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](#_Toc41836569)

[2. Preface and acknowledgments 3](#_Toc41836570)

[3. Abstract 4](#_Toc41836571)

[4. Introduction 5](#_Toc41836572)

[5. Chapter 1 - Problem Definition 6](#_Toc41836573)

[5.1 Definition 6](#_Toc41836574)

[5.2 Elaboration 6](#_Toc41836575)

[5.3 Requirements 7](#_Toc41836576)

[5.3.1 Functional Requirements 7](#_Toc41836577)

[5.3.2 Non-Functional Requirements 7](#_Toc41836578)

[6. Chapter 2 – Solution Architecture 9](#_Toc41836579)

[6.1 Actors 9](#_Toc41836580)

[6.1.1 Knowledge Providers 9](#_Toc41836581)

[6.1.2 End Users 9](#_Toc41836582)

[6.2 System 9](#_Toc41836583)

[6.2.1 Scope 10](#_Toc41836584)

[6.3 Knowledge Base 10](#_Toc41836585)

[6.4 AI Diagnosis Engine 10](#_Toc41836586)

[6.4.1 Plant Deficiency Dataset 10](#_Toc41836587)

[6.4.2 Actor Model 10](#_Toc41836588)

[6.5 Database 10](#_Toc41836589)

[6.5.1 Relational (SQL) 11](#_Toc41836590)

[6.5.2 Non-Relational (NoSQL) 11](#_Toc41836591)

[6.5.3 Firebase Storage & Realtime Database 11](#_Toc41836592)

[6.6 Interaction Platform 11](#_Toc41836593)

[6.7 Conclusion 11](#_Toc41836594)

[7. Chapter 3 – Technical Implementation 12](#_Toc41836595)

[7.1 Software Design 12](#_Toc41836596)

[7.1.1 Domain Model 12](#_Toc41836597)

[7.1.2 Implementation Designs 14](#_Toc41836598)

[7.2 Diagnosis Engine Implementation 14](#_Toc41836599)

[7.2.1 Design 14](#_Toc41836600)

[7.2.1 Firebase Admin / Pyrebase 14](#_Toc41836601)

[7.3 AI Implementation 14](#_Toc41836602)

[7.3.1 Anaconda Distribution 14](#_Toc41836603)

[7.3.1 AI Model 15](#_Toc41836604)

[7.3.2 Training & Testing 15](#_Toc41836605)

[7.4 Version Control & Development Tools 17](#_Toc41836606)

[7.4.1 GitHub 17](#_Toc41836607)

[7.4.2 PyCharm 17](#_Toc41836608)

[7.4.3 Android Studio 17](#_Toc41836609)

[8. Conclusion 18](#_Toc41836610)

[8.1 Future 18](#_Toc41836611)

[8.2 Multi-purpose Architecture 18](#_Toc41836612)

[8.2.1 Step 1: AI 18](#_Toc41836613)

[8.2.2 Step 2: Database 19](#_Toc41836614)

[8.2.3 Step 3: Interaction platform 19](#_Toc41836615)

[8.3 Resulting impact 19](#_Toc41836616)

[9. Critical Reflection 20](#_Toc41836617)

[10. Bibliography 21](#_Toc41836618)

[11. Appendix 22](#_Toc41836619)

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



, 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.

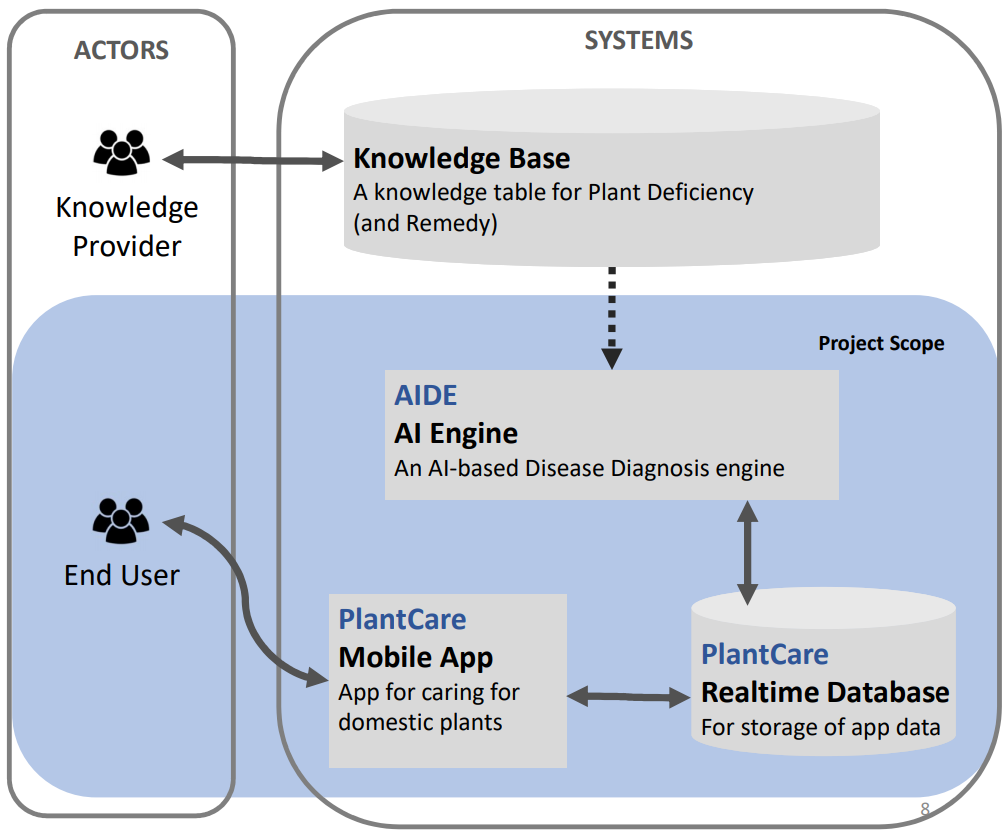


Figure 10: Solution Architecture

## 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, 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

### Scope

As seen on figure 10, the blue part of the solution architecture represents the project

## Knowledge Base

This is the first functional component of the system that I will cover. It’s a dictionary

## AI Diagnosis Engine

This functional component is a high-level representation of multiple smaller

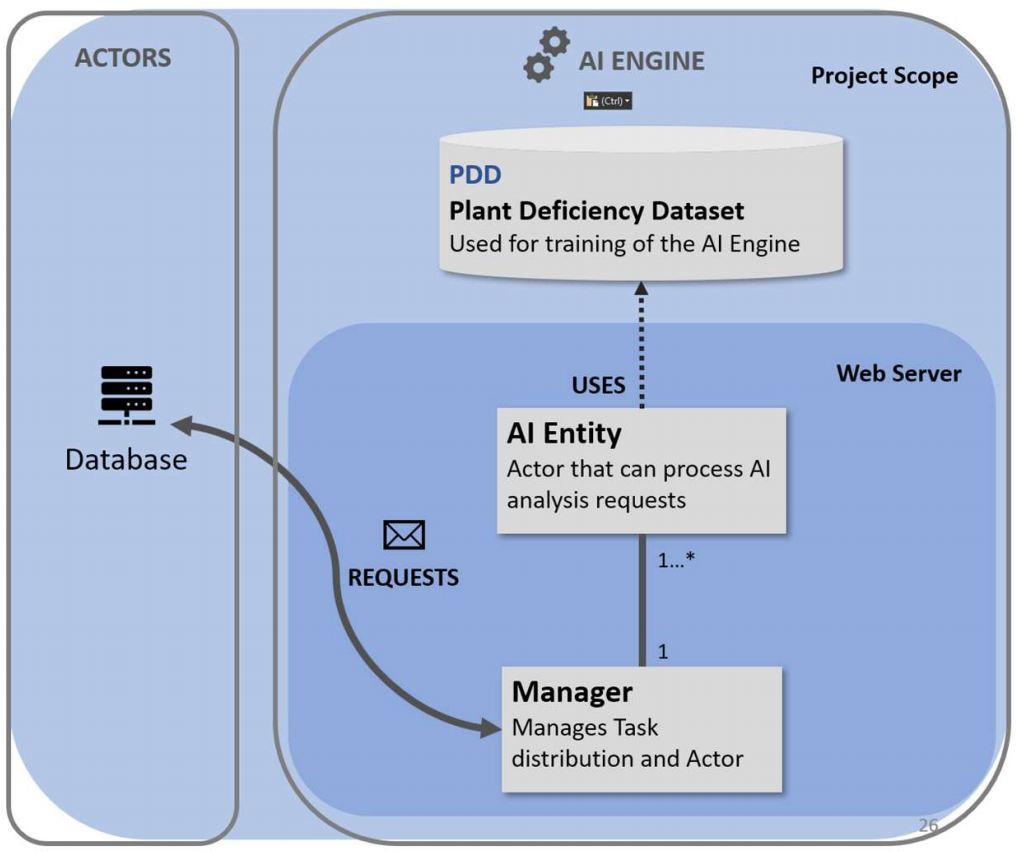


Figure 11: AI Engine Architecture

### Plant Deficiency Dataset

The first sub-component of AIDE that I will cover is also the most critical part. The PDD

### Actor Model

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

## 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

### Non-Relational (NoSQL)

Examples - Firebase, MongoDB, Redis, ...

This type of database is not as well-known as the previous type. But opposite to 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

## 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

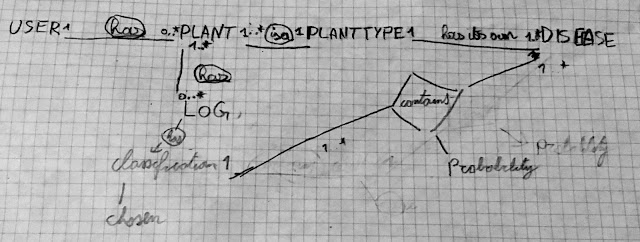


Figure 12: Sketch of problem domain model

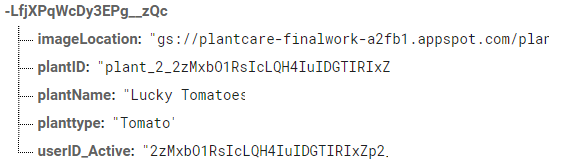
### Domain Model

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.

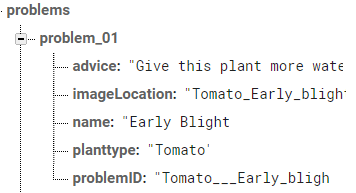
#### Plant

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.

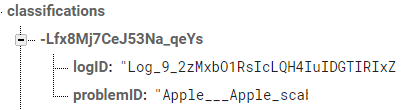
#### Plant Type

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.

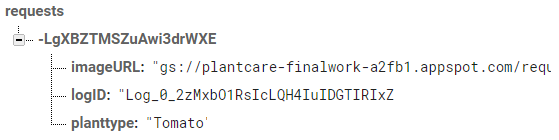
#### Classification

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.

## Diagnosis Engine Implementation

This implementation is another critical part of this project. It is implemented as a Py

### 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.

## 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

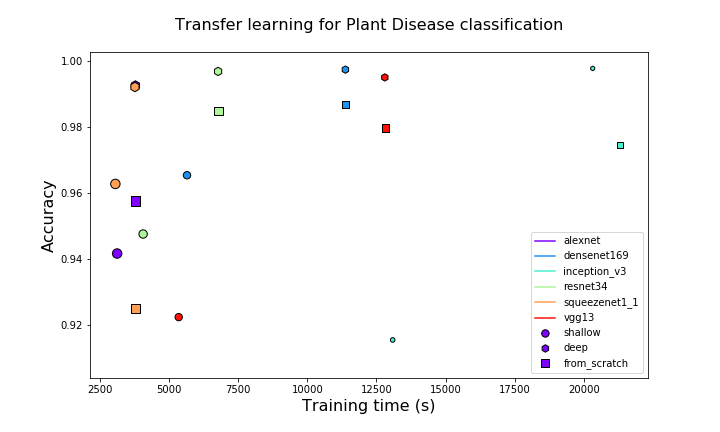


Figure 14: plant disease classification scores3

|  |  |
| --- | --- |
| 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

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. Appendix

