prentice Hall PTR.
8. DAGS. (2006). Scene Dataset. From Stanford.edu: http://dags.stanford.edu/projects/scenedataset.html
9. Das, S. (2017). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ... From medium: https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
10. De Geyter, K. (2018). A comparison of the environmental impact of vertical farming, greenhouses and food import. Environmental Management. Ghent: University College Ghent.
11. Firman, E. (2019, March 23). Android Image Picker. From GitHub: https://github.com/esafirm/android-image-picker
12. Google Inc. (n.d.). Firebase Admin Setup. From firebase.google.com: https://firebase.google.com/docs/admin/setup
13. Google Inc. (n.d.). Firebase Documentation. From firebase.google.com: https://firebase.google.com/docs
14. Homan, J. (2014, April 5). Relational & Non-Relational Databases. From pluralsight.com: https://www.pluralsight.com/blog/software-development/relational-non-relational-databases
15. JetBrains s.r.o. (2019). PyCharm. From JetBrains: https://www.jetbrains.com/pycharm/
16. Jodal, S. M. (n.d.). From pykka.org: https://www.pykka.org
17. Judd, S. (2019, June 7). Glide. From GitHub: https://github.com/bumptech/glide
18. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional. From papers.nips.cc: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
19. Mohanty, S. (2018, September 23). PlantVillage-Dataset. From GitHub: https://github.com/spMohanty/PlantVillage-Dataset/
20. MongoDB. (2018, June). Top 5 Considerations When Evaluating NoSQL Databases. From mongoDB: https://webassets.mongodb.com/\_com\_assets/collateral/10gen\_Top\_5\_NoSQL\_Considerations.pdf
21. Muntenescu, F. (2016, November 1). Android Architecture Patterns Part 1: Model-View-Controller. From medium.com: https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6
22. PlantVillage. (n.d.). About Us. From Penn State University: https://plantvillage.psu.edu/about-us
23. Richardson, C. (2018). From Microservices: https://microservices.io/
24. Stern, S. (2019). FirebaseUI for Android. From GitHub: https://github.com/firebase/FirebaseUI-Android/
25. Storti, B. (2015, July 9). The Actor Model. From brianstorti.com: https://www.brianstorti.com/the-actor-model/
26. Sykes, N. (2018, November 17). What to know about the impact of data quality and quantity in AI. From SmartDataCollective: https://www.smartdatacollective.com/what-to-know-about-impact-data-quality-quantity-in-ai/
27. Torch Contributors. (n.d.). Documentation. From pytorch.org: https://pytorch.org/docs/stable/index.html
28. United Nations DESA. (2018, May 16). Revision of world urbanization prospects. From UN.org: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html

**Legend References:**

1. Downtime (n.d.). From dictionary.cambridge.org: https://dictionary.cambridge.org/dictionary/english/downtime
2. Software Architecture (2019). From wikipedia.org: https://en.wikipedia.org/wiki/Software\_architecture
3. Database (2019) From wikipedia.org: https://en.wikipedia.org/wiki/Database
4. User experience (2019). From wikipedia.org: https://en.wikipedia.org/wiki/User\_experience
5. What is REST (2018). From restfulapi.net: https://restfulapi.net/
6. Application programming interface (2019). From wikipedia.org: https://en.wikipedia.org/wiki/Application\_programming\_interface
7. Deep learning (2019). From wikipedia.org: https://en.wikipedia.org/wiki/Deep\_learning

