Project Report - SEP

Group 2

Group members:

Adrian - Gabriel Vaitis (304486)

Adriana Grecea (304149)

Agostina Mezzabotta (305170)

George Andronache Eduard (304460)

Georgiana Ion (304216)

Ioan Vlad Dirlea (304182)

Khaled Hammoun (275241)

Luis Fernandez Ponton (304272)

Morten Frederik Hansen (304668)

Stefania Tomuta (304173)

Tabita Roxana Varlan (304181)

Tomas Ondrejka (304150)

Supervisors:

Ib Havn

Kasper Knop Rasmussen

Knud Erik Rasmussen

74225 characters

Software Engineering

4th Semester

02/06/2022

VIA University College

Table of Contents

Abstract

The administrators of Farmerama, a small pig farm in Horsens, agreed on investing into a system which monitors the living conditions of the animals in terms of noise, temperature, humidity and CO_2. In this way, the live data and evaluations provide better supervision over the natural conditions within the barn with the goal of enhancing the pigs' health. In regard to this future system need, a series of discussions and agreements were conducted by the Product Owner, the stakeholders and the farm's administrators. This process resulted in 50 functional and 6 non-functional requirements used by a set of actors: employees, administrators and guests. Therefore, the main functionalities of the system are: authentication, managing the environment related to the threshold changes which can trigger the movements of the windows inside the barn, managing accounts, viewing historical and latest data regarding the measurements, administering areas at choice and managing personal profile.

The system consists of a client-server application with the client being the presentation tier, which takes care of the user experience within the system through an Android application. The server, responsible for handling the business logic and persisting data, consists of a web service implementation, a gateway application and a data warehouse. The IoT devices, that are running using FreeRTOS, are responsible for measuring data and sending it through the LoRaWAN network which the server then receives. The communication between the IoT devices and the server is bidirectional as the IoT devices also receive data in order to modify measurement thresholds and these instructions are sent all the way from the Android application through the server.

The quality of the system was concluded after a certain amount of tests were conducted. White Box and Black Box testing was performed with an emphasis on the latter in order to test all the use cases. According to the results, the system delivered fulfils the expectations of the Product Owner's and the Farmerama administration.

1. Introduction

Farmerama is a small farm right outside Horsens Kommune that deals with raising pigs in an ethical way in order to deliver the best quality meat in the local area. However, the owner has been facing issues with managing his livestock, as the pigs are often agitated and restless, with bad influences on weaning and their behaviour. The main goals of the farm's owner are to improve the health of the animals living in the farm as well as improve their welfare.

The owner of Farmerama has repeatedly expressed that there are several concerns that interfere with the pigs' wellbeing. Some of the issues relate to increased levels of CO_2 caused by lack of ventilation that can lead to an increase in temperature, which in turn leads to higher levels of humidity. Higher levels of humidity lead the hay inside the barn to mould faster, which prompts losses on the owner's side as they need to change the hay more often than desired.

Moreover, the farm is currently ventilated by manually operating the windows which influences the temperature. The employees are in charge of the ventilation of the farm which is a matter of concern as they are not properly aware of the pigs' ideal needs regarding the temperature. The current situation does not permit further checking on the noise and CO_2 level in the barn which could be problematic in regards to the pigs' welfare.

Farmerama is interested in knowing whether or not there might be a correlation between the temperature, humidity and/or CO_2 level in the pig pen and the noise level. If the farm had a way to properly monitor and gather relevant data, it would be easier for their employees to take care of the animals living at the farm. In this way, the farm can provide the best living conditions such that the quadrupeds can develop and grow in a stress-free environment which can supply better first-hand products such as fresh meat and other prime materials to be exported for other resources.

The following chapter, Analysis, contains a detailed description of the requirements proposed by the stakeholders and the Design chapter elaborates on the manner the requirements were implemented. The Implementation chapter covers specific implementation details of the system and finally, the different testing approaches used to verify the functionality of the system are explained in the Test chapter.

2. Analysis

2.1. Requirements

In the aftermath of several discussions with the stakeholders and the Product Owner, the development team came up with a list of prioritized requirements that covered the customer's desired needs. Their purpose was to provide an additional understanding of the required functionalities that were requested.

The requirements were split between functionalities targeting a total of 3 actors, the Guest, the Employee, and the Administrator. An inheritance relationship is present among the 3 actors. Therefore the Administrator, who has absolute access, can perform any action available to either the Guest or the Employee, but in turn, is the only one who is allowed to register new users within the system as well as add, edit and remove areas. On the other hand, the Guest has limited access to the system, being able to only log into an already existing account and view the lasted measured data. The Employee has access to all of the system's functionalities except for the aforementioned ones that are exclusive to the Administrator.

The SMART principles were taken into consideration whilst devising the functional requirements. Therefore, the requirements are:

- Specific, as each requirement focuses on distinct and individual topics
- Measurable, due to being disparate and diverse
- Achievable, due to being realistic
- Reachable, by being specific and clear
- Time-bound, due to the project having a deadline

Furthermore, the requirements were also designed with the FURPS+ principles in mind. While the Functionality principle refers to the previous paragraph and the system's functional requirements, the non-functional requirements adhere to the remaining principles. As such, the non-functional requirements are:

- Usable, as per the GUI specifications of developing an Android application
- Reliable, due to requesting the data be stored in a database
- Performance related, by allowing for multiple user interactions through a web API

- Supportable, due to the clear architecture prerequisite

Table 1 is an extract of the complete list of prioritised functional requirements. For the full list of all 50 functional requirements, see Appendix G (the Product Backlog) as part of the Process Report.

2.1.1. Functional requirements

1	Critical	As a guest, I want to view the latest measurement of the temperature inside the barn, so that I can monitor the values of the environment in which the pigs live
2	Critical	As a guest, I want to view the latest measurement of the CO2 levels inside the barn, so that I can decide to ventilate or not
3	Critical	As a guest, I want to view the latest measurement of the humidity inside the barn, so that the food provided will not spoil
4	Critical	As a guest, I want to view the latest measurement of decibels inside the barn, so that I can interpret the pigs' stress levels
5	Critical	As an owner, I want to add accounts to the system, so that I can ensure only my employees log in and access the system's features
6	Critical	As a guest, I want to log into my account, so that I can gain access to the system's features
7	Critical	As an employee, I want to view reports measuring the noise at the pigs' area, so that I know they have not been under stress
8	Critical	As an employee, I want to view reports measuring the temperature in the pigs' area, so that I can be sure the pigs are in good condition during extreme temperatures
9	Critical	As an employee, I want to view reports measuring the CO2 in the pigs' area, so that I make sure the pigs receive the right amount of oxygen that ensures a quality life
10	Critical	As an employee, I want to view reports measuring the humidity in the pigs' area, so that I can be sure that the infectious diseases are avoided
11	High	As an employee, I want to see the historical data regarding temperature because I am interested in data statistics
12	High	As an employee, I want to see the historical data regarding CO2 because I am interested in data statistics
13	High	As an employee, I want to see the historical data regarding sound because I am interested in data statistics
14	High	As an employee, I want to see the historical data regarding humidity because I am interested in data statistics
15	High	As an employee, I want to edit thresholds for CO2 levels, so that I can ensure the pigs' well-being
16	High	As an employee, I want to edit thresholds for temperature, so that I can ensure the pigs' well-being
17	High	As an employee, I want to edit thresholds for humidity, so that I can ensure the pigs' well-being

18	High	As an employee, I want to edit thresholds for sound, so that I can ensure the pigs' well-being
19	High	As an employee, I want to view a log of CO2 measurements that have exceeded a certain threshold, so that I can act accordingly
20	High	As an employee, I want to view a log of humidity measurements that have exceeded a certain threshold, so that I can act accordingly
21	High	As an employee, I want to view a log of sound measurements that have exceeded a certain threshold, so that I can act accordingly
22	High	As an employee, I want to view a log of temperature measurements that have exceeded a certain threshold, so that I can act accordingly
23	High	As an employee, I want the window to open when the temperature exceeds the upper threshold
24	High	As an employee, I want the window to close when the temperature exceeds the lower threshold

Table 1 - Extract of the list of functional requirements

2.1.2. Non-functional requirements

56	The system should use a client-server architecture
57	The system should persist data on the server and expose the data through a web API
58	The presentation tier should support offline access, thus providing a local database for persistence
59	The system should receive data from an IoT device approximately every 5 minutes
60	The IoT device should still operate the window even without a radio signal
61	The system should store historical data in a data warehouse

Table 2 - Non-functional requirements

2.2. Use case diagram

After the functional requirements were analysed, they were divided into seven use cases based on shared or similar goals. Besides, five actors were defined. Figure 1 shows the use case diagram defining and describing the relationship between use cases and actors.

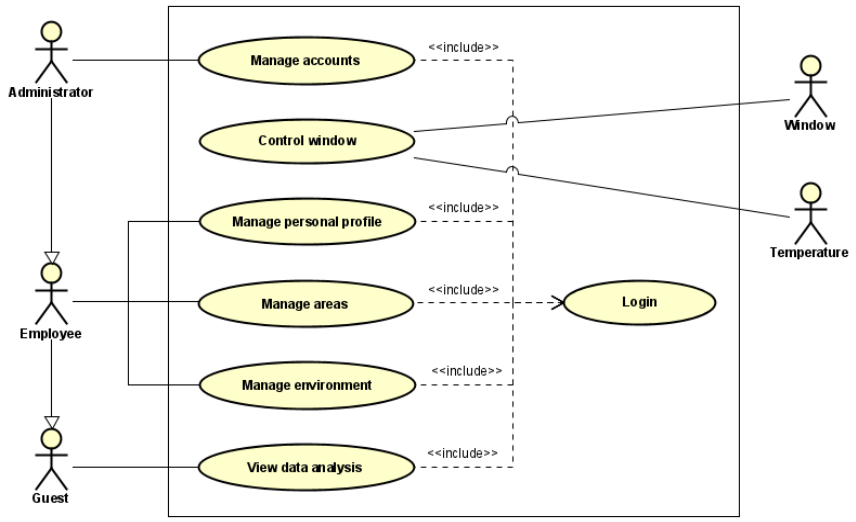


Figure 1 - Use case diagram

Each use case is linked to the functional requirements outlined before. Table 3 summarises these relationships.

Use case	Covered requirements
Manage accounts	5, 31, 32, 33
Manage personal profile	33, 34, 44, 54, 55
Manage areas	33, 35, 36, 41, 42, 43
Manage environment	15, 16, 17, 18, 19, 20, 21, 26, 33, 45, 46, 47, 48, 49, 50, 51, 52, 53
View data analysis	1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 33
Login	6
Control window	22, 23

Table 3 - Use cases and their covered requirements

2.3. Use cases

Use case descriptions were used to define each use case and their flow. Below is a summary of all the use cases in the system as well as the full use case description of the main use case (see next page). A full list of all use case descriptions can be found in Appendix C.

2.3.1. Use case summaries

a. **Manage accounts**: The administrator can add and remove accounts from the system
b. **Manage personal profile**: The employee can view and edit their own personal account information, choose to enable notifications and remove local storage
c. **Manage areas**: The administrator can add, view, edit and remove areas
d. **Manage environment**: The actor can view and edit an area's measurement thresholds, view threshold changes based on a specific date and view measurements that exceeded set thresholds
e. **View data analysis**: The actor can view the latest measurements. The employee can view the historical measurements based on a specific date
f. **Login**: The guest can log into the system in order to gain access to the system's features
g. **Control window**: The window can open or close dependent on the measured temperature and the set thresholds

2.3.2. Use case description

<table>
<tr><td>Use case</td><td>View data analysis</td></tr>
<tr><td>Summary</td><td>The actor can view the latest measurements of an area
The employee can view the historical measurements of an area based on a specific date</td></tr>
<tr><td>Actor</td><td>Employee, Guest</td></tr>
<tr><td>Precondition</td><td>In order to view historical measurements, the actor must be logged in</td></tr>
<tr><td>Postcondition</td><td></td></tr>
<tr><td>Case scenarios</td><td>View latest measurement:
<table>
<tr><th>Actor action</th><th>System responsibility</th></tr>
<tr><td>1. The actor chooses to view the latest data</td><td>2. The system displays a collection of measurement types (temperature, humidity, CO_2, sound) as well as a collection of areas</td></tr>
<tr><td>3. The actor specifies a measurement type (temperature, humidity, CO_2, sound) as well as an area</td><td>4. The system displays the latest measurement for the given measurement type in the given area</td></tr>
</table>
View historical measurements:
<table>
<tr><th>Actor action</th><th>System responsibility</th></tr>
<tr><td>1. The employee chooses to view the historical data</td><td>2. The system displays a collection of measurement types (temperature, humidity, CO_2, sound), a collection of areas as well as a date</td></tr>
<tr><td>3. The employee specifies a measurement type (temperature, humidity, CO_2, sound), an area as well as a date</td><td>4. The system displays the historical measurements for the given measurement type in the given area on the given date [E1]</td></tr>
</table></td></tr>
<tr><td>Exception scenarios</td><td>At any time the actor can log out
1. The actor chooses to log out
2. The system reverts the changes
3. The system logs out the actor

[E1] There are no measurements matching the given measurement type, area and date
4. The system displays an error message “No data available”
5. The base sequence continues from 3</td></tr>
<tr><td>Note</td><td>The actor can terminate the process at any time

If the application has no connection to the internet, the data already stored in the local database will be used to display the previous viewed information

Covered requirements: 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 33</td></tr>
</table>

Table 4 - “View data analysis” use case description

2.4. Activity diagram

Figure 2 gives a more visual representation of the aforementioned main use case description, specifically the “View latest measurement” case scenario. A full list of activity diagrams can be found in Appendix D.

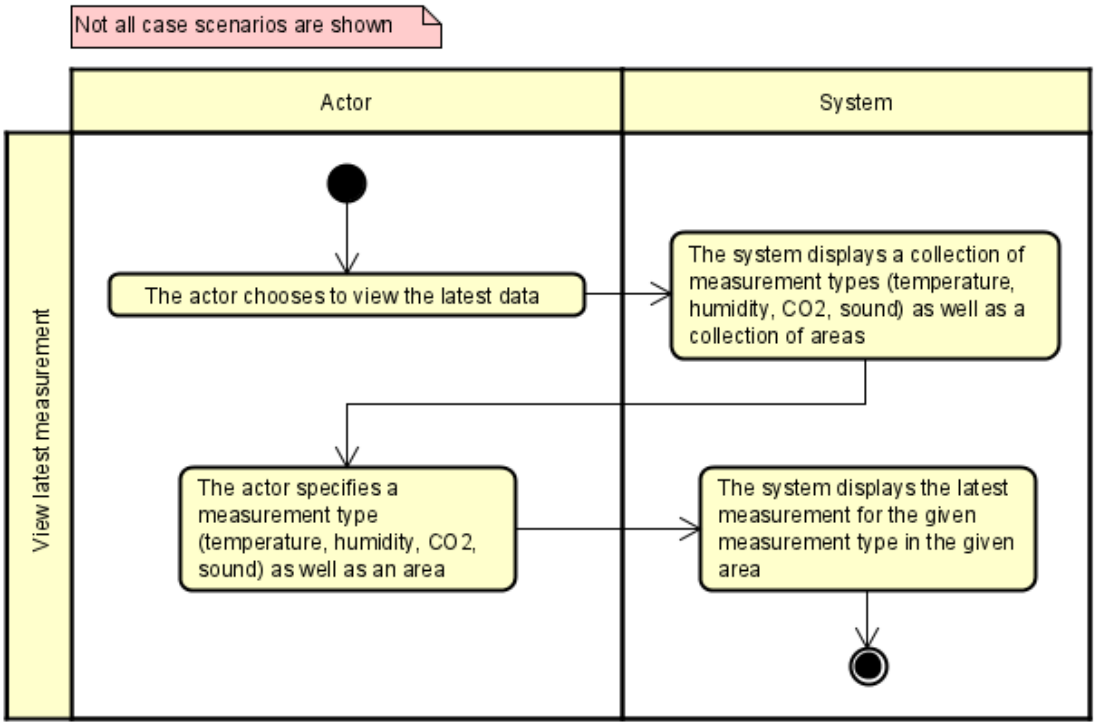


Figure 2 - “View latest measurement” case scenario in the “View data analysis” use case

2.5. Domain model

The following figure was created in order to show the relationships between the actors and the different components in the system. As shown in the diagram and mentioned previously, the Guest can view the most recent measured data and the time the data was collected while the Employee also can view historical measurements, manage devices' thresholds as well as receive notifications regarding measurements that have exceeded said thresholds. The Administrator is able to manage their employees as well as the areas within the barn. Additionally, an IoT device consists of 4 different types of sensors as well as an actuator, and each sensor belongs to an area within the barn. Each sensor also has a minimum and maximum threshold, although in the case of the sound sensor specifically, it only has a maximum threshold.

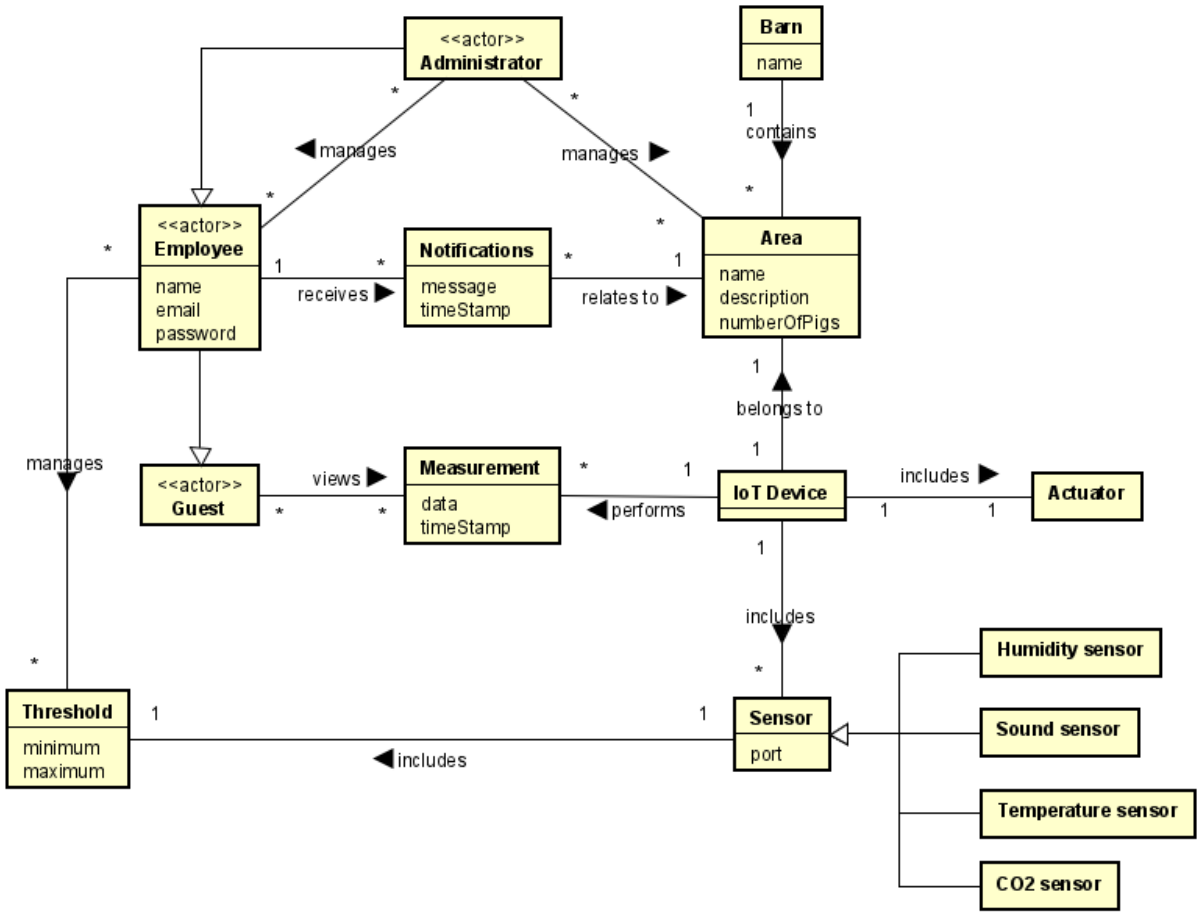


Figure 3 - Domain model

3. Design

3.1. Architecture

The architecture diagram, figure 4, shows the system structure. The system consists of a presentation tier which is an Android application, implemented in Java. This application communicates with a RESTful web service (also made in Java) by sending HTTP requests. For data storage, Microsoft SQL Server is used for both the source database as well as the data warehouse. In order to send data from a micro-controller unit, programmed in C, the LoRaWAN network is utilised and the data is received in the gateway application using a web socket (made in Java).

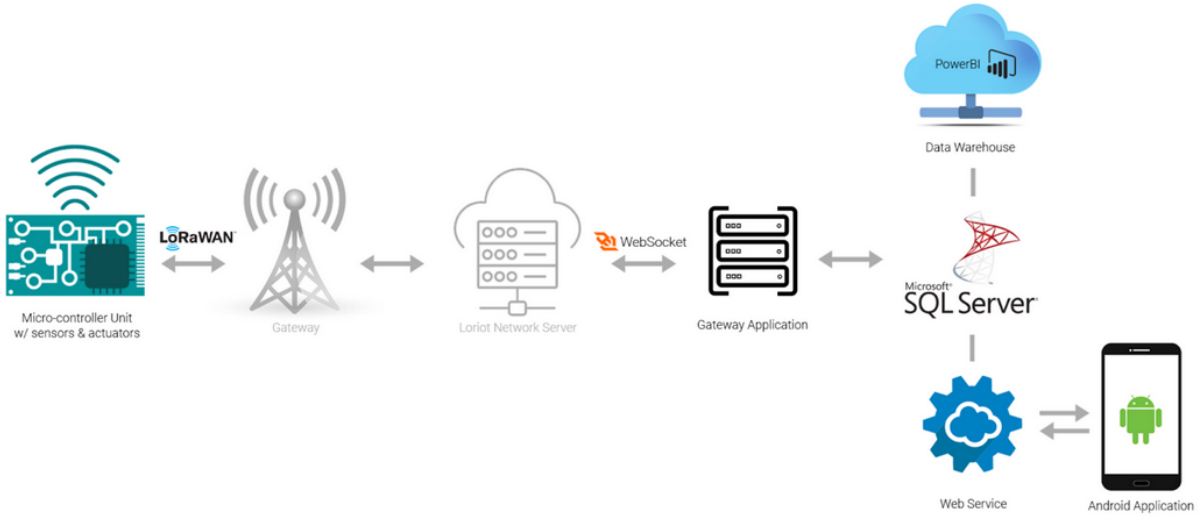


Figure 4 - Architecture diagram

3.2. Android

3.2.1. Technologies

Georgiana, Vlad, Stefania, Tabita

Room, SQLite

As one of the non-functional requirements was offline persistence, Room was chosen to have access to data when the user has no internet access. Therefore, as Android comes with a built-in SQLite database, sql queries were used to retrieve data and provide it to the user when the web service cannot be reached. On the other hand, while the user is still online and has internet access, the retrieval is done from the web service. The user can also choose to remove local storage at preference. Consequently, the local database was used as a backup in case the user loses internet connection.

Retrofit

In order to establish a type-safe REST connection to the API provided by the server, the Retrofit library was used. It was implemented to manage the process of receiving, sending and creating HTTP requests and responses. The connection to the database is also responsible for catching errors given by invalid data or forbidden actions and displaying them to the user.

Firebase Storage

For storing and sharing the users' profile pictures the application uses Firebase Storage API. For each user it was created a reference(path) containing the user id to a specific file stored in a Cloud Storage bucket. These references can then be used to upload, edit or download data. If a new image will be stored at the same path this will override the old picture. Displaying the images was done by using a third-party visual representation library, Picasso which improved the image quality for the application, made possible the scaling and resizing of the images and loading the images into image views by accessing the image via a URL.

Shared Preferences

For storing relatively small quantities of primitive data, SharedPreferences APIs was the option. After an onboarding intro was created for the first user experience, but also after the user logged in, SharedPreferences played the role in persisting the login information and also that the user already interacted with the onboarding screen. When the app was killed and relaunched, it immediately gave users access to the app's features without requiring

them to go through the onboarding and login process again, resulting in a more pleasant experience.

3.2.2. Design Patterns

MVVM

Georgiana, Vlad, Stefania, Tabita

The system design is based on the MVVM architecture where the behaviour and state is separated from the structure of the user interface. Therefore, multiple goals are achieved: code scalability, reliability and maintainability. In the figure below, the models are represented by the repositories, the view model by MeasurementsViewModel and the views are represented by LatestMeasurmentFragment. The abstract connection between the layers allows multiple changes without refactoring as the dependency flows only in one direction: the LatestMeasurementFragment has a reference to the MeasurementsViewModel and the MeasurementsViewModel has a reference to both Measurement and Area Repository classes.

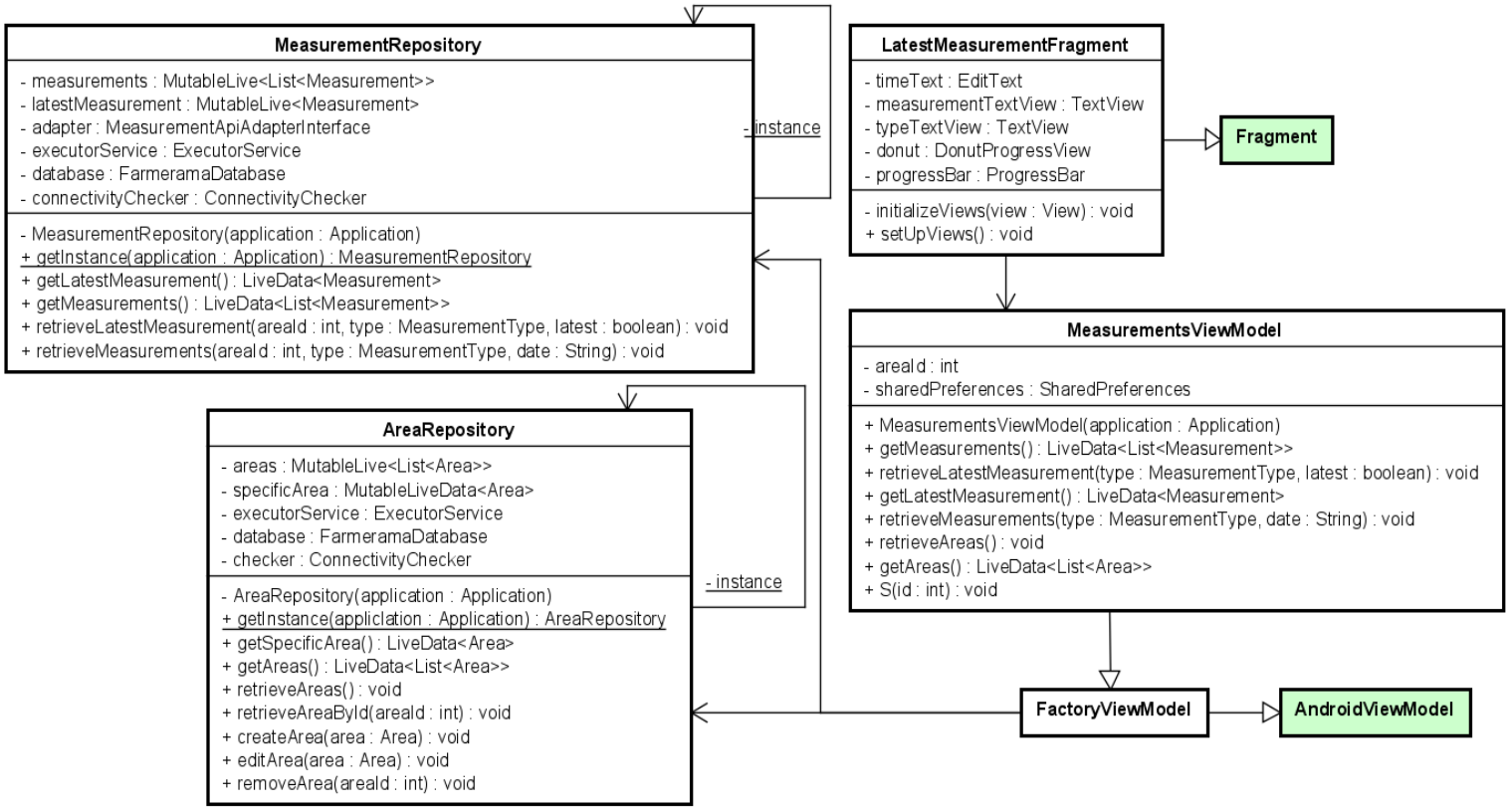


Figure 5 - MVVM Design Pattern Class Diagram

Observer Design Pattern

Georgiana, Vlad, Stefania, Tabita

The purpose is to introduce loose coupling and ensure that the data represented in the view matches the data in the model. With LiveData, the view components observe the specific data and all the active observers will be notified the moment the state of the object is changed in the model. UI Controllers (Fragments and Activities) are observing changes in the LiveData. The ThresholdModificationsFragment in this example is checking to see if any thresholds had changed, and then presenting the new modifications to the user.

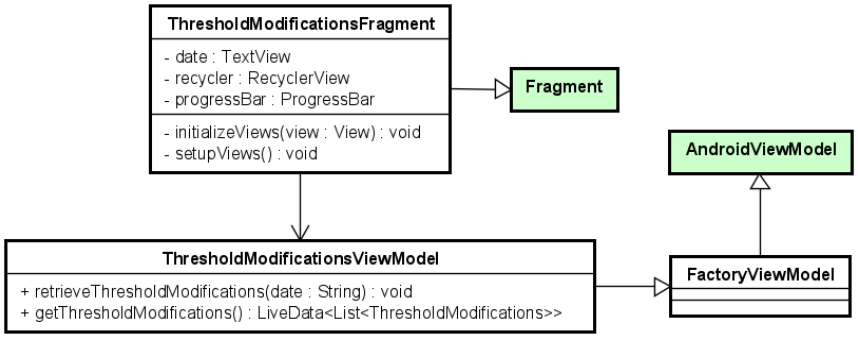


Figure 6 - Observer Design Pattern Class Diagram

Adapter Design Pattern

Vlad

Given the fact that the view for displaying measurements in Android is reused for any type of measurements, the Adapter Design Pattern was used to match with the REST controllers and to specify which type of measurement the system is retrieving. The retrieving methods in MeasurementRepository class require a MeasurementType which is used in the MeasurementApiAdapter to call the specific controller in the MeasurementApi interface.

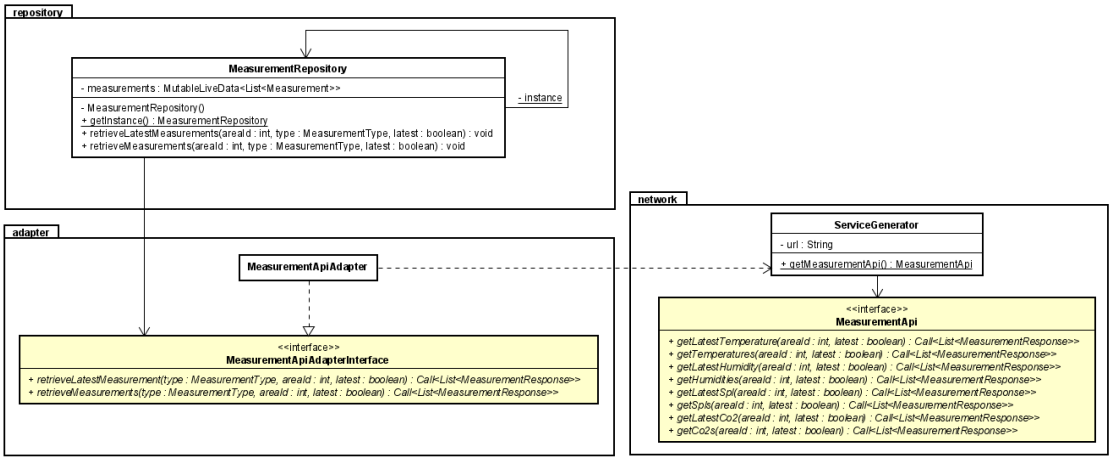


Figure 7 - Adapter Design Pattern

Repository Design Pattern

Vlad, Stefania

The reasoning behind the Repository Design Pattern is to access the source data in a logical and flexible manner. In the Repository class, the response from the server is deserialized, converted to match the application model and set in MutableLiveData objects. The repository is also responsible for retrieving data when there is no internet connection so, in case of offline mode, the data is fetched from the local database.

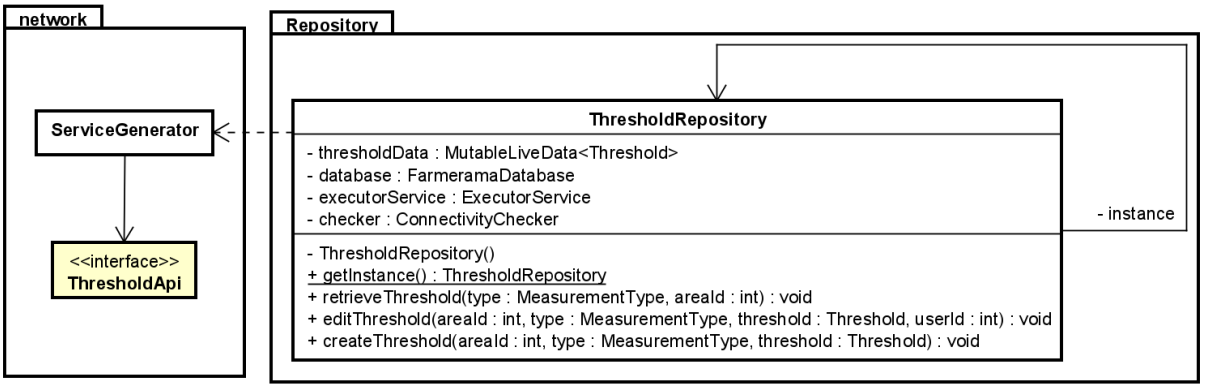


Figure 8 - ThresholdRepository Class Diagram

Singleton

Georgiana, Vlad, Stefania, Tabita

The purpose of this design pattern is to control object creation by limiting the reference of this class to be one. On the other hand, it has the advantage of eliminating unnecessary connections to the Farmerama database.

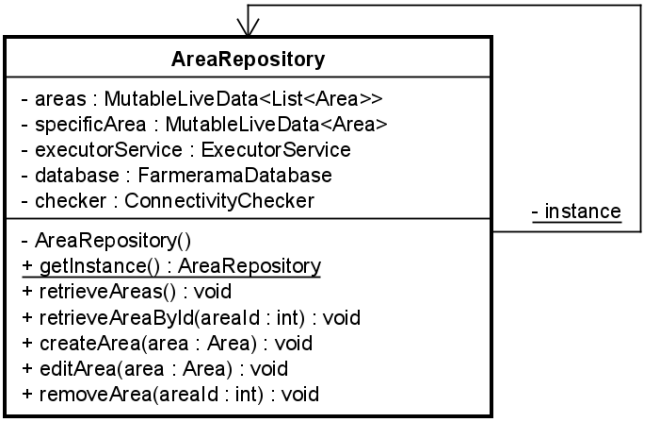


Figure 9 - Area Repository Singleton Class Diagram

DAO Design Pattern

Stefania

In order to implement a local storage of the data in case of offline mode, Dao design pattern was used in accordance with Room. Due to the fact that the implementation details of the IMeasurementDAO, IUserDAO and so on is hidden behind the interfaces, it has to be noted that it follows the separation of concerns principle. Therefore, it provides freedom of usage and more opportunities for further upgrade within the code.

For example, two different DAO methods were used in order to interact with the database. Methods like createArea, updateArea and removeAreas are considered to be convenient methods as no SQL code was used. On the other hand, getAreas, getAreaById required manual input of SQL queries in order to retrieve the desired data. FameramaDatabase, which is an abstract class and a singleton by itself has abstract instance variables of the DAO interfaces that are called through the FarmeramaDatabase in each representative repository.

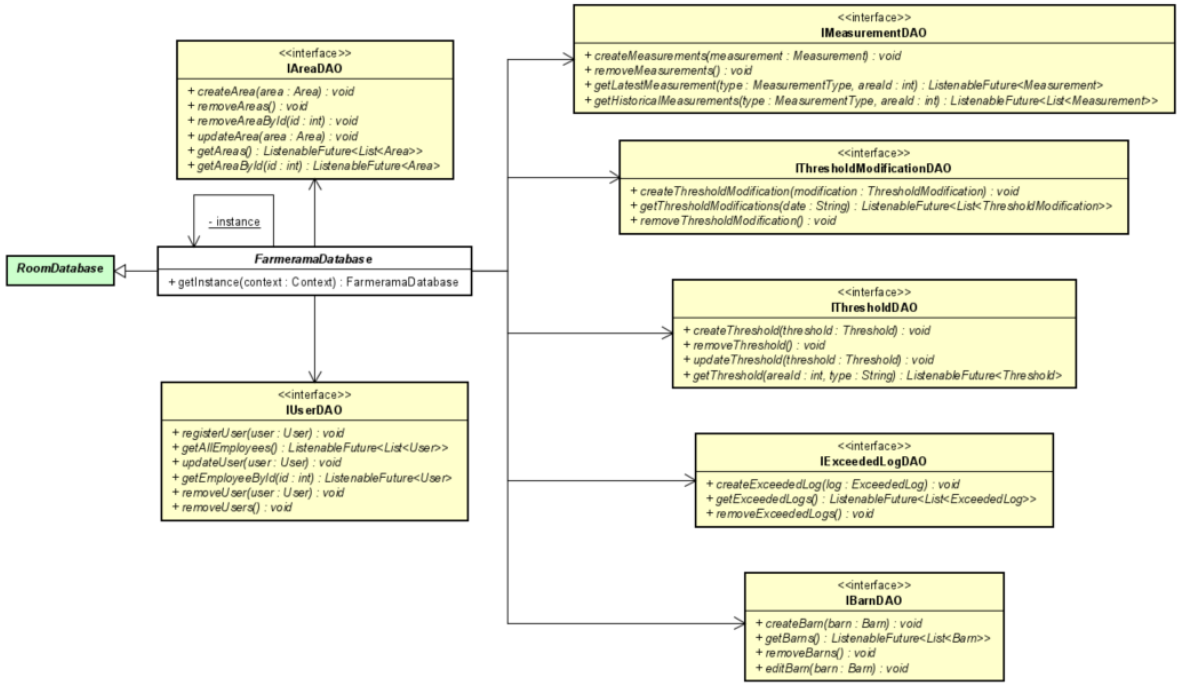


Figure 10 - DAO Design Pattern Diagram

Factory Design Pattern

Stefania

In the viewmodel package, the FactoryViewModel has the responsibility of initialising the instances of the repositories: AreaRepository, UserRepository, MeasurementRepository etc. In this way, the repositories, which are singletons by themselves, are implemented only once in the constructor of the FactoryViewModel. On the other hand, the factory extends the AndroidViewModel which contains the application context and is used to preserve one instance of each repository.

Therefore, the rest of the ViewModels that extend the factory can call its getters to obtain the references of the repositories' implementations. It has to be noted that all the existing ViewModels extend the FactoryViewModel, but due to preserving the simplicity of the class diagram, the rest are not shown.

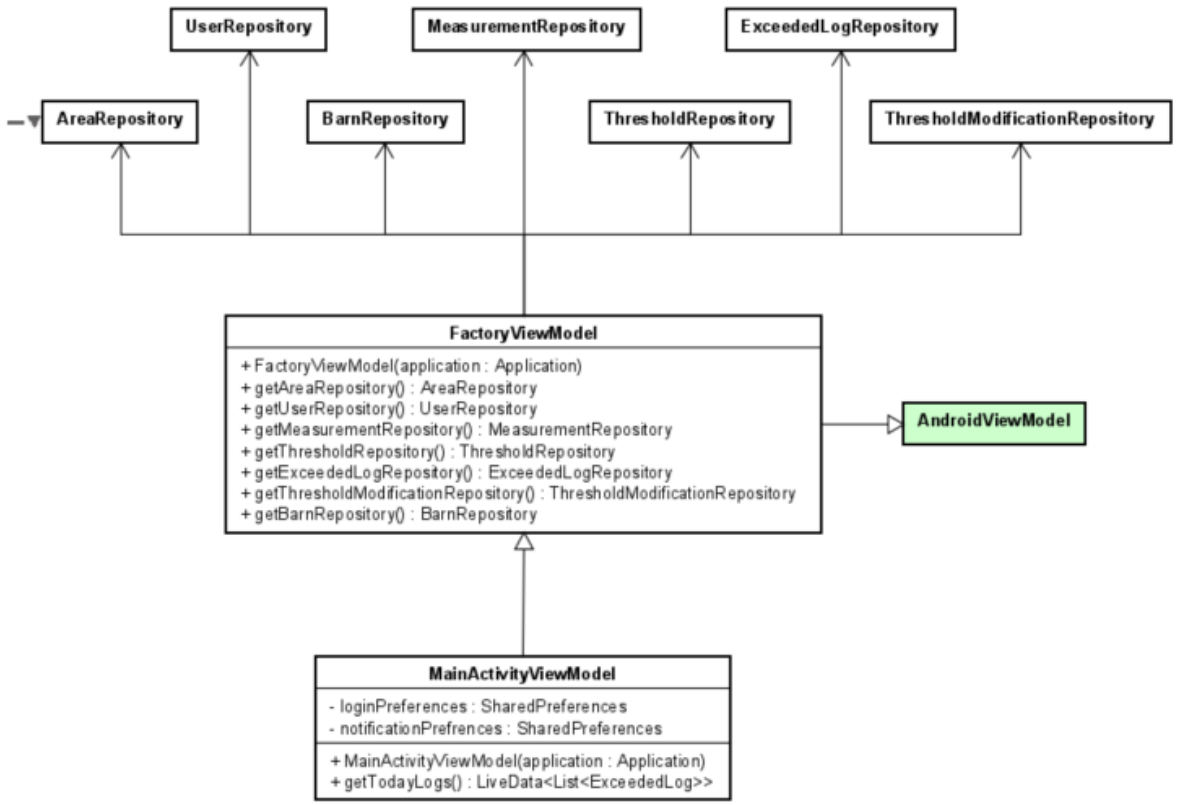


Figure 11 - Factory Design Pattern Class Diagram

3.2.3. Entity Relationship Diagram

Stefania

Keeping in mind the fact that the local data storage takes action only when there is no internet connection, the following entities were identified: Barn, Area, Measurement, Threshold, User, ExceededLog and ThresholdModification. Each entity has one primary key, with the exception of the Measurement which has a composite key: measurementId and measurementType. On the other hand, specific TypeConverters were used to map objects such as LocalDate and LocalDateTime.

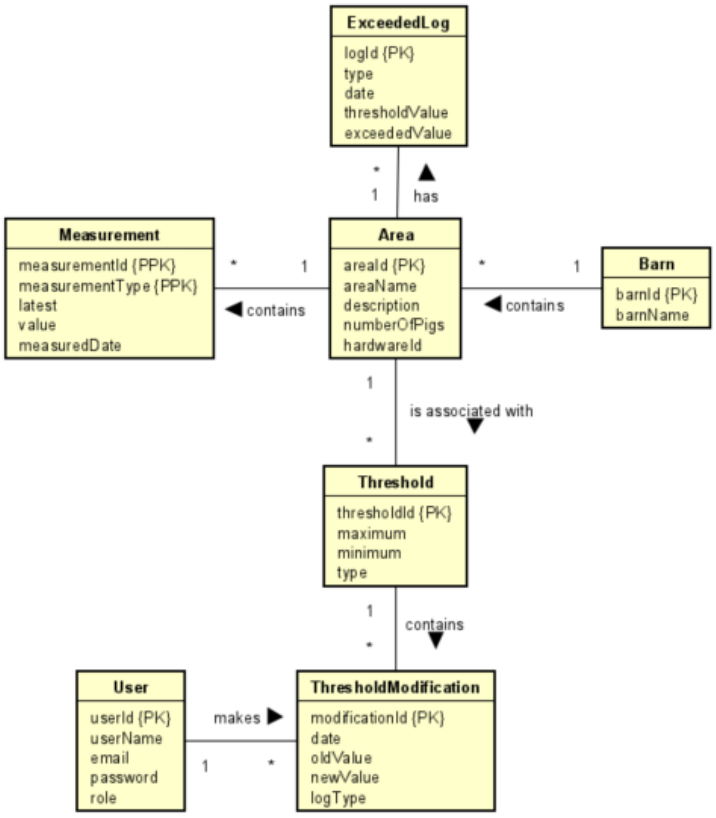


Figure 12 - Entity relationship diagram for Room database

3.2.4. Sequence diagram

Vlad, Stefania

The following diagram gives an overview of how the user interacts with the application when a new threshold is created. It is part of the "Manage environment" use case. After the user creates a new threshold, the live data which is observed in the view is going to be updated by the response received from the Web Service: either a successful message with a threshold created or a Toast with an error message.

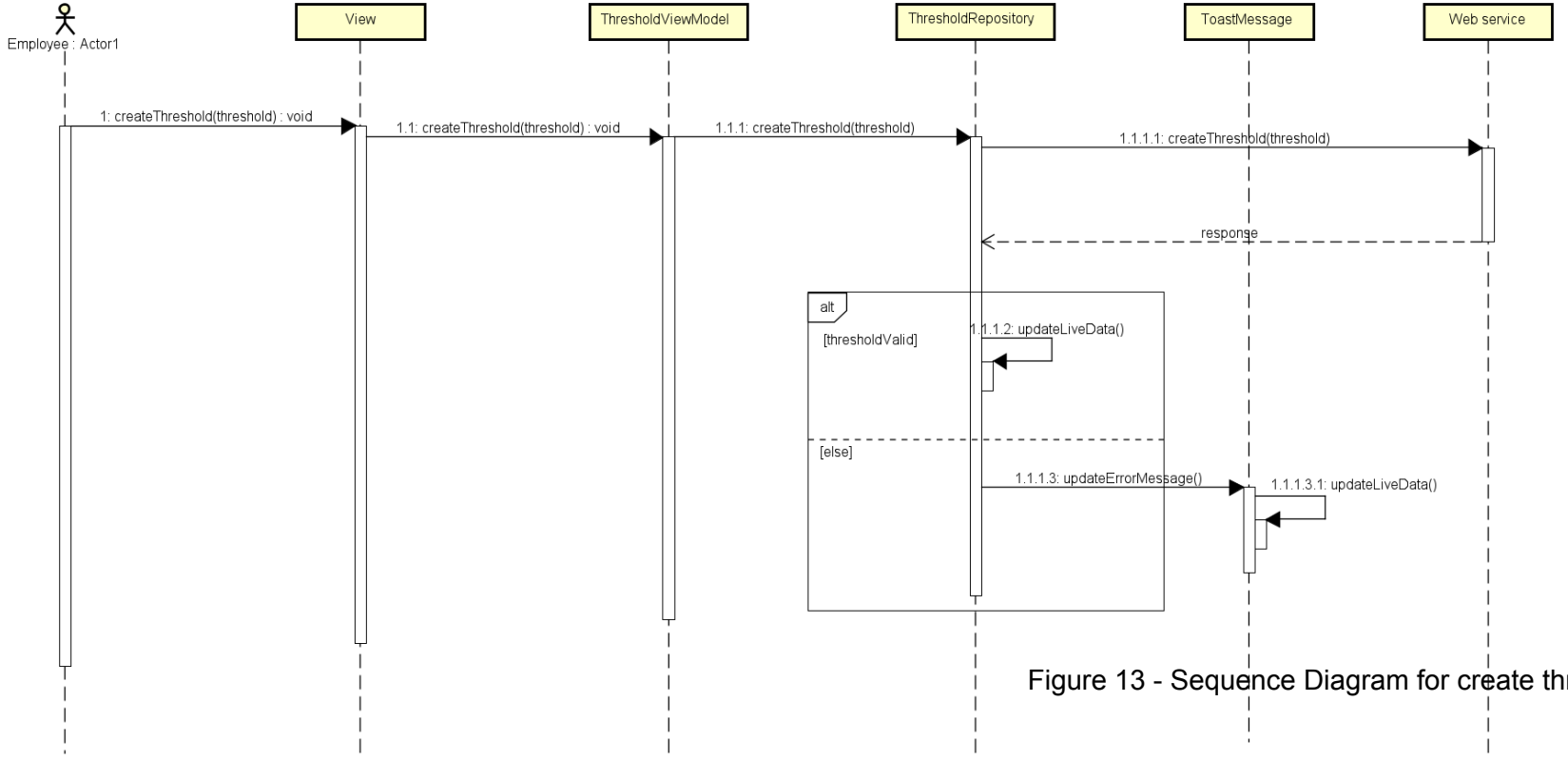


Figure 13 - Sequence Diagram for create threshold

3.2.5. Graphical User Interface Design

George

Initial Concept Design and Idea

The initial idea of the design started when the development team wanted to make a professional looking corporate application so that the vast majority of the users can use it easily. The initial mockup was designed in Figma where some details were established regarding the colours used, text style, buttons shape, etc. to match both light and dark mode. During development, several improvements have been made over the original idea mock up design based on the stakeholders' requirements.

The initial app idea was designed with a shared main page for all of the measurements (CO_2, Humidity, Temperature, Sound) but this concept was changed during development. The navigation drawer was also redesigned to sort the categories more efficiently.

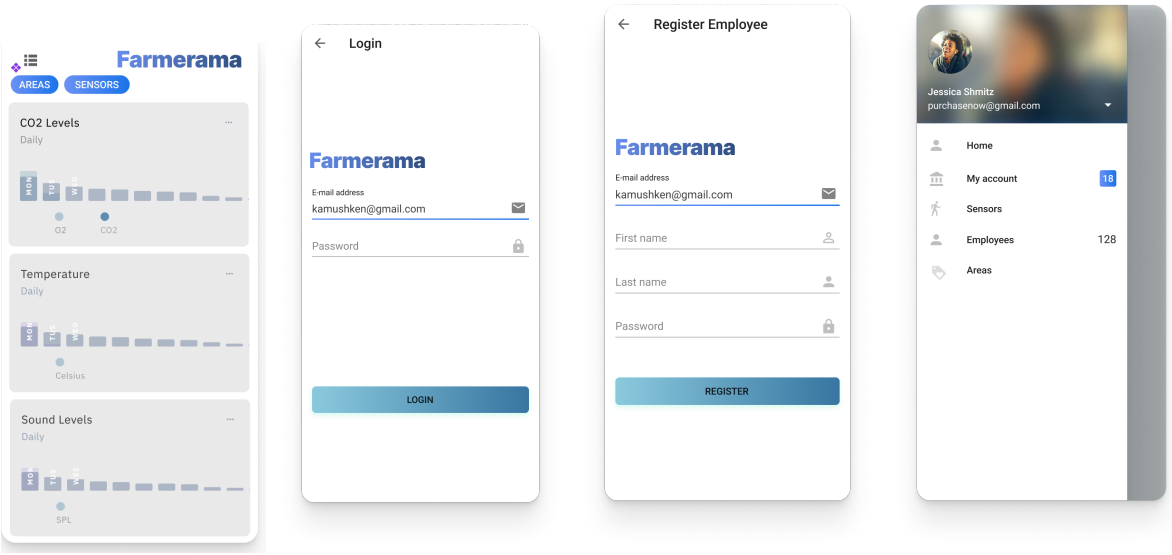


Figure 14, 15, 16, 17 - Mockups of the Android application made in Figma

XML layouts were used to create the app's design. The software is designed to give the user a modern experience, providing automation in the possible scenarios. The design outcomes are modern when combined with components from Material Design and custom graphics.

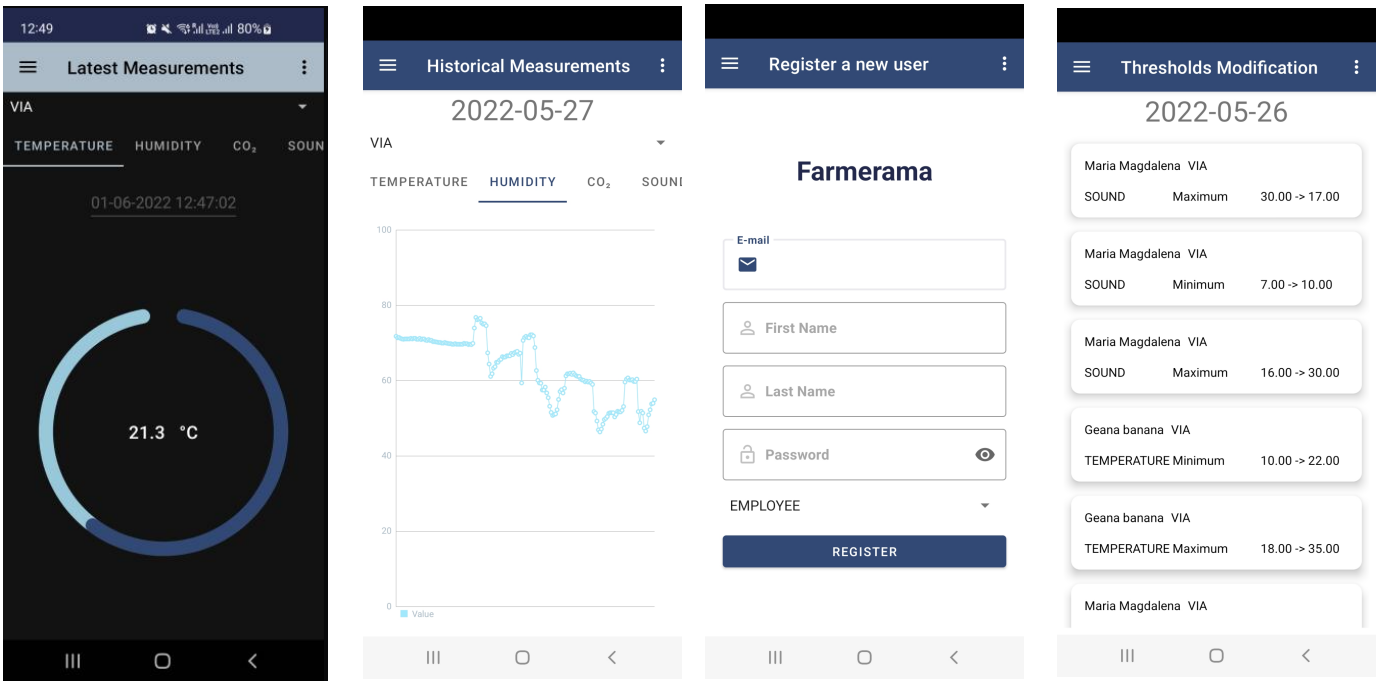


Figure 18, 19, 20 and 21 - End results of the Android application

3.3. Data Engineering

Adriana, Agostina, Luis, Tomas

3.3.1. Technologies

Java Spring Boot

The Spring Framework is a mature, robust, and extremely flexible framework for developing Java web applications.

One of the main advantages of Spring is that it handles most of the low-level parts of application development, allowing for concentration on features and business logic. Due to this fact and the team's prior experience with it, the Spring framework was chosen for the implementation of the data service.

JPA

JPA (Java Persistence API) is a Java specification that is used to store data. JPA serves as a connector between object-oriented domain models and relational database management systems. Using this technology, saving objects in a database is as simple as calling a single save() method with the object. This allows the team to focus on other aspects of software development rather than writing SQL statements.

Swagger

Documentation of the data service API is important to ensure discoverability and ease of use for developers that implement it. Swagger has been chosen as a solution for this reason. Swagger is an API documentation tool that automatically generates HTML documentation based on the service configuration. This tool makes it easier for the Android team to implement API endpoints. For each documented endpoint, Swagger shows information about the URL, HTTP method, inputs and expected outputs.

Cloud computing

Cloud computing refers to the delivery of various services through the Internet. These resources include data storage, servers, databases, networking, and software, among other tools and applications. Amazon has been chosen as the project cloud provider due to its generous free tier that matches the system requirements.

During the design, it was decided that the system would use a number of cloud services, including:

- AWS RDS - for MSSQL database hosting
- AWS ElasticBeanstalk - for hosting data service that will continuously listen to LoraWan socket for incoming measurements
- Docker - for generating images and running docker containers
- AWS ECR - for storing the docker images
- CircleCi - for operating continuous delivery and testing pipelines

3.3.2. Source Database

3.3.2.1. Conceptual Model

After analysing the domain model and the requirements devised during analysis the DAI development team identified the following entities: Barn, Area, Hardware, Measurement, Threshold, and User. While it can be seen in Figure 22 that all of the entities have at least one relationship between them, with the majority being one-to-many relationships, there is also a one-to-one relationship between hardware and area. This was decided by the development team after determining the fact that the hardware is unique to each area, as there is only one device measuring data in each designated area.

Moreover, the User entity is only connected to the rest of the entities through the Threshold entity as the user is allowed to perform changes on the data stored in the threshold table. The relationship between the entities is many to many as the user can perform changes on various thresholds multiple times.

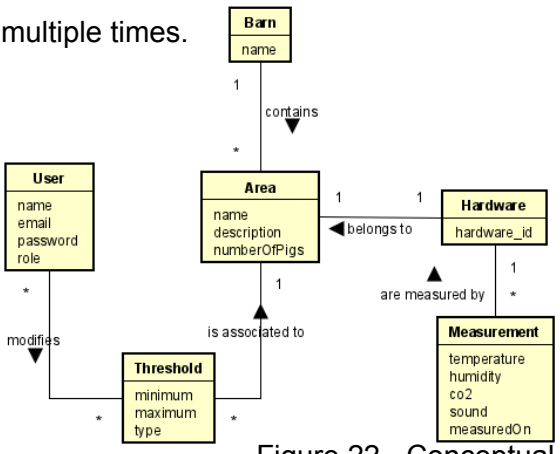


Figure 22 - Conceptual Model of the Source Database

3.3.2.2. Logical Model

The Logical Model was developed after taking into consideration the entities and the relationships between them from the aforementioned Conceptual Model. It has been decided by the development team that the Hardware entity did not possess any important attributes other than its unique id, therefore the entity was merged together with the Area entity as they had a one-to-one relationship between them.

Furthermore, the rest of the entities' attributes were updated according to the 9 mapping rules. Thus the entities with one-to-many relationships got an additional id as a foreign key, such as in the case of Area and Measurement. Since many measurements can be measured within an area, the areaId field was added to the Measurement entity to help keep track of proper data insertions. Moreover, because the entities Threshold and User had a many-to-many relationship between them, an additional table ThresholdLogs was created to keep a record of changes in the thresholds done by the users of the system.

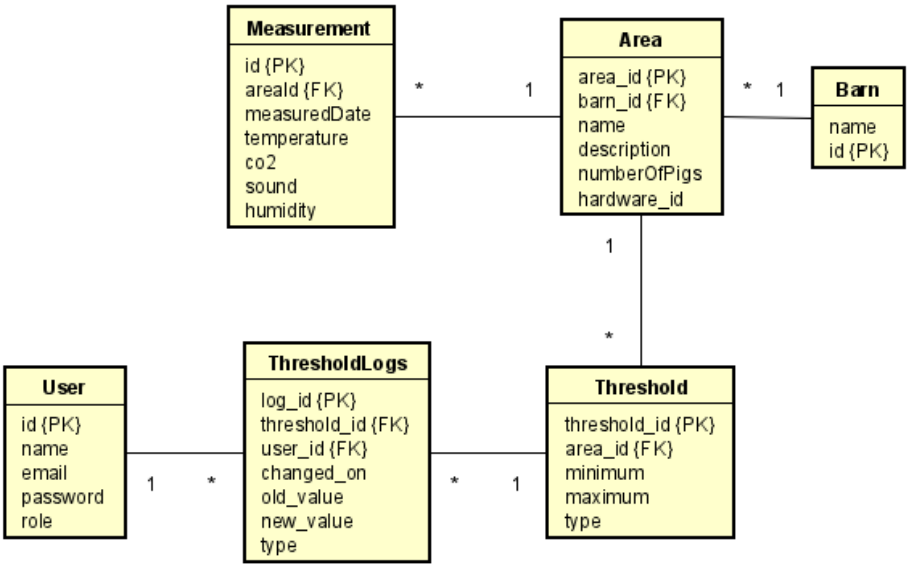


Figure 23 - Logical Model of the Source Database

3.3.3. Data Warehouse

In this section, the design of the data warehouse is presented. The data warehouse was designed to support analytical decision-making following Kimball's Dimensional Modelling techniques and principles, as well as ETL system design and development (Kimball, 2002).

3.3.3.1 Dimensional Modelling

Dimensional Modelling is a four-step design process that includes:

1. Selection of the business process
2. Declaration of the grain
3. Identification of the dimensions
4. Identification of the facts

These steps were developed based on the requirements of the business and users, and after having an understanding of the underlying operational data.

Selection of the Business Process

The business process selected for the system is the quality assurance of the pigs' welfare, in order to deliver the best possible quality of meat and comply with animal welfare regulations.

Declaration of the Grain

The lowest granularity of the system is represented by a measurement (temperature, humidity, CO2, and sound) collected by each sensor from an IoT device.

Identification of Dimensions

The dimensions included in the data warehouse are area, date, and time.

The area represents a meaningful location at the barn for the employees. Areas contain pigs and a device measuring temperature, humidity, CO2, and sound. Important attributes of the areas are: name, the id of the device, and the barn they belong to.

The date and time dimensions are relevant because measurements must be analysed over time. Date includes days, months, weeks, quarters, years and days of week. Time is required because the devices are sending data approximately every 5 minutes. In the time

dimension, meaningful textual descriptions for periods within the day (Morning, Afternoon, Evening and Night).

Identification of Facts

The fact selected reflects measurements in the environment of an area in the barn. This environment is a relevant result of the business process of assuring the quality of the pig's welfare.

All the measurements have a uniform grain and are temperature (celsius), humidity (percentage), CO2 (part per million), and sound (decibels).

3.3.3.2 Star Schemas

In this subsection, the star schemas are presented and explained. There are two star schemas, because the data warehouse includes a database schema for staging the data (stage schema) and a final database schema where the extracted and transformed data is loaded to (dwh schema).

Stage Schema

The stage star schema is a simple schema used to represent the DimArea dimension and the FactEnvironment table existing in the data warehouse's stage schema. Because the purpose of the stage schema is to extract the data from the source database and to make transformations to it, there is no need to include attributes like surrogate keys and attributes relevant to the incremental load and type 2 changes.

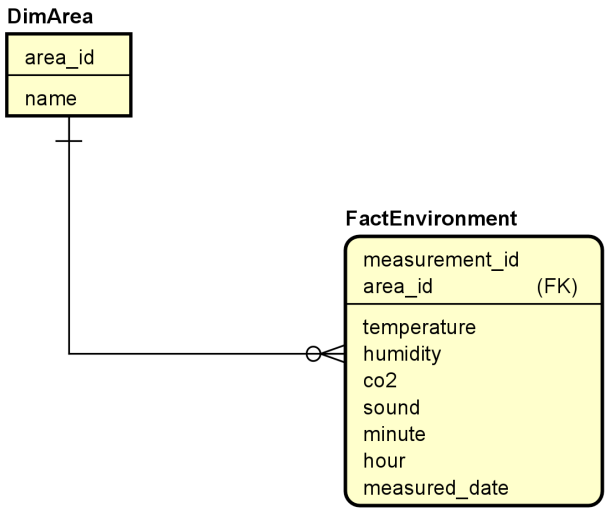


Figure 24 - Stage Star Schema

Data Warehouse Schema

The data warehouse star schema includes the final result of the initial loading process and the incremental loads.

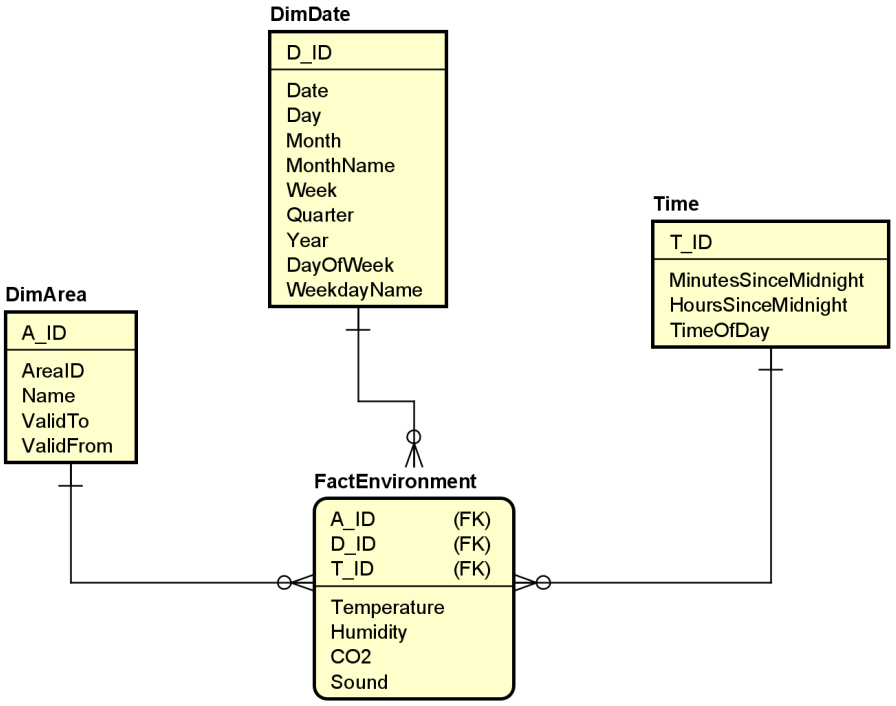


Figure 25 - Data Warehouse Star Schema

Date and time dimensions are included with a surrogate key each that acts as a foreign key in the fact table and is also a part of the fact table's primary key. These dimensions include more verbose attribute names and more possibilities for filtering the fact table by, for example, weekdays or time of day (morning, afternoon, evening, night).

The Area dimension also includes a surrogate key, which acts as a foreign key in the fact table and is also part of the fact table's primary key. There are two attributes to track for type 2 changes: ValidTo and ValidFrom.

The fact table FactEnvironment has a primary key composed of the surrogate key of the previously mentioned dimensions. The attributes in the fact table are Temperature, Humidity, CO2 and Sound.

3.3.3.3 Source Target Mappings

As the Data Warehouse's purpose is to process and store historical data from the source database, source target mappings were designed in order to ensure that the data is mapped from the correct fields across the two schemas. Table 5 contains the fields from the source database and their corresponding parts in the data warehouse. While the number of rows is scarce, this is due to the fact that the data containing business value is only being stored in the area and measurement tables in the source database.

Source			Target		
Table	**Field**	**DataType**	**Table**	**Field**	**DataType**
area	area_id	int	DimArea	AreaID	int
area	name	varchar(255)	DimArea	Name	varchar(50)
measurement	temperature	float	FactEnvironment	Temperature	float
measurement	humidity	float	FactEnvironment	Humidity	float
measurement	co2	int	FactEnvironment	CO2	int
measurement	sound	float	FactEnvironment	Sound	float

Table 5 - Source Target Mapping

3.3.3.4 ETL

The Extract-Transform-Load process is presented in this section. In summary, this process involves extracting data to a staging area, where it is transformed to match the stage star schema before being loaded into the data warehouse's presentation area. This step is crucial for a successful data warehouse. Thus, it requires consistency between previously designed star schemas, source-target mappings, and scripts.

These steps are presented in their logical order according to the design. However, these do not necessarily represent the order of the implementation. Therefore, some parts of the implementation are mentioned here, so that the ETL process used can be better understood in reference to the SQL scripts.

The scripts used to run the ETL process can be found in Appendix K - Data Warehouse. All the scripts are numbered (note that the scripts include the ETL process and the last scripts are the incremental load and type 2 changes).

The ETL process design is summarised in the following activity diagram (Figure 26).

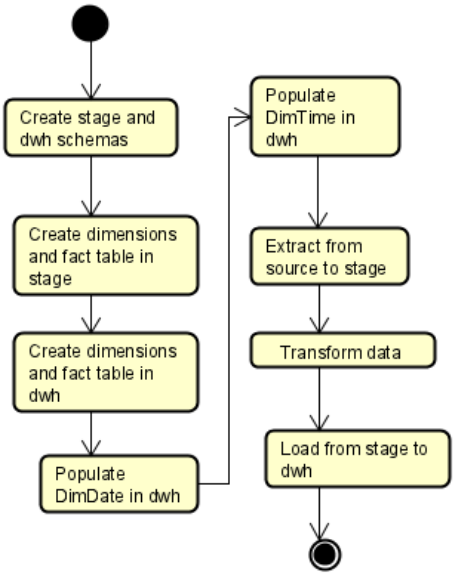


Figure 26 - Initial load ETL process activity diagram

Setting up the Data Warehouse

The first five steps include the setting up of the data warehouse so that the ETL process can be executed. This setting up includes:

- Creating stage and dwh schemas: the stage schema contains the staged data extracted from the source database and the dwh schema contains the data loaded from the stage schema and matching the data warehouse's dimensional model.
- Creating dimension tables in both the stage schema and the dwh schema.
- Populating the DimDate and DimTime with auto generated values.

Extract

The data transfer from the source database to the data warehouse's staging area takes place in this step. The script that performs this function begins keeping track of the date the data is extracted and stored in the etl schema (The etl schema and the LogUpdate table

appear in the next subsection about the incremental load, because their purpose is to log the last load date. However, in the implementation elements from the initial load and incremental load are combined, for example, the creation of the etl schema and LogUpdate table are done when the data warehouse is created). Thus, it can be used in future incremental loads.

Then, data is extracted into the staging dimension and fact table. The data is inserted into the stage DimArea and stage FactEnvironment tables. The entire script for extraction can be found in Appendix K - Data Warehouse.

Transform

The next phase is data transformation. It is performed in the same script as the extraction. Its main goal is to ensure the quality of the data in the process. The focus was put on the following data quality dimensions: Completeness, Consistent Representation and Free-of-Error (Pipino et al 2002).

The first transformation focuses on eliminating duplicate combinations of the same measuredDate, minute and hour fields in the FactEnvironment table in the stage schema after its population. Duplicates can happen if the source database has records within the same date and minute. The importance of this transformation is that the combination of the three fields will create the dwh's FactEnvironment's primary key once the load to the dwh schema is performed. Therefore, the transformation is mostly focused on having a consistent representation and avoiding duplicate key conflicts..

Another transformation checks for missing measurement values (temperature, humidity, CO2 and sound) and validates that the values do not exceed certain maximums and minimums agreed upon by convention. If the values are missing or exceed the maximum or minimum values, then the value will be replaced by the average of the currently staged values of the same type. Hence, this transformation focuses on preserving completeness by filling in missing values, and the Free-of-Error data quality dimension, by making sure that the values are validated and therefore correct and reliable.

Load

After the transformation, the data is loaded from the staging schema into the dwh schema. In this script, the DimArea's ValidFrom and ValidTo fields are reset. Although this step logically belongs to the type 2 changes, the update of ValidFrom and ValidTo are implemented in this script so that the right values are already present in the data warehouse when the

incremental load and type 2 changes are executed. The ValidTo date will match the first record in the database. The ValidTo field has been set to a distant future date.

Loading data to the dwh schema from the stage schema is done by joining the FactEnvironment in the dwh schema with the FactEnvironment in the stage schema, in order to include the right surrogate keys (A_ID, D_ID and T_ID) as the primary key of records in the FactEnvironment. The DimArea is loaded directly from the stage and a surrogate key is auto generated for each record.

Incremental Load

The activity diagram below presents the main steps in the ETL process for the incremental load. As with the previous ETL diagram, these steps are presented in their logical order, which means that they do not necessarily represent the order of the implementation. For example, in this case, setting up the data warehouse and the initial load in the implementation already contain some elements relevant to the incremental load and type 2 changes. For example, the ValidTo and ValidFrom attributes of the dimensions are only used in the type 2 changes, however, in the implementation, these attributes are already included in the initial creation of tables.

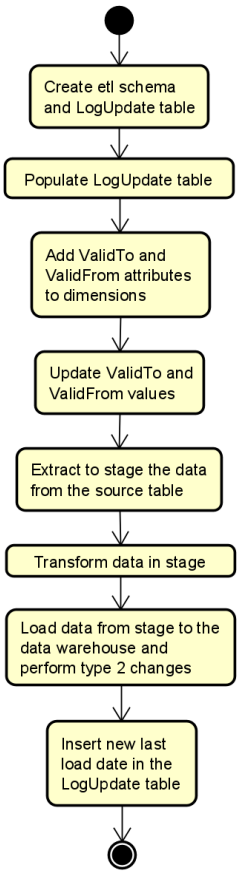


Figure 27 - Incremental load ETL process activity diagram

First the ETL schema must be created to contain the logs about last load dates. The LogUpdate table is created for this purpose and an initial date is inserted. In the case of this data warehouse, the initial date inserted was the date when the initial load was completed. The last load date will be used to extract only data from the source database previous to that date.

Next, the ValidTo and ValidFrom attributes must be included in each dimension, in order to account for type 2 changes. The value included in ValidTo is a date in the distant future and in ValidFrom the date of the first record in the source database.

The next step is to extract the relevant data from the source database. The relevant data is the data that was inserted after the last load date. The FactEnvironment and the DimArea are truncated every time before extracting the data.

After extracting the data, the same transformations as described in the initial load are performed.

Then, the data is loaded to the data warehouse. To do this, the DimArea, DimDate and DimTime are used to join the FactEnvironment table in stage with the FactEnvironment table in the data warehouse. Type 2 changes are also performed (see next subsection).

The final step is to insert the new last load date in the LogUpdate table. In the implementation, the last load date is inserted earlier in the process because running the scripts in AWS was slow and the next incremental load would miss records that were inserted while the previous incremental load was running (for a more detailed account see the Implementation section).

Type 2 Changes

To handle changes in the dimensions, type 2 changes were selected. Type 2 changes consist of creating a new dimension entry whenever changes are detected.

The changes can be adding a new record (for example, a new area), editing an existing record (for example, changing the name of an area), deleting an existing record (for example, deleting an area).

The type 2 changes or Change Data Capture by comparison work as follows. A ValidTo and ValidFrom attribute are added to the relevant dimension. In the case of the system, the attributes are added to the DimArea. Changes are handled as follows:

- If a **new row** has been inserted in the source database, then a new row is inserted in the data warehouse with ValidTo set to the date of insertion and ValidFrom to a date in the distant future.
- If a **row has been edited** in the source database, then the ValidTo date of the record corresponding to the edited row (with the old values) is updated to yesterday and the ValidFrom is left unchanged. A new record with the content from the updated record in the data source is created with a ValidFrom set to the date of edition and a ValidTo to a date in the distant future.
- If a **row is deleted** in the source database, then the ValidTo attribute is set to yesterday in the data warehouse and the ValidFrom is left unchanged.

Visualisation

PowerBI was used to create the data visualisations. The visuals used are meant to respond to the following Farmerama staff questions:

a) Are there any extreme temperatures in a month?
b) Are there significant variations in temperature depending on the month?
c) What is the humidity like inside the barn?
d) How does humidity behave?
e) When comparing different months, what are the highest and lowest humidity levels?
f) What are the sound levels inside the barn?
g) How does sound change throughout the year?
h) Do the volume levels fluctuate during the week?
i) When is it the loudest during the day?
j) What are the CO2 levels like inside the barn? What do a week's CO2 measurements look like? What about a month's measurements?
k) Have CO2 measurements changed since more pigs were included in the barn in early 2022?

These questions are designed to help specialists better understand the pig's environment so that they may make decisions to ensure their health.

All of the questions were answered using PowerBi data visualisations. The visualisations were divided into four dashboards, one for each sensor and one for the main dashboard, which answered the most relevant queries briefly:

- Dashboard #01: Farmerama’s environment - main dashboard
- Dashboard #02: Temperature and Humidity
- Dashboard #03: CO2
- Dashboard #04: Sound

The complete collection of dashboards can be found in Appendix I - Data Visualisations. Because all dashboards follow PowerBI visualisation best practices (Russo, 2017), they were included in a one-page report for each related topic. It is possible to identify the following recommendations to create a perfect dashboard:

- Design for a target: Dashboard #01 was designed to be shared with those interested in having an overview of the environment at the barn. The rest of them are meant for specialists.
- Keep everything at glance: Using Dashboard #01 as an example, it is possible to get a quick and broad sense of how the barn looks on one page
- Keep it simple: All of the visualisations are easy to interpret, including the captions, which make the graph self-readable in the eyes of specialists. Besides, our main dashboard creates a selection of the most relevant aspects of the analysis to give an overview of the barn
- Align elements: all elements in the dashboards are aligned, including titles
- Be consistent: same selection of colours for minimum and average values, as well as a consistent colour for each kind of measurement

Overall, the intention was to be clear, highlight the most relevant information and be consistent by choosing the right charts, colours and leaving the noise off. The complete dashboard is available in Appendix I - Data Visualisations. The snippet below is an example of the visuals chosen, answering question a. It shows a line chart as an example of choice of colours and charts.

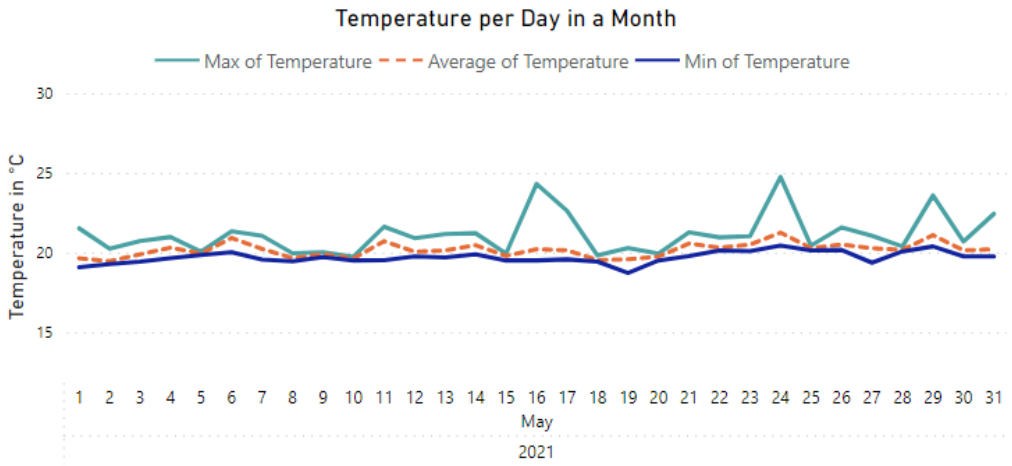


Figure 28 - Temperature in a month

To display the behaviour of temperature in a month a line chart was used since it is necessary to relate the measurements and the relation with time. Because PowerBI does not support small numbers in the y axis, values were modified in order to show with more clarity the data variations.

The image below shows an example of the sound measurements within a week, bringing answers to questions h and i.

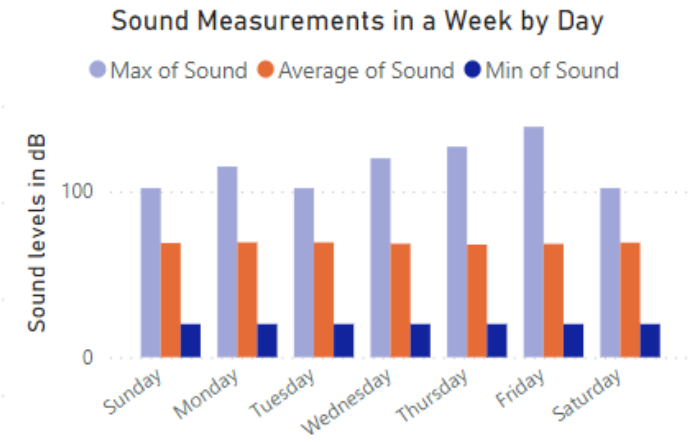


Figure 29 - Sound measurements in a week by day

Because the purpose is to compare measures based on the day of the week, a bar chart was used. In this case the y axis starts from 0 and days of the week were selected as temporal series.

Figure 30, shows the distribution of sound measurements during the day.

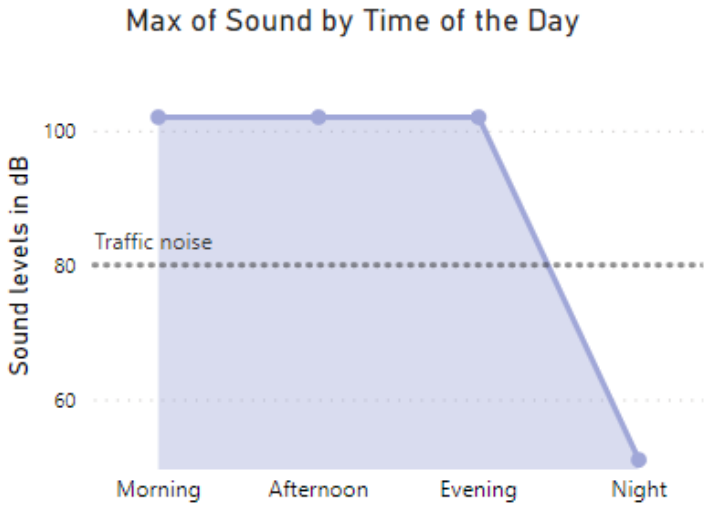


Figure 30 - Max of Sound by time of the Day

Again, a line chart was used to correlate sound behaviour with the time of day. Only the highest measurements are given in this situation, as they are the ones that may cause disturbance to animals and employees. To distinguish those maximums that may have a negative impact on the animals' well-being, a constant line was applied. Values below 80 dB

would not be a significant issue. Data markers were added to aid readability because the time intervals were long and the data behaviour was flat.

3.4. IoT

Adrian, Khaled, Morten

3.4.1. Technologies

Hardware

The IoT devices that were installed in areas that were to be monitored by Farmerama required a custom-built VIA ARDUINO MEGA2560 Shield that utilised a temperature, humidity, CO_2 and a sound sensor. In addition to the aforementioned sensors, the device also used an antenna and a LoRaWAN driver using the RN2483 module for receiving and sending data as well as a RC servo used to open and close a window. The device also consisted of a micro-controller that was responsible for powering and running the entire device.

FreeRTOS

To control access to shared resources and to support the use of tasks, thus simulating tasks running in parallel, the Free Real Time Operating System was used. FreeRTOS introduced easy access to several data structures such as event groups, message buffers, message queues and mutexes as well as dynamic task priorities and greatly simplified the entire development process by taking care of concepts like task scheduling while providing an easy-to-use API.

Event groups were used in order to ensure the correct flow of the application, thus guaranteeing that the FarmeramaTask would wait for all tasks responsible for measuring data to have signified being done before fetching the data and passing it on to the SenderTask.

Message buffers and message queues were used in order to pass data between tasks. The message buffer was used between the Lora Driver and the ReceiverTask to automatically enqueue downlink messages received from the LoRaWAN network. Message queues were used between the main FarmeramaTask and each task measuring data.

In order to control access to the low and high temperature thresholds stored on the device, a mutex was introduced. By putting the mutex in Configuration.c itself, it is ensured that the shared resource itself is responsible for giving and taking back the mutex when its methods are called. This way, the ReceiverTask (updating the data) and the ServoTask (reading the data) do not need to handle the mutex and as such, the protection cannot be violated by mistake. This is shown in figure 31.

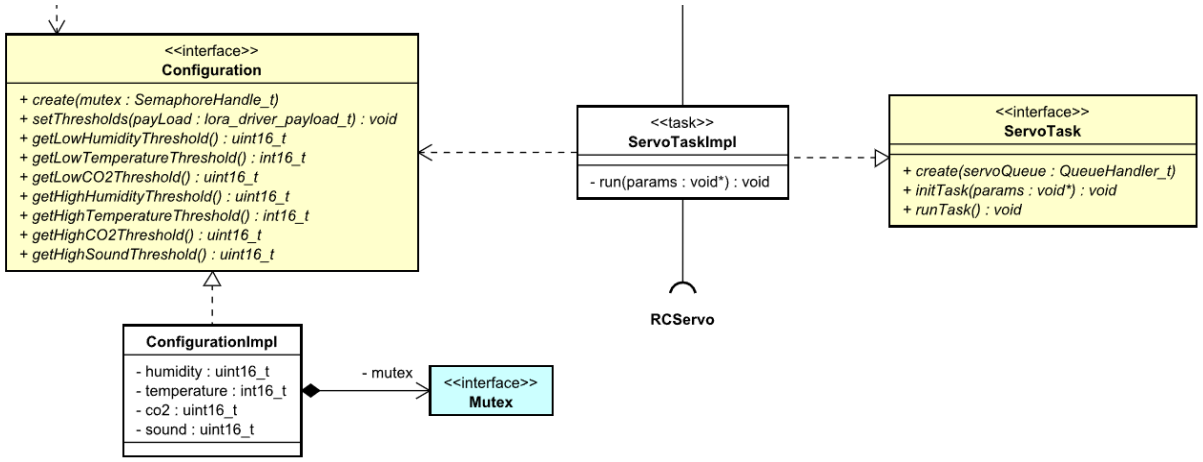


Figure 31 - Usage of a mutex to protect a shared resource in C

Sensors

Regarding user story 1, 2, 3 and 4, the following sensors were utilised:

1. HIH8120 - for humidity and temperature
2. MH-Z19 - for CO_2
3. SEN14262 - for sound

As Farmerama is located just outside of Horsens Kommune, the farm is within the range of the LoRaWAN network. To connect to the LoRaWAN network and send/receive uplink- and downlink messages, the RN2483 driver was used. Using the LoRaWAN network allows the device to be located in a remote area that might not have internet access.

Actuators

Regarding user story 23 and 24 which relates to opening and closing the window of the specific area the device is located in, an RC servo is connected to each device and is rotated depending on a given input value. This allows the task responsible for measuring temperatures to pass a measured temperature to the task responsible for operating the window. The value of the received temperature is then compared with the low and high thresholds stored in memory and the servo position is set based on this comparison.

3.4.2. Class diagram

In the design phase of the development it was decided to have a main task that would be responsible for controlling the rest of the tasks. The Farmerama task would notify the HumidityTemperature task when it should run, so that all the rest of the task could follow. The two event groups were responsible for this task synchronisation, so that when one of the tasks was done, the next task would proceed. At the end, when all the measurement tasks had completed their responsibilities, the Farmerama task would proceed to request the UplinkMessageBuilder to build an uplink payload that would be further sent to the Sender task. The Sender task would then send the payload through the LoRaWAN network to the gateway application. Once the payload had been sent, the Farmerama task would be delayed by 5 minutes, and this delay would cause the aforementioned measurement tasks and the sender task to wait.

Figure 32 (see next page) is a small part of the class diagram of the application running on an IoT device. The full diagram can be found in Appendix E.

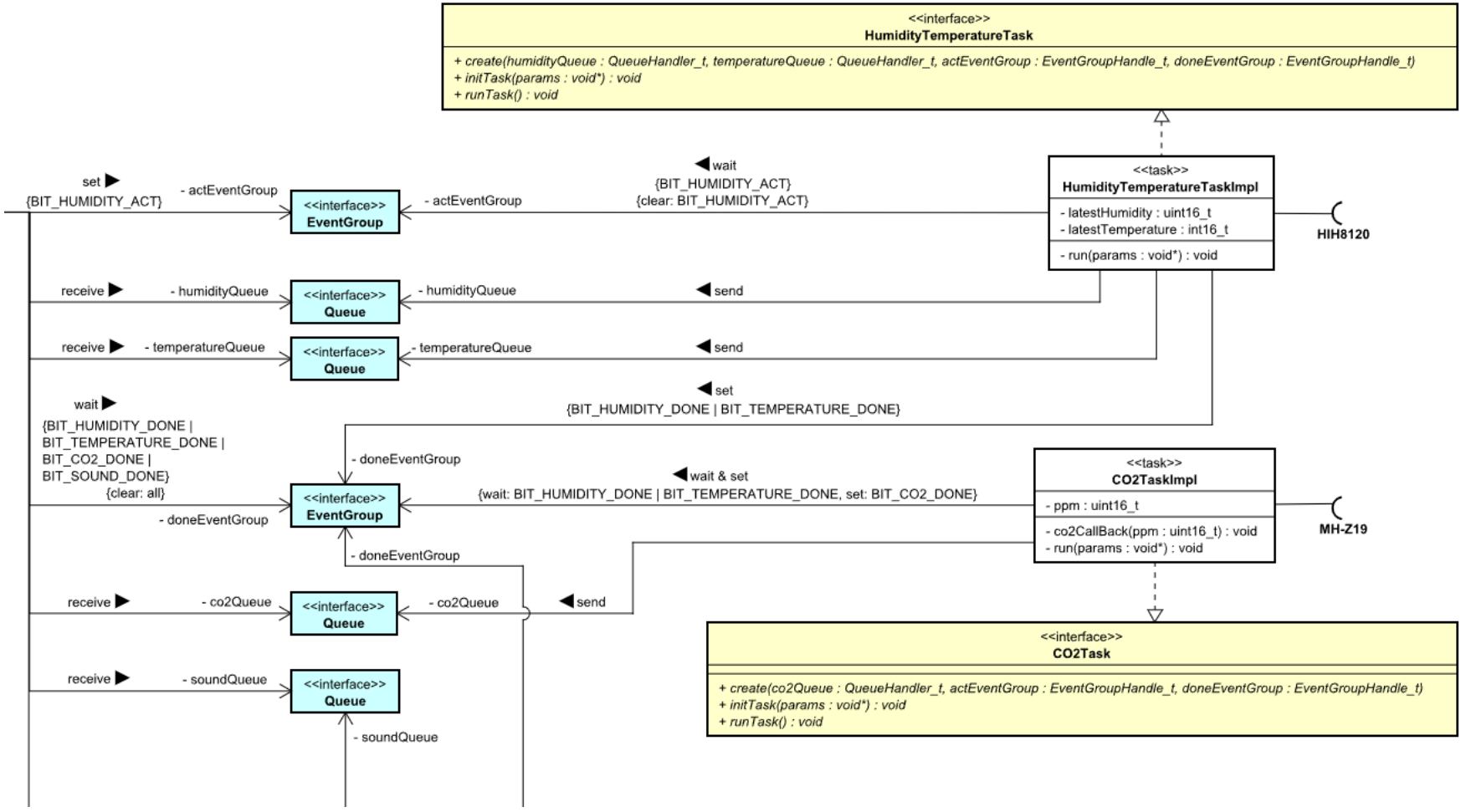


Figure 32 - Extract of the IoT micro-controller class diagram

3.4.3. Sequence diagram

The sequence diagram contributed greatly to the pre-implementation phase, as it concluded the design phase and served as a blueprint for the implementation. Overall, the sequence diagram formed a critical part of the solution.

As the sequence diagram is too large to be put here, it can be found in Appendix F. It is important to note that the provided sequence diagram follows a single use cese. Only the humidity measurement is presented in the “View latest measurement” sequence diagram.

3.5. SOLID Principles

Adriana, Vlad, Stefania

While developing Farmerama’s system, some of the SOLID principles were used to facilitate implementation and future maintenance of the product. The section shows an overview of the most relevant principles in this project.

3.5.1. Single Responsibility Principle

The Single Responsibility Principle was applied throughout the system to help with testing, organisation, and minimise dependencies between classes. There are many examples of this principle as it was used by all three sub-teams. An example is the design of the controllers in the web service application, where each controller is responsible for a different part of the data transmitted. Moreover, the MVVM architecture also follows this principle as the Views, ViewModels and Models each have their own responsibility. In the IoT application, each task also has its own responsibility.

The design of the controllers can be seen in Appendix E - Data Service Class Diagram while the design of the MVVM in E - MVVM class diagram.

3.5.2. Open-closed Principle

While the system follows several of the SOLID principles, it fails to adhere to the Open-closed principle. One example is the Measurement entity. As the entire system is centred around measurements that are passed from the IoT device all the way to the Android application, the class is used all across the project. The design of the class was decided upon by the DAI team after establishing the contents of data sent from the hardware device.

As such the Measurement class has four attributes, one for each data measurement used within the system. This decision was taken in order to minimise redundant and repetitive data. If in the foreseeable future the system is undergoing expansion and will add new measurements, a new attribute will have to be added to the class, which is a clear violation of the Open Closed principle. Henceforth, a solution that follows the principle would have been to have value and type attributes, however that will result in storage of repetitive data.

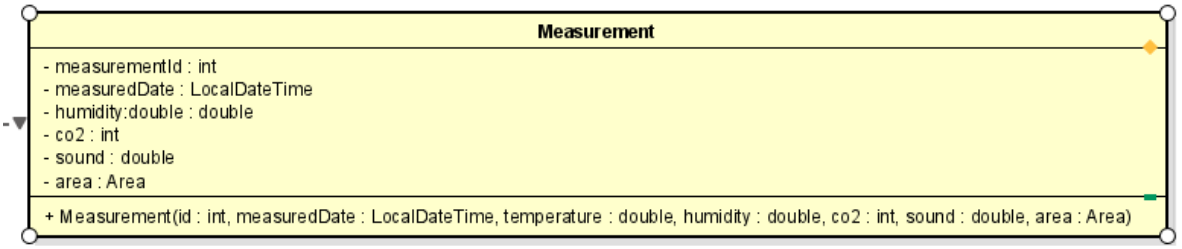


Figure 33 - Violation of Open Closed Principle

3.5.3. Dependency Inversion Principle

The Dependency Inversion principle allows for higher level classes to not depend on explicit implementations of lower level classes. The abstraction occurs by depending on interfaces, as their abstraction allows for the separation of different components. Henceforth, if in the foreseeable future the system undergoes changes in any of the tiers, for example utilising a different persistence framework, the abstraction layer will stay the same and only the implementation will change.

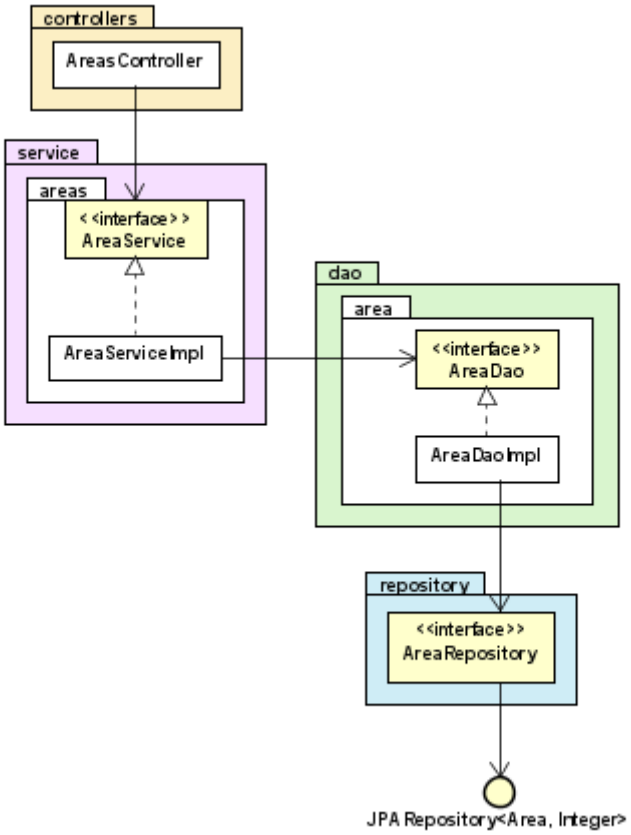


Figure 34 - The Dependency Inversion Principle used on the server

4. Implementation

Georgiana, Vlad, Stefania, Tabita

4.1. Android application

In the ServiceGenerator class an implementation of the MeasurementApi class is generated by Retrofit. Due to the common representation of data received from the web service, the GsonConverterFactory was used to automatically map the JSON data into the POJOs.

```
8   public class ServiceGenerator {
9       private static String url = "http://sep4-data-service.eu-central-1.elasticbeanstalk.com/";
10      private static UserApi userApi;
11      private static MeasurementApi measurementApi;
12      private static AreaApi areaApi;
13      private static BarnApi barnApi;
14
15      public static MeasurementApi getMeasurementApi() {
16          if (measurementApi == null) {
17              measurementApi = new Retrofit.Builder()
18                      .baseUrl(url + EndpointsHelper.AREAS)
19                      .addConverterFactory(GsonConverterFactory.create())
20                      .build()
21                      .create(MeasurementApi.class);
22          }
23          return measurementApi;
24      }
```

Figure 35 - ServiceGenerator class

In the MeasurementApi interface, Retrofit was used in order to turn the HTTP API into a Java interface. Therefore, the MeasurementApi interface implementation was generated by the Retrofit class. An asynchronous HTTP request, in this case a GET request, is made to the remote webserver with each Call that is created from the MeasurementApi. On the other hand, annotations such as “@Path”, “@Query” were used in order to handle the requests and manipulate the url.

```
12  public interface MeasurementApi {
13
14      @GET("{areaId}/temperatures")
15      Call<List<MeasurementResponse>> getTemperatures(@Path("areaId") int areaId,
16                                                      @Query("date") String date);
17
18      @GET("{areaId}/sounds")
19      Call<List<MeasurementResponse>> getSpls(@Path("areaId") int areaId,
20                                              @Query("date") String date);
21
```

Figure 36 - MeasurementApi interface

Figure 37 represents the implementation of the getMeasurements method in the class MeasurementApiAdapter. The adapter returns the necessary api call with the specified areaId and date. The measurementType was sent to switch between the methods in order to retrieve the list of measurements desired by the user at the time.

```
31      @Override
32  @   public Call<List<MeasurementResponse>> getMeasurements(MeasurementType type,
33                                                              int areaId, String date) {
34          MeasurementApi measurementApi = ServiceGenerator.getMeasurementApi();
35          switch (type) {
36              case TEMPERATURE:
37                  return measurementApi.getTemperatures(areaId,date);
38              case HUMIDITY:
39                  return measurementApi.getHumidities(areaId, date);
40              case SPL:
41                  return measurementApi.getSpls(areaId, date);
42              case CO2:
43                  return measurementApi.getCo2s(areaId, date);
44              default:
45                  throw new IllegalStateException("Unexpected value: " + type);
46          }
47      }
```

Figure 37 - MeasurementApiAdapter class

The ToastMessage class was implemented to set and return a String message. The message was encapsulated in a MutableLiveData object which has been observed in the MainActivity class. The methods were set as static so the message can be set from any part of the system. Therefore, the main activity will display a toast with the specified message at any time the setToastMessage method is called.

```
public class ToastMessage {

    private static MutableLiveData<String> toast = new MutableLiveData<>();

    public static LiveData<String> getToastMessage() { return toast; }

    public static void setToastMessage(String string) { toast.setValue(string); }
}
```

Figure 38 - ToastMessage class

In figure 39, the following method is used in two different scenarios: the user is online or the user is offline. In case the user is online, which is verified by the ConnectivityChecker class, the latestMeasurement variable which is MutableLiveData is going to be set to the result received from the web service, in this case the latest measurement for a specific area and a specific measurement type. At the same time, the measurement is loaded in the database. In case the response from the web service is a bad one, the toast message is displaying the error. As for the second scenario, when the user is offline, the latest measurement is going to be retrieved from the database with the use of asynchronous queries, ListenableFuture. ExecutorService was also used in order to handle and track progress in asynchronous tasks.

```
public void retrieveLatestMeasurement(int areaId, MeasurementType type, boolean latest) {
    if(onlineChecker.isOnlineMode()) {
        Call<List<MeasurementResponse>> call = adapter.retrieveLatestMeasurement(type, areaId, latest);
        call.enqueue(new Callback<List<MeasurementResponse>>() {
            @EverythingIsNonNull
            @Override
            public void onResponse(Call<List<MeasurementResponse>> call, Response<List<MeasurementResponse>> response) {
                if (response.isSuccessful()) {
                    executorService.execute(() -> {
                        for (MeasurementResponse measurement : response.body()) {
                            database.measurementDAO().createMeasurement(measurement.getMeasurement(type));
                            latestMeasurement.postValue(measurement.getMeasurement(type));
                        }
                    });
                } else {
                    ErrorReader<List<MeasurementResponse>> responseErrorReader = new ErrorReader<>();
                    ToastMessage.setToastMessage(responseErrorReader.errorReader(response));
                }
            }

            @EverythingIsNonNull
            @Override
            public void onFailure(Call<List<MeasurementResponse>> call, Throwable t) {
                Log.i( tag: "Retrofit",  msg: "Could not retrieve data");
            }
        });
    }
    else {
        ListenableFuture<Measurement> result = database.measurementDAO().getLatestMeasurement(type, areaId);
        result.addListener(() -> {
            try {
                latestMeasurement.postValue(result.get());
            }
            catch (Exception e) {
                Log.i( tag: "Room",  msg: "Could not retrieve data");
            }
        }, Executors.newFixedThreadPool( nThreads: 5));
    }
}
```

Figure 39 - MeasurementRepository class

The implementation of the NotificationWorker class was made through extending the Worker class. The method doWork runs in the background regardless of the state of the application. The worker class is called in the main activity as a PeriodWorker and it is set to run every 15 minutes. The method is responsible for calling the API to check if there are any exceeded measurements lately and for every log the publishNotification method is called to notify the users about the measurements exceeding the thresholds.

```
public class NotificationWorker extends Worker {

    @NonNull
    @Override
    public Result doWork() {
        Call<List<LogResponse>> call = ServiceGenerator.getThresholdsApi().getLatestLogs();
        try {
            Response<List<LogResponse>> response = call.execute();
            for (int i = 0; i<response.body().size(); i++) {
                publishNotification(response.body().get(i).getLog(), i);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return Result.success();
    }
```

Figure 40 - NotificationWorker class

The purpose of the ErrorReader is to convert the error messages received from the web service into Strings which are later displayed to the user in a Toast message. In this way, the response and the class were made generic to allow different response types to be read from the buffer. After the string has been received, the stream is closed so that the resources associated with it can be released. It is also advantageous for further code reuse.

```
public class ErrorReader<T> {

    public String errorReader(Response<T> response) {
        StringBuilder stringBuilder = new StringBuilder();
        BufferedReader bufferedReader = null;
        String errorMessage = null;
        if(response.errorBody() != null) {
            bufferedReader = new BufferedReader(new InputStreamReader(response.errorBody().byteStream()));
            try{
                while((errorMessage = bufferedReader.readLine()) != null) {
                    stringBuilder.append(errorMessage);
                }
                bufferedReader.close();
            }
            catch (Exception e) {

            }
        }
        return stringBuilder.toString();
    }

}
```

Figure 41 - ErrorReader class

The LineChart was used for a graphical display of the historical data. The implementation of the view was built on the dependency 'com.github.PhilJay:MPAndroidChart:v3.1.0' and was used in HistoricalMeasurementsFragment class. The minimum value is set to 0 and the maximum value of the chart is set based on the type of the retrieved measurements. Additionally, by pressing one of the values a custom MarkerView is created and displays the time the value was registered.

```
private void setupViews() {
    viewModel.getMeasurements().observe(getViewLifecycleOwner(), measurements -> {
        List<Entry> entries = new ArrayList<>();
        for (int i = 0; i < measurements.size(); i++) {
            entries.add(new Entry(i, (float) measurements.get(i).getValue()));
        }

        ILineDataSet lineDataSet = new LineDataSet(entries,  label: "Value");
        LineData lineData = new LineData(lineDataSet);
        lineChart.setData(lineData);
        if (!measurements.isEmpty()) {
            lineChart.getAxisLeft().setAxisMaximum(measurements.get(0).getMeasurementType().getMaximum());
            lineChart.getAxisLeft().setAxisMinimum(0);
        }

        lineChart.setOnChartValueSelectedListener(new OnChartValueSelectedListener() {
            @Override
            public void onValueSelected(Entry e, Highlight h) {
                TimeMarkerView tmv = new TimeMarkerView(getContext(), R.layout.marker_date);
                tmv.refreshContent(DateFormatter.formatDate(measurements.get((int) e.getX()).getDateTime().toString()));
                lineChart.setMarker(tmv);
            }

            @Override
            public void onNothingSelected() {
            }
        });
    });
}
```

Figure 42 - HistoricalMeasurementsFragment

As for the IUserDAO, both convenient and manually imputed queries were used. When a new user is added in the database, in case the tables are going to be loaded again upon a unique or primary key constraint violation occurrence, the “Replace” makes sure the pre-existing rows that are the cause of this violation will be deleted and reinserted normally. ListenableFuture was used for retrieval of users/employees as well as employee by id due to the fact that in the repository, the LiveData which is observed in the viewmodels is set to the result of the Future.

```java
@Dao
public interface IUserDAO {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void registerUser(User user);

    @Query("DELETE FROM user_table")
    void removeUsers();

    @Query("DELETE FROM user_table WHERE userId = (:id)")
    void removeUserById(int id);

    @Update
    void updateUser(User user);

    @Query("SELECT * FROM user_table")
    ListenableFuture<List<User>> getAllEmployees();

    @Query("SELECT * FROM user_table WHERE userId = (:id)")
    ListenableFuture<User> getEmployeeById(int id);

}
```

Figure 43 - The IUserDAO interface

The DonutProgressView is used in the Latest Measurements page. It is a View in a ConstraintLayout that is binded with the use of the id in the specific Fragment. As default, a lighter blue colour is displayed that will be overridden by a darker one to show the measurement value.

```xml
<app.futured.donut.DonutProgressView
    android:id="@+id/latestMeasurements_donut"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:donut_bgLineColor="@color/teal_200"
    app:donut_gapAngle="270"
    app:donut_gapWidth="20"
    app:donut_strokeWidth="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Figure 44 - DonutProgressView

4.2. DAI Team

Adriana, Agostina, Luis, Tomas

Controller

The Temperatures controller (figure 45) is a class that accepts, validates, processes, and returns a response to the API temperature requests. When the following requirements are satisfied, the method getLatest() is invoked: the path must be in the form "/areas/id/temperatures" and the parameter latest must be set to true. The id of the area from which the client wants to get measurement is an input into the method. This value is passed to the model that handles the request. If an error occurs during the request processing, the controller returns a "Bad Request" status code along with the explanation for the failure.

```
@GetMapping(value = "/areas/{id}/temperatures", params = "latest=true")
public List<SentMeasurement> getLatest
        (@PathVariable int id, @RequestParam("latest") Optional<Boolean> isLatest) {
    try {
        return temperatureModel.getLatest(id);
    } catch (Exception e) {
        throw new BadRequestException(e.getMessage());
    }
}
```

Figure 45 - Temperature controller

Service

The service classes' responsibility is to implement business rules that have been determined via analysis. The getLatest() request is simply delegated to temperatureDao since it has no business rules (figure 46). After calling the temperatureDao, the future object response from it needs to be awaited before returning the List of Measurements.

```
@Override
public List<SentMeasurement> getLatest(int areaId) throws Exception {
    return Helper.await(temperatureDao.getLatest(areaId));
}
```

Figure 46 - Temperature model

Dao

The dao layer implementation for the getLatest() method is shown in figure 47. The method delegates the call further into the JPA repository interface. To make this call asynchronous, the Future and AsyncResult classes have been used. AsyncResult is an implementation of the Future interface that's used to wrap the repository's result, such that getLatestByArea() call does not block the main thread This ensures that the data service has high performance and therefore can handle large numbers of requests.

```
@Override
@Async
public Future<List<SentMeasurement>> getLatest(int areaId) {
    return new AsyncResult<>(repository.getLatestByArea(areaId));
}
```

Figure 47 - Temperature DAO

JPA Repository

A JPA native query has been developed to retrieve the most recent temperature measurement. (figure 48) The query returns the most recent temperature by area id. The SentMeasurement projection returned by the query method has two values: MeasuredDate and Value.

```
14  @Repository
15  public interface TemperatureRepository extends JpaRepository<Measurement, Integer> {
16
        1 usage  tomas.ondrejka2000@gmail.com <304150@via.dk> *
17      @Query(value = "SELECT TOP 1 measured_date as measuredDate, temperature as value, measurement_id as measurementId, area_id as areaId FROM measurement " +
18              "WHERE measurement.area_id = :area_id Order By measurement_id DESC", nativeQuery = true)
19      List<SentMeasurement> getLatestByArea(@Param("area_id") int area_id);
20
```

Figure 48 - Temperature JPA query

Socket listener for measurements

One of the requirements for data service was to be able to receive measurements from IoT devices. It was decided that the LoraWan network would be used for this purpose. The LoraWan network exposes data to a socket that our data service can listen to. Figure 49 shows a WebsocketClientController that connects to the socket and receives incoming data via the onText() method. This method then takes the input JSON string and parses it into a Java Object, which is used as input to the model's saveSocketData() method.

To ensure that the service is listening to the socket only in the prod environment, spring annotation @Profile was used. This prevents data from being received and stored multiple times while developing on localhost.

```
@Profile("prod")
@Component
public class WebsocketClientController implements WebSocket.Listener {

    public CompletionStage<?> onText(WebSocket webSocket,
                                     CharSequence data, boolean last) {
        String json = data.toString();
        Gson gson = new Gson();
        SocketData socketData = gson.fromJson(json, SocketData.class);

        try {
            measurementModel.saveSocketData(socketData);
        } catch (Exception e) {
            e.printStackTrace();
        }

        webSocket.request( n: 1);
        return new CompletableFuture().completedFuture("onText() completed.")
                .thenAccept(System.out::println);
    }
```

Figure 49 - Websocket listener

Incremental load script scheduling

Class Etl was created to ensure that the data warehouse is always up to date (figure 50). It has a method called performEtl() that connects to the database, loads an incremental load script and applies it. The database connection is closed after a successful run. An exception message is sent to the console if the trigger is not successful.

In the Etl class, two Spring annotations were used: @Profile for only activating the script in production to eliminate duplicate script triggers, and @Scheduled for scheduling the method call at midnight every day.

```
@Profile("prod")
@Component
public class Etl {

    @Scheduled(cron = "0 0 0 ? * *")
    @Async
    public void performEtl() {
        System.out.println("ETL Triggered");
        try {
            Connection c = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);
            if (c == null)
                return;

            Reader reader = new BufferedReader(new InputStreamReader(
                    getClass().getClassLoader().getResourceAsStream( name: "Etl.sql")));
            ScriptRunner sr = new ScriptRunner(c);
            sr.setSendFullScript(true);
            sr.runScript(reader);
            c.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
```

Figure 50 - Incremental load script trigger

Data warehouse incremental load

The incremental load and type 2 changes were implemented in a single script that is being executed on a daily basis by the scheduler presented above.

The implementation of the script mainly follows the incremental load and type 2 changes ETL process outlined in the Design section. Therefore, the overall implementation aspects are trivial, except for some details that were presented in the Design section, in order to make the ETL process used easier to understand. Here, the details presented are related to the new last load update which appears in figure 51. Note that @NewLoadDate is used for type to changes (therefore, it is an int, and it is assigned by converting @NewLastLoadDate), and @NewLastLoadDate is used to set the new last load date in the LogUpdate.

```
-- Incremental load and Type 2 changes script
-- New and future load dates.
DECLARE @NewLoadDate INT;
DECLARE @NewLastLoadDate DATETIME;
-- Two hours have to be added because of the server's timezone
SET @NewLastLoadDate = dateadd(HOUR, 2, getdate());
SET @NewLoadDate = CONVERT(CHAR(8), @NewLastLoadDate, 112);
-- Add one minute to the last load date to make sure it doesn't load records in that minute in the next load */
SET @NewLastLoadDate = DATEADD(mi, 1, @NewLastLoadDate);
```

Figure 51 - Incremental load script load date assignment

First, because the script is running in the cloud (AWS), there is a time zone difference between the local time and the data warehouse's time zone. The difference is two hours. In order to avoid having a new last load date with two hours difference, two hours were added to the new last load date. This way, it is ensured that no duplicate records, and thus primary key conflicts, occur in the data warehouse. These would occur the first time the incremental load runs in the cloud, after the initial load has been completed and the last load date has been set to the local date.

Second, a minute is added to the new last load date. This is the case, because the data warehouse is not taking into account seconds. Therefore, the new last load date is set to the same minute in which the incremental load was executed. Then, it is possible that the next time the incremental load executes, it will load the same records within the same minute. This triggers a duplicate key conflict in the data warehouse. Adding a minute to the new last load date fixes the issue and the risk of losing data in the process is minimal.

Third, the declaration and assignment of the new last load data is located at the beginning of the script and not after the extraction and transformations to the stage schema. The reason for this is that the script takes longer time (minutes) to run because it is running in the cloud and not locally. Therefore, setting the new last load date after the extraction and transformation process could potentially result in data loss. The lost data would be data inserted in the source database right before the new last load date was assigned.

4.3. IoT

Adrian, Khaled, Morten

The IoT application, built in C, was responsible for gathering data and sending the data to the gateway application located on the business tier using the LoRaWAN network, such that the data can be saved in the SQL Server database. In addition to the production code, the application also consists of a project used for implementing Google tests in order to guarantee the functionality of certain components in the system. For mocking, a dependency on the Fake Function Framework was introduced.

The application starts in the main.c file shown below in figure 52 with the creation of queues, event groups, tasks, mutexes as well as initialisation of drivers. Queues, event groups and message buffers are passed to different components as required through dependency injection. Once completed, tasks are started with the vTaskStartScheduler function and system control is passed on to FreeRTOS.

```
int main(void) {
    stdio_initialise(ser_USART0);

    _createQueues();
    _initDrivers();
    _createEventGroups();
    _createTasks();
    _createMutexes();
    configuration_create(_mutex);

    puts("Starting...");
    vTaskStartScheduler();
}

static void _initDrivers(void) {
    puts("Initializing drivers...");
    hih8120_initialise();
    mh_z19_initialise(ser_USART3);
    rc_servo_initialise();
    sen14262_initialise();
    lora_driver_initialise(ser_USART1, _messageBuffer);
}

static void _createTasks(void) {
    farmerama_create(_senderQueue, _humidityQueue, _temperatureQueue,
    humidityTemperatureTask_create(_humidityQueue, _temperatureQueue,
    co2Task_create(_co2Queue, _actEventGroup, _doneEventGroup);
    soundTask_create(_soundQueue, _actEventGroup, _doneEventGroup);
    servoTask_create(_servoQueue);
    senderTask_create(_senderQueue);
    receiverTask_create(_messageBuffer);
}
```

Figure 52 - The system's starting point in main.c

The following code snippet is taken from Farmerama.c and shows the function which the task runs during every iteration. As seen in the figure, the task starts by setting 2 bits in the _actEventGroup which will cause the HumidityTemperatureTask to start measuring. When the bits have been set, the Farmerama task proceeds to wait for all tasks responsible for measuring data to have signified that they have completed their tasks. The measurements are received from queues and the values are sent to a separate function in order to be parsed into a lora_driver_payload_t type. Its length is checked to ensure a valid payload has been created and the payload is then sent to the SenderTask through a queue. Each measurement is also sent to the ServoTask through another queue. At last, the xTaskDelayUntil method is called to put the task into a Blocked state for 5 minutes. xTaskDelayUntil is preferred over vTaskDelay to guarantee exact waiting times versus relative waiting times.

```
void farmerama_runTask(void) {
    EventBits_t uxBits = xEventGroupSetBits(_actEventGroup, BIT_HUMIDITY_ACT | BIT_TEMPERATURE_ACT);

    if ((uxBits & (BIT_HUMIDITY_ACT | BIT_TEMPERATURE_ACT))
                == (BIT_HUMIDITY_ACT | BIT_TEMPERATURE_ACT)){
        _errorState = true;
    }

    EventBits_t waitBits = xEventGroupWaitBits(_doneEventGroup,
                        BIT_HUMIDITY_DONE | BIT_TEMPERATURE_DONE | BIT_CO2_DONE | BIT_SOUND_DONE,
                        pdTRUE,
                        pdTRUE,
                        pdMS_TO_TICKS(TASK_INTERVAL)
    );

    if ((waitBits & (BIT_HUMIDITY_DONE | BIT_TEMPERATURE_DONE | BIT_CO2_DONE | BIT_SOUND_DONE))
                != (BIT_HUMIDITY_DONE | BIT_TEMPERATURE_DONE | BIT_CO2_DONE | BIT_SOUND_DONE)) {
        _errorState = true;
    }

    uint16_t humidity;
    int16_t temperature;
    uint16_t ppm;
    uint16_t sound;

    if (xQueueReceive(_humidityQueue, &humidity, pdMS_TO_TICKS(10000)) != pdTRUE){
        humidity = CONFIG_INVALID_HUMIDITY_VALUE;}
    if (xQueueReceive(_temperatureQueue, &temperature, pdMS_TO_TICKS(10000)) != pdTRUE){
        temperature = CONFIG_INVALID_TEMPERATURE_VALUE;}
    if (xQueueReceive(_co2Queue, &ppm, pdMS_TO_TICKS(10000)) != pdTRUE){
        ppm = CONFIG_INVALID_CO2_VALUE;}
    if (xQueueReceive(_soundQueue, &sound, pdMS_TO_TICKS(10000)) != pdTRUE){
        sound = CONFIG_INVALID_SOUND_VALUE;}

    uplinkMessageBuilder_setHumidityData(humidity);
    uplinkMessageBuilder_setTemperatureData(temperature);
    uplinkMessageBuilder_setCO2Data(ppm);
    uplinkMessageBuilder_setSoundData(sound);

    if (true == _errorState){ uplinkMessageBuilder_setSystemErrorState(); }

    lora_driver_payload_t message = uplinkMessageBuilder_buildUplinkMessage(PORT);
    if (message.len > 0) { xQueueSendToBack(_senderQueue, &message, pdMS_TO_TICKS(10000));}

    if (false == _errorState){
        xQueueSendToBack(_servoQueue, &humidity, pdMS_TO_TICKS(10000));
        xQueueSendToBack(_servoQueue, &temperature, pdMS_TO_TICKS(10000));
        xQueueSendToBack(_servoQueue, &ppm, pdMS_TO_TICKS(10000));
        xQueueSendToBack(_servoQueue, &sound, pdMS_TO_TICKS(10000));
    }

    _errorState = false;
    TickType_t lastWakeTime = xTaskGetTickCount();
    xTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(TASK_INTERVAL));
}
```

Figure 53 - The runTask function in Farmerama.c

The HumidityTemperatureTask is responsible for measuring humidity and temperature. Figure 54 shows the task's main function which is run at every iteration. It starts by waiting for its "act bits" to be set by the Farmerama task. Once set, the task wakes up the HIH8120 driver and is put on a 100ms delay to give time for the driver to wake up. It then calls the measure() method on the HIH8120 driver and waits for an additional 50ms to give time for the driver to perform the measurements. The measured humidity and temperature are then retrieved, set and sent to the FarmeramaTask using queues.

In case the driver fails to wake up or perform a measurement, the measurements are defaulted to designated invalid values such that they can later be flagged as invalid measurements before being sent through the LoRaWAN network.

```
void humidityTemperatureTask_runTask() {
    xEventGroupWaitBits(_actEventGroup,
                        BIT_HUMIDITY_ACT | BIT_TEMPERATURE_ACT,
                        pdTRUE,
                        pdFALSE,
                        portMAX_DELAY
    );

    if (hih8120_wakeup() == HIH8120_OK) {
        vTaskDelay(pdMS_TO_TICKS(100));

        if (hih8120_measure() == HIH8120_OK) {
            vTaskDelay(pdMS_TO_TICKS(50));
            _latestHumidity = hih8120_getHumidityPercent_x10();
            _latestTemperature = hih8120_getTemperature_x10();
        } else {
            _latestHumidity = CONFIG_INVALID_HUMIDITY_VALUE;
            _latestTemperature = CONFIG_INVALID_TEMPERATURE_VALUE;
        }
    } else {
        _latestHumidity = CONFIG_INVALID_HUMIDITY_VALUE;
        _latestTemperature = CONFIG_INVALID_TEMPERATURE_VALUE;
    }

    xQueueSendToBack(_humidityQueue, &_latestHumidity, portMAX_DELAY);
    xQueueSendToBack(_temperatureQueue, &_latestTemperature, portMAX_DELAY);
    xEventGroupSetBits(_doneEventGroup, BIT_HUMIDITY_DONE | BIT_TEMPERATURE_DONE);
}
```

Figure 54 - The runTask function in HumidityTemperatureTask.c

Figure 55 shows the class responsible for constructing a lora_driver_payload_t payload as previously mentioned. Here, the data is passed in through different setter functions and stored in the class as static variables. Each function is also responsible for ensuring the data's validity and will flag the specific bit representing the given data as invalid (0) or valid (1) depending on the data itself. Once the buildUplinkMessage function is called, the payload is populated with the values stored in the static class variables.

```
lora_driver_payload_t uplinkMessageBuilder_buildUplinkMessage(uint8_t port) {
    lora_driver_payload_t payload;

    payload.portNo = port;
    payload.len = 9;

    payload.bytes[0] = _humidity >> 8;
    payload.bytes[1] = _humidity & 0xFF;
    payload.bytes[2] = _temperature >> 8;
    payload.bytes[3] = _temperature & 0xFF;
    payload.bytes[4] = _ppm >> 8;
    payload.bytes[5] = _ppm & 0xFF;
    payload.bytes[6] = _sound >> 8;
    payload.bytes[7] = _sound & 0xFF;
    payload.bytes[8] = _validationBits;

    return payload;
}

void uplinkMessageBuilder_setHumidityData(uint16_t data) {
    _humidity = data;

    if (data == CONFIG_INVALID_HUMIDITY_VALUE) {
        _validationBits |= 0 << 3;
    } else {
        _validationBits |= 1 << 3;
    }
}
```

Figure 55 - Extract of the UplinkMessageBuilder.c class

The Sender task is responsible for everything related to LoRaWAN as well as the LoRa driver, and Figure 56 shows the task's initTask and its runTask functions. Upon initialisation, the task will attempt to establish a connection with the LoRaWAN network and once started, it will wait for a lora_driver_payload_t payload from the _senderQueue. When a payload is read, the payload is sent to the gateway application over the LoRaWAN network using the lora_driver_sendUploadMessage function.

```
void senderTask_initTask(void* params) {
    lora_driver_resetRn2483(1);
    vTaskDelay(2);
    lora_driver_resetRn2483(0);
    vTaskDelay(150);
    lora_driver_flushBuffers();

    _connectToLoRaWAN();
}

void senderTask_runTask() {
    lora_driver_payload_t uplinkPayload;
    xQueueReceive(_senderQueue, &uplinkPayload, portMAX_DELAY);
    lora_driver_sendUploadMessage(false, &uplinkPayload);
}
```

Figure 56 - The SenderTask.c class' initTask and runTask functions

5. Test

In order to ensure that the final product met the Product Owner's expectations, the system was tested all throughout the development, guaranteeing the functionality of each user story. The tests consisted of various system- and API tests that utilised the black-box testing approach, unit tests on specific components that were deemed crucial as well as tests on the data warehouse.

5.1. Test cases

George, Georgiana, Vlad, Morten, Stefania, Tabita

System tests were based on the use case descriptions, including all case scenarios as well as exception scenarios. A total of 86 system tests were performed and all tests were passed.

Scenario	Test scenario	Test steps	Test data	Expected result	Actual result	Pass / Fail
1	View the latest temperature measurement	1. Choose temperature as a measurement category	VIA	The system displays the latest temperature measurement	As expected	Pass
2	View the latest humidity measurement	1. Choose humidity as a measurement category	VIA	The system displays the latest humidity measurement	As expected	Pass
3	View the latest CO2 measurement	1. Choose CO2 as a measurement category	VIA	The system displays the latest CO2 measurement	As expected	Pass
4	View the latest sound measurement	1. Choose sound as a measurement category	VIA	The system displays the latest sound measurement	As expected	Pass
5	View the historical temperature measurements	1. Choose temperature as a measurement category	VIA 20-05-2022	The system displays the historical temperature measurements	As expected	Pass

Figure 57 - Extract of the "View data analysis" black-box test cases

For a full list of test cases performed using the black-box testing approach, see Appendix G.

5.2. Unit tests

Stefania

AND

Validator tests

It has to be noted that unit tests were not the main emphasis for the Android team as Black Box Testing and Espresso tests were more used, due to the lack of the business logic. Unit tests were performed on the TypeConverters, in this case “Converters” class. One method converts a string to local date and the other one from local date to string.

```
25  public class DateConverterTest {
26      @Test
27      public void fromStringToLocalDate() {
28          assertEquals(LocalDate.of( year: 2022, month: 5, dayOfMonth: 31),
29                  Converters.fromString( changedOn: "2022-05-31"));
30          assertNotEquals( unexpected: null, Converters.fromString( changedOn: "2022-05-31"));
31          assertNotEquals( unexpected: "31-05-2022", Converters.fromString( changedOn: "2022-05-31"));
32          assertNotEquals( unexpected: "Yes", Converters.fromString( changedOn: "2011-08-21"));
33      }
34
35      @Test
36      public void fromLocalDateToString() {
37          assertEquals( expected: "2022-07-31", Converters.fromLocalDateTime(
38                  LocalDate.of( year: 2022, month: 7, dayOfMonth: 31)));
39          assertNotEquals( unexpected: "31-05-2022", Converters.fromLocalDateTime(
40                  LocalDate.of( year: 2022, month: 5, dayOfMonth: 31)));
41          assertNotEquals( unexpected: null, Converters.fromLocalDateTime(
42                  LocalDate.of( year: 2022, month: 4, dayOfMonth: 7)));
43          assertNotEquals( unexpected: "Hello", Converters.fromLocalDateTime(
44                  LocalDate.of( year: 2015,  month: 5,  dayOfMonth: 9)));
45      }

Tests passed: 2 of 2 tests – 24 ms
```

Figure 58 - Unit tests on the DateConverter class

Data Service

Unit testing has been conducted to guarantee that the developed data service application meets the requirements of the product owner. The system's most crucial and critical components have been chosen for testing.

Because of its Java nature and the developers' prior familiarity, JUnit was chosen as the unit testing framework. This framework simplifies the creation and execution of automated tests.

The AAA pattern was utilised for the unit tests (figure 59). This structure makes it easier to read and understand unit tests. Each test has three sections: Arrange, Act, and Assert. The Arrange section provides code for initialization and setup, Act contains the test action, and Assert compares the expected and actual results.

```
@Test
void validateThresholdTemperatureBothOutOfBoundaries() {
    //Arrange
    Threshold threshold = createThreshold( min: 10,  max: 40, ThresholdType.TEMPERATURE);
    Exception exception = null;

    //Act
    try {
        validator.validateThreshold(threshold);
    } catch (Exception e) {
        exception = e;
    }

    //Assert
    assertNotNull(exception);
}
```

Figure 59 - AAA pattern

Figure 60 illustrates the successful results of unit test classes after they have been executed.

These tests are also being run in a continuous delivery pipeline before the deployment of new versions to the production. This guarantees that the version that is being deployed is accurate and behaves as planned. A new version is not deployed if a test fails in the pipeline.

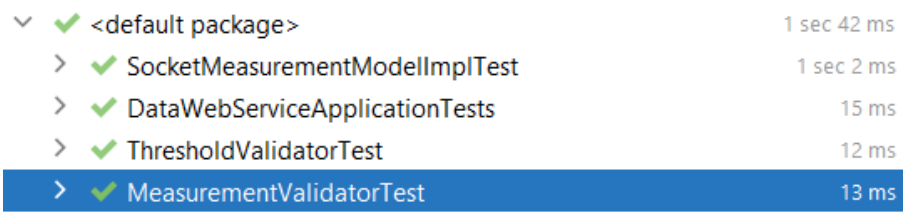


Figure 60 - Successful unit tests

The function saveSocketData() of the class SocketMeasurementImpl is responsible for parsing and saving incoming data to the database. Three separate dependencies are present in this class. The dependencies have been mocked using the Mockito framework to fully isolate the class's responsibilities.

Mockito is a prominent open source mocking object framework for software testing. Developing tests for classes with external dependencies is much easier using Mockito. Mock objects are fake implementations of interfaces or classes. It enables the output of certain method calls to be customised.

The faked dependencies' behaviour is defined and dummy data is produced in the saveSocketData() test (figure 61). After calling the method, the predicted and actual outcomes are compared to evaluate the test's result.

```
@Test
public void saveSocketData() throws Exception {
    //Arrange
    when(measurementDao.create(any())).then(i -> new AsyncResult<>(i.getArguments()[0]));
    when(areaDao.readByHardwareId(anyString())).then(i -> new AsyncResult<>(new Area( id: 1,
            new Barn( name: "Barn"), name: "Area Name", description: "Description", numberOfPigs: 100, i.getArgument( i: 0))));

    when(measurementValidator.isCo2ValueValid(anyInt())).thenReturn(true);
    when(measurementValidator.isHumidityValueValid(anyDouble())).thenReturn(true);
    when(measurementValidator.isTemperatureValueValid(anyDouble())).thenReturn(true);

    model = new SocketMeasurementServiceImpl(measurementDao, areaDao, measurementValidator, thresholdDao);

    SocketData data = new SocketData( cmd: "rx", EUI: "0004A30B002528D3",
            ts: "1654003641848", data: "02C100FD022B00320E");
    LocalDateTime expectedDateTime = LocalDateTime.of( year: 2022, month: 5, dayOfMonth: 31,
            hour: 15, minute: 27, second: 21, nanoOfSecond: 848000000);
    Measurement result;

    //Act
    result = model.saveSocketData(data);

    //Assert
    assertEquals( expected: 555, result.getCo2());
    assertEquals( expected: 70.5, result.getHumidity());
    assertEquals( expected: 25.3, result.getTemperature());
    assertEquals( expected: -1000, result.getSound());
    assertEquals(expectedDateTime, result.getMeasuredDate());
}
```

Figure 61 - saveSocketData() test

5.3. API tests

Adriana, Agostina, Luis, Tomas

API tests were performed in order to ensure that the endpoints created by the DAI team were producing correct data and appropriate responses in case of different invalid scenarios. For each endpoint, a test table was created in order to document all the possible scenarios and their response status. The following table represents an extract of the Temperatures Controller test table.

For a full list of API test cases, please refer to Appendix H.

Endpoint	Request	Result	Fail/Pass

GET '/areas/{id}/temperatures'	Latest temperatures valid Id:
	id = 9
	Latest = true
		HTTP 200
		{
		"value": 22.3,
		"areaId": 9,
		"measurementId": 28046,
		"measuredDate": "2022-05-30T13:49:10.453"
		}
			Pass

	Temperatures valid Id: Id = 9 Latest = false	HTTP 200 { "value": 22.3, "areaId": 9, "measurementId": 28037, "measuredDate": "2022-05-30T12:33:14.083" }, { "value": 22.6, "areaId": 9, "measurementId": 28038, "measuredDate": "2022-05-30T12:39:03.317" }, { "value": 22.7, "areaId": 9, "measurementId": 28039, "measuredDate": "2022-05-30T12:50:41.903" }, { "value": 22.4,	Pass

		"areaId": 9, "measurementId": 28040, "measuredDate": "2022-05-30T13:08:09.897" },	

	Temperatures valid date, valid id:
	Id = 9
	Date = '2022-05-26'
		{
		"value": 23.7,
		"areaId": 9,
		"measurementId": 27720,
		"measuredDate":
		"2022-05-26T09:34:07.823"
		},
		{
		"value": 24.3,
		"areaId": 9,
		"measurementId": 27721,
		"measuredDate":
		"2022-05-26T09:51:19.477"
		},
		{
		"value": 24.5,
		"areaId": 9,
		"measurementId": 27722,
		"measuredDate":
		"2022-05-26T09:57:09.233"
		},
		{
		"value": 24.7,
		"areaId": 9,
		"measurementId": 27723,
		"measuredDate":
		"2022-05-26T10:02:58.91"
		}
			Pass

	Temperatures invalid id: Id = 96 Date = '2022-05-26'	HTTP 400 Area with the given Id doesn't exist	Pass
	Temperatures invalid date Id = 9 Date = 'dfifd'	HTTP 400	Pass

Table 6 - Temperatures Controller Test Table

Additionally, Swagger UI was another tool used for testing the API as it provided a simulation of a possible API client, thus allowing for mocking API endpoint calls. This in turn allowed for

a simple and consistent way of testing the endpoints and fixing bugs before delivering them to the Android team for further development.

To exemplify, the following figure depicts a GET request for the latest measured temperature from a given area. As the implementation of the endpoint requires two optional parameters, latest and date, the response will depend on the data given. Therefore by setting the latest to true, the response body represents the latest measurement. The false value will retrieve all the values measured in the current day and by setting the date the response body returns a list with all the data measured that day.

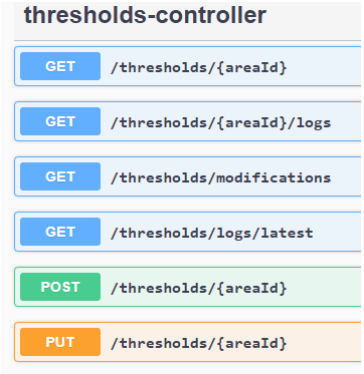


Figure 62 - Swagger UI for Thresholds Controller

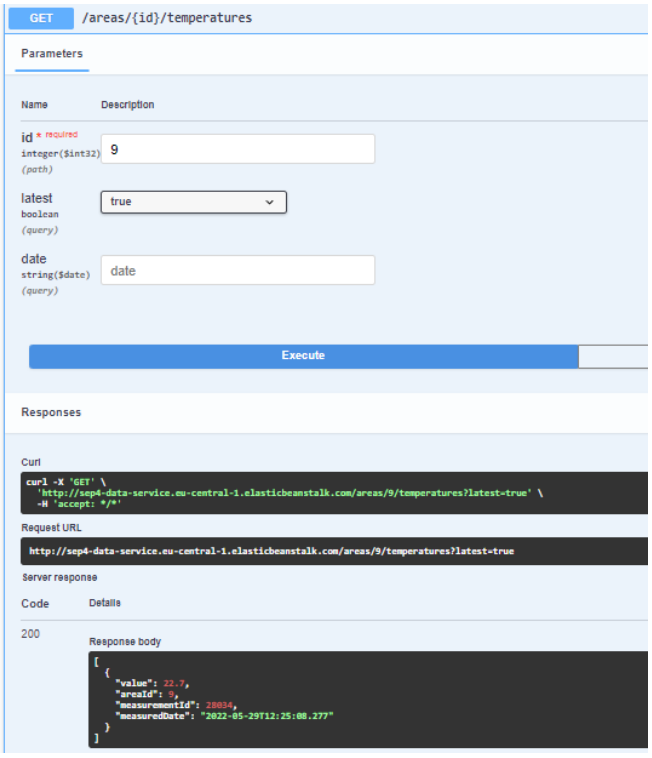


Figure 63 - Swagger UI used for GET/areas/{id}/temperatures

5.4. Google tests

Adrian, Khaled, Morten

Google tests on the IoT application were performed with the help of Visual Studio 2022. In order to perform the tests, the Fake Function Framework was added to mock external functions used in the unit under test. The development team decided to perform Google tests on the main task, the Farmerama task, as this was the task responsible for the flow of the application and as such it was the most complex and important task. Figure 64 shows the different Google tests that were performed. These ensured that the task implementation was properly done by verifying certain functions were called and that the arguments passed in were correct in all methods within the task.

As seen in the figure, a total of 18 Google tests were performed, and all tests passed. Refer to the Test folder in the IoT src project for the full tests.

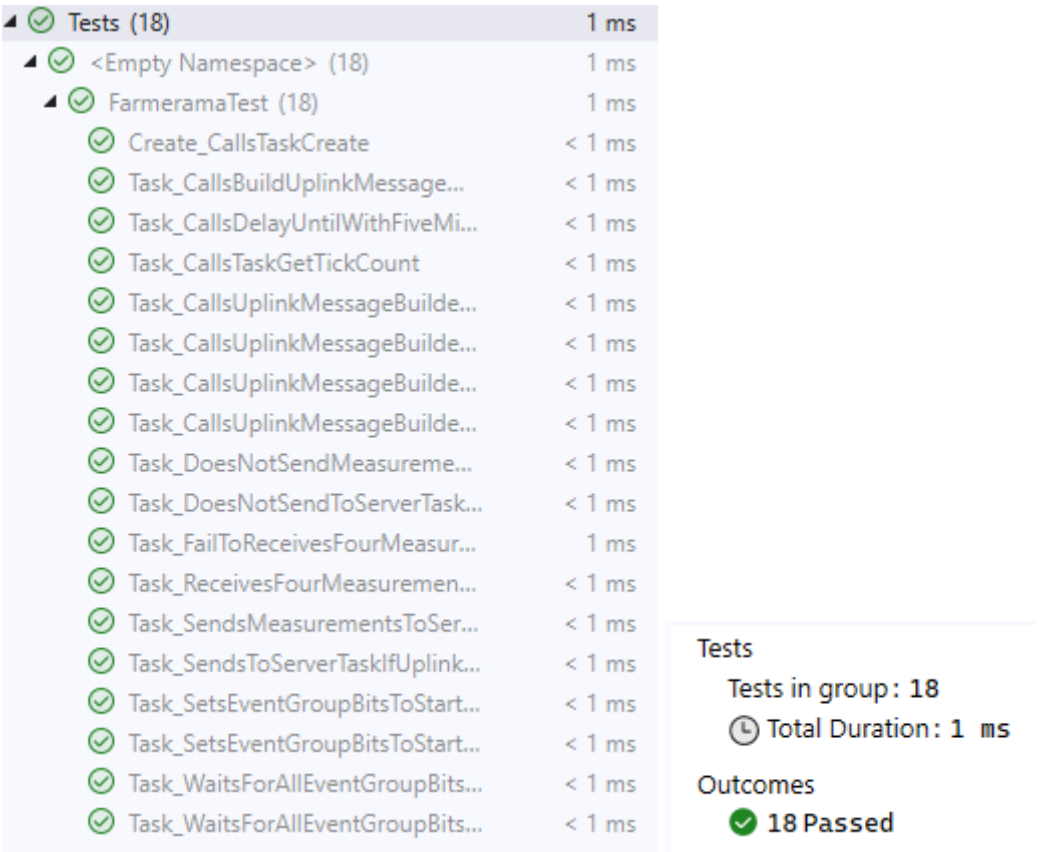


Figure 64 - Google tests of Farmerama.c

5.5. Data Warehouse tests

Adriana, Agostina, Luis, Tomas

The Data Warehouse was tested using Microsoft SQL Server Management Studio to ensure that the ETL is producing the right results when executed. This includes testing the initial load, the incremental load and the type 2 changes.

To test the data warehouse a test source database was created by mocking the source database. The source database contained dummy data for several months. Around ten percent of the data from the measurements in the source database was inserted into the test source database.

A test data warehouse was created using the scripts for the actual data warehouse. These scripts, modified to refer to the test source database and test data warehouse, were used to test the initial load and the type 2 changes and can be found in Appendix K - Data Warehouse.

Initial load

The initial load was tested by counting the amount of records in the test source database and the amount of records in the data warehouse after the initial load was performed. These tests were performed for the fact table and the dimension table DimArea. No other tests were performed because there are no calculations being made from the source database to the data warehouse.
These are the results of testing FactEnvironment and DimArea after the initial load (figure 65):

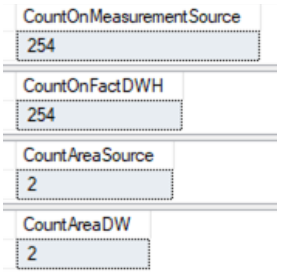


Figure 65 - FactEnvironment and DImArea after initial load

The counts match, therefore the test is passed.

Incremental Load and Type 2 Changes

To test the incremental load and type 2 changes several modifications were made in the test source database. In figure 66, the SQL code for the modifications is displayed.

```
-- New Area
INSERT INTO [dbo].[area] ([name], [barn_id])
VALUES ('Area3', 1)

-- Changed Area
UPDATE [dbo].[area]
SET name = 'Main Area'
WHERE area_id = 1

-- Deleted Area
DELETE FROM [dbo].[area]
WHERE area_id = 2

-- New Measurement (the measure_date is tomorrow, which could not be possible in a real-life setting,
-- but here it is just for testing purposes
-- In the etl the last load date is set to today, so it will include the record in the incremental load
INSERT INTO [dbo].[measurement] ([measured_date], [area_id], [temperature], [humidity], [co2], [sound])
VALUES (DATEADD(day, 1, GETDATE()), 1, 23.56, 40.48, 335, 74)
```

Figure 66 - Modifications in the test source database

The first three modifications are an insertion, an edition, and a deletion of an area. These are used to test the type 2 changes.

The last modification includes a record to the measurements table in the test source database. This modification is used to test the incremental load. In this case, the date of the record is being set to tomorrow, and even though it would not make sense to have such a record in the test source database, it makes the testing possible without changing parts of the ETL script tested (because after the initial load the last load date is set to the current date, and then the incremental load will used this date, therefore it is convenient to use have tomorrow's date in the test source database).

In summary, these modifications include a new area, an edited area, a deleted area and one more record in the measurement table. The results can be seen in figure 67.

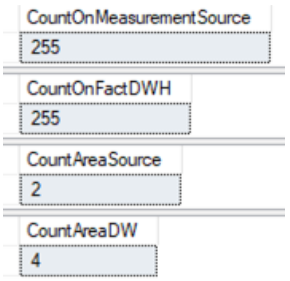


Figure 67 - FactEnvironment and DimArea after incremental load

The counts for measurement in the test source database and FactEnvironment match after the incremental load. Moreover, there is one more record than in the initial load test, which means that the new record in the measurement table has been included in the data warehouse.

The count of areas differs because of the type 2 changes. But this is expected, because a new area was included and an old area was modified. Therefore, these tests pass.

In figure 68, the resulting DimArea table can be seen.

	A_ID	AreaId	Name	ValidTo	ValidFrom
1	1	1	Area1	20220526	20220401
2	2	2	Area2	20220526	20220401
3	3	3	Area3	99990101	20220527
4	4	1	Main Area	99990101	20220527

Figure 68 - DimArea after the type 2 changes

The first area's name is modified to "Main Area", therefore ValidTo is set to yesterday to the old record and the new record is included with ValidTo referencing a date in the distant future and a ValifFrom set to the day the type 2 changes were performed (2022-05-27). Area2 was deleted in the test source database, therefore ValidTo is set to yesterday. Area3 is the new record, therefore ValidTo day refers to a data in the distant future and ValidFrom is set to the day the type 2 changes were performed.

5.6. Espresso

George

Several UI Tests were made for the Android app using the plugin Espresso. The tests were created in order to test if the UI is working correctly and if it can be navigable by creating different tasks. To create an Espresso Test the following dependencies were required:

androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
androidTestImplementation 'androidx.test.ext:junit:1.1.4-alpha06'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.0-alpha06'

The code behind the test shown in the images below shows the methods written for the IDE to compile the real-life steps that should be taken one-by-one in order to complete the tasks written. The given example below shows a LoginTest that takes an eventual user through these automated steps such as the written input into the desired fields and pressing on the different layout items such as the “LOGIN” button. In order to run these tests correctly, the Android project must be migrated into Android X, where most of the other dependencies might be interrupted.

```java
@LargeTest
@RunWith(AndroidJUnit4.class)
public class ViewAndEditAreasTest {

    @Rule
    public ActivityScenarioRule<MainActivity> mActivityScenarioRule =
            new ActivityScenarioRule<>(MainActivity.class);

    @Test
    public void viewAndEditAreasTest() {
        ViewInteraction materialButton = onView(allOf(withId(R.id.skip), withText("SKIP"),
                childAtPosition(childAtPosition(withClassName(is( value: "android.widget.FrameLayout")), position: 0), position: 2), isDisplayed()));
        materialButton.perform(click());

        ViewInteraction textInputEditText = onView(allOf(withId(R.id.LoginEmailAddress),
                childAtPosition(childAtPosition(withId(R.id.login_emailContainer), position: 0), position: 0)));
        textInputEditText.perform(scrollTo(), replaceText( stringToBeSet: "mariamgd@gmail.com"), closeSoftKeyboard());

        ViewInteraction textInputEditText2 = onView(allOf(withId(R.id.LoginPassword),
                childAtPosition(childAtPosition(withId(R.id.login_passwordContainer), position: 0), position: 0)));
        textInputEditText2.perform(scrollTo(), replaceText( stringToBeSet: "password"), closeSoftKeyboard());

        ViewInteraction materialButton2 = onView(allOf(withId(R.id.loginButton), withText("Login"),
                childAtPosition(childAtPosition(withClassName(is( value: "android.widget.ScrollView")), position: 0), position: 3)));
        materialButton2.perform(scrollTo(), click());

        ViewInteraction appCompatImageButton = onView(allOf(withContentDescription( text: "Open navigation drawer"),
                childAtPosition(allOf(withId(R.id.toolbar), childAtPosition(withId(R.id.constraintLayout), position: 0)), position: 1), isDisplayed()));
        appCompatImageButton.perform(click());

        ViewInteraction navigationMenuItemView = onView(allOf(withId(R.id.areasFragment),
                childAtPosition(allOf(withId(androidx.appcompat.R.id.activity_chooser_view_content), childAtPosition(withId(R.id.nav_view), position: 0)), position: 4), isDisplayed()));
        navigationMenuItemView.perform(click());

        ViewInteraction recyclerView = onView(allOf(withId(R.id.areas_recycleView),
                childAtPosition(withClassName(is( value: "androidx.constraintlayout.widget.ConstraintLayout")), position: 0)));

        ViewInteraction materialButton3 = onView(
                allOf(withId(R.id.addArea_createButton), withText("Save"),
                        childAtPosition(childAtPosition(withClassName(is( value: "android.widget.ScrollView")), position: 0), position: 6)));
        materialButton3.perform(scrollTo(), click());

        ViewInteraction imageButton = onView(allOf(withId(R.id.areas_fab), withParent(withParent(withId(R.id.fragment))), isDisplayed()));
        imageButton.check(matches(isDisplayed()));
    }

    private static Matcher<View> childAtPosition(
            final Matcher<View> parentMatcher, final int position) {

        return new TypeSafeMatcher<View>() {
            @Override
            public void describeTo(Description description) {
                description.appendText("Child at position " + position + " in parent ");
                parentMatcher.describeTo(description);
            }

            @Override
            public boolean matchesSafely(View view) {
                ViewParent parent = view.getParent();
                return parent instanceof ViewGroup && parentMatcher.matches(parent)
                        && view.equals(((ViewGroup) parent).getChildAt(position));
            }
        };
    }
}
```

Figure 69 - Espresso Tests

6. Results and Discussion

Adriana, Morten

The product backlog consisted of 50 functional- and an additional 6 non-functional requirements and was updated throughout the entire development process, with the development team putting emphasis on analysing, designing and implementing highly prioritised requirements first. Once critical requirements were considered complete and approved by the Product Owner, focus was shifted to requirements with high priority, leaving low prioritised requirements to be completed closer to the end of the development process.

During development, the use case “Manage devices” that was originally part of the system as well as its related requirements, was entirely removed due to the development team realising that its functionality and requirements were already technically included in other use cases. An additional change that occurred was the inclusion of a new use case, “Control window”, which was related to the Temperature and Window actors and deals with changing the window position.

Despite all obstacles, the development team managed to implement all 48 functional requirements proposed by Farmerama, and throughout each development iteration, requirements selected for that specific iteration were thoroughly tested using various system tests, taking the blackbox testing approach, and ultimately approved by the Product Owner. Unit tests and google tests were performed to a lesser extent and only on certain selected components that were deemed crucial to the functionality of the core system.

Ultimately, the final product was considered satisfactory by the Product Owner based on the stakeholders’ requirements and wishes.

7. Conclusions

Adriana

Farmerama's request prompted the development team to analyse the issues the farm was facing and devise a potential solution that could satisfy the owner's needs and obtain the stakeholders' approval. The solution offered was based on the fact that the farm currently is monitoring their livestock with human resources which caused issues with the pigs raised within the farm due to human error. The development team offered to make a system monitoring the animals' living area, allowing for automation and a quicker response time.

The system was developed with the help of three teams that focused on separate parts of the project that consisted of a total of 55 requirements covering 7 use cases. While all users can view the latest measurements, the rest of the features that the system offers require authentication. Therefore, a login system was implemented in order to provide access to authorised members and also to differentiate between actors and their allowed interactions with the system. As such, a regular Employee has access to the rest of the features of the system except for the ones covered by the "Manage accounts" use case, which are functionalities exclusively available to the Administrator.

As per the non-functional requirements of the system, the graphical user interface was developed as an Android application. The data displayed by the system is processed in a SQL server data warehouse and is measured by a hardware device with sensors for humidity, temperature, sound and CO_2. Moreover, the historical data stored and processed in the data warehouse is available to external users in the form of data visualisations designed in PowerBI. For the development of the project, Java and C were used for the main parts of the system, with the source database and data warehouse using SQL Server and SQLite for the local Android database.

Implementation and testing were carried out simultaneously in order to ensure that any issues were caught and fixed in no time. Each part of the project was tested separately by the respective subteam, such as IoT using Google tests, DAI testing the API's endpoints and the Android team performing Black Box tests in order to test the whole system's functionality.

Conclusively, the delivered project satisfies all the stakeholders' wishes. The system is functional and is able to complete all the customer's requirements.

8. Project Future

Stefania

Currently, the end product focuses on transforming the pigs' experience into real time data. In this way, the administrators and the employees can interact with it and take action to improve the quality of life of the pigs. Thus, the main focus was to deliver the best features for the users in the given amount of time and provide a satisfying user experience.

One way to improve the system in the future is to support multiple barns. In the present, the data is shown only for the Farmerama barn, which can be changed to comprise more barns and have control overall. Barns could be added and removed at the choice of the administrators, providing more flexibility to the system.

Regarding IoT devices, monitoring the amount of light in the barn can be a good idea. In this way, the employees can be aware of how much light, or lack of light, is in the building and act accordingly. Moreover, having a PIR sensor that can detect the heat energy and movement can automate the system even more.

On the other hand, security can be implemented as the API is hosted on an insecure server. Implementation of Secure Socket Layer can be a better way of guaranteeing security. Consequently, the system will not be as vulnerable to attacks that can compromise the JSON data and the system itself. All the current data that is sent to the server can be protected from malicious interference or important data leaks.

As for the android application, more visualisations can be implemented to depict statistics. A future feature can be an announcements page where the administrators can post statistics or urgent matters to the employees. Furthermore, another feature can be to have a chat system where the employees and administrators can interact with each other.

9. Sources of Information

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd ed. Addison Wesley Professional.

Kimball, R., & Ross, M. (2002). The Data Warehouse Toolkit. Wiley.

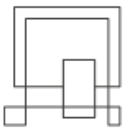
Millington, S. (2021, May 18). A solid guide to solid principles. Baeldung. Retrieved May 25, 2022, from https://www.baeldung.com/solid-principles

Russo, M. (2017, May 31). How to design beautiful dashboards in power bi – the cure: 15 rules for everyone #powerbi. SQLBI. Retrieved May 23, 2022, from https://www.sqlbi.com/blog/marco/2017/05/31/how-to-design-beautiful-dashboards-in-power-bi-the-cure-15-rules-for-everyone-powerbi/

The 15 rules to design a perfect dashboard - SQLBI. (n.d.). Retrieved May 31, 2022, from https://www.sqlbi.com/wp-content/uploads/videotrainings/dashboarddesign/rules-A3.pdf

Pipino et al (2002) Data quality assessment. Communications of the ACM, 45(4), 211

Bring ideas to life
VIA University College



Process Report - SEP4

Group 2

Group Members:

Adrian - Gabriel Vaitis (304486)

Adriana Grecea (304149)

Agostina Mezzabotta (305170)

George Andronache Eduard (304460)

Georgiana Ion (304216)

Ioan Vlad Dirlea (304182)

Khaled Hammoun (275241)

Luis Fernandez Ponton (304272)

Morten Frederik Hansen (304668)

Stefania Tomuta (304173)

Tabita Roxana Varlan (304181)

Tomas Ondrejka (304150)

Supervisors:

Ib Havn

Kasper Knop Rasmussen

Knud Erik Rasmussen

45441 characters

Software Engineering

4th Semester

02/06/2022

VIA University College

Table of Contents

1. Introduction

The aim of this report is to describe the process carried out to develop our Fourth Semester Project. The group included 12 people divided in 3 teams: Internet of Things, Data Engineering and Interactive Media.

The methodologies used were SCRUM to manage the workflow and Agile Unified Process to guide the specific product development.

Through our fourth semester, the following milestones have been met:

Feb 20	Project Idea and Group contract
Mar 06	Inception handin delivered, including the Project Description
Mar 20	Group integrants choose a specialisation (Internet of Things, Data Engineering and Interactive Media)
Apr 18	Interfacing contract, establishing the responsibilities for each team
May 08	Elaboration hand-in. Presentation of a proof of concept of the project
May 18	Project weeks. Working in the Construction and Transition phases of the project
Jun 02	Final handin

Table 1 - Milestones

The following sections will describe in detail the different stages of the process, focusing on group formation and characteristics, Scrum documentation, personal reflections and a brief conclusion.

2. Group Description

This group was formed just before the 4th semester started, as everybody knew with whom they wanted to work and also which specialisation to choose, therefore the process of creating and splitting the group worked smoothly and quickly. All of us had an idea about how the group should look: it was required to have a 9 to 12 group. The team started with 10 people, afterward Adrian and George were added.

This section describes the group based on cultural backgrounds. Six countries of the world are represented by one of our group members. These countries include Denmark, Romania, Bulgaria, Slovakia, Spain and Argentina. The following chart is based on Hofstede Insights Compare Countries tool.

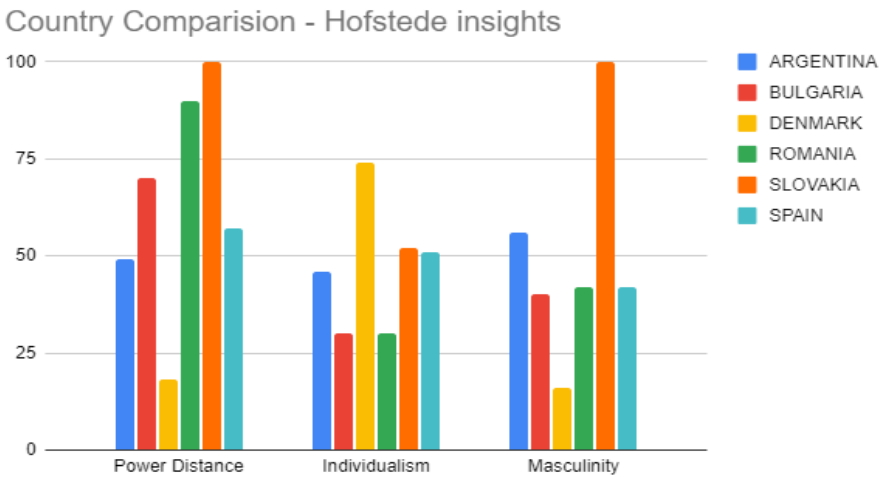


Figure 1 - Country comparison

It goes without saying that we had a heterogeneous group. Considering power distance, Slovakia and Denmark are at different extremes. Denmark shows high levels of individualism in contrast to the rest of the nationalities. Slovakia leads the board again when it comes to Masculinity. Understanding our backgrounds prevents issues due to different ways of work and communication.

Complex group formations can be extremely rewarding when well structured. We found Scrum as a good ally to achieve our goals. Organising the work flow and the way we communicate was fundamental to complete our project. More details about the project and the methodology we used will be described in the next sections.

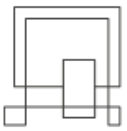
3. Project Initiation

This semester the variety of ideas regarding the project topic was reduced and constrained by the specific conditions that were required to be covered. Therefore the team was satisfied by the initial project idea they proposed to the supervisors, as it seemed to be original in terms of previous projects of the same kind. However, after consulting with the supervisors it appeared to be that the baby monitoring system that the team wanted to develop was too crucial and required a different equipment than the one the university was going to provide. Therefore, after an additional brainstorming session and another meeting with the supervisors, the team decided to develop a system monitoring the environment and therefore indirectly the wellbeing of pigs living at the Farmerama farm.

As it came to be, there were several elements that comprised the functionality of the developed system. Among them, there was handling the hardware device by measuring and sending data as well as receiving it from the data server, managing a data server and devising a data warehouse to process and store the historical data, and displaying the refined information in an Android application that included multiple features such as offline usage. Despite the vast differences that came both with the project requirements and with the growth of the development team, this project was reachable, as there was an abundant amount of information online regarding the domain model.

The following tools were used throughout the development of the project:

- Git and Github was used to store and merge the different codebases
- Amazon Web Services was used to host the server in the cloud
- Google Drive was used for storing documentation like Project- and Process report as well as everything related to Scrum
- Astah Professional was used to create diagrams used for documentation of Analysis and Design
- Figma was used for mocking graphic user interfaces
- PowerBI was used to create reports based on the data stored in the data warehouse
- Android Studio, Intellij IDEA, Microchip Studio and Visual Studio IDEs were used for writing Java and C code
- Datagrip and SQL Server Management Studio were used to created database



queries and tables

- Discord was used as the main tool of communication between group members, both through text- and voice chat

4. Project Description

This section describes the process of our project description during the Inception phase of the AUP. We worked in our Project Description during the weeks 08 and 09 and delivered the Project Description as part of the Inception hand-in.

The Project Description gave us the opportunity to discuss a context for our SEP, as well as a better understanding of the problem and domain we were working on. In particular, the most meaningful sections were: Background Description, Problem Statement and Definition of Purpose.

Since there is not a real customer, we used the Background Description to read and investigate the field, trying to give the project a proper context. This process allowed us to understand the field, project's scope and delimitations.

The Problem Statement helped us to define the problem and which aspects should our project bring solutions to.

The definition of Purpose finally narrowed the goal of this project:

The purpose of this project is to help Farmerama effectively measure the environmental conditions in terms of noise, temperature, humidity and CO_2 at the barn with the aim of giving detailed analyses that allow Farmerama to make the right decisions in order to improve the pigs' wellbeing.

After understanding the project goal, we defined the methodology and planned the time schedule as well as identified possible risks that could affect our progress. To see the complete Project Description see Appendix A.

5. Project Execution

This part of the report is meant to reflect our process during the project execution. This semester, the team worked with the Scrum methodology together with Agile Unified Process, due to the fact that the members had a lot of experience with these methodologies from previous semesters. Using Scrum and AUP facilitated the process of work among the 12 members, so that everybody succeeded in working effectively.

5.1. Agile Unified Process

Together with the Scrum methodology, the team also used Agile Unified Process. It was used to split the project execution in the following phases: inception, elaboration, construction and transition.

5.2. Inception

During the inception phase, the team focused on analysing the proposed system and introducing use cases, identifying the actors of the system and the functionalities each actor would have access to. Based on this, user stories were created and assigned to use cases and actors. During this phase, the development team also began the creation of the description and flow of the core use cases of the system. During this phase, initial user stories were created and specialisations were chosen.

5.3. Scrum

As it was mentioned before, Scrum was the methodology used to manage the workflow through the AUP phases.

Before starting sprints, Stefania was assigned to be the Product Owner and Khaled assigned the role of Scrum Master.

Role	Name
Product Owner	Stefania Tomuta
Scrum Master	Khaled Hammoun
Scrum Team	Adrian-Gabriel Vaitis

	Adriana Grecea Agostina Mezzabotta George Andronache Eduard Georgiana Ion Ioan Vlad Dirlea Luis Fernandez Ponton Morten Frederik Hansen Tabita Roxana Varlan Tomas Ondrejka

Table 2 - Team members

The entire process was organised into six sprints of three days each. Every sprint consisted of 4 different meetings:

Sprint planning meetings: these meetings were the longest, due to their importance. We established the Sprint goal and created a Sprint Backlog during these meetings. The Sprint Backlog's user stories were divided into tasks. We all agreed on an estimated time to complete the task. Following that, preliminary tasks were assigned. During the sprint, whenever one person completed a task, they could take another one. The next table shows the sprint planning meeting for Sprint 4.

Date-time	20 May 2022
Sprint Days	20/05/2022, 23/05/2022, 24/05/2022
Attendees	All
Comments	-
Sprint goal	User stories: 7, 9, 15, 16, 17, 18, 19, 20, 21, 22, 26, 31, 32, 34, 35, 36, 43, 44

Table 3 - Sprint Planning Meeting

Appendix A - Scrum planning meetings contains the details of all our Sprint Planning meetings.

Daily scrum meetings: The second and third days of each sprint began with a daily scrum meeting, during which all teams provided a brief update on what had been accomplished the previous day and what needed to be completed before the next meeting.

A daily meeting lasted approximately 20-30 minutes, allowing teams to return to work on their tasks.The goal of the daily scrum meeting was to keep the team focused and track what was being worked on and by whom, ensuring that everyone was up to date. The following table shows an example of our daily scrum meetings for sprint 4.

Dates	Updates
Day 2	The Android team worked on implementing the list of employees, removing an area, updating a profile, setting the thresholds and creating the view for seeing all the exceeded measurements. The DAI Team worked on creating models for thresholds and threshold logs. Implementing endpoints and repositories to set minimum and maximum thresholds and to get a list of threshold logs. Areas can now be deleted, users can be edited. We also wrote some documentation in the project and process report. The IoT team discussed the format of the downlink payload for receiving data. Designed black box test scenarios and attempted to get Google Tests working
Day 3	The Android team managed to implement the view for the account, the logs for the exceeded threshold and the modifications of the thresholds. The IoT team finally managed to get the Google tests “working”. A meeting with Ib is required for them to be fully completed. The team also worked on receiving downlink messages through the LoRaWAN network and hooking up the RC servo to be rotated depending on the measured temperature and the received temperature threshold