Technische Universiteit
Eindhoven
University of Technology

/ Department of
Mathematics and
Computer Science
/ PDEng Software
Technology

DASSICO - Data Storing and Structuring for Information Discovery in AgTech

Dimas Satria

DASSICO - DATA STORING AND STRUCTURING FOR INFORMATION DISCOVERY IN AGTECH

Dimas Satria

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners	Eindhoven University of Technology Proeftuin voor Precisielandbouw
Steering Group	Dr. Yanja Dajsuren, PDEng Prof. Dr. Jacob de Vlieg Paul van Zoggel MA
Date	November 2019
Document Status	Public
SAI report no.	2019/118

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.

Contact Address

Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080A, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by

Eindhoven University of Technology
Stan Ackermans Institute

Printed by

Eindhoven University of Technology *UniversiteitsDrukkerij*

SAI report no.

2019/118

Abstract

N/A

Keywords

Precision agriculture, Data structure, Sensor data, Information discovery, PDEng, TU/e, Software technology

Preferred reference

DASSICO - Data Storing and Structuring for Information Discovery in AgTech: SAI Technical Report, November 2019. 2019/118

Partnership

This project was supported by Eindhoven University of Technology and Proeftuin voor Precisielandbouw.

This PDEng thesis is approved by the supervisors and the Thesis Evaluation Committee is composed of the following members:

Chair:	Dr. Yanja Dajsuren, PDEng
Supervisors:	Prof. Dr. Jacob de Vlieg Paul van Zoggel MA
Members:	Dr. George H. L. Fletcher Dr. Gijs Dubbelman

Foreword

The future of food depends on better time and matter management than we globally today do in order to feed ourselves on the planet. We are largely resource inefficient, as a species we haven't found our durable place in nature yet like all other non-invasive plants and creatures.

High-tech sensors, processing possibilities, and artificial intelligence discoveries are making it possible to support us in really listening to mother nature and do what is best for us and her. In many domains from mobility to healthcare, technology and data have pushed us forward, yet all these engineering advances seem to have difficulty to help us feeding ourselves better. We believe this is because we scaled up industrial food production but for 2 generations we stopped using our senses and minds observing how are soil and nutrient cycles are doing.

Farmers have a passion for growing great crops to feed all, research shows 80% of farmers want to do everything sustainable within nature's boundaries if they get the chance. There are a few front runners on the planet in modern crop production and luckily one of them runs a family farm just 20 minutes south of Eindhoven. Jacob and Jan van den Borne took over the family grower business in 2006. They had a choice; will we invest in quantity or quality. Luckily they chose quality. How can we have better yields with the same resources was the question. After adopting all possible precision technologies available after 3 years they started to harvest huge amounts of data. Listening to mother nature and do as she would do is the goal. Smaller high tech electric machines, drones, sensors, software, and dashboards are making this possible for a few people growing in balance with nature the food for whole urban settlements.

Van Den Borne got a question; You might have the most advanced open-air food resilient enterprise on the planet? How can more farmers benefit from your developed working process? After TUe PDEng student Bulgaa Enkhtaivan unfolded the Van Den Borne data processes and a pilot system (DIPPA 1), Dimas Satria picked up from there and started to focus on the absolute necessary future proof base of every farmer managing sustainable food production; harvesting, storing and structuring the data for students and researchers around the globe to dive in and discover information helping to learn to listen to mother nature better. (DIPPA 2).

Globally there is a lot of focus and research in the interpretation of data output. The 'boring' part - saving and structuring data future proof – is a research domain in food security underestimated. Artificial Intelligence in food resilience begins with manually and automatic collecting data on the growth of plants and resources needed. Independent and neutral data storage by a farmer is the first step to train future systems. Systems students and researchers all over the world imagine but lacking manual and machine data to feed them.

Imagine a student who has an internship on a farm anywhere on the planet. The student brings a soil moisture and temperature sensor. He or she installs it on a field and creates for the farmer an Open Source Diskstation Instance for the data to be stored and structured for future purposes. A community

of students and volunteers help the farmer based on the data he is gathering manually and the sensor. The yield on the plot increases, investments in sensors, machines, software, and dashboards can be made.

Dimas Satria did a wonderful job in analyzing the status quo in agricultural data storage, mapping, and standards. The void is indeed independent open storage and structuring off data. With the little time he had in coding the backend of the data structuring system, the proof of concept is a working demo of the open-source deployment package for any future service provider on the planet interested in helping farmers forward. The documentation is thorough and decisions in collaboration with TU/e experts solid. By collaborating with 3th-year TU/e Computer Science students, the system got a truly user-friendly interface so we will be happy to deploy the system in trial settings for students, researchers and farmers to collect and structure data.

I would like to thank Dimas for his amazing focus, punctuality, and effort to bring theory and practice together in this thesis, documentation and software solution. I hope for Dimas this was a warm welcome in the world of (glocal) food resilience and technology solutions and will help him find his future career pathways.

Paul van Zoggel
TU/e W&I and the Precision Agriculture Trial Center Reusel, The Netherlands.

14 November 2019

Preface

This document summarizes the project "DASSICO - Data Storing and Structuring for Information Discovery in AgTech". This project's goal is to provide a software system to store and structure data collected from various sources with diverse data structures into centralized database. This project addresses some challenges in structuring the data to enable researchers or analytics tools finding required data efficiently to perform analysis.

This project was executed by Dimas Satria as his final design project from the Professional Doctorate in Engineering (PDEng) program in Software Technology. This program is offered by the Department of Mathematics and Computer Science of the Eindhoven University of Technology in the context of the 4TU School for Technological Design, Stan Ackerman's Institute.

The project's proposal comes from Proeftuin voor Precisielandbouw. They provide test facilities to help farmers in adopting precision agriculture concept in The Netherlands. The project was carried out in eight months under the supervision of the Proeftuin voor Precisielandbouw and the Eindhoven University of Technology (TU/e).

This document explains the process for achieving the project goal and the project results under general software engineering terms. Nonetheless, no specialized knowledge in software engineering is required for fully understanding this document. Readers who are interested in understanding the basics and results of this project should read Chapters 1 - 4 and 10 - 12. For readers who are interested in a more detailed technical solution, Chapters 5 - 9 cover requirement analysis, architecture, design, and verification and validation processes in the project.

Dimas Satria
14 November 2019

Acknowledgements

I would like to thank people who helped me with advice and support throughout this wonderful project. The project would not be successful without their assistance.

First of all, my special gratitude goes to my supervisors from TU/e and Proeftuin voor Precisielandbouw, Yanja Dajsuren, Paul van Zoggel, and Jakob de Vlieg for their continuous support, motivation, and feedback. Thank you very much for all of your valuable guidance that helped me develop myself further professionally. I completed this project with better skills than I initially had at the beginning of the project.

Thanks to these stakeholders who gave significant contributions to this project: Jacob van den Borne, Ger Snijkers, and Puck Mulders. Jacob provided the data sources needed for this project and shared extensive knowledge of Precision Agriculture in practice on his farm. Puck Mulders and Ger Snijkers provided feedback to make the outcome of the project more useful for researchers.

I would also like to extend my gratitude to George Fletcher and Nicola Zannone who shared their expertise to shape solutions to the problems, which are addressed in this project.

Thanks to the SEP students for a nice collaboration in developing the software system in this project. Without your contribution, the outcome of this project would be different. The SEP students were Sander van de Ven, Jurrien van Winden, Léon van de Beek, Luuk Roozen, Bram van Leeuwen, Arthur Groote Woortmann, Han Swinkels, Kas Burgers, Willem van Santvoort, Mohamed Ghanem, and Mart Keizer.

Furthermore, I am grateful for everybody who has been involved in the Software Technology PDEng program. Special thanks to all my PDEng colleagues for sharing experience and giving feedback during the two years of our program. Thanks to Désirée van Oorschot who helped me a lot during my study in the program.

Last but not least, I owe my deep and sincere gratitude to my wife, Arum, for her unconditional love, support, and encouragement. To my daughter, Ashalina, thank you for all the joy you bring that washes away all my worries. To my parents and brother in Indonesia, thank you for your endless support.

Dimas Satria
14 November 2019

Executive Summary

The world's population is continuously growing each year and expected to increase by two billion in the next 2050 by United Nations[1]. Naturally, we need to have more food available to feed all people. To answer this problem, farmers have been applying Precision Agriculture (PA) concept to increase farm productivity with high-efficiency resource usage in an eco-friendly manner. This concept requires observation and measurement data that is collected from fields, crops, environment, and so forth. This data can be analyzed to get valuable knowledge for farmers to treat the crops and fields properly and minimize the negative environmental impact that can reduce the quality and quantity of the crops.

The goal of this project was to develop a software system to integrate observation and measurement farm data from various sources with diverse formats. This project analyzes the design of a previous prototype as a reference. The software system in this project needs to structure the data for researchers or analytics tools to find the required data efficiently for performing analysis. It is difficult for a system maintainer to structure farm data with different structures produced by all farms in the world because he needs to have a solid knowledge of every data to be put at proper database tables in the system. It is also challenging for farmers to structure their data in the system because they need to understand the system's database schemas.

In order to achieve this project's goal, a metadata-based ETL was designed for the software system. With this design, the users only need to provide farm data and its metadata to the system. Then, the system stores the data at the proper tables in the database based on the metadata. The metadata can help researchers to understand the stored data and perform analysis. Role-based access permission was implemented in the system for the users to protect or share their data with the other users.

Apart from the documentation generated in the project, a prototype was developed in this project that fulfilled the main requirements of its stakeholders. The design of the system allows the extension of new software components and deployment independently. General users can make use of a user interface to interact with the system. Asking service requests directly with the system via REST APIs is also possible for data analytics or machine learning algorithms to retrieve and perform analysis on the data from the system.

In conclusion, the results demonstrate the benefits of integrating data from various sources to provide more data for researchers to perform analysis and then provide a data-driven decision for farmers to manage their farms properly. It is recommended to integrate data from other companies or departments near to the farms to see the impact of their activities on the farms' productivity.

[1] https://www.un.org/development/desa/en/news/population/world-population-prospects-2019.html

Contents

List of Figures

List of Tables

Abbreviation and Acronyms

AgTech Agriculture Technology

DIPPA Data Integration Platform for Precision Agriculture

ETL Extract-Transform-Load

JWT JSON Web Token

OAuth Open Authorization

PSG Project Steering Group

RDBMS Relational Database Management System

REST Representation State Transfer

SEP Software Engineering Project

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

1 Introduction

This chapter presents the context, the problem description, and the goal of this project. It also gives an outline of this report.

1.1 Context

The project "DASSICO - Data Storing and Structuring for Information Discovery in AgTech" was conducted by Dimas Satria, the trainee, as part of the PDEng program in Software Technology. This program is offered by the Department of Mathematics and Computer Science of Eindhoven University of Technology in the context of the 4TU.School for Technological Design, Stan Ackerman's Institute [4].

The project was initiated by Jacob de Vlieg and Paul van Zoggel as representatives from Proeftuin voor Precisielandbouw and as customers of this project. The customers aim to develop and operate a testing ground for precision agriculture in order to speed up innovation and the adoption of precision agriculture in The Netherlands [5]. The customers also collaborate with test facility farms, universities, agriculture organizations, and various companies in The Netherlands to help farmers in solving crucial problems in agriculture: over consumption of resources and negative environmental impact.

Precision agriculture [6] is a farming management concept to manage variations in the field in order to grow crops optimally with minimum required resources and environmental risks. Collecting data from field, crop, environment, and many other variabilities in the field is one of the key points in precision agriculture. By analyzing the collected data, we can discover useful information for farmers to increase farm productivity with high efficiency resource usage in an eco-friendly manner.

An initial prototype, namely Data Integration Platform for Precision Agriculture (DIPPA), was developed to study the challenges in utilizing farm data from a farmer [1]. Data integration in DIPPA requires Extract-Transform-Load (ETL) tasks to store and structure diverse data from many sources into a centralized system. Creating an ETL task needs solid knowledge of not only farm data with various data schemas, but also internal database schemas in DIPPA. It is rather challenging to create ETL tasks for farmers, who have domain knowledge in agriculture, to learn database schemas or for software engineers to understand farm data owned by each farmer.

By having the context explained in this section, the following section presents the goal and the contribution of this project.

1.2 Project Goal

As part of innovation and research projects in Agriculture Technology (AgTech), the customers require a solution to utilize data in precision agriculture from various sources with different data schemas. The solution should enable researchers to retrieve the data for information discovery without necessarily asking assistance from farmers to explain their data.

The main contribution of this project is to build a software system for farmers to store and structure precision agriculture data with different schemas easily. The software system was designed and realized as a prototype so that farmers can import their data to the system manually, harvest data from active sensors placed at farms, or retrieve farm data from their cloud servers directly.

The system enables farmers to import farm data without the necessity of knowing internal database schemas of the system by providing metadata of farm data. The metadata explains the contents of the data to be imported. The system uses the given metadata to store and structure imported farm data into the system. Researchers or other system users who are registered in the system may access the data and its metadata. They can gain benefit from the provided metadata in understanding the data for analysis purposes.

1.3 Outline

The report addresses domain analysis to introduce the concept of precision agriculture in Chapter 2. Chapter 3 presents an analysis of the problem that this project aims to solve. Chapter 4 analyzes the main stakeholders of this project whose interests are affected by the project execution or result. Chapter 5 lists the functional and non-functional requirements of this project. Chapters 6-9 (software architecture, design, implementation, and verification and validation) give elaborated explanation from design to its implementation and validation of the results. Chapter 10 summarizes the project results and future work. The final chapters, Chapters 11-12, show an overview of project planning and an evaluation of the system design and the design process.

2 Domain Analysis

This chapter introduces the domain where this project is carried out, namely precision agriculture. Several ways to collect data in precision agriculture are presented. Farmer levels in adopting technologies in precision agriculture are also described.

2.1 Precision Agriculture

Precision agriculture is a concept in agriculture that involves observing, measuring, and responding to inter- and intra-field variability in crops properly [6]. This concept aims to help farmers in lowering the production cost by using input resources efficiently, optimizing crop yields, and lessening negative impact to the environment.

Farmers cultivating crops in fields with precision agriculture are like parents taking care of children. They need to assess and prepare their living space for the children to grow well. They provide high nutrient foods and sufficient water. Giving too little or too much food and water will not be good for the children's growth progress. A doctor can give advice to achieve optimal health and development of the children and to prevent disease by asking the parents some information about their environment and the amount of food and water consumed each day.

Collecting data about crops and their environment is one of the important steps in precision agriculture. All features of the environment in the field, such as soil, weather, and water, differ from one location to another. All these variables affect the growth of crops in the field. Farmers have to take appropriate actions at the correct time and location on a field to optimize crop yield. Those actions should be motivated by the knowledge that is discovered from analyzing observation and measurement data.

Figure 2.1 illustrates an example of data gathering in precision agriculture for one agricultural cycle. An agricultural cycle is activities from preparing soil in fields to harvesting crops. In the figure, a farmer prepares a field in winter. The farmer scans the soil in the field to gather soil nutrient data. Soil moisture data can be monitored using a soil sensor. The farmer can take some crop sample data, for examples crop height, root length, and number of leaves, periodically to monitor crop growth. During the harvesting period, the farmer records data about the crop yield per area by using sensors on a tractor.

All of the collected data is further examined to get valuable knowledge to help farmers in managing their fields well and achieving high productivity. For example, farmers can apply a proper amount of fertilizer in the field based on soil nutrient data analysis. They can determine how much water to use to irrigate the field after analyzing data from a soil sensor reading and ensure that the crops have a sufficient amount of water to grow.

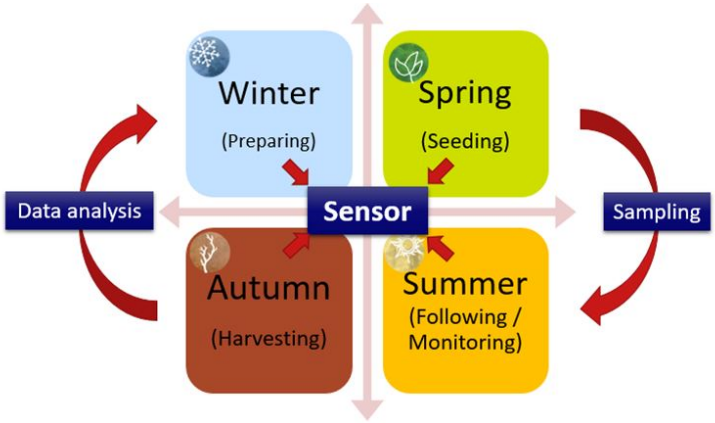


Figure 2.1: An example of data gathering in precision agriculture

2.2 Data Collection in Precision Agriculture

With the help of technology, farmers who adopt precision agriculture can collect data comprehensively to be analyzed. In this project, observing and measuring activities to collect data are further called sensing. The data produced by a sensing activity is named sensing data. The variabilities of data are generally in space with geolocation and time. Based on data gathering method, sensing is grouped into manual sensing, automated sensing, and remote sensing.

Manual sensing is a labor-intensive activity in collecting data by humans. An example of manual sensing is taking a potato crop sample in a field by a farmer to measure its weight. Taking a crop sample is usually performed to observe crop growth periodically until harvest time.

Automated sensing collects data with the assistance of devices at an observation location. Automated sensing can use a static or moving sensor. A static sensor is placed on a field and only acquires data at a certain geolocation. A moving sensor is generally shifting from one location to another on a field to obtain data. Examples of static and moving sensors for automated sensing are a weather station and soil nutrient scanner on a tractor, respectively.

In remote sensing, data acquisition of an object is performed at a distance without physical contact with the object. Acquiring data in remote sensing can be carried out via satellite or drone to observe a field.

2.3 Application of Technology for Precision Agriculture

Farmer levels in terms of technology usage on precision agriculture are illustrated in Figure 2.2. In this figure, farmer level 1 uses manual sensing to collect crop sample data and other data related to the farm field. Some examples of field data are total area of a field, soil type, and the types of crops planted in the field. This farmer is usually a traditional farmer who is going to apply precision agriculture. A farmer at level 2 may combine manual, automated, and remote sensing to collect field, crop sample, and environment data. At this level, the farmer only collects data without doing any analysis. A farmer at level 3 already has sensing data and leverages some data analysis algorithms or artificial intelligence to get knowledge and wisdom for increasing farm productivity. At the highest

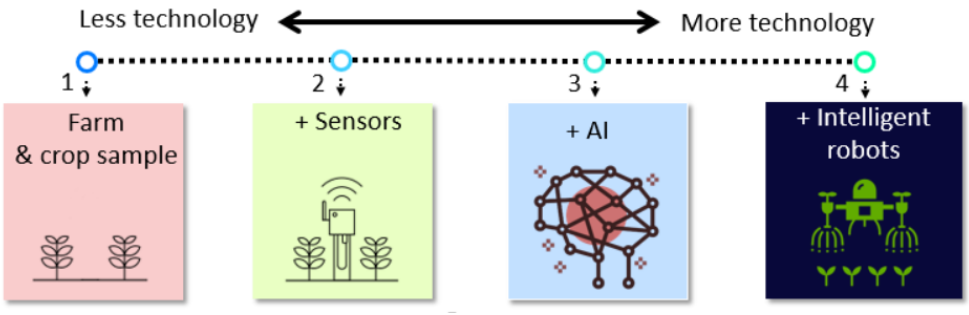


Figure 2.2: Farmer levels in using technologies for precision agriculture

level, a farmer employs intelligent robots to replace all the tedious work in manual sensing. In the scope of this project, only farmer levels 1 and 2, which need to utilize manual and automated sensing data, are relevant.

3 Problem Analysis

This chapter describes an analysis of the DIPPA prototype. The problems that this project aims to solve and the challenges are also explained.

3.1 Analysis of DIPPA Prototype

In 2018, Enkthaivan conducted a PDEng project to study the prospects and challenges in utilizing data in precision agriculture [1]. In her project, an initial prototype was designed and developed to store farm data of Van den Borne Aardappelen farm into a data storage. The prototype is called DIPPA. Figure 3.1 presents an architecture design of DIPPA [1].

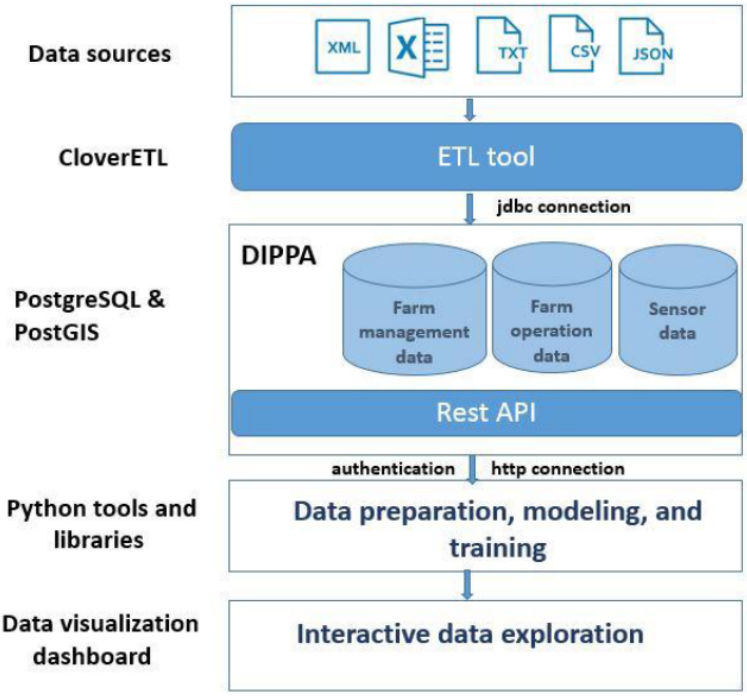


Figure 3.1: An architecture design of DIPPA [1]

The prototype employed ETL tool was employed to integrate files from various sources into a DIPPA storage or database. This ETL processes files with specific data structures for one farmer. Thus, a new procedure or task in the ETL process has to be created whenever a file with a different data structure needs to be loaded in the database. That limits the flexibility to store data from other farms or other

files with different data structures.

Creating a task in ETL needs a good understanding of farm data and database schemas of DIPPA. Farmers who know their farm data well might take a considerable amount of time to learn database schemas of DIPPA in order to create ETL tasks to store their data in DIPPA. It is also challenging for a software engineer to understand farm data well without missing information to be stored in DIPPA. Some researchers who used the data stored in the prototype reported missing data context or metadata. That hampered them in understanding and analyzing the data easily.

Several maintainability issues were identified from the prototype from the technology viewpoint. Firstly, an ETL tool in DIPPA uses CloverETL that is obsolete [7]. Secondly, data preparation and visualization parts of DIPPA were developed using Python 2.7. Python 2.7, as the latest version in Python 2, will come to the end of its life soon and not be maintained starting from January 2020 [8]. A major risk of using obsolete tools or unsupported programming language is vulnerabilities will sooner or later appear and nobody fixes them. That can definitely harm the security of the data as stated by National Cyber Security Centre of United Kingdom [9].

Even though migrating code from Python 2 to 3 would be possible within the project timeframe, another maintainability issue was identified in the source code of the prototype. The source code of the prototype is not modular and has many hardcoded values. Those codes need major modifications and would take a considerable amount of time to satisfy the current project requirements that are listed in Chapter 5.

DIPPA has three database schemas to store different types of data. The database schemas are farm operation, farm management, and sensor data. Farm operational data consists of farm name, employees, operations or activities on fields, and other data related to the fields, soils, and crops. Farm management data is the data related to farm management including fuel use, crop storage, and product quality. Sensor data contains observation data in the field acquired using sensor devices. Further analysis of database schemas of DIPPA can be seen in Appendix A.

3.2 Project Goal

The project goal is to develop a software system to store and structure manual and automated sensing data from various sources with different data structures without necessarily knowing internal database schemas of the system. The software system is called DASSICO - Data Storing and Structuring for Information disCOvery. The data source of DASSICO can be files, sensors, or farm cloud servers. DASSICO enables users or external systems in retrieving the data for their purposes. Figure 3.2 depicts a data flow diagram of DASSICO.

Farmers are not only able to upload farm data to DASSICO, but also able to ask the system getting farm data from farm clouds or sensor devices in farm fields. Users can register themselves in DASSICO to search for available farms. Researchers can send a request to the system to access farm data. The external systems are data analytics or machine learning algorithms that access the data for data analysis purposes.

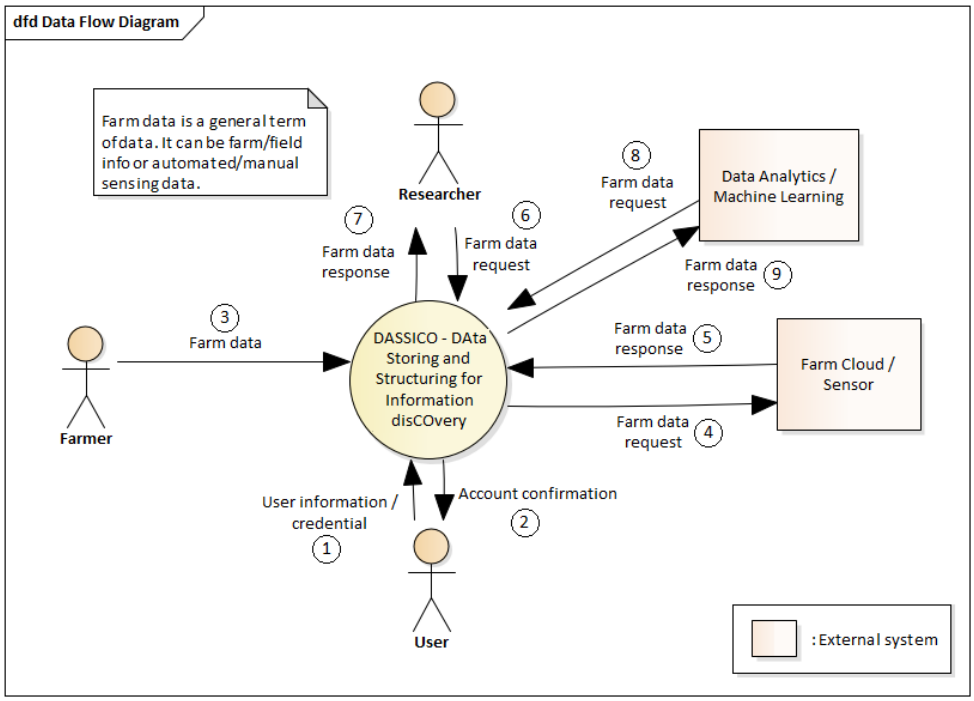


Figure 3.2: A diagram showing interactions of users and external systems with DASSICO

3.3 Project Challenges

Based on the analysis of the DIPPA prototype and these project goals, the following challenges are identified:

- **Integrating sensing data with varied data structures**

  Sensing data from files, sensors, or farm clouds may have diverse data structures to be utilized in DASSICO. Data mapping process is necessary to map and structure each element in sensing data as data source to a destination or a database in DASSICO. A limitation in the DIPPA prototype is on creating a task for data mapping process that needs good understanding of farm data and internal database schema in DIPPA. Finding specialists who know both domains is also difficult.

- **Metadata of the stored sensing data**

  Some researchers who used data from the previous prototype found that some sensing data lacked context or metadata. That hindered them in performing data analysis. Ensuring the input sensing data to have sufficient metadata would be a considerable challenge in the system design.

The following chapters present the stakeholders who were considered in the project to achieve the project goals and describe how the system solves these challenges during system design and development.

4 Stakeholder Analysis

This chapter discusses stakeholder analysis on the project. It describes the interests, concerns, and involvement of each stakeholder in the project that need to be considered in shaping and achieving project goals.

4.1 Stakeholders

4.1.1 Proeftuin voor Precisielandbouw

The stakeholders from Proeftuin voor Precisielandbouw share knowledge and project requirements on the problem domain for the trainee. Table 4.1 shows the main stakeholders from Proeftuin voor Precisielandbouw.

Table 4.1. List of stakeholders from Proeftuin voor Precisielandbouw

Jakob de Vlieg	
Role	Project owner
Responsibility	· Monitoring and controlling the progress, quality of the design, and development process · Providing regular feedback on the status of the project · Reviewing PDEng thesis and documentations
Interest	· Having proper quality of project process and result · Ensuring that the project gives added value to the farmers, the universities, and the companies

Paul van Zoggel	
Role	Project mentor
Responsibility	· Sharing domain knowledge and related documents · Providing detailed guidance on the project for the trainee · Monitoring and controlling the progress, quality of the design, and development process · Providing regular feedback on the status of the project · Reviewing PDEng thesis and documentations
Interest	· Having proper quality of project result and its process · Ensuring the project to give added value to the farmers, the universities, and the companies

Jacob van den Borne (Van den Borne Aardappelen)	
Role	Farm owner of Van den Borne Aardappelen
Responsibility	· Providing current data sources · Sharing domain knowledge, experience, challenges, and vision on precision agriculture
Interest	· Utilizing farm data without a vendor locked-in (dependent on a certain supplier) · Harvesting data from a new sensor in the farm · Saving time and money to explain the meaning of sensing data to researchers for data preparation and analysis

4.1.2 Eindhoven University of Technology

The university plays a role in the academic aspects of the project. The university also provides a supervisor to guide and share expert knowledge on the problem domain for the trainee. Table 4.2 lists the main stakeholders from the university.

Table 4.2. List of stakeholders from TU/e

Yanja Dajsuren	
Role	University supervisor and Program director of PDEng Software Technology (ST)
Responsibility	As university supervisor: · Monitoring and controlling the quality and progress of the project and project report · Supporting the trainee with relevant knowledge for the problem domain As program director of ST PDEng: · Evaluating the trainee · Ensuring the quality of the PDEng ST program
Interest	· Having proper quality of project result and its process matching PDEng criteria · Keeping successful cooperation with the industrial companies and universities to continue

Dimas Satria	
Role	PDEng trainee
Responsibility	· Managing and executing the project (making project plan, executing it, and reviewing it regularly) · Writing PDEng thesis and documentations
Interest	· Gaining project management and software design experience · Completing the project on time

Software Engineering Project (SEP) Students	
Role	Group of bachelor students to develop subset of the system
Responsibility	· Developing subset of the system · Writing documentation
Interest	· Completing project on time

Table 4.3. List of other stakeholders

Contact Person	**Role / Affiliation**	**Responsibility**	**Interest**
Puck Mulders	Researcher at TU/e	Getting farm data to be analyzed	· Having clean and structured data · Accessing farm data
Ger Snijkers	Centraal Bureau van Statistiek		

4.1.3 Other Stakeholders

Other stakeholders are potential future users of the system. Table 4.3 shows the interest and involvement of these stakeholders.

4.2 Stakeholder Prioritization Map

The power and interest of the stakeholders of this project is depicted in Figure 4.1. In the figure, the power axis shows how much the stakeholder can influence the project. The interest axis displays the level of interest of the stakeholders in the project. Each quadrant in the figure is associated with communication strategy as follows:

- High power – High interest (Manage closely)

  The stakeholders in this area are the key players for the project. The inputs from them have much impact on the project direction.

- High power – Low interest (Keep satisfied)

  These stakeholders have high influence with the project, but they do not have much interest in the project. The needs of these stakeholders must be satisfied to maintain or increase their interest.

- Low power – High interest (Keep informed)

  These stakeholders may not have big influence on the project. Discussing with them about the project in detail can give valuable information for the project.

- Low power – Low interest (Monitor)

  They do not want to be regularly updated about the project. They also do not have a big impact on the project. It is good to keep them informed about the project once in a while.

The detailed communication plan with each stakeholder is described in Table 4.4.

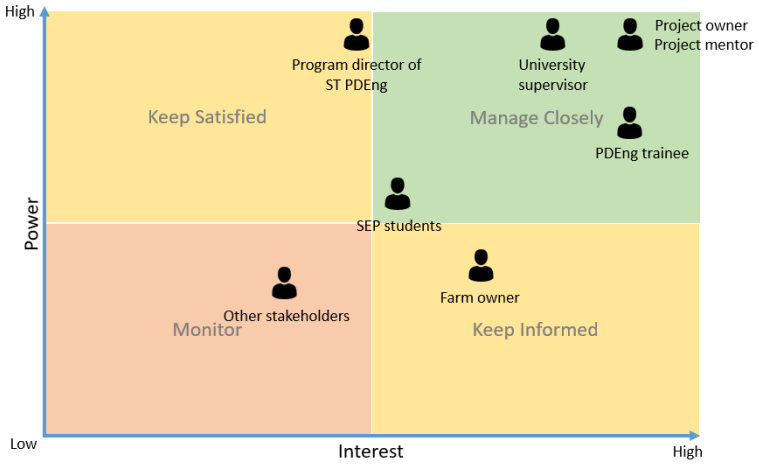


Figure 4.1: Stakeholder prioritization map

Table 4.4. Detailed communication plan with the stakeholders

Name	**Frequency**	**Communication Method**
Jakob de Vlieg	Throughout the project via Project Steering Group meetings	Face-to-face meeting, email
Paul van Zoggel	Throughout the project via weekly progress and Project Steering Group meetings	Face-to-face meeting, email, chat via WhatsApp
Jacob van den Borne	Throughout on demand-based meetings	On demand face-to-face meeting or via Paul van Zoggel
Yanja Dajsuren	Throughout the project via weekly progress and Project Steering Group meetings	Face-to-face meeting, email, phone call, chat via WhatsApp
SEP Students	Weekly progress and on demand-based meetings from 2 September until 8 November 2019	Email and face-to-face meeting
Puck Mulders (Researcher at TU/e)	Throughout on demand-based meetings	Email and face-to-face meeting
Ger Snijkers (Centraal Bureau van Statistiek)	Throughout on demand-based meetings	Email and face-to-face meeting

5 Requirements Analysis

This chapter presents a list of requirements that have to be satisfied within the project time frame. Use case analysis is carried out to describe expected behaviors of the system from the point of view of the end users. Several discussions with relevant stakeholders were conducted to collect a set of use cases. Functional requirements and non-functional requirements are derived from the use cases.

5.1 Use Cases

A use case diagram was depicted in Figure 5.1 based on the interaction of the system and its users. The main use cases, in yellow color, are described in detail in this section. Main actors in the system are:

- **System admin** represents the end user in charge of managing the system.
- **General user** represents the general end user who is registered in the system.
- **Farm admin** represents the end user who has full access to a certain farm in the system, manages its resources, and sets access permission of other users on the resources.
- **Farmer** represents the end user who is responsible to manage resources of a certain farm in the system.
- **Researcher** represents the end user who has more access permission than a general user to restricted resources of a farm in the system.

A matrix of role-based permission of a user is shown in Table 5.1. In the table, a user with a certain role is allowed to perform an action (read, edit, or delete) on farm data if it shows a green-colored box. If the box is red, the user is not permitted to perform the action. The yellow box is defined for a user permission that depends on an accessibility status of the data. The accessibility status can be public or private. If an accessibility status of a data is set as public, a general user can read it. Otherwise, the user may not carry out the action.

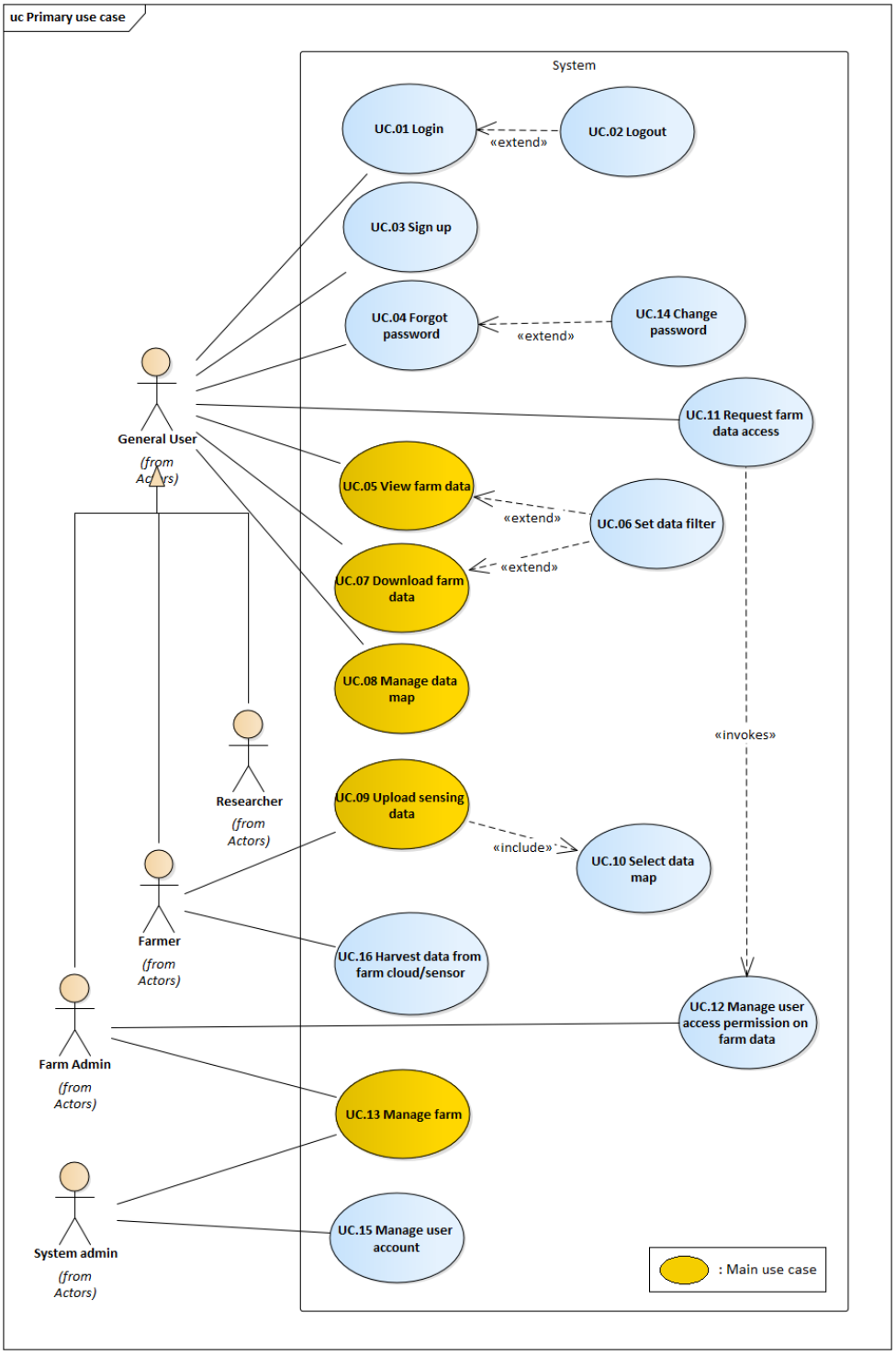


Figure 5.1: Use case diagram of the system

Table 5.1. Role-based permission matrix of a user on a certain farm

Role	Farm Data														
	Farm Information			Field / Crop Field			Farm User Access			Sensing Data			Equipment		
	R	C/U	D	R	C/U	D	R	C/U	D	R	C/U	D	R	C/U	D
Farm admin															
Farmer															
Researcher															
General user															

Legend:
R : Read
C/U : Create or update
D : Delete

Table 5.2. UC.05 View farm data

Name	**UC.05 View farm data**
Actor	General user
Pre-condition	The user has already logged in.
Post-condition	The user gets available farm data.
Main success scenario	1. The user sets parameters to filter data. 2. The user views available farm data.
Extensions	N/A

Table 5.3. UC.07 Download farm data

Name	**UC.07 Download farm data**
Actor	General user
Pre-condition	The user has already logged in.
Post-condition	The user gets farm data.
Main success scenario	1. The user sets parameters to filter data. 2. The user downloads selected farm data.
Extensions	N/A

Table 5.4. UC.08 Manage data map

Name	**UC.08 Manage data map**
Actor	General user
Pre-condition	The user has already logged in.
Post-condition	Data map is created, updated, or deleted.
Main success scenario	1. The user chooses the option to view a list of data file maps. 2. The system presents a list of data maps.
Extensions	1.A The user creates a new data map. 1.A.1 The user chooses an option to create a new data map. 1.A.2 The user describes data type of each column in a file to be uploaded. 1.A.3 The user saves the data map. 1.B The user updates an existing data map. 1.B.1 The user chooses an option to update a data map. 1.B.2 The user edits the data type of selected columns. 1.B.3 The user saves the changes. 1.C The user deletes an existing data map. 1.C.1 The user chooses an option to delete a selected data map. 1.C.2 The system sends a confirmation message. 1.C.3 The user confirms to delete a selected map. 1.C.4 The system deletes the data map.

Table 5.5. UC.09 Upload sensing data

Name	**UC.09 Upload sensing data**
Actor	Farmer
Pre-condition	Farmer has already logged in and selected a field.
Post-condition	Sensing data is stored in the system.
Main success scenario	1. Farmer uploads a sensing data file. 2. Farmer selects an available data map. 3. The system stores the data.
Extensions	2.A Farmer does not have a data map for the file. 2.A.1 Go to UC.08.1.A in Table 5.4

Table 5.6. UC.13 Manage farm

Name	**UC.13 Manage farm**
Actor	Farm admin
Pre-condition	Farm admin has already logged in.
Post-condition	Farm is created, updated, or deleted.
Main success scenario	1. Farm admin chooses the option to view a list of all owned farms. 2. The system presents a list of the farms.
Extensions	1.A Farm admin would like to add new farm. 1.A.1 Farm admin chooses the option to add new farm. 1.A.2 Farm admin inputs farm information. 1.A.3 Farm admin saves farm information. 1.B Farm admin would like to update farm information 1.B.1 Farm admin chooses the option to update farm information. 1.B.2 Farm admin updates farm information. 1.B.3 Farm admin saves the changes. 1.C Farm admin would like to delete an existing farm 1.C.1 Farm admin chooses the option to delete a selected farm. 1.C.2 The system sends a confirmation message. 1.C.3 Farm admin confirms to delete a selected farm. 1.C.4 The system deletes selected farm.

5.2 Functional and Non-Functional Requirements

Each requirement is assigned a priority level to indicate the importance of it in the project. The priority categories for the requirements in this project are defined based on MoSCoW approach as follows [10]:

- Must have —absolute requirement of the specification
- Should have —important requirement, but not necessary for the delivery of the project
- Optional have —optional requirement

5.2.1 Functional requirements

The functional requirements are categorized into three groups based on system behaviors: data import and export, data filter and search, and authorization and authentication. The relationships between use cases and functional requirements are shown in Figures 5.2 to 5.4. The detailed functional requirements with acceptance criteria and priority are shown in Tables 5.7 to 5.9.

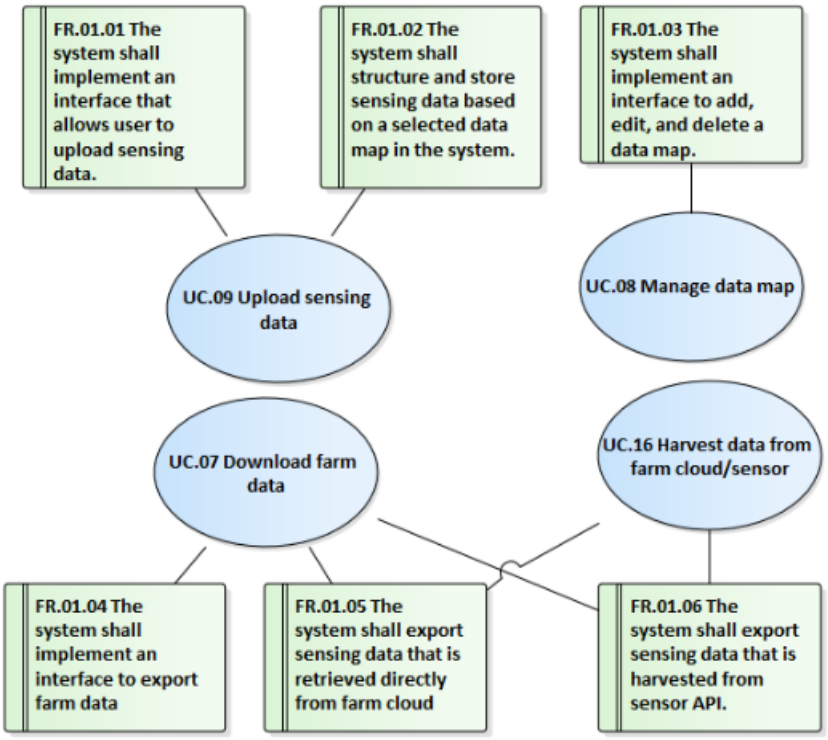


Figure 5.2: Relationship between use cases and functional requirements for data import and export

5.2.2 Non-functional requirements of the system

Besides the functional requirements, relevant non-functional requirements for the system are identified after discussions and analyses with the stakeholders. These requirements are categorized based on ISO/IEC 25010 standard [11]. Relevant non-functional requirements that needs to be satisfied by the system within this project time frame are listed in Table 5.10.