**Code References** (excluding above)**:**

1. Garg, I. (2017). How do you create an Android View Pager with a dots indicator? (2016). From stackoverflow.com: https://stackoverflow.com/questions/38459309/how-do-you-create-an-android-view-pager-with-a-dots-indicator/38459310#38459310
2. Florian (March 2018). How to Create an Image Slider with Picasso + ViewPager - Android Studio Tutorial. From youtube.com: https://www.youtube.com/watch?v=tM7rwJoK-18&t=15s
3. Sheena (July 2019) Python Lists in Depth. From codementor.io: https://www.codementor.io/sheena/python-lists-in-depth-lrtmk7w4q
4. Alder, G.; Benson, D. (n.d.). From draw.io: https://draw.io/
5. Python Software Foundation (2019). Queue. From docs.python.org: https://docs.python.org/3/library/queue.html
6. Python Dictionaries (n.d.). From w3schools.com: https://www.w3schools.com/python/python\_dictionaries.asp
7. Miguens, J. (July 2017). How to detect changes in firebase child with python? From stackoverflow.com: https://stackoverflow.com/questions/45154853/how-to-detect-changes-in-firebase-child-with-python
8. Google Inc. (2019). Spinners. From developer.android.com: https://developer.android.com/guide/topics/ui/controls/spinner#java
9. Google Inc. (2019). Spinner. From developer.android.com: https://developer.android.com/reference/android/widget/Spinner
10. Google Inc. (2019). SpinnerAdapter. From developer.android.com: https://developer.android.com/reference/android/widget/SpinnerAdapter
11. Google Inc. (2019). Take photos. From developer.android.com: https://developer.android.com/training/camera/photobasics
12. Google Inc. (2019). Create swipe views with tabs. From developer.android.com: https://developer.android.com/guide/navigation/navigation-swipe-view#java
13. Google Inc. (2019). ViewPager. From developer.android.com: https://developer.android.com/reference/android/support/v4/view/ViewPager
14. Google Inc. (2019). Create a List with RecyclerView. From developer.android.com: https://developer.android.com/guide/topics/ui/layout/recyclerview#java
15. Google Inc. (2019). CardView. From developer.android.com: https://developer.android.com/reference/android/support/v7/widget/CardView
16. Saini, A. (n.d.). Spinner tutorial with examples in Android Studio. From abhiandroid.com: https://abhiandroid.com/ui/spinner
17. Praveen K. (September 2018). How do I read all child key values of a child from Firebase database in Android? From quora.com: https://www.quora.com/How-do-I-read-all-child-key-values-of-a-child-from-Firebase-database-in-Android
18. Vargas J. (July 2018). How to get a String List from Firebase to fill a Spinner? From stackoverflow.com: https://stackoverflow.com/questions/38492827/how-to-get-a-string-list-from-firebase-to-fill-a-spinner
19. Philio. (March 2011). Passing ArrayList through Intent. From stackoverflow.com: https://stackoverflow.com/questions/5374546/passing-arraylist-through-intent
20. Send data from activity to fragment in Android. (October 2012). From stackoverflow.com: https://stackoverflow.com/questions/12739909/send-data-from-activity-to-fragment-in-android
21. Action Bar's onClick listener for the Home button (July 2012). From stackoverflow.com: https://stackoverflow.com/questions/11079718/action-bars-onclick-listener-for-the-home-button
22. Android - How to show App logo in action bar? (March 2015). From stackoverflow.com: https://stackoverflow.com/questions/19540648/android-how-to-show-app-logo-in-action-bar
23. Google Inc. (2019). Work with Lists of Data on Android From firebase.google.com: https://firebase.google.com/docs/database/android/lists-of-data
24. “Aws Rh” (February 2018). RecyclerView + Cardview with GridLayout | Android studio tutorial. From youtube.com: https://www.youtube.com/watch?v=SD2t75T5RdY
25. TVAC Studio (January 2017). Retrieving Basic Data - Android Firebase Tutorial - Full Backend Course (Part - 12) From youtube.com: https://www.youtube.com/watch?v=9rSDvKV23bI&list=PLGCjwl1RrtcSi2oV5caEVScjkM6r3HO9t&index=12
26. Prabeesh, R. K. (May 2018). Android tutorial (2018) - 53- Swipe View with Tabs From youtube.com: https://www.youtube.com/watch?v=QU5-IpnEub4
27. Simplified Coding (November 2017). #3 Firebase Authentication Tutorial - Saving User Information From youtube.com: https://www.youtube.com/watch?v=qVaBY92sOSI&list=PLk7v1Z2rk4hi\_LdvJ2V5-VvZfyfSdY5hy&index=3
28. Mohit G. (June 2015). Android Tabs Example – With Fragments and ViewPager From truiton.com: https://www.truiton.com/2015/06/android-tabs-example-fragments-viewpager/
29. Carney, K. (n. d.). Create a Custom Adapter for an Android GridView. From homeandlearn.co.uk: https://www.homeandlearn.co.uk/android/grid\_view\_custom\_adapter.html
30. Tamada R. (July 2017). Android working with Card View and Recycler View. From androidhive.info: https://www.androidhive.info/2016/05/android-working-with-card-view-and-recycler-view/
31. Eddydn (December 2017). Android Grid Layout demo with Click / Select Item From github.com: https://github.com/eddydn/AndroidGridLayout
32. Appendix



## Android UI

|  |  |  |
| --- | --- | --- |
| Fragments | V1 | Final |
| Plant Detail |  |  |
| Logs | / |  |
| Activities | V1 | Final |
| Sliding Tabs | / |  |
| Login |  |  |
| Register |  |  |
| Plant Overview |  |  |
| Plant Add |  |  |
| Log Details  (Processing...) | / |  |
| Log Details | / |  |
| Classification  (Log Details Processed) | / |  |
| Problem | / |  |
| About | / |  |
| Popup Alert | V1 | Final |
| Delete Log | / |  |
| Delete Plant | / |  |
| Taking a good picture for log  (only first time) | / |  |