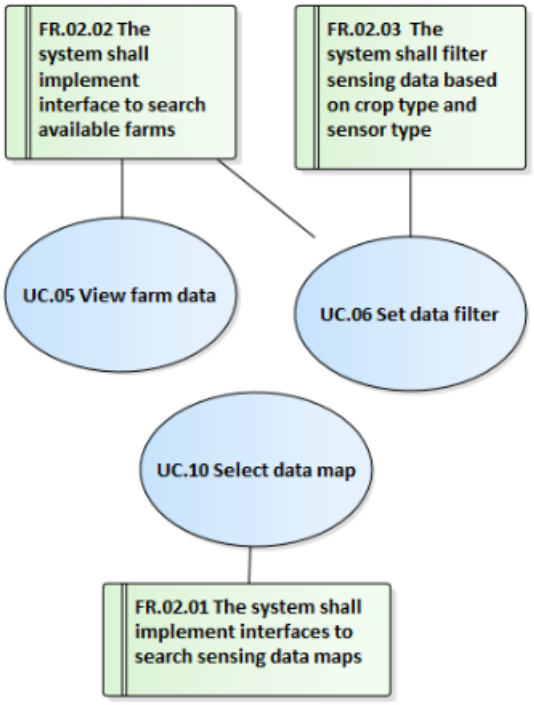


Figure 5.3: Relationship between use cases and functional requirements for data filter and search

Table 5.7. Functional requirements for data import and export

ID	Description	Priority
FR.01.01	The system shall implement an interface that allows user to upload sensing data. *Acceptance criteria:* A CSV file containing sensing data is stored in the system.	Must
FR.01.02	The system shall structure and store sensing data based on a selected data map in the system. *Acceptance criteria:* Sensing data are stored in correct database tables and columns based on a data map.	Must
FR.01.03	The system shall implement an interface to add, edit, and delete a data map. *Acceptance criteria:* A new data map can be added, edited, or deleted.	Must
FR.01.04	The system shall implement an interface to export farm data *Acceptance criteria:* Farm data is downloaded.	Must
FR.01.05	The system shall export sensing data that is retrieved directly from farm cloud. *Acceptance criteria:* Sensing data from farm cloud is downloaded.	Should
FR.01.06	The system shall export sensing data that is harvested from sensor API. *Acceptance criteria:* Sensing data from sensor API is downloaded.	Should

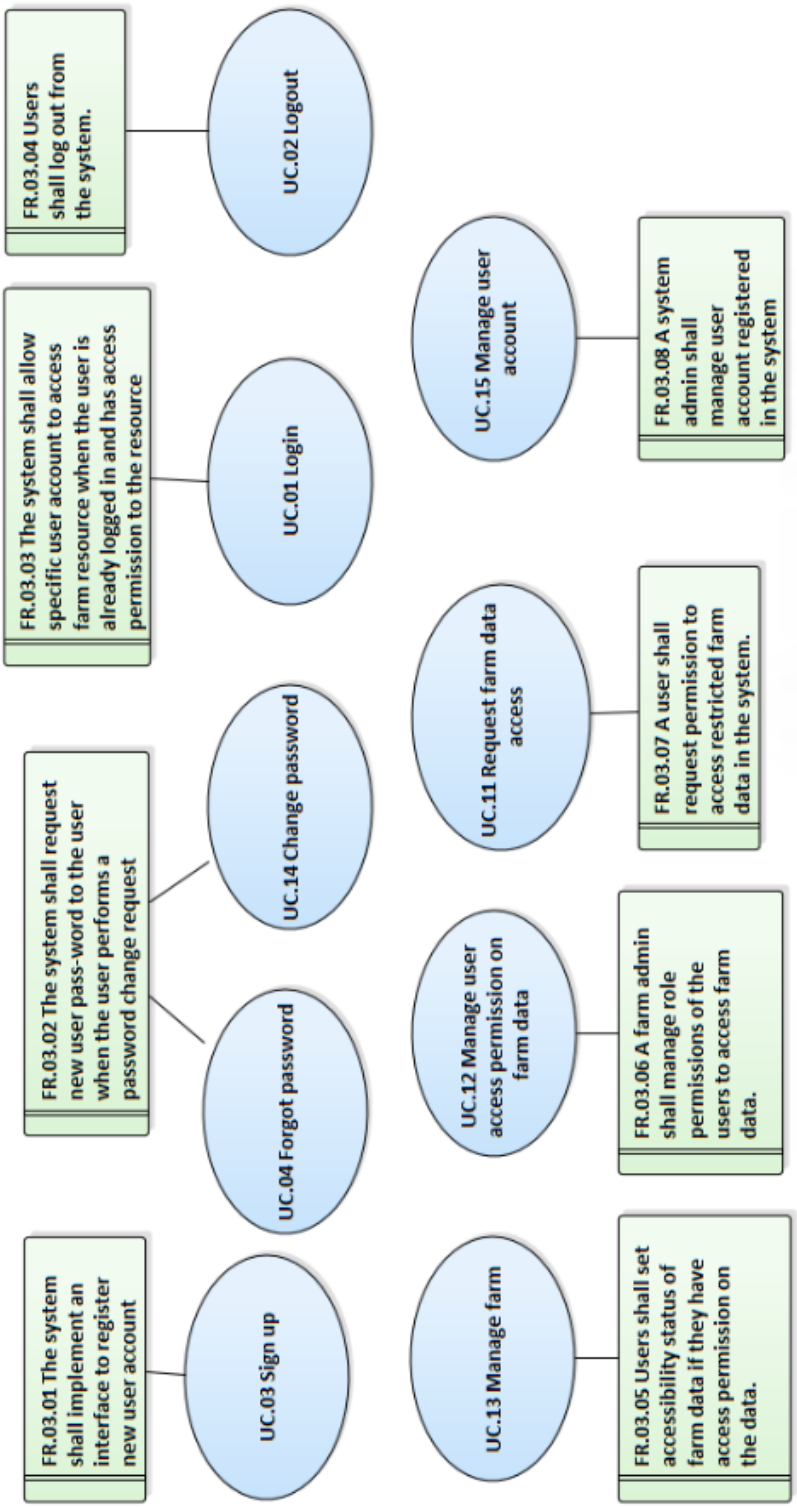


Figure 5.4: Relationship between use cases and functional requirements for authorization and authentication

Table 5.8. Functional requirements for data filter and search

ID	Description	Priority
FR.02.01	The system shall implement interfaces to search available data maps. *Acceptance criteria:* User can see a list of available data maps based on data map name.	Should
FR.02.02	The system shall implement interface to search available farms. *Acceptance criteria:* User can see a list of available farms	Optional
FR.02.03	The system shall filter sensing data based on crop type and sensor type. *Acceptance criteria:* User can see a list of available sensing data based on crop type and sensor type	Should

Table 5.9. Functional requirements for authorization and authentication

ID	Description	Priority
FR.03.01	The system shall implement an interface to register new user account *Acceptance criteria:* A user creates an account in the system.	Must
FR.03.02	The system shall request new user password to the user when the user performs a password change request *Acceptance criteria:* A user changes the password.	Optional
FR.03.03	The system shall allow specific user account to access farm resource when the user is logged in and has access permission to the resource *Acceptance criteria:* A user can perform an action on the resource if the user has access permission to it.	Must
FR.03.04	Users shall log out from the system. *Acceptance criteria:* A user cannot perform any action on the resource in the system after logging out.	Should
FR.03.05	Users shall set accessibility status of farm data if they have access permission on the data. *Acceptance criteria:* A user can set accessibility status of farm data if permitted.	Should
FR.03.06	A farm admin shall manage role permissions of the users to access farm data. *Acceptance criteria:* A farm admin can set a permission role to a user for accessing farm data.	Optional
FR.03.07	A user shall request permission to access restricted farm data in the system. *Acceptance criteria:* A farm admin gets notification and can accept or reject the request.	Optional
FR.03.08	A system admin shall manage user account registered in the system. *Acceptance criteria:* A farm admin can edit or remove other user accounts.	Optional

Table 5.10. Non-functional requirements

ID	Category	Description	Priority
NFR.US.01	Usability	The system publishes API documentation for clients or external systems to discover available services.	Must
NFR.US.02	Usability	A user interface of the system is designed with a good sense of aesthetic to ensure better interaction with the users.	Optional
NFR.CO.01	Compatibility	The system has access to get farm data of Van den Borne Aardappelen in Dacom server (farm cloud)	Should
NFR.CO.02	Compatibility	Data in the system is accessible by clients or external systems	Must
NFR.MA.01	Maintainability	The system is implemented with modular design to enable developing and maintaining parts of the system independently	Must
NFR.MA.02	Maintainability	The system is designed to add or modify features on one component has minimal impact on other components	Must
NFR.MA.03	Maintainability	The system is flexible to store data collected by sensors with various data structures	Must
NFR.MA.04	Maintainability	The system reuses design and implementation as much as possible from DIPPA prototype	Must
NFR.SE.01	Security	The system manages access permission with external system or clients to protect restricted resources or data that is stored in the system	Must
NFR.SE.02	Security	The system can delete all data copied from other farm clouds when the data owner requests data deletion (GDPR).	Optional
NFR.PE.01	Performance efficiency	When data request is received, the system gives feedback within two seconds	Must
NFR.RE.01	Reliability	The system can provide services to the clients or external systems despite the presence of software faults at other unrelated services	Must
NFR.RE.02	Reliability	The system has mechanisms to recover from failure in less than a day	Should

6 Software Architecture

This chapter presents a software architecture that defines a set of components and their relationships in DASSICO. Several architectural pattern alternatives for DASSICO were analyzed based on the requirements in Chapter 5. Important design decisions and technology choices are explained as well in this chapter.

6.1 Design Challenges

DASSICO needs to aggregate sensing data from various sources. The data sources can be files uploaded by farmers or sensing data collected from farm clouds or sensor devices (**NFR.CO.01**). Structuring the data allows the system to find requested data efficiently to satisfy business requests from data consumers. The data consumers can be system users or external systems, for example data analytics and machine learning algorithms.

DASSICO also needs to present available sensing data with its metadata for the data consumers. The users of DASSICO may access the data via a user interface or a front end. The external systems do not need the front end to access the data to perform analysis (**NFR.CO.02**).

A possibility to extend functionalities of DASSICO with minimal impact on other components is important (**NFR.MA.02**). Some stakeholders want to have some features to be implemented in the future that are beyond the scope of this project, for example a billing functionality for the users to download data from DASSICO.

6.2 Design Alternatives

Three design alternatives to address the design challenges in Section 6.1 are identified: monolithic, service-oriented architecture, and microservice. Figure 6.1 illustrates these architectural alternatives. The alternatives are studied and compared to address the quality attributes of the system as given in Table 5.10.

- **Monolithic**
  The concept of monolithic architecture is to combine different components of an application into one module on a single platform. The components in this architecture are not independently executed [12]. The front end and back end of the system are integrated into one application to be hosted and delivered together in this architecture.
- **Service-Oriented Architecture (SOA)**

Table 6.1. Comparison of design alternatives

Criteria	Monolothic	Service-Oriented Architecture	Microservices
Maintainability	-	+	+
Reliability	-	+	++
Performance	+	-	-

An application with SOA consists of a number of services in its logic part. Each service performs different tasks to satisfy different business requests. SOA employs Enterprise Service Bus (ESB) as a middleware to manage communications or route messages among services in the back end. The ESB also handles data transformation and mapping from data consumers to the back end and vice versa. All services in this architecture share a data storage [13, 14].

- **Microservice**
  Microservice architecture emerged from SOA [15]. It is structured by a set of independent services that collaborate to satisfy business requests from clients. The services in this architecture communicate with each other directly via HTTP protocol. An API gateway in this architecture routes incoming requests from clients to appropriate services [16]. This architecture can have shared data storage for many services or separate data storage for each service to allow development, deployment or scaling it independently.

A monolithic architecture can offer good performance due to fast communication among software components by sharing memory and code [17, 18]. However, maintaining a monolithic application can be difficult to do in isolation by different teams because the application is not modular. Making modifications on a component or adding new features can also be challenging due to high dependencies between components in the application. As all components are combined into one application, there is a big risk to system reliability. If one critical error occurs in a component of a monolithic application, the whole application will stop working and other service requests to other components cannot be satisfied.

Different from monolithic architecture, services in SOA are designed and developed in modular forms for better maintainability purposes. Thus, modifying a service or adding a new one is relatively easy due to isolation from other services. Singh and Tyagi stated that ESB also accounted for the reliability of SOA [19]. ESB can be quite complex to manage all service communications in one place. It can also be a single point of failure in SOA. If a service slows down and takes lots of requests with other services to complete a business request from a user, ESB will be clogged up with requests for that service. This might affect system reliability and performance of other services in SOA [20].

Microservice offers good maintainability because its services are modular and independent. It also provides better reliability compared to SOA and monolithic architectures if a failure occurs in a service, the system still can handle other requests to other services due to independent of the services. Microservice might negatively affect system performance because a message passing between services can increase response time for each request [21].

The architecture of DASSICO follows the microservice pattern because of its benefits in maintainability and reliability compared to other alternatives. Table 6.1 shows a summary of design alternative comparison. In the table, the scale of comparison is from double minus (- -) as the worst to double plus (++) as the best.

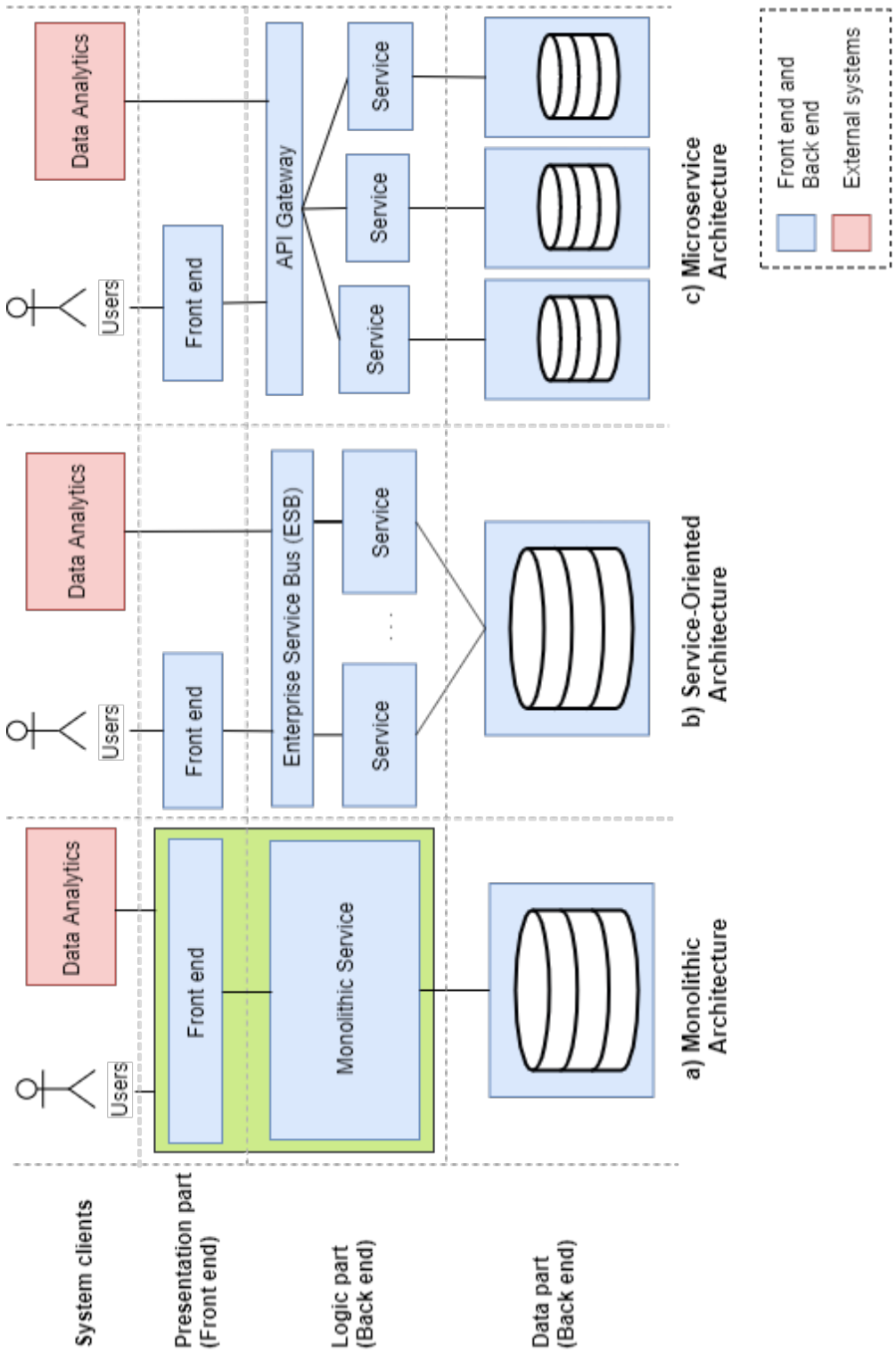


Figure 6.1: Illustrations of (a) monolithic, (b) SOA, and (c) microservice architecture

Table 6.2. A relationship matrix between the functional requirements and the services in DASSICO

Service	Functional Requirements
User auth	FR.03.01, FR.03.02, FR.03.03, FR.03.04, FR.03.06, FR.03.07, FR.03.08
Data map	FR.01.03, FR.02.01
Sensing	FR.01.01, FR.01.02, FR.01.04, FR.02.02, FR.02.03, FR.03.05
Data link	FR.01.05

6.3 High-Level Design of DASSICO

A high-level design of DASSICO can be seen in Figure 6.2. In this figure, a front end is a user interface for users to interact with DASSICO (**NFR.US.02**). An API gateway is directing incoming requests from users or external data consumers, for example data analytics, to be handled by responsible services in the back end.

A set of functional requirements of DASSICO were grouped into several services based on system subdomains as shown in Figure 6.2. A system subdomain contains a set of strongly related functionalities to satisfy a business request [16]. Table 6.2 shows functional requirements to be satisfied by the respective services in DASSICO. The services are:

- **User Auth Service**

  User Auth service is responsible for security requirement **NFR.SE.01**. This service consists of functionalities for user authentication and authorization. This service manages personal data and access permission of system users.

- **Sensing Service**

  This service contains functionalities to manage general farm data and sensing data. It is also responsible for structuring sensing data to be stored in a database.

- **Data Map Service**

  This service manages the metadata of sensing data for mapping and structuring processes in Sensing service.

- **Data Link Service**

  This service manages connections and handles incoming data from farm clouds and/or sensor devices as data producers. Farm clouds are cloud servers that store sensing data of farmers. Sensor devices that can be connected with this service are the sensors with APIs to harvest their data logs or sensor readings.

Those services handle business requests that operate with different data categories in the system. Three main categories of data are user data, farm data, and data-link data as illustrated in Figure 6.2. User data contains the roles of users in farms and personal data, such as name, password, and email address. Farm data contains farm operational data and sensing data. Examples of farm operational

Table 6.3. Some tactics to mitigate performance issue

Attribute: Performance				
ID	**Architectural decisions**	**Sensitivity**	**Tradeoff**	**Risk**
T.PE.01	Increase resources	Server	-	-
T.PE.02	Computational concurrency	Services and database	-	-
T.PE.03	Reduce computational overhead	Services	-	-
T.PE.04	Data caching	Front end and database	-	-

data are field locations, type of crops in a field, and soil type. Data-link data contains configurations to build connections with sensor devices or farm clouds. Chapter 7 explains the services and the data categories in DASSICO in more detail.

To mitigate performance issue that is identified in Table 6.1, some tactics were addressed as shown in Table 6.3. Increasing resources, such as using faster processors and networks, to the server can potentially improve latency in DASSICO to satisfy business requests. Additionally, introducing computational concurrency to process requests in parallel for the services and database can reduce blocked time to satisfy multiple requests. On top of that, decreasing computational overhead is a tactic to optimize the process in the services and improve latency. Then, employing a data caching tactic on the front end and database can reduce the time to load the same data multiple times. All of these tactics have no relevant trade-offs and risks to other architectural decisions.

6.4 Technology Choices

The selection of technologies is an important step to design and implement DASSICO. The DASSICO's design gives flexibility in using different technology to develop each system component. The decision to choose the technology depends on who is responsible for building the component. Figure 6.2 shows the responsibility distribution with the SEP students, as mentioned in Chapter 4, to design and implement system components. The PDEng trainee was responsible to design and develop the components in blue and the students' part was in green.

6.4.1 The PDEng Trainee's Part

In this section, the decisions to choose technologies in DASSICO taken by the PDEng trainee are explained.

API Gateway

An API gateway in microservice architecture routes all incoming requests from data consumers to services and encapsulates the internal structures of the system. When a service is added or removed from the system, the gateway's routing settings need to be updated as well. Thus, the gateway should have good performance and easy to configure.

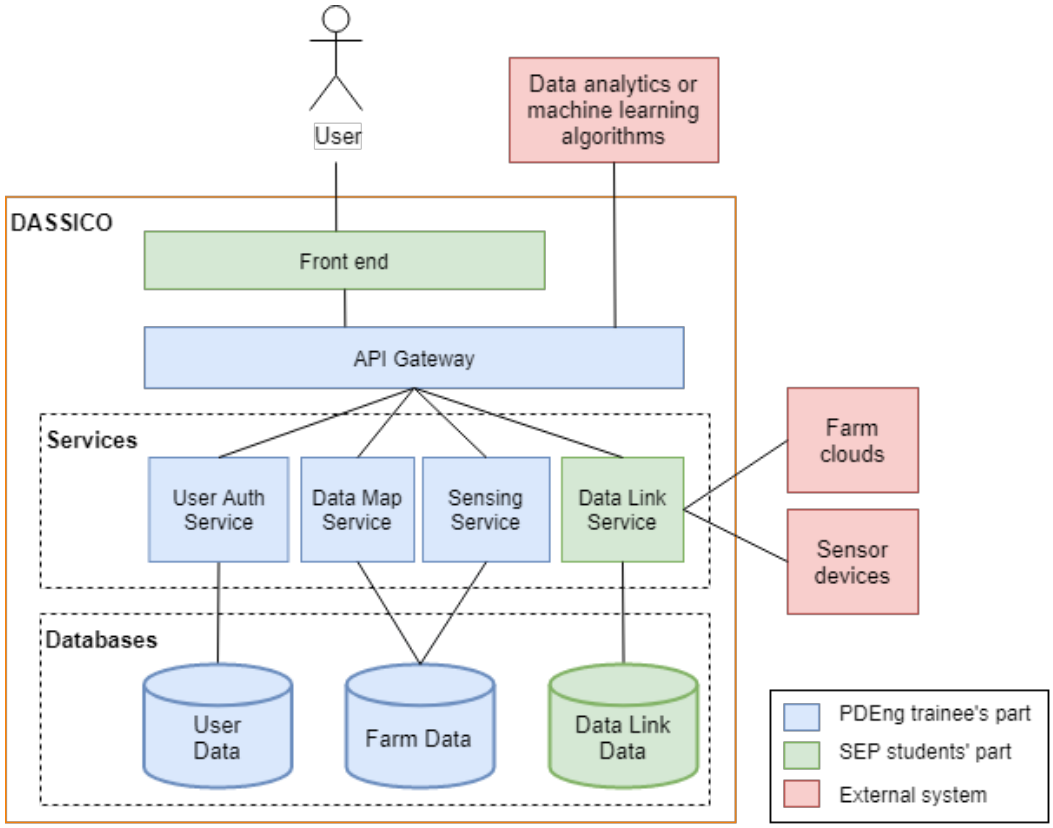


Figure 6.2: Architecture design of DASSICO

Compatibility requirement **NFR.CO.02** applies to API gateway to enable data analytics or machine learning algorithms accessing data in DASSICO. Two major protocol options to allow web-based applications to interoperate with external systems are Representation State Transfer (REST) and Simple Object Access Protocol (SOAP) [22]. Table 6.4 shows comparison between REST and SOAP. REST protocol is chosen because it requires less bandwidth and allows data caching to improve performance. Moreover, it supports more data formats than SOAP. REST protocol can use Secure Sockets Layer (SSL) and Hypertext Transfer Protocol Secure (HTTPS) to improve security on data exchange between DASSICO with the external systems.

Tyk[1] and Kong[2] are two platform alternatives explored in this project to implement API gateway. Both alternatives have community editions that are free to use and open-source. Kong outperformed Tyk in processing requests [23]. Moreover, Kong is relatively easy to install and manage compared to Tyk. Based on those reasons, Kong community edition was chosen for an API gateway in this project.

System Services

The PDEng trainee was responsible to design and implement User Auth, Data Map, and Sensing Services. A programming language chosen to develop those services is Python because the trainee has more experience in using Python than other existing languages for building web-based applica-

[1] https://tyk.io/
[2] https://konghq.com/

Table 6.4. Comparison between REST and SOAP for compatibility

Parameter	**REST**	**SOAP**
Data format	Supporting many data formats including plain text, HTML, XML, and JSON	Supporting only XML
Bandwidth	Requiring fewer resources (lightweight)	Requiring more resources and bandwidth that can affect performance
Data cache	Possible caching to provide better performance	Caching not supported
Security	Using SSL and HTTPS	Using WS-security and SSL

tions. That would save time to develop the services than to learn other languages within the project timeframe.

Python can also run on many different operating systems. That enables the system to be deployed in a machine with different operating systems that are supported by Python. During the analysis phase of this project, Python version 3.7 was selected because it was the latest stable version and would be kept maintained longer by Python organizations compared to other its predecessors.

Two alternatives of Python web frameworks to build services in a microservice architecture are Django[3] and Flask[4]. Django is a free and open-source web framework that enables developers to create a common web application with its built-in libraries. The features in Django can help the developers to build complex and large systems easily without necessarily using third-party libraries or extensions.

Flask is a lightweight web framework that can accelerate the development of simple web applications with required functionalities. It offers flexibility for developers to choose and use extensions for Flask if necessary.

Lazar stated that Flask had better performance than Django in loading and responding data from a remote server [24]. He also mentioned that Flask generally required a shorter time to load data from a database compared to Django (**NFR.PE.01**). Thus, the Flask framework was selected to build User Auth, Data Map, and Sensing Services.

Database Technology

User, farm, and sensing data are the data to be stored in the DASSICO's database. They are all categorized as structured data, considering the sensing data in the scope of this project limited to automated and manual sensing data. A Relational Database Management System (RDBMS) is suitable to store structured data [25]. RDBMS can ensure integrity and consistency of the data. RDBMS allows flexibility to explore the data in database using SQL queries that is necessary for information discovery.

This report revisits two alternatives of the most popular RDBMSs that were compared for the implementation of the DIPPA prototype by Enkhthaivan [1]. They are PostgreSQL[5] and MySQL[6]. PostgreSQL was chosen over MySQL for the prototype because of its excellence in extensibility to

[3]https://www.djangoproject.com/
[4]http://flask.palletsprojects.com/en/1.1.x/
[5]https://www.postgresql.org/
[6]https://www.mysql.com/

add new data types, functions, index types, and so on while MySQL does not support for extensibility [26]. PostgreSQL also supports geospatial data handling via PostGIS extension that is necessary for geolocation (longitude and latitude) of sensing data. This extension provides dedicated data types and functions on database level that can reduce the development effort for a developer to handle geospatial data. MySQL has built-in geospatial data support, but it provides fewer functionalities than PostGIS. Thus, PostgreSQL was chosen for database technology for this project by considering its geospatial data support and extensibility.

6.4.2 SEP Student's Part

SEP students were responsible for designing and developing the front end, the data link service and the database for data link service, as shown in Figure 6.2. To increase the chance in achieving an optimal result, they chose the technologies that are familiar to them. React framework[7] with TypeScript programming language were selected to develop the front end. They also used TypeScript to develop Data Link Service. The chosen database technology for Data Link Service was PostgreSQL.

Using different technologies from the other parts that the PDEng trainee chose do not create any issue to both parties because of the modular design of DASSICO following microservice architectural pattern. The PDEng trainee and the SEP students worked in parallel and independently to develop their parts (**NFR.MA.01**).

After discussing about the high-level architecture design and chosen technologies in this chapter, the detailed designs of the services and the data models in DASSICO are explained in the next chapter.

[7] https://reactjs.org/

7 Detailed Design of DASSICO

This chapter gives a more detailed designs of DASSICO as introduced in Chapter 6, mainly the parts of DASSICO that were designed and implemented by the PDEng trainee. The other parts which the SEP students designed and developed, can be found in the documentation of their project, specifically in the Software Design Document [2].

7.1 DASSICO Services

The services that DASSICO provides are User Auth, Sensing, Data Map, and Data Link services. User Auth service manages user data, role and permission to access the resources of many farms. Sensing service manages the farm and sensing data objects. Data Map service handles metadata to explain the contents of sensing data. Data Link service handles configurations to harvest data from farm clouds or sensor devices directly.

This section only explains the designs of User Auth, Sensing, and Data Map services. The detailed design of Data Link service can be found in the project documentation from the SEP students [2]. A domain model was created to show the responsibility of services in DASSICO. Figure 7.1 depicts a domain model that describes an overview of the objects and their relationships in the services. The model is represented in a UML class diagram.

7.1.1 User Auth Service

This service is responsible for authorization and authentication of any incoming service request for security purposes (**NFR.SE.01**). Authorization is a process to verify that the service requester has permission to perform a certain operation. Authentication is a process to verify the identity of a service requester.

Open Authorization (OAuth) with JSON Web Token (JWT) is a common protocol for authentication and authorization in a microservice architecture [27–29]. In this project, User Auth service acts as an authorization server in OAuth protocol. This service can create a JWT token for a service requester with valid credentials. The token can be used to access protected services to perform business requests. Without a valid token, a request is rejected by the services. A JWT token can be created when a user performs a user registration action (**FR.03.01**) or logs in via the front end (**FR.03.03**).

An activity diagram to show a JWT token creation during user login can be seen in Figure 7.2. In this figure, a user performs a login action via the front end. Then, the User Auth service processes the request and retrieve user data from User database, a database for this service. If the user credentials are valid, this service creates a JWT token. The token can be used to request resources from other

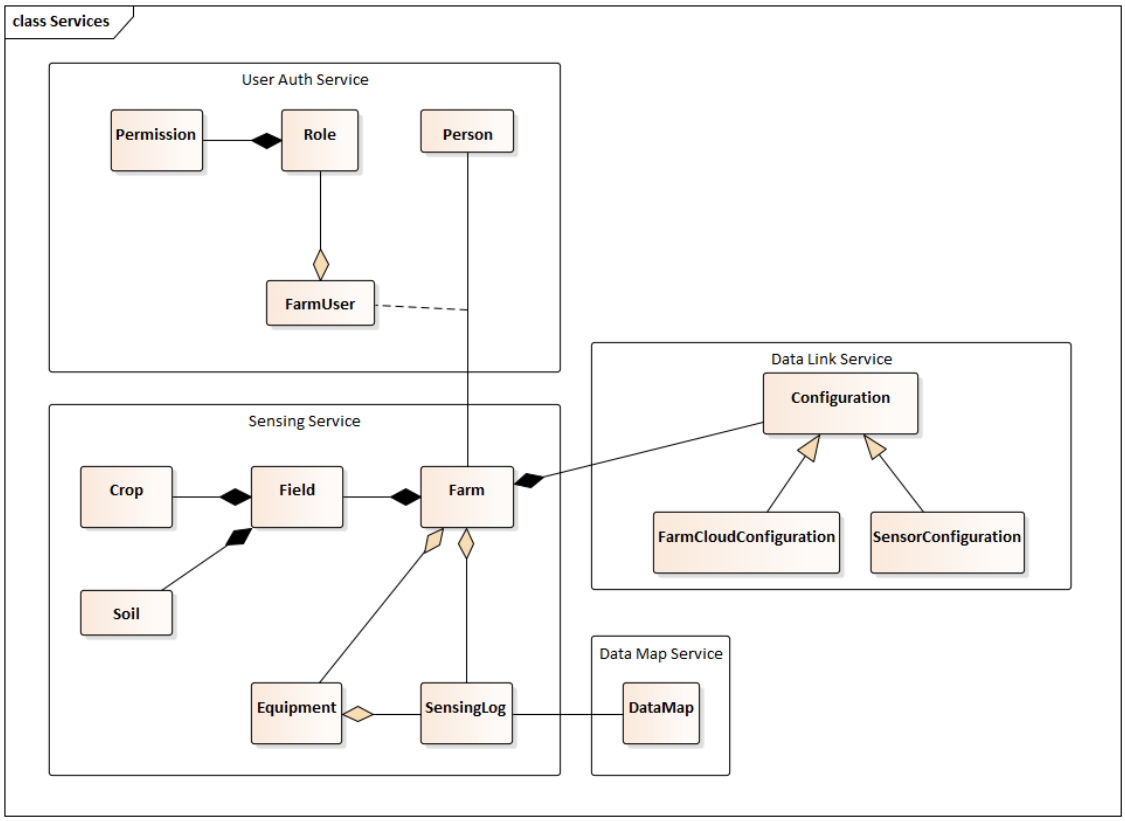


Figure 7.1: A domain model for the services in DASSICO

services in the system. Two alternatives to validate the token were identified as follows:

- Centralized token validation

  A User Auth service is the only service in the system that can create and validate JWT token. If a request with a JWT token comes to any service in the system, the service have to send the token to a User Auth service to validate.

- Global token validation

  All services in the system can validate a token that is created by a User Auth service. The services do not need to send the token to User Auth service to be validated.

Both alternatives offer some advantages and disadvantages to the system. A comparison of both approaches can be seen in Table 7.1. On one hand, applying centralized token validation offers better maintainability of the system than another alternative. On the other hand, global validation can give better performance because of no additional latency required for validating the token in a specific service.

The global validation approach can offer good reliability by spreading the loads of validation to the requested services. Thus, if there is any critical failure that makes User Auth service stop working, other services can validate JWT token. This risk can be mitigated in the central validation approach by load balancing the requests to multiple instances of User Auth service to improve reliability.

In this project, the centralized token validation was chosen because all services would be deployed in one server with an assumption that the additional latency for network communication between services within one machine was small. The latency would be much higher if the services were deployed on different servers.

Table 7.1. Token validation approaches for access control management

Criteria	Centralized validation	Global validation
Maintainability	Easy to maintain code to validate token only in one service Easy to apply another encryption algorithm or a validation mechanism in one service to create a token for security purposes	Code duplication to validate token in each protected service More effort to implement and test all protected services if there is a change in the token validation mechanism
Performance	Additional latency due to communication network to validate token	No additional latency for communication network
Reliability	Risk of critical failure in User Auth service making all services inaccessible	Offering better reliability by spreading the loads of validation process on all services

Figure 7.3 shows an activity diagram for verifying access permission on other services using centralized token validation approach. In this figure, authorization process is performed in User Auth service. If a token in an incoming request to a protected service is invalid, the request is denied. User role and permission in this project are already defined in Table 5.1 of the requirement analysis chapter.

7.1.2 Data Map Service

This service is responsible for managing metadata of sensing data to be utilized in DASSICO. The metadata should explain what types of the objects are observed, what parameters or characteristics of the objects to observe, when and where the observation is done, what measurement unit is used, and which equipment or sensor device is used to take measurements. Those are necessary for researchers to perform data analysis.

As a guideline for system users who want to provide the metadata without missing important details, a structure of metadata was defined in this project. A structure of a metadata is described in detail in Appendix B.

7.1.3 Sensing Service

Sensing service is responsible to utilize farm and sensing data. It stores and structures sensing data that is uploaded by a system user or harvested by Data Link service from sensor devices or farm clouds. Sensing service implements ETL process to structure sensing data at proper tables in the database.

Generally, creating tasks in an ETL process needs to know an internal database schemas of the system. In this case, the schemas of Farm Data could be revealed to the users. To encapsulate the Farm Data's schemas from the users, the ETL process makes use of a data map that contains metadata of the sensing data to be stored. A data map can be retrieved from Data Map service. Figure 7.4 illustrates that ETL process uses a data map to structure and load sensing data to the database.

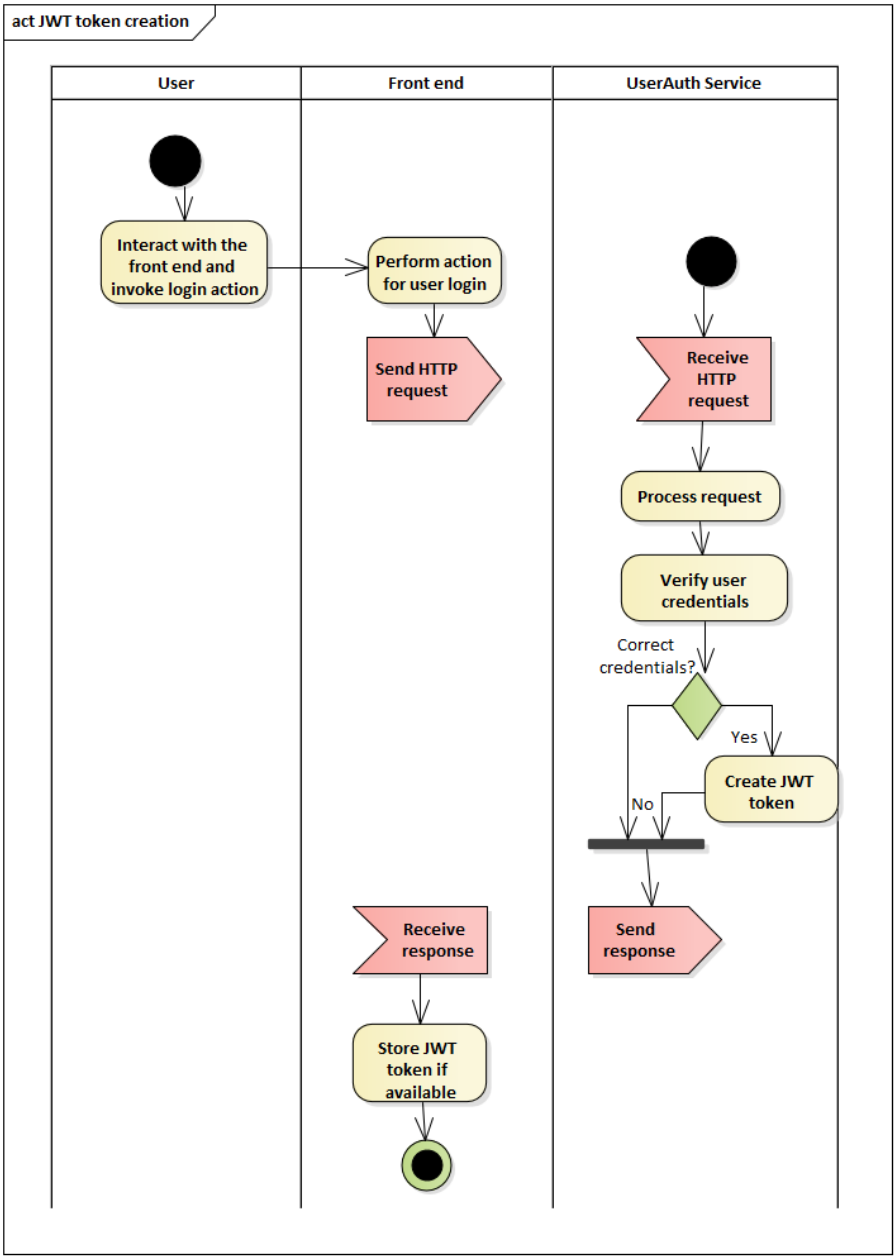


Figure 7.2: JWT token creation in User Auth Service

The ETL process in the prototype uses Pandas as a common Python library to manipulate data and transform it into SQL queries to store it in the database. Pandas was chosen because it could be deployed together with the service in any machine as long as a Python program could run in it.

An activity diagram for data integration in Sensing Service can be seen in Figure 7.5. In this figure, Sensing service receives data from the front end or Data Link service. Sensing service checks the data and sends a request to Data Map service to get a data map. If the selected data map is available, an ETL process runs to structure the data and load it into a database.

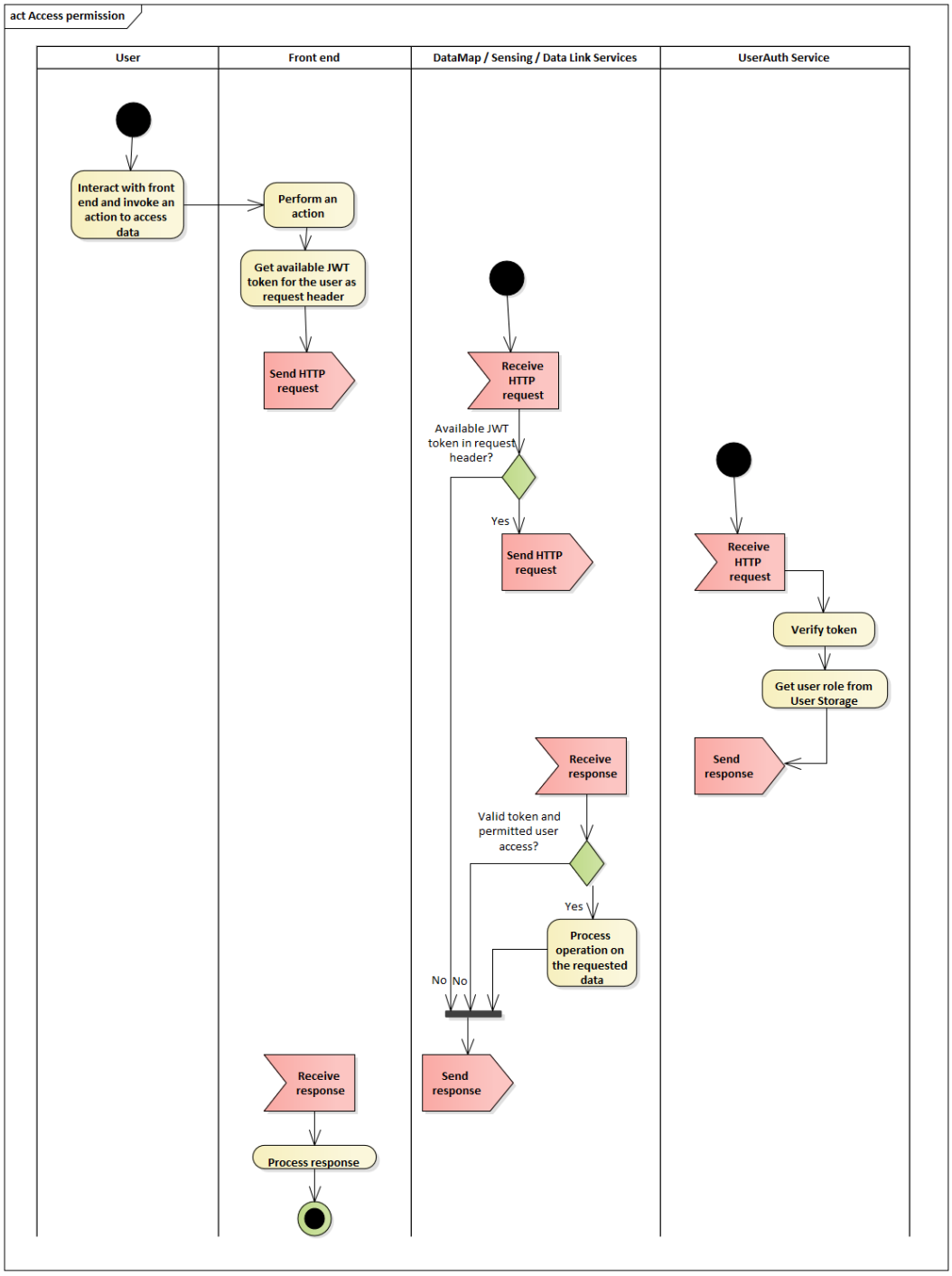


Figure 7.3: Handling access permission with User Auth Service as an authorization server

7.2 Data Model

This section explains data models in the databases to persist data from the system services. Two reference models were used to define the data models in this project. They are a DIPPA prototype and

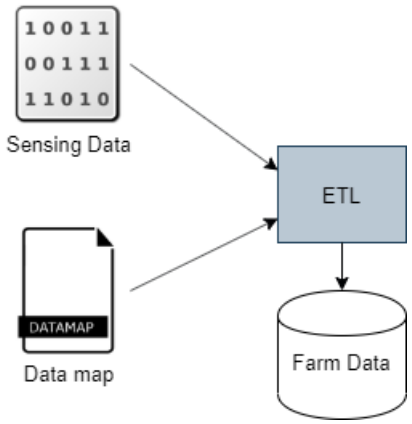


Figure 7.4: Structuring sensing data using a data map in ETL process

rmAgro as described in Appendix A.

7.2.1 User Database

A data model in a dabase for User Auth service keeps user data and role permissions. The data model can be seen in Figure 7.6. In this data model, User table contains data of the users that are registered in DASSICO. Role-based access permission's settings are stored in Role, RolePermission, PermissionType, and Resource tables. FarmUser table defines the roles of users in farms.

7.2.2 Farm Database

A database to store farm data is structured by three different data models: farm data, sensing data, and data map. In this section, each data model in this storage is presented.

Farm Data Model

Farm data model holds general farm data (name, address, postal code, and so on), field and crop field data. Field data describes field area, size, and soil type. Crop field data determine one or more areas within the borders of a field that define which crop is planted and the planting period. Figure 7.7 represents a farm data model in the database.

Sensing Data Model

Sensing data model defines a flexible structure to store data from a variety of sensor systems or equipments as seen in Figure 7.8. This model also keeps data about an object or context being observed and measurement condition, for example the depth of an equipment under ground level to take measurements.

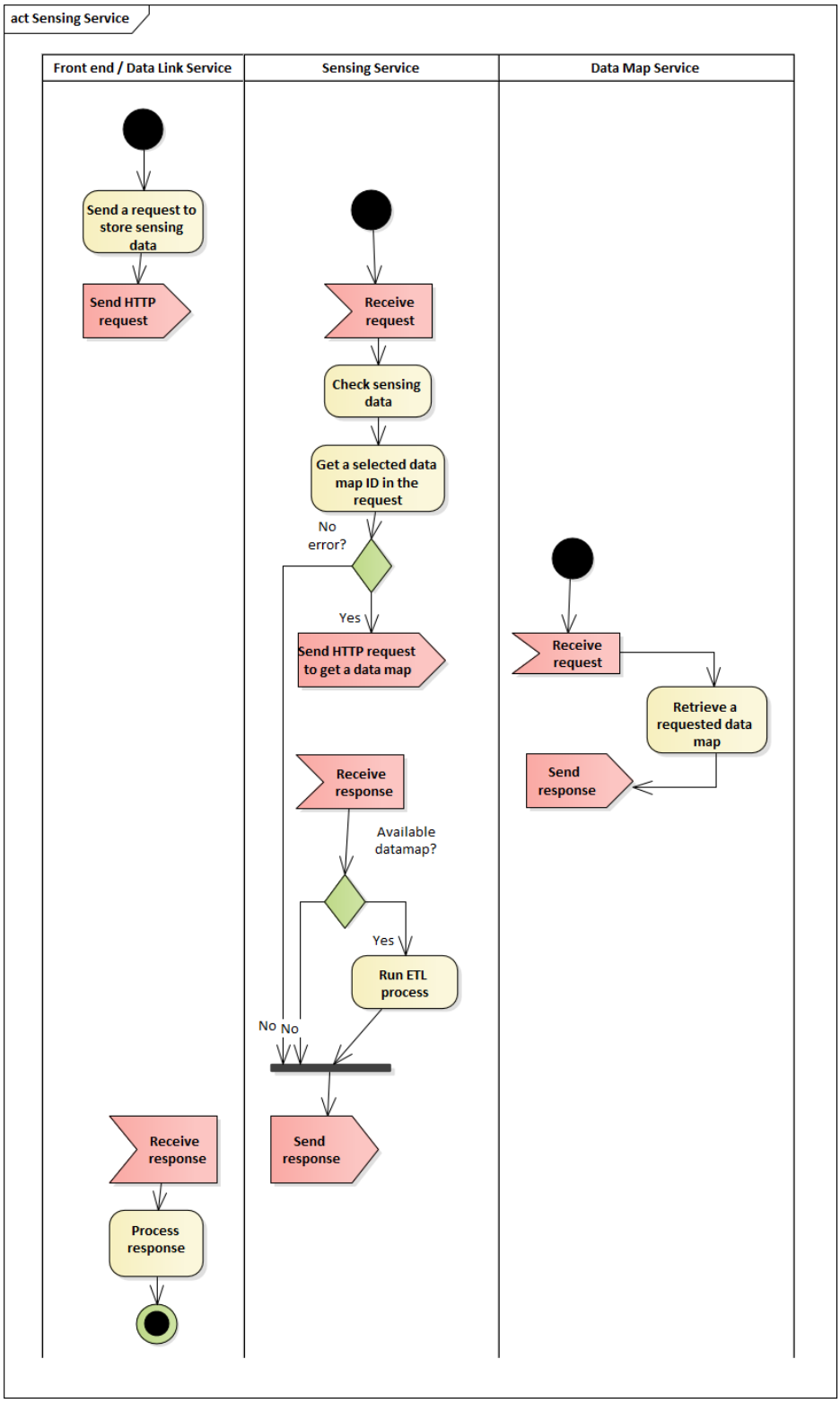


Figure 7.5: An activity diagram for data integration in Sensing Service

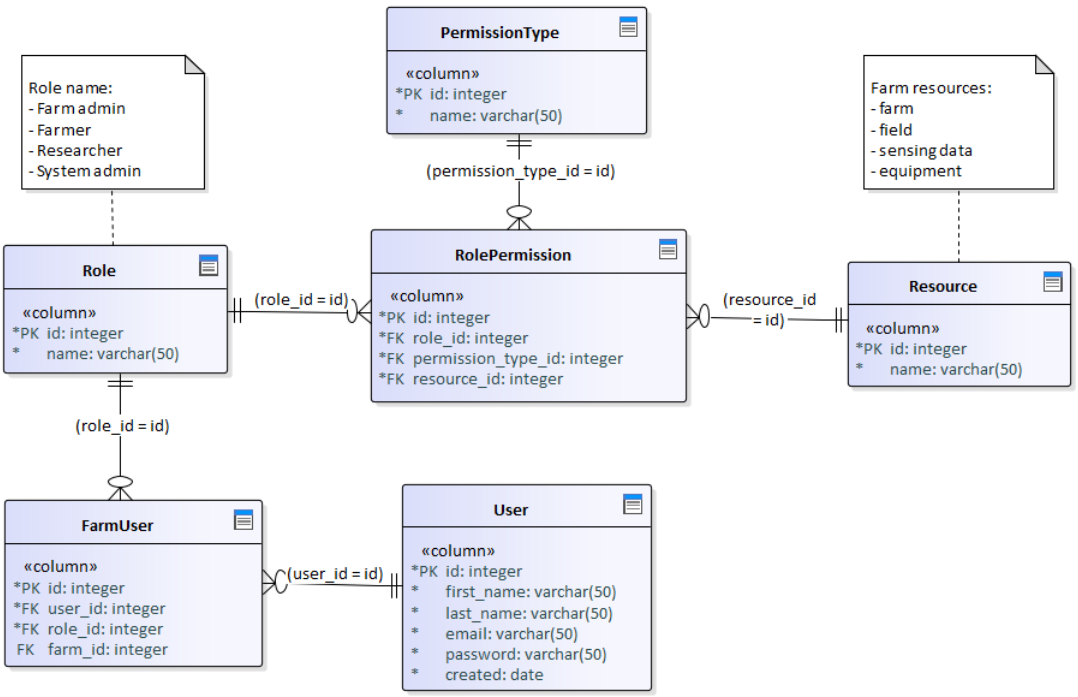


Figure 7.6: User data model

Data Map Model

Equipments or sensors with the same model or brand, series, and software version generally produce same data structure. From that assumption, a data map model was designed to organize data maps or metadata of sensing data that are related to particular equipment models. The data map is necessary for structuring sensing data in Sensing service. System users can use it to understand the measured data from a particular equipment. Figure 7.9 shows a data map model in Farm Data storage.

7.2.3 Data-Link Database

A data model in the database for Data Link service was designed by the SEP students. The data model is supposed to hold configuration data for farm clouds and sensor devices. Due to limited time, the students could design a data model for a specific sensor device, namely WolkyTolky sensor[1]. This data model keeps the location of equipment in a field, API key and equipment ID, and a table to cache or store the last saved data temporarily. Figure 7.10 shows a data model for Data Link service. The detailed explanation of the data model can be found in the project documentation of the SEP students [2].

[1] https://www.wolkytolky.com/en/

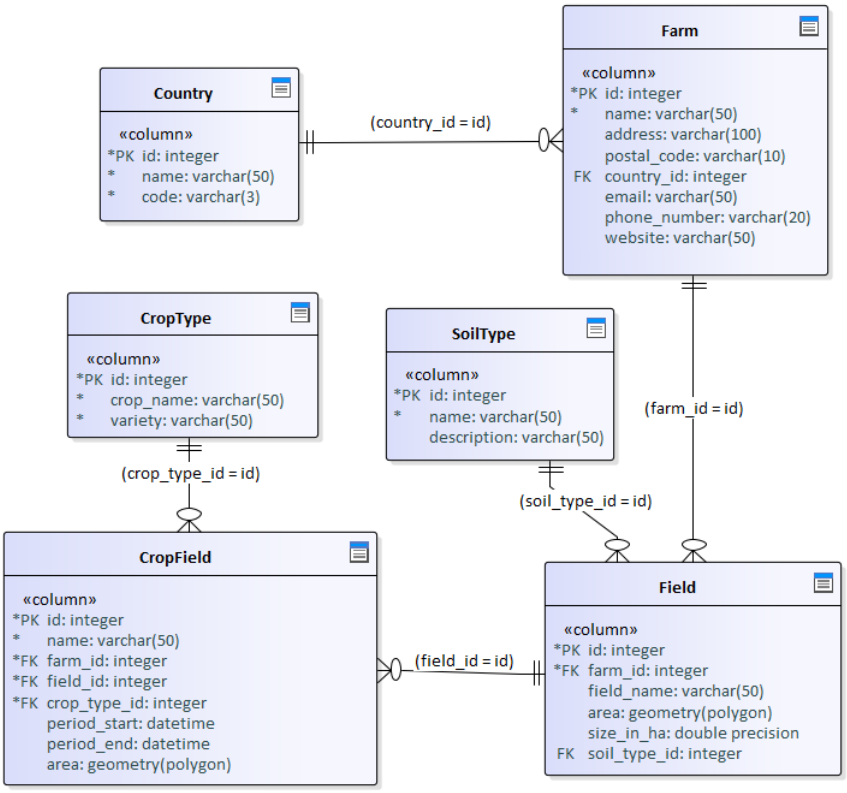


Figure 7.7: Farm data model

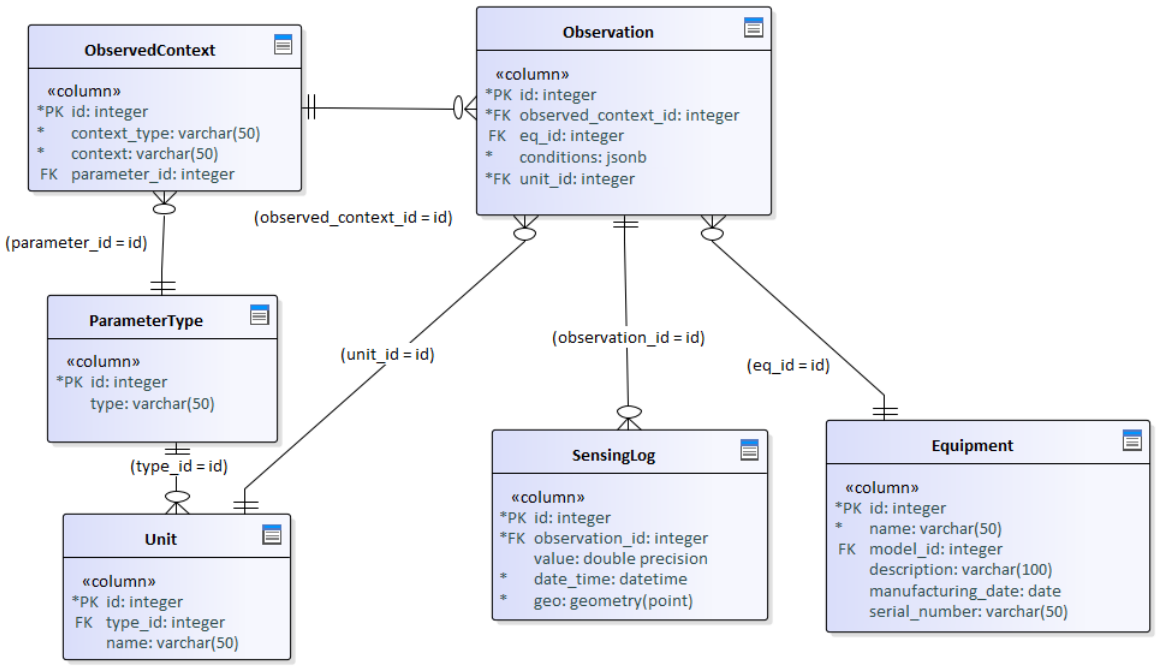


Figure 7.8: Sensing data model

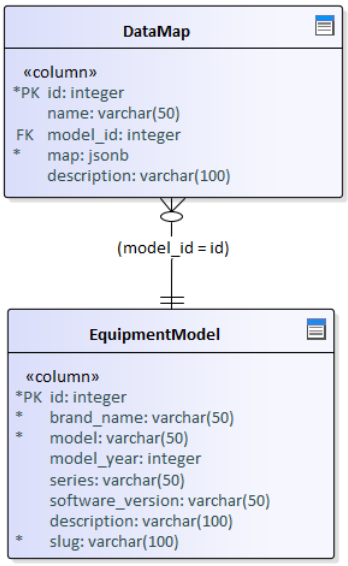


Figure 7.9: Data map model

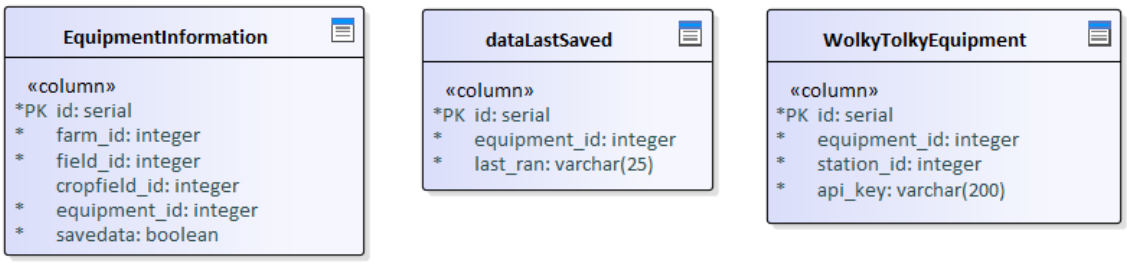


Figure 7.10: Data link model [2]

8 Implementation and Deployment

This chapter addresses a realization of the system prototype. The following sections explains implementation and deployment of each part in DASSICO.

8.1 Implementation

8.1.1 Front End

The front end of DASSICO was developed as a user interface by the SEP students. The front end design can be seen in Figure 8.1. It provides a login page for users to login or register as a new user. A dashboard view can be accessed if a user has already logged in. The available views in the dashboard are:

- Farm View: for creating a new farm or searching available farms from the database, shown in Figure C.2
- Live View: for presenting updated sensor readings in the fields of a farm, shown in Figure C.3
- History View: for showing available sensing data of selected fields, shown in Figure C.4
- Fields View: for managing fields and planted crops of a farm, shown in Figure C.5
- Datamaps View: for creating and searching data maps of equipments, shown in Figure C.6
- Equipments View: for showing and managing available equipments in a farm, shown in Figure C.7
- Farm Settings View: for managing farm settings, for example accessibility status, name, website, and the roles of other users in a farm, shown in Figure C.8
- Personal Settings View: for managing personal data of the user, shown in Figure C.9

8.1.2 Back End

In this section, implementation of the DASSICO's back end is described. The functionalities that are supported by the services in the back end are documented in Swagger Hub[1]. Publishing the documentation in Swagger Hub enables external systems to discover all services in DASSICO easily

[1] https://app.swaggerhub.com/apis-docs/dimsc/DataStorage/1.0.0/

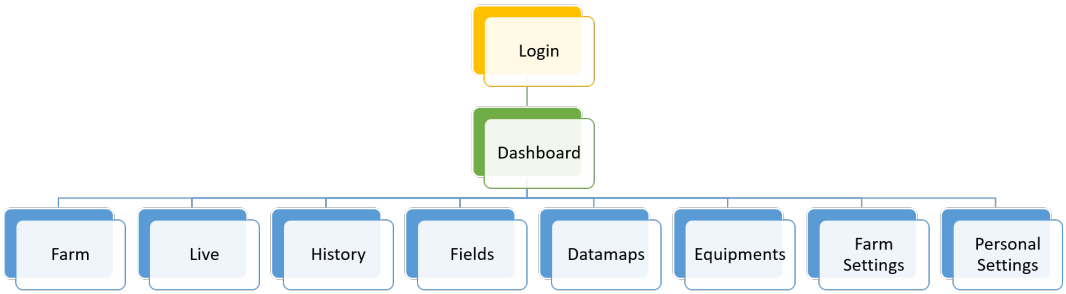


Figure 8.1: The front end design [2]

(**NFR.US.01**). Any external systems that need to use system services can look at the supported functionalities in the documentation. Figure 8.2 shows a screenshot of the API documentation in Swagger.

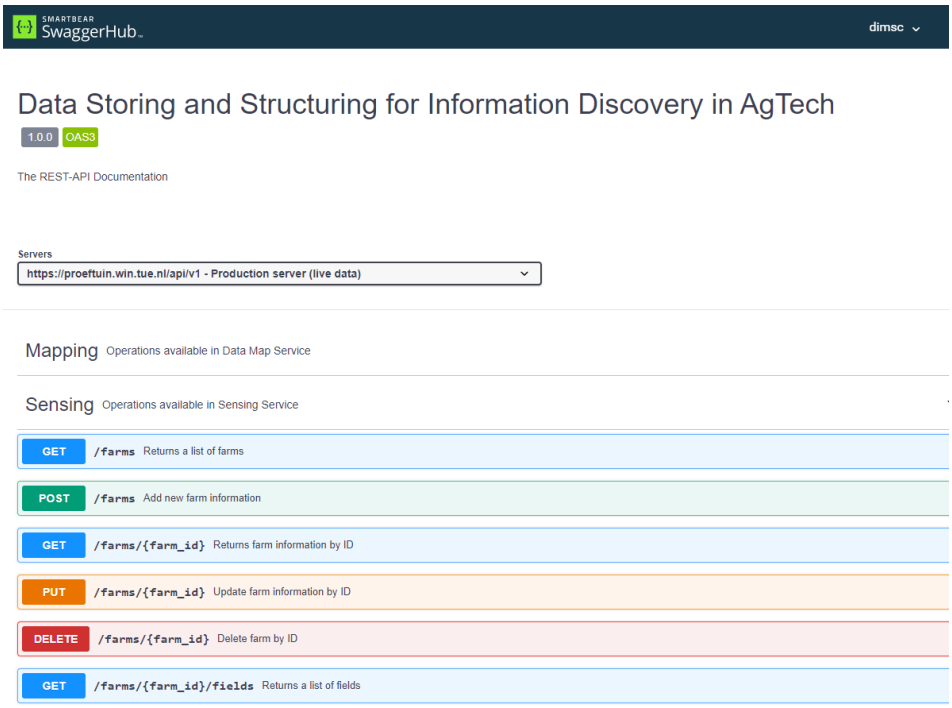


Figure 8.2: An API documentation of the supported functionalities in the prototype

The UserAuth and Sensing Services created the database structures of User and Farm Data Storages, respectively, by using an Object-Relational Mapping (ORM). Figure 8.3 shows the database structures in User and Farm databases. The implementation of a database for Data Link service is described in a project documentation from the SEP students [2].

8.2 Deployment

This section describes the deployment of DASSICO prototype. Figure 8.4 shows the deployment diagram of the prototype. All components were hosted in the servers of the Eindhoven University of

```
             List of relations
 Schema |      Name        | Type  |  Owner
--------+------------------+-------+----------
 public | farm_user        | table | postgres
 public | resource         | table | postgres
 public | role             | table | postgres
 public | role_permission  | table | postgres
 public | user             | table | postgres
(5 rows)
```

(a) Database tables for user data

```
             List of relations
 Schema |         Name         | Type  |  Owner
--------+----------------------+-------+----------
 public | accessibility_status | table | postgres
 public | country              | table | postgres
 public | crop_field           | table | postgres
 public | crop_type            | table | postgres
 public | data_map             | table | postgres
 public | equipment            | table | postgres
 public | equipment_model      | table | postgres
 public | farm                 | table | postgres
 public | field                | table | postgres
 public | observation          | table | postgres
 public | observation_location | table | postgres
 public | observed_context     | table | postgres
 public | owner                | table | postgres
 public | parameter_type       | table | postgres
 public | sensing_log          | table | postgres
 public | soil_type            | table | postgres
 public | spatial_ref_sys      | table | postgres
 public | unit                 | table | postgres
(18 rows)
```

(b) Database tables for farm data

Figure 8.3: Screenshots of database tables created by User Auth and Sensing Services

Technology. The specifications of the machine are:

- Operating System: Ubuntu 16.04.3 LTS 64-bit
- Hard disk space: 32 GB
- RAM space: 4 GB
- CPU core(s): 1 core @ 2.60GHz

The API gateway and system services are deployed separately using application containers to offer better maintainability. They can be deployed to run on other servers easily by using containers.

There are many alternatives to choose for the containers, such as Docker [2], KVM [3], rkt [4], and so forth. The performance of Docker and rkt containers are almost similar [30]. The performance comparison between Docker and KVM is also quite similar [31]. Docker was chosen because it has good supports from the company and the community, compared to other container technologies. Having a good support is related to maintainability requirements of the software that can be crucial if there is any issue or functional limitation in Docker in the future. The company can solve software bug and functional limitation issues and the community can help in providing solution to use the technology.

Docker has functionalities to monitor health condition of the component running in it. It also offers an automatic restart functionality. If a component deployed in Docker has a failure and crashed, Docker will automatically restart the component to run again (**NFR.RE.02**).

[2] https://www.docker.com/
[3] https://www.linux-kvm.org/
[4] https://coreos.com/rkt/

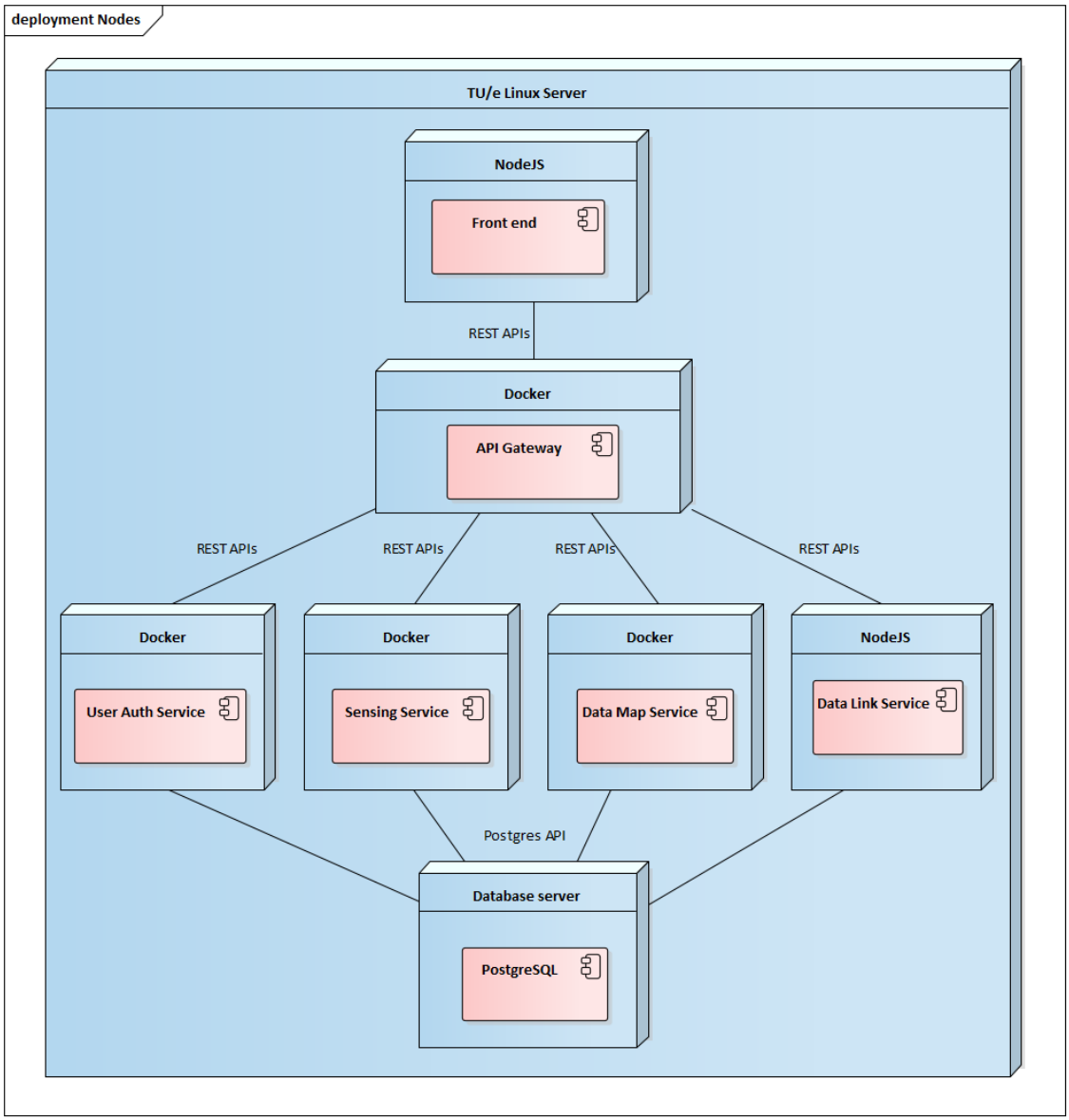


Figure 8.4: Deployment diagrams of the system

9 Verification and Validation

This chapter discusses the verification and validation processes of the software system against the requirements, listed in Chapter 5.

9.1 Verification

Verification of the system was accomplished by executing unit and integration tests. The integration tests were performed mainly during software components' integration by the SEP students and the PDEng trainee in the end of the SEP project. The components were verified formally passing a test plan [32]. The verification status of the functional requirements of the system is shown in Tables 9.1 to 9.3. The verification status of the non-functional requirements of the system is shown in Table 9.4.

Table 9.1. The verification status of the functional requirements for data import and export

ID	Description	Priority	Status
FR.01.01	The system shall implement an interface that allows user to upload sensing data.	Must	Satisfied
FR.01.02	The system shall structure and store sensing data based on a selected data map in the system.	Must	Satisfied
FR.01.03	The system shall implement an interface to add, edit, and delete a data map.	Must	Satisfied
FR.01.04	The system shall implement an interface to export farm data.	Must	Satisfied
FR.01.05	The system shall export sensing data that is retrieved directly from farm cloud.	Should	Not satisfied
FR.01.06	The system shall export sensing data that is harvested from sensor API.	Should	Not satisfied

FR.01.05 could not be satisfied because building a connection to Dacom, as a farm cloud, needed an application key and access request to be accepted by Dacom. Dacom accepted the request in the last weeks of system development phase. The SEP students who worked on that did not have time enough time to complete the functionality.

The system can collect data from a WolkyTolky[1] sensor via its API. FR.01.06 could not be satisfied because due to limited time to develop the functionality, the data from this sensor was only presented

[1] https://www.wolkytolky.com/en/

Table 9.2. The verification status of the functional requirements for data filter and search

ID	Description	Priority	Status
FR.02.01	The system shall implement interfaces to search available data maps.	Should	Satisfied
FR.02.02	The system shall implement interface to search available farms.	Optional	Satisfied
FR.02.03	The system shall filter sensing data based on crop type and sensor type.	Should	Satisfied

Table 9.3. The verification status of the functional requirements for authorization and authentication

ID	Description	Priority	Status
FR.03.01	The system shall implement an interface to register new user account.	Must	Satisfied
FR.03.02	The system shall request new user password to the user when the user performs a password change request.	Optional	Satisfied
FR.03.03	The system shall allow specific user account to access farm resource when the user is logged in and has access permission to the resource.	Must	Satisfied
FR.03.04	Users shall log out from the system.	Should	Satisfied
FR.03.05	Users shall set accessibility status of farm data if they have an access permission on the data.	Should	Satisfied
FR.03.06	A farm admin shall manage role permissions of the users to access farm data.	Optional	Satisfied
FR.03.07	A user shall request permission to access restricted farm data in the system.	Optional	Not satisfied
FR.03.08	A system admin shall manage user account registered in the system.	Optional	Not satisfied

directly in the front end. The data from this sensor was not stored in the system storage. Thus, the data cannot be exported from the system as a file.

For usability requirements, NFR.US.01 is satisfied by publishing API documentation in SwaggerHub as mentioned in Chapter 8. NFR.US.02 is not satisfied because the front end was designed without feedback from real users.

NFR.CO.01 could not be satisfied in this project due to time limitation. Access permission to Dacom API needs an approval from the company. It was approved by Dacom at the end of project time line. Thus, it needs more time to implement the code to utilize data from Dacom. NFR.CO.02 was satisfied by providing APIs to enable external systems accessing data in the DASSICO system.

Regarding to maintainability requirements, NFR.MA.01 and NFR.MA.02 were satisfied by applying microservice architectural pattern in the system. Each component in the system can be maintained and deployed independently. Employing metadata-based ETL approach in the Sensing service and the Sensing data model in the database satisfied NFR.MA.03 requirement to store and structure data with various data structures. For NFR.MA.04, the DASSICO system reused the data model designs

Table 9.4. The verification status of the system' non-functional requirements

ID	Category	Description	Priority	Status
NFR.US.01	Usability	The system publishes API documentation for clients or external systems to discover available services.	Must	Satisfied
NFR.US.02	Usability	A user interface of the system is designed with a good sense of aesthetic to ensure better interaction with the users.	Optional	Not satisfied
NFR.CO.01	Compatibility	The system has access to get farm data of Van den Borne Aardappelen in Dacom server (farm cloud)	Should	Not satisfied
NFR.CO.02	Compatibility	Data in the system is accessible by clients or external systems	Must	Satisfied
NFR.MA.01	Maintainability	The system is implemented with modular design to enable developing and maintaining parts of the system independently	Must	Satisfied
NFR.MA.02	Maintainability	The system is designed to add or modify features on one component has minimal impact on other components	Must	Satisfied
NFR.MA.03	Maintainability	The system is flexible to store data collected by sensors with various data structures	Must	Satisfied
NFR.MA.04	Maintainability	The system reuses design and implementation as much as possible from DIPPA prototype	Must	Satisfied
NFR.SE.01	Security	The system manages access permission with external system or clients to protect restricted resources or data that is stored in the system	Must	Satisfied
NFR.SE.02	Security	The system can delete all data copied from other farm clouds when the data owner requests data deletion (GDPR).	Optional	Not satisfied
NFR.PE.01	Performance efficiency	When data request is received, the system gives feedback within two seconds	Must	Not satisfied
NFR.RE.01	Reliability	The system can provide services to the clients or external systems despite the presence of software faults at other unrelated services	Must	Satisfied
NFR.RE.02	Reliability	The system has mechanisms to recover from failure in less than a day	Should	Satisfied

Table 9.5. Load test to determine the performance of DASSICO

Concurrent API calls / second	TU/e Server (1 thread)			Trainee's laptop (1 thread)			Trainee's laptop (multithread)		
	Min	Average	Max	Min	Average	Max	Min	Average	Max
5	362	7488	14401	358	9616	20800	349	363	418
10	825	12898	21290	363	12963	25719	352	272	436
20	2699	14975	25728	359	15444	29608	351	510	673

Table 9.6. Machine specifications for load testing

Parameter	TU/e server	The trainee's laptop
OS	Ubuntu 16.04.3 LTS 64-bit	Windows 10 64-bit
Hard disk space	32 GB	768 GB
RAM space	4 GB	32 GB
CPU core(s)	1 core @ 2.60 GHz	4 cores @ 2.80 GHz

of DIPPA prototype as explained in Appendix A.

NFR.SE.01 was satisfied by the DASSICO system by defining role-based access permissions for users. OAuth protocol with JWT was implemented to manage authentication and authorization of the users to access available services in the system.

Performance requirement NFR.PE.01 could not be satisfied by the system when it is deployed in TU/e server because the server only has one CPU. This requirement was satisfied by the system when the trainee performed the test in another machine that enables multithreading. The result of load test on DASSICO can be seen in Table 9.5. The machine specification for the TU/e server and the trainee's laptop can be seen in Table 9.6.

9.2 Validation

Validation of the system was a continuous effort. The system was validated by the main stakeholders via discussions and demonstrations. During the SEP project, the PDEng trainee was in charge of validating the results as a project client during the weekly demo sessions.

10 Conclusions

This chapter covers the results and the recommendations for the future work.

10.1 Results

This project goal is to provide a software system to store and structure sensing data for system users, as data integrators, without necessarily knowing any internal database schemas of the system. The system in this project employs metadata-based ETL approach to aggregate sensing data from many sources with diverse structures.

This approach separates the concern of the users away from the database schemas in the system. This will help the users to integrate their data in the system easily because they only need to provide sensing data and its metadata. A maintainer of the system has more flexibility to make necessary changes on the database in the future compared to the solution in the previous prototype that the users needed to know the database schema to integrate data.

The system's design has enabled the components in the system to be built in isolation. This design offers flexibility for developers to choose any programming language to implement the components and to deploy the components independently. Moreover, evolving the system to have new functionalities implemented in a new component in the future can be accommodated easily due to the structure of the system.

The system implements role-based access permissions to enable users in protecting or sharing their data with others. The system uses OAuth protocol with JWT to handle user authentication and authorization for any incoming request from the users.

10.2 Future Work

As future work, the following tasks could be carried out as follow-up of this project.

- The system performance is not optimized yet in this project. The implemented solution to increase the performance is limited to multi threading in the services. In the future, more design decisions, implementation, and tests are needed to increase the system performance.
- The data utilized in this project are manual and automated sensing data. Remote sensing data can be integrated for researchers to have more data analysis.
- The sensor device that is connected to the system is only a WolkyTolky sensor by calling its API. It should be possible to harvest data from other sensors in future work.

- Connecting the system with DACOM farm cloud could not be satisfied in this project. Access permission to DACOM via its API needs an approval from the company. They approved the request at the end of the project. Thus, connecting to DACOM needs more time to implementation work.

11 Project Management

This chapter discusses the project management —the planning and the risk analysis throughout the project. A mitigation strategy was devised for each risk to reduce threats in achieving the project goals within specified project time frame.

11.1 Project Planning

The project was carried out from May 1 until December 31, 2019. A process model and a detailed activity plan were defined to show the flow and activities to complete the project. The project activity plan was reviewed and discussed together with the company and/or university supervisors in weekly progress meetings and monthly Project Steering Group (PSG) meetings.

11.1.1 Project Process Model

A model represented in Business Process Model and Notation (BPMN) was created to show a visual representation of the process and activity in the project initially, as seen in Figure 11.1. The project was divided into four phases in this model: project initialization, research, design, and implementation and test. The research phase uses waterfall approach because this approach is suitable for planning tasks when the starting and ending time are defined. Researching

At the beginning of the project, the requirements of the project were not determined clearly. Thus, understanding the domain, stakeholders, and requirements were important to define a project direction. A concrete project planning was defined with a detailed tasks to be executed during system design, implementation, and test using using agile approach with one-week sprints.

The research phase focuses on understanding the data to be utilized, the design decisions, and the chosen technologies in the DIPPA prototype. The result of this phase was a report containing the designs and technologies in the prototype to be reused in this project.

Based on the result from the research phase, the architecture and design of the system are created. The responsibility in design phases are shared between the PDEng trainee and the SEP students. In the last phase, system implementation and test, the PDEng trainee and the SEP students worked in parallel and independently. The system components were integrated and tested incrementally with the SEP students for six weeks.

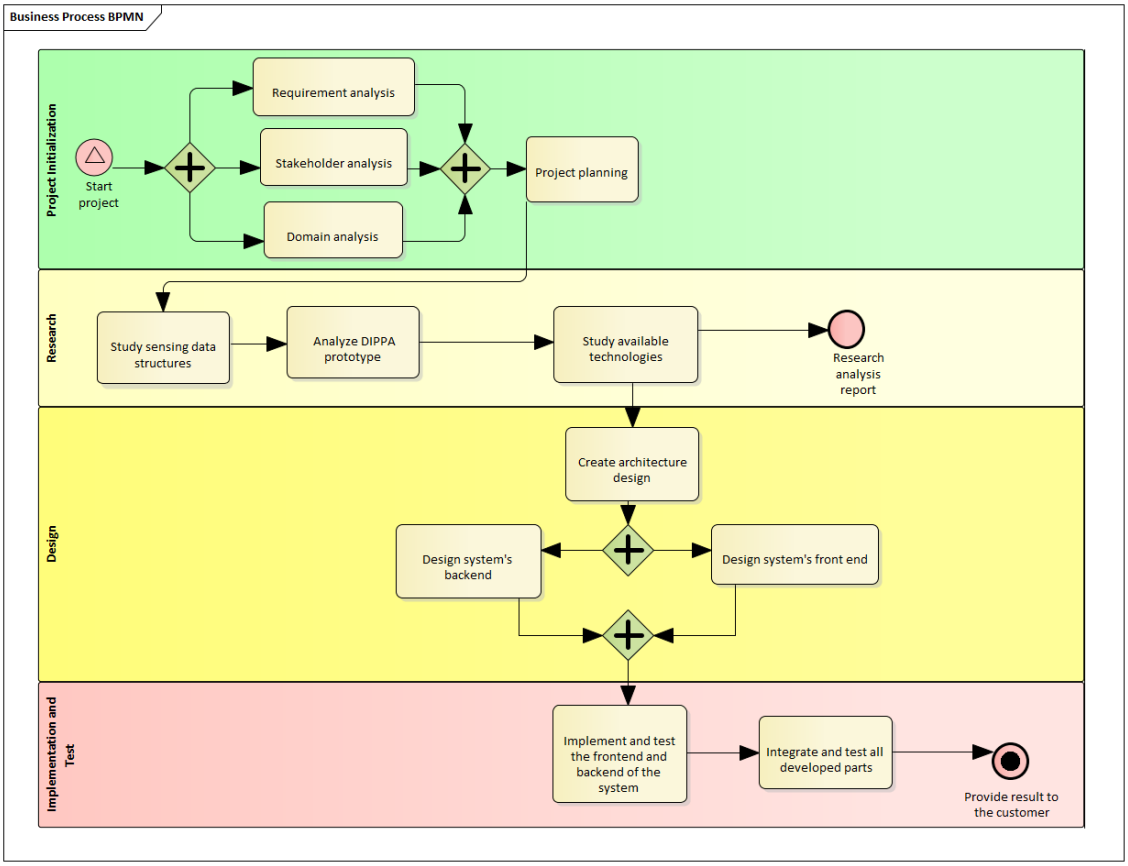


Figure 11.1: A BPMN model to show the process and activities in the project initially

11.1.2 Project Activity Plan

The project activity plan shows a detailed tasks executed in the project as seen in Figure 11.2. This plan was reviewed in PSG meetings and updated based on the discussions during the meetings. The preparation, kick-off meeting, and sprint planning for the SEP project is also integrated in this planning.

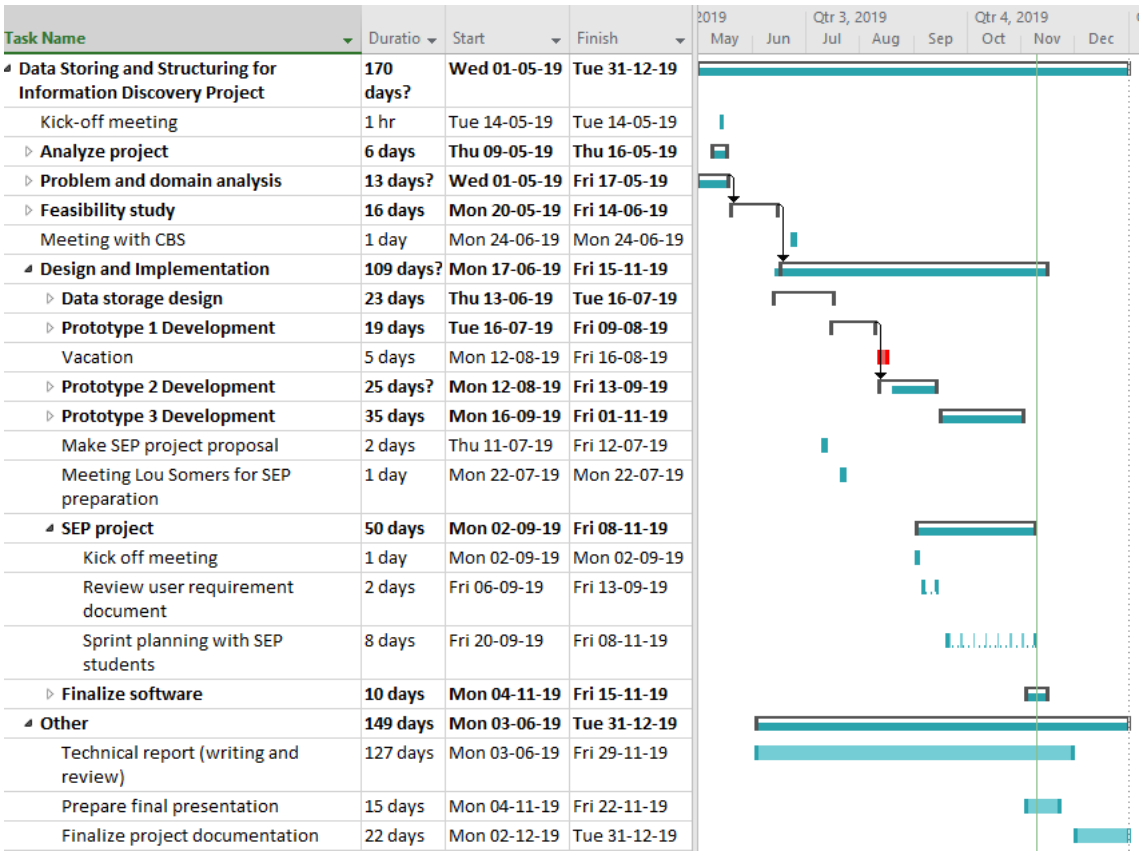
11.2 Risk Analysis

A list of the identified risks during the project is shown below. Impact level and likelihood of the risks are estimated in three scales: low, medium, and high.

- **R1: Incorrect time and effort estimation in project plan**

  Impact: **Medium**
  Likelihood of event: **High**

  The detailed tasks in the project are defined and executed based on the plan. The possible consequences of incorrectly estimating the time and effort for each task are the tasks do not



Task Name	Duratio	Start	Finish
Data Storing and Structuring for Information Discovery Project	**170 days?**	**Wed 01-05-19**	**Tue 31-12-19**
Kick-off meeting	1 hr	Tue 14-05-19	Tue 14-05-19
Analyze project	**6 days**	**Thu 09-05-19**	**Thu 16-05-19**
Problem and domain analysis	**13 days?**	**Wed 01-05-19**	**Fri 17-05-19**
Feasibility study	**16 days**	**Mon 20-05-19**	**Fri 14-06-19**
Meeting with CBS	1 day	Mon 24-06-19	Mon 24-06-19
Design and Implementation	**109 days?**	**Mon 17-06-19**	**Fri 15-11-19**
Data storage design	**23 days**	**Thu 13-06-19**	**Tue 16-07-19**
Prototype 1 Development	**19 days**	**Tue 16-07-19**	**Fri 09-08-19**
Vacation	5 days	Mon 12-08-19	Fri 16-08-19
Prototype 2 Development	**25 days?**	**Mon 12-08-19**	**Fri 13-09-19**
Prototype 3 Development	**35 days**	**Mon 16-09-19**	**Fri 01-11-19**
Make SEP project proposal	2 days	Thu 11-07-19	Fri 12-07-19
Meeting Lou Somers for SEP preparation	1 day	Mon 22-07-19	Mon 22-07-19
SEP project	**50 days**	**Mon 02-09-19**	**Fri 08-11-19**
Kick off meeting	1 day	Mon 02-09-19	Mon 02-09-19
Review user requirement document	2 days	Fri 06-09-19	Fri 13-09-19
Sprint planning with SEP students	8 days	Fri 20-09-19	Fri 08-11-19
Finalize software	**10 days**	**Mon 04-11-19**	**Fri 15-11-19**
Other	**149 days**	**Mon 03-06-19**	**Tue 31-12-19**
Technical report (writing and review)	127 days	Mon 03-06-19	Fri 29-11-19
Prepare final presentation	15 days	Mon 04-11-19	Fri 22-11-19
Finalize project documentation	22 days	Mon 02-12-19	Tue 31-12-19

Figure 11.2: The detailed activities in the project plan

have enough time to finish and the customer will be unsatisfied. Therefore, it is better to review project plan continuously and keep the customer updated about the challenges in executing the tasks.

- **R2: Technical risk (limitation in the framework)**

  Impact: **High**
  Likelihood of event: **Medium**

  There is a possibility that the chosen technology has a certain limitation due to software bugs in the framework or other reasons. This problem can delay the project progress or the development of the feature in the prototype cannot go further using the chosen technology. To mitigate this issue, we can try to find a workaround or choose another technology alternative. We can discuss with the supervisors about possible changes in project deliverables due to this issue.

- **R3: Wrong in defining priority for each requirement**

  Impact: **Medium**
  Likelihood of event: **Medium**

  Based on discussion with the stakeholders, the project requirements were identified. Then, the requirements were prioritized to be satisfied in the project time frame. There can be a

mistake in setting the priority of the requirements. If an important requirement was missing and not fulfilled, the stakeholders might be unsatisfied. Moreover, architecturally-significant requirement can be difficult and expensive to fix in the software system. To avoid this, we need to review requirement priority in every sprint if necessary.

- **R4: The SEP students need much support from the PDEng trainee to finish their project**

  Impact: **High**
  Likelihood of event: **Medium**

  Collaborating with the SEP students can put additional work for the PDEng trainee to prepare their project, meetings, and discussions. Those activities can consume too much time of the PDEng trainee. It is better to make sure that the requirements for the SEP students are defined as detailed and clear as possible, so that they can work independently and efficiently.

- **R5: Unavailability of the stakeholders for long time (unplanned event)**

  Impact: **Medium**
  Likelihood of event: **Low**

  Gathering relevant information and discussion to take some decisions are necessary to involve the stakeholders. If there is an unplanned event that the stakeholders are unavailable for long time, the project deliverables might be delayed. To mitigate this, we can work on the tasks that do not require immediate assistance from stakeholders.

- **R6: The PDEng trainee is absent for long time**

  Impact: **Medium**
  Likelihood of event: **Low**

  The PDEng trainee might be absent for a long time due special circumstances, for example falling sick, there is no other PDEng trainee to replace within the project timeline. The project scope and deliverables might need to be negotiated with the supervisors based on priority.

12 Project Retrospective

This chapter finalizes the report by providing a reflection based on the author's perspective.

Working on this project was a challenging and interesting experience. The challenges from managerial and technical perspectives have enabled me to spot my strengths and improvement points.

At the beginning, the project goal was rather high level and general. I had to allocate a significant amount of time to get the detailed requirements and prioritize them to be satisfied within the tight time of this project. Discussing with the stakeholders helped me in understanding the domain and customer's needs.

During the project, we decided to collaborate with the SEP students to develop the software system. I was aware that if the collaboration did not go well, that would affect the project negatively, for example possible delay in finishing project tasks due to counter-productive in the development. Integrating the system components that were developed by the trainee and SEP students was tough because many bugs were found from both parts. Fortunately, the cooperation went well and resulted in a working software system.

Overall, the project was a chance for me to improve my skills, both technical and non-technical. In my working experience, I used to work in team projects. Managing the project individually has been challenging because I had to position myself in different roles, such as project manager, product owner, researcher, software architect, developer, and tester. By playing those roles, I learnt to think more critically and speak at the proper level of abstraction with the stakeholders depending of the role that I took at a time.

Bibliography

[1] B. Enkhtaivan. *DIPPA : Data integration platform for precision agriculture*, Eindhoven University of Technology, PDEng Thesis, October 2018.

[2] L.V.D. Beek, K.K.J. Burgers, M. Ghanem, A.N.G. Woortmann, M.J.B. Keizer, B. van Leeuwen, L.A. Roozen, W.J.A. van Santvoort, and H.P.A. Swinkels. Software design document. Technical report, Eindhoven University of Technology, 2019.

[3] D. Goense. rmAgro, a reference model for data exchange in precision agriculture. `https://zenodo.org/record/893666#.XcQgr-hKhpg`, 2017. [Online; accessed 03-October-2019].

[4] Stan Ackermans Institute. 4TU Federation. `https://www.4tu.nl/sai/en/`. [Online; accessed 03-October-2019].

[5] Project doelstelling. `https://www.pcvpl.nl/nl/project-doelstelling`. [Online; accessed 03-October-2019].

[6] Earth Observing System. Precision Agriculture: from concept to practice. `https://eos.com/blog/precision-agriculture-from-concept-to-practice/`, 2019. [Online; accessed 03-October-2019].

[7] CloverDX. What happened to CloverETL Community Edition? `https://www.cloverdx.com/what-happened-to-cloveretl-community-edition`, 2019. [Online; accessed 03-October-2019].

[8] Python 2.7 Release Schedule. `https://legacy.python.org/dev/peps/pep-0373/`, 2008. [Online; accessed 03-October-2019].

[9] M. Rich. Time to shed Python 2. `https://www.ncsc.gov.uk/blog-post/time-to-shed-python-2`, 2019. [Online; accessed 29-October-2019].

[10] D. Clegg. *Case Method Fast-Track: A RAD Approach*. Addison-Wesley, 1994.

[11] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.

[12] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara. From monolithic to microservices: An experience report from the banking domain. *IEEE Software*, 35(3):50–55, May 2018.

[13] I. Miri. Microservices vs SOA. `https://dzone.com/articles/microservices-vs-soa-2`, 2016. [Online; accessed 03-October-2019].

[14] Z. Xiao, I. Wijegunaratne, and X. Qiang. Reflections on soa and microservices. In *2016 4th International Conference on Enterprise Systems (ES)*, pages 60–67, Nov 2016.

[15] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference model for service oriented architecture 1.0. *OASIS standard*, 12(S 18), 2006.

[16] Chris Richardson. *Microservices patterns*. Manning Publications Shelter Island, 2018.

[17] J. Lumetta. Monolith vs Microservices: Which is the best option for you? `https://www.webdesignerdepot.com/2018/05/monolith-vs-microservices-which-is-the-best-option-for-you/`, 2018. [Online; accessed 29-October-2019].

[18] Takanori Ueda, Takuya Nakaike, and Moriyoshi Ohara. Workload characterization for microservices. In *2016 IEEE international symposium on workload characterization (IISWC)*, pages 1–10. IEEE, 2016.

[19] N. Singh and K. Tyagi. Important factors for estimating reliability of SOA. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 381–386, March 2015.

[20] Tom Laszewski, Kamal Arora, Erik Farr, and Piyum Zonooz. *Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud*. Packt Publishing Ltd, 2018.

[21] Adalberto R Sampaio, Julia Rubin, Ivan Beschastnikh, and Nelson S Rosa. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications*, 10(1):4, 2019.

[22] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[23] Banco Bilbao Vizcaya Argentaria SA. API Gateways: Kong vs. Tyk). `https://www.marketscreener.com/BANCO-BILBAO-VIZCAYA-ARGE-69719/news/API-Gateways-Kong-vs-Tyk-25590337/`, 2017. [Online; accessed 03-October-2019].

[24] A. Lazar. Python web development: Django vs Flask in 2018. `https://hub.packtpub.com/python-web-development-django-vs-flask-2018/`, 2018. [Online; accessed 03-October-2019].

[25] A. Ashwini. Should you use NoSQL or SQL DB or both? `https://medium.com/swlh/should-you-use-nosql-or-sql-db-or-both-349cb26c9add`, 2017. [Online; accessed 03-October-2019].

[26] 2ndQuadrant. PostgreSQL vs MySQL. `https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/`. [Online; accessed 03-October-2019].

[27] Umesh Ram Sharma. *Practical Microservices*. Packt Publishing Ltd, 2017.

[28] T. Yarygina and A. H. Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20, March 2018.

[29] Kennedy A Torkura, Muhammad IH Sukmana, and Christoph Meinel. Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 171–180. ACM, 2017.

[30] X. Xie, P. Wang, and Q. Wang. The performance analysis of docker and rkt based on kubernetes. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 2137–2141, July 2017.

[31] W. Takase. A comparison of performance between kvm and docker instances in openstack. Technical report, High Energy Accelerator Research Organization, 2015.

[32] L.V.D. Beek, K.K.J. Burgers, M. Ghanem, A.N.G. Woortmann, M.J.B. Keizer, B. van Leeuwen, L.A. Roozen, W.J.A. van Santvoort, and H.P.A. Swinkels. Acceptance test plan. Technical report, Eindhoven University of Technology, 2019.

A Data Model Analysis

A.1 DIPPA Prototype

DIPPA prototype has data models for structuring farm operational data, sensor data, and farm management data. The farm operational data model shows relations among farm information, its farm fields, and operations/actions executed on each field [1]. The sensor data model structures all observation logs (manual and automated sensing data) from farm fields. The farm management data model shows the amount of crop yield in the warehouse and tractors' tracks on farm fields.

Looking at the project requirements in Chapter 5, not all data tables in the data models of DIPPA are necessary to satisfy these project goals. Several other tables also need some modification as well. The data to be stored and structured in the project are manual and automated sensing data. Thus, a farm management data model is also out of scope of this project.

Figure A.1 shows the farm operational data model of DIPPA. The shaded tables shown in the figure are not used for this project. Based on discussion with the stakeholders, soil nutrient data is generated from soil scanning sensor and it is categorized as automated sensing data. A harvesting table stores automated sensing data from sensors on tractors. Soil nutrient and harvesting tables are no longer needed because there is another table to store sensing data generally.

The sensor data model is reused in this project to store manual and automated sensing data from farm fields as shown in Figure A.2. Some changes are needed for this data model to provide more context or metadata of sensing data.

A.2 rmAgro Reference Model

rmAgro is a standard data model as a reference for precision agriculture domain [3]. This reference model was designed by Goense from Wageningen University and Research. Reference data models in rmAgro for farm data and sensor can be seen in Figures A.3 and A.4.

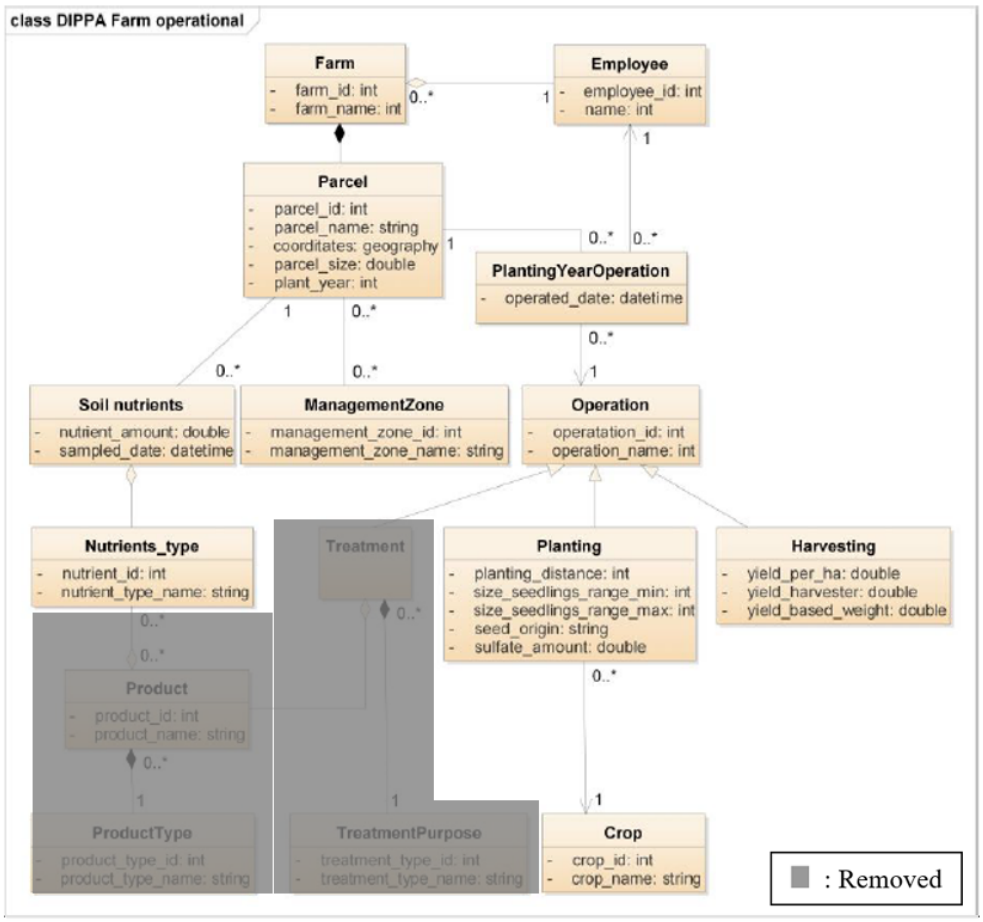


Figure A.1: Farm operational data model in DIPPA [1]

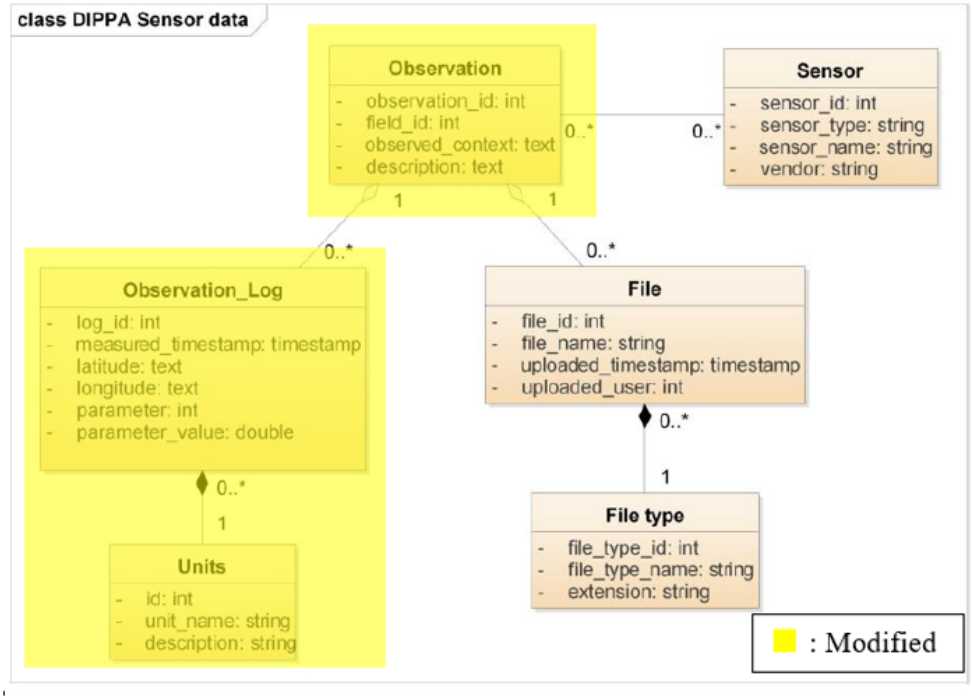


Figure A.2: Sensor data model in DIPPA for storing sensing data [1]

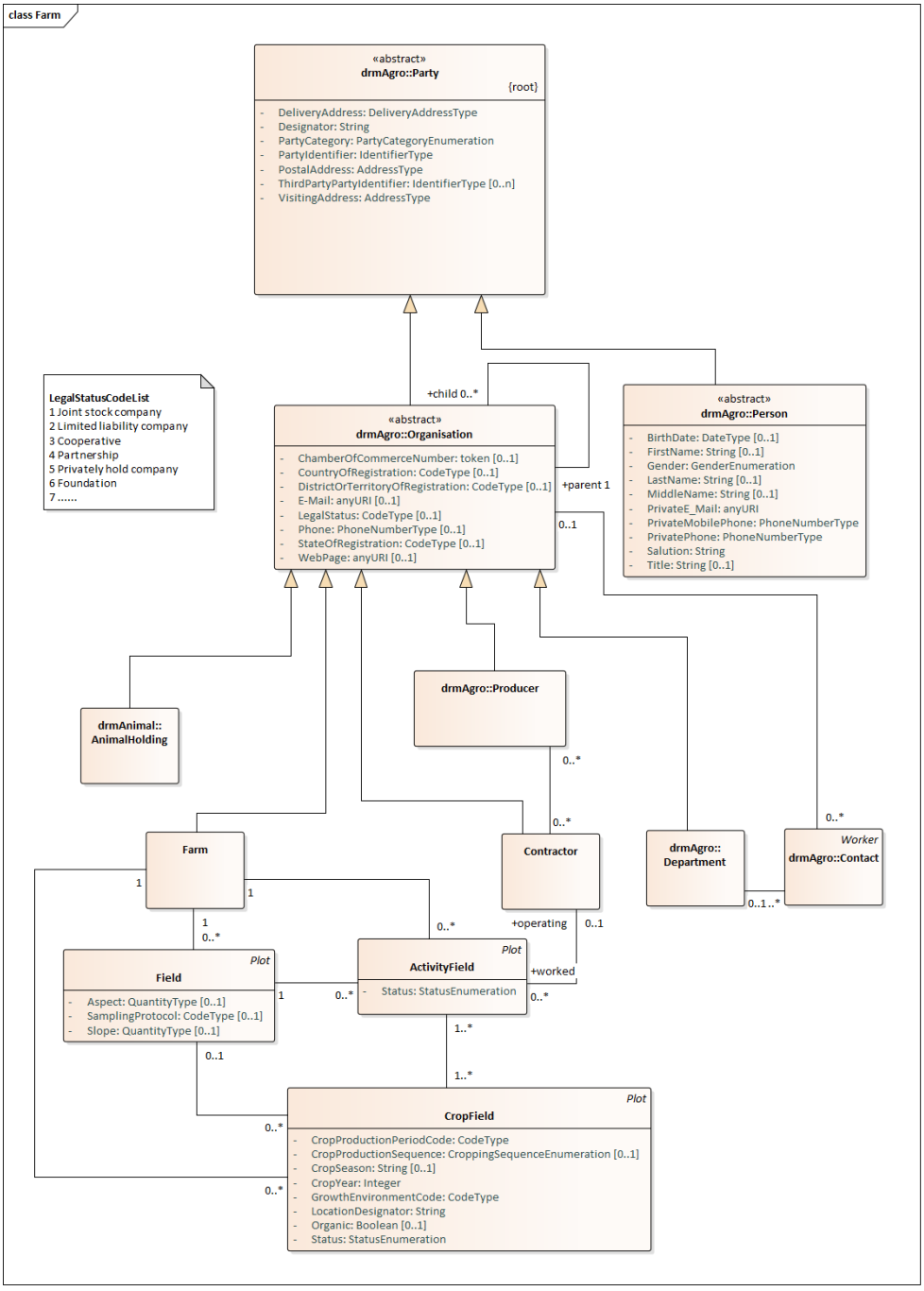


Figure A.3: Data model for farm data in rmAgro [3]

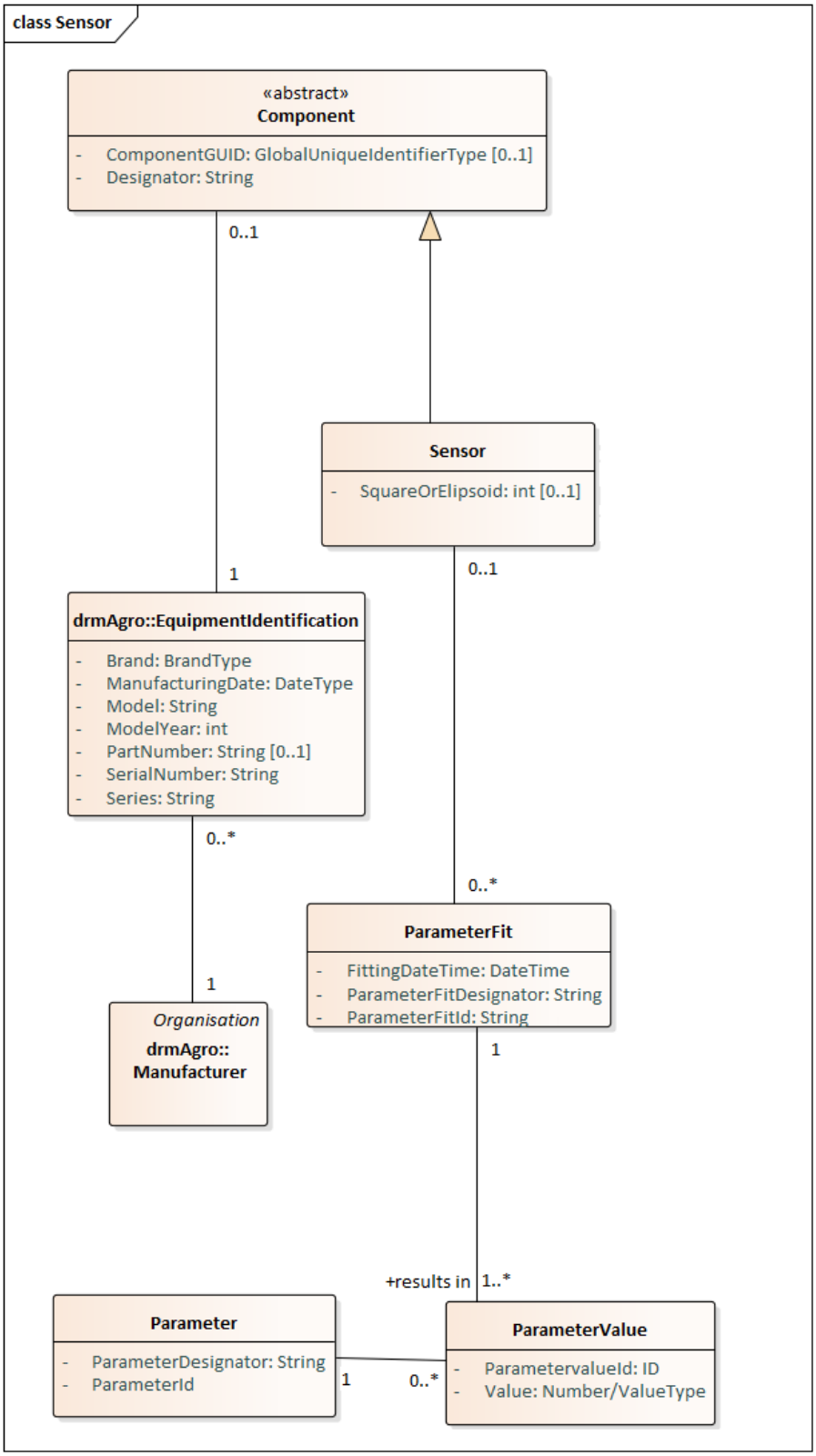


Figure A.4: Data model for sensor data in rmAgro [3]

B Data Map Structure

A data map is used to define metadata of the data contained in an input file. Variables in a data map file that need to be defined by system users are:

- *name*: a name of data map file.
- *description*: a description of a data map file.
- *has_header*: a boolean parameter that should be "True" if the input files have column names as header, otherwise "False".
- *has_coordinate*: a boolean parameter that should be "True" if the input files contain latitude and longitude information, otherwise "False".
- *has_date*: a boolean parameter that should be "True" if input files contain observation date, otherwise "False".
- *has_time*: a boolean parameter that should be "True" if an input file contain observation time, otherwise "False".
- *model_id*: an equipment model ID.
- *maps*: metadata of data columns in an input file. In *maps*, has certain structure as follows:
  1. *column*
     If *has_header* variable is True, this variable should contain a column name in input file. Otherwise, this variable has an integer value starting from 1 to define which column position in input file that this metadata describes.
  2. *observation*
     (a) *type*
     Observation type has several value options:
     i. “datetime” for the column containing date or time data
     ii. “coordinate” for the column containing coordinate data
     iii. “environment” for the column containing environment observation data
     iv. “crop” for the column containing data about the crop to monitor its growth
     v. “harvest” for the column containing data on crop yield / harvesting operation
     (b) *context*
     This variable describes an object or a context that is observed.

(c) *parameter*
This variable describes a parameter of the object or the context that is observed.

(d) *description*
An additional description about the observation.

(e) *unit*
A measurement unit of the observation.

(f) *condition*
An additional observation condition.

Some default settings for observation types "datetime" and "coordinate" columns are described in Table B.1.

Table B.1. Default settings for "datetime" and "coordinate" column types

Type	Context	Parameter	Example Values
datetime	datetime	day-month-year hh:mm:ss	02-01-2017 10:45 PM
		month-day-year hh:mm:ss	06-02-2017 22:45:00
		year-month-day hh:mm:ss	2017-12-02 22:45
	date	day-month-year	02-01-2017 02/01/2017 02-January-2017 02/January/2017 02-Jan-2017 02/Jan/2017
		month-day-year	06-02-2017 06/02/2017 June-02-2017 June/02/2017 Jun-02-2017 Jun/02/2017
		year-month-day	2017-12-02 2017/12/02 2017-Dec-02 2017/Dec/02 2017-December-02 2017/December/02
	time	null	22:45:00
	year	null	2019
	month	null	03
	day	null	15
	hour	null	10
	minute	null	45
	second	null	30
coordinate	longitude	null	110.19982379
	latitude	null	-7.629398

An example of a CSV file containing sensing data from a sensor with a header as column names can be seen in Figure B.1. An example of a data map for this file to store only values in columns “Temperature 1.50m” and “Moisture -10cm” in the system can be seen in Figure B.2.

```
Date and time;Wind speed (10 min avg);Rain;Solar;Temperature 1.50m;Humidity;Moisture -10cm;Moisture
-20cm;Moisture -30cm;Temperature -10cm;Barometer;Solar Panel Voltage;Battery Voltage;Dew point
Temperature;WindChill
2019-08-28 00:04;0;0;0;17;74.9;21;21;20;24.4;1011.9;0;13;12.5;17
2019-08-28 00:09;0;0;0;17.1;75.2;21;21;20;24.4;1011.9;0;13;12.7;17.1
2019-08-28 00:14;0;0;0;17;75.5;21;21;20;24.3;1011.9;0;13;12.6;17
2019-08-28 00:19;0;0;0;17;75.6;21;21;20;24.3;1011.9;0;13;12.7;17
2019-08-28 00:24;0;0;0;17;75.8;21;21;20;24.3;1011.9;0;13;12.7;17
```

Figure B.1: An example of a CSV file containing sensing data

```
{
    "name":"Sensor1 Data Map",
    "description":"Data map for Sensor1 from X company",
    "has_header":True,          //True if input file has column names as header
    "has_coordinate": False,    //True if input file contains latitude and longitude information
    "has_date": True,           //True if input file contains date information
    "has_time": True,           //True if input file contains time information
    "model_id":1,
    "maps":[
        {
            "column": "Date and time",
            "observation": {
                "type": "datetime",
                "context": "datetime",
                "parameter": "year-month-day hh:mm:ss",
                "description": "Date and time of observation",
                "unit": null,
                "condition": null
            }
        },
        {
            "column":"Temperature 1.50m",
            "observation":{
                "type":"environment",
                "context":"weather",
                "parameter":"temperature",
                "description":"Weather temperature in height 1.5m above soil",
                "unit":"Celsius",
                "conditions":[
                    {
                        "parameter":"height",
                        "value":1.5,
                        "unit":"m"
                    }
                ]
            }
        },
        {
            "column": "Moisture -10cm",
            "observation":{
                "type":"environment",
                "context":"soil",
                "parameter":"moisture",
                "description":"Soil moisture 10 cm underground",
                "unit":"mg/L",
                "conditions":[
                    {
                        "parameter":"height",
                        "value":-10,
                        "unit":"cm"
                    }
                ]
            }
        }
    ]
}
```

Figure B.2: An example of a data map to store temperature and moisture values

C Front End Screenshots

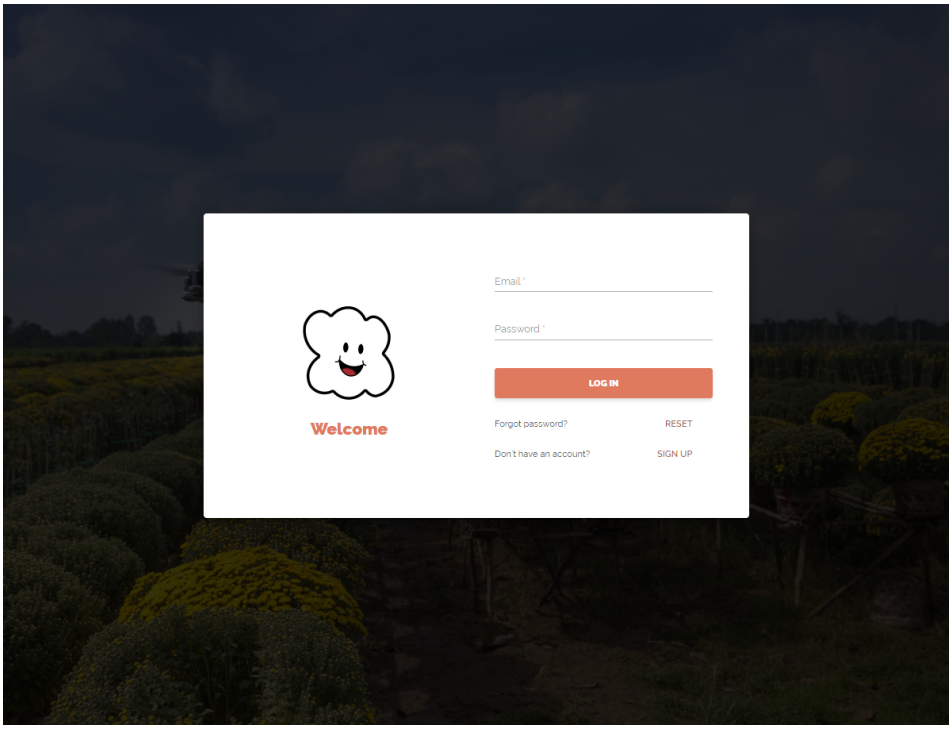


Figure C.1: Login view

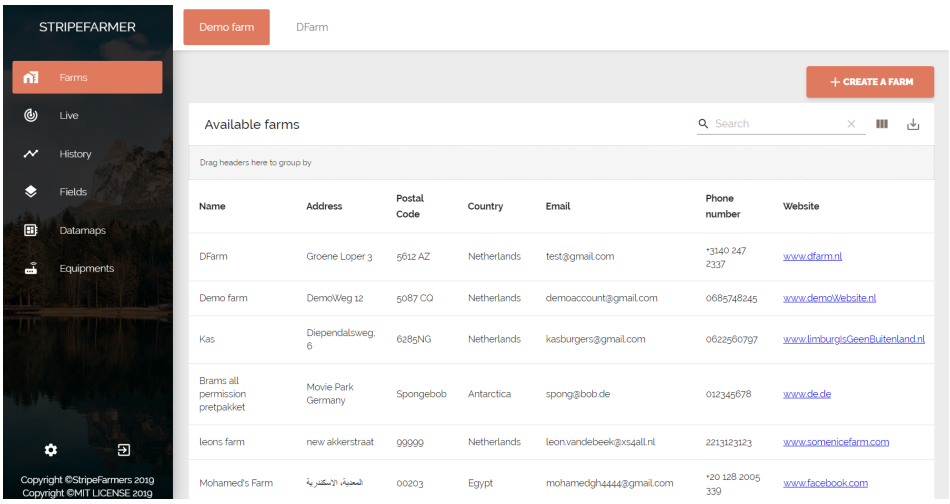


Figure C.2: Farm search and view of the dashboard

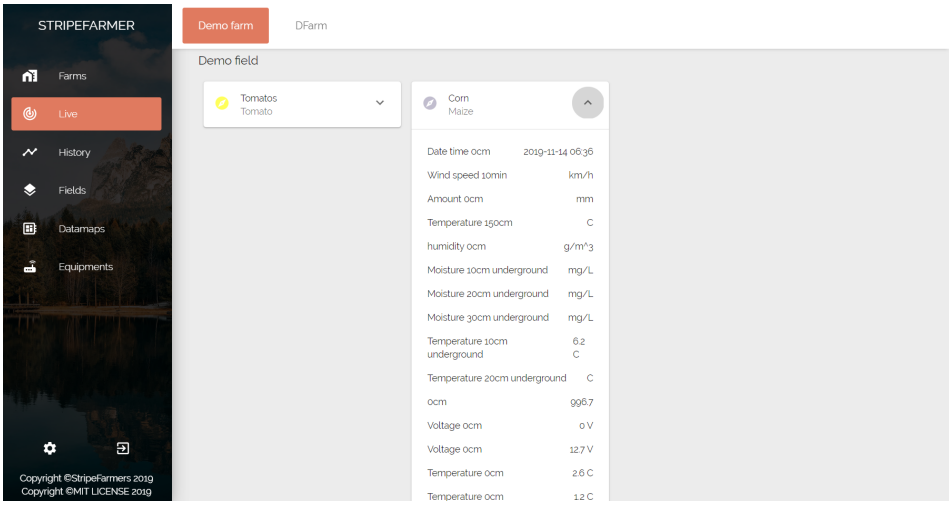


Figure C.3: Live view of sensor reading in the dashboard

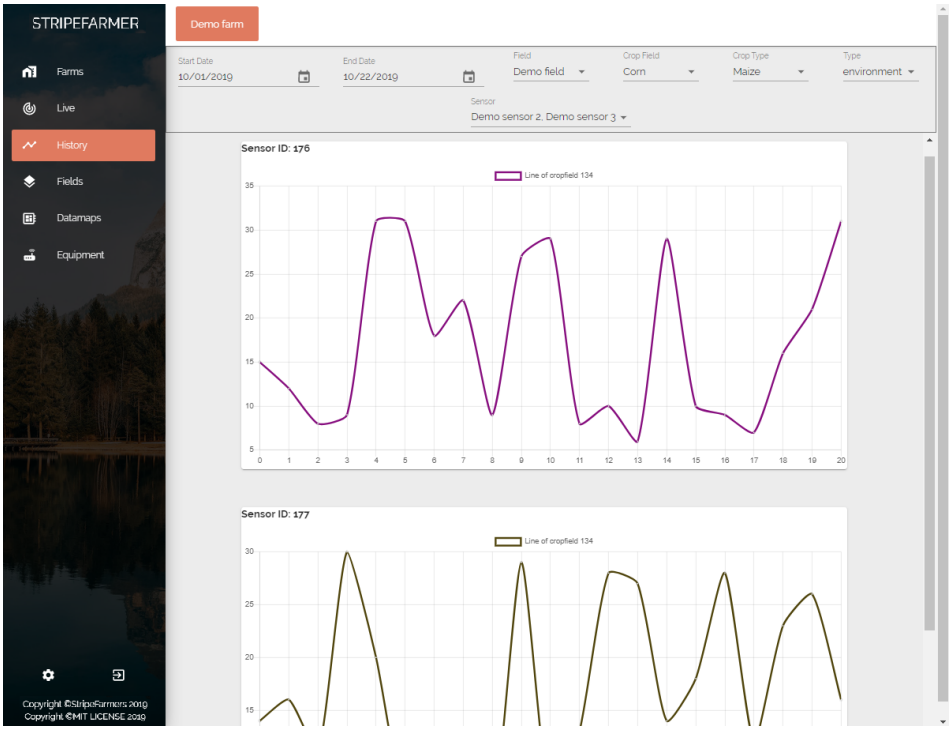


Figure C.4: Sensing data view of the dashboard

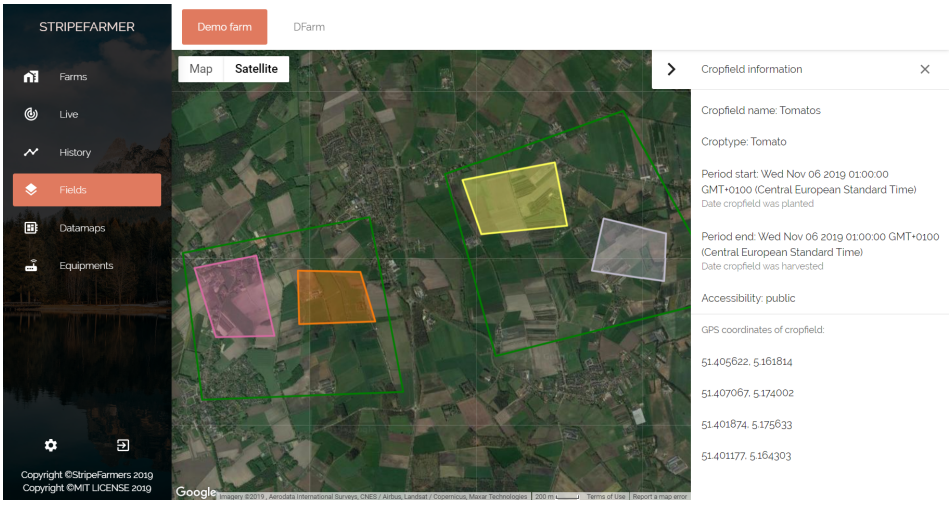


Figure C.5: Farm field and crop field views of the dashboard

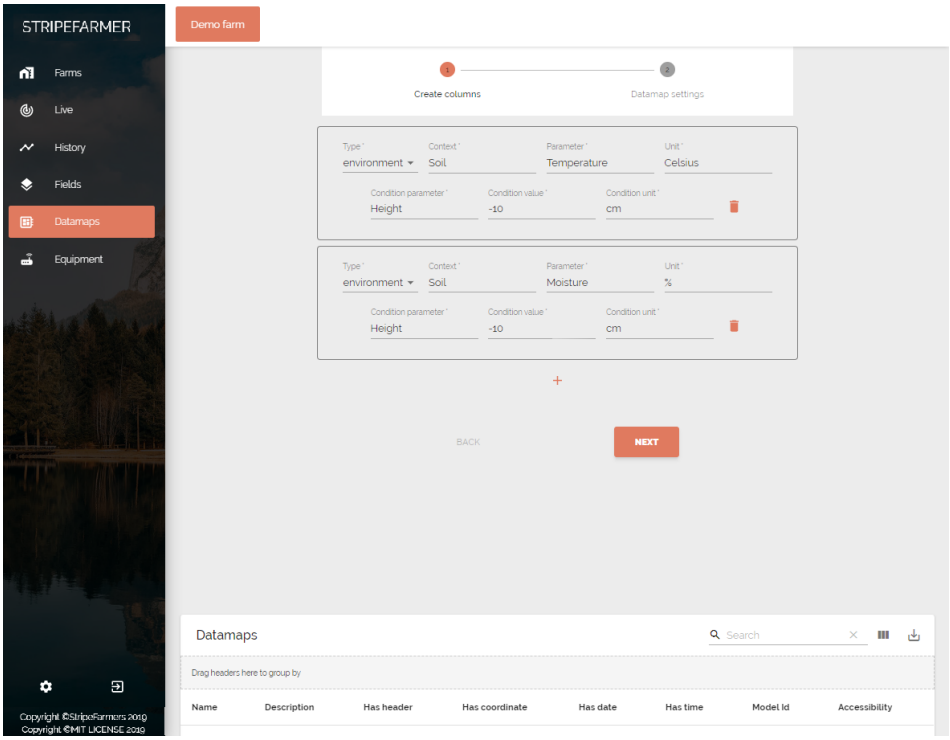


Figure C.6: Data map configuration view of the dashboard

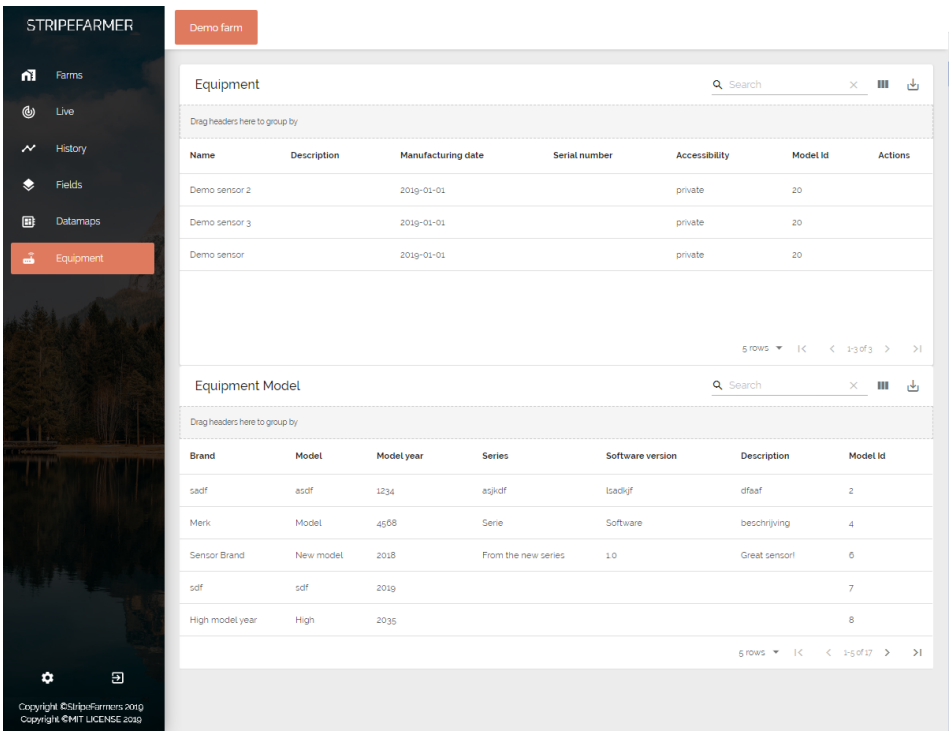


Figure C.7: Farm equipment view of the dashboard

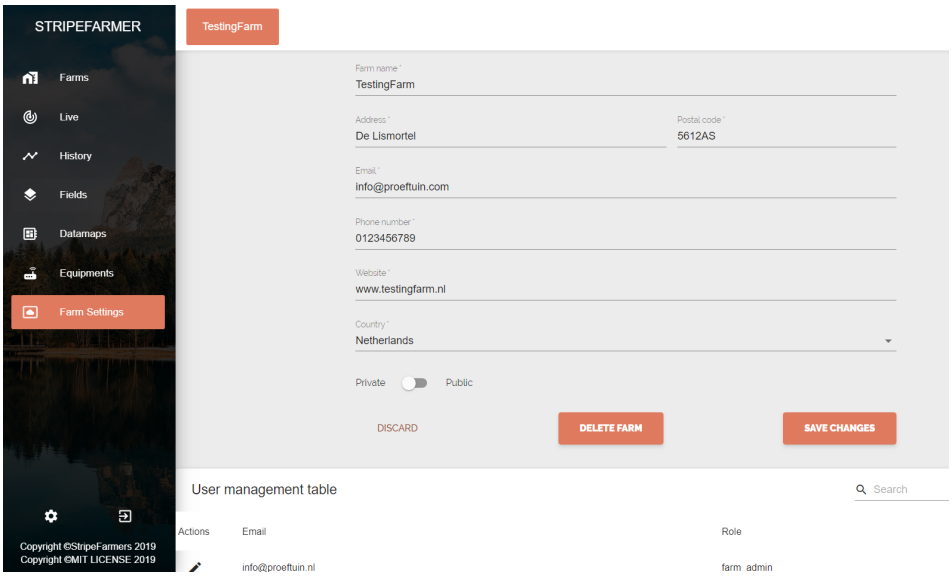


Figure C.8: Farm settings view of the dashboard

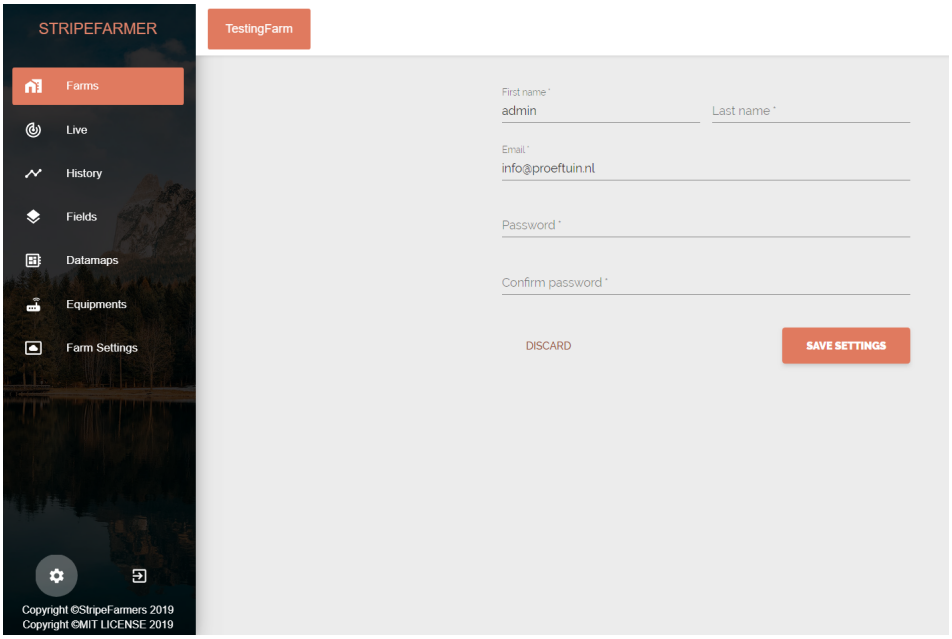


Figure C.9: Personal settings view of the dashboard

About the Author

Dimas Satria received two Bachelor degrees of engineering in electrical engineering from Diponegoro University, Indonesia, and Fontys University of Applied Sciences, The Netherlands in 2011. After his graduation, he worked for four years in The Netherlands as a software engineer in Grass Valley, Bosch Security Systems, and ASML. He received his Master's degree in Computer Science from Korea University in August 2017, South Korea. During his Master's studies, he was a reviewer for *the KSII Transactions on Internet and Information Systems journal* and *the IEEE Wireless Communications Magazine*. His Master's thesis was "Data Packet with Priority Arrangement in LTE – Licensed Assisted Access." He is interested in Embedded Systems, Internet of Things, and AgTech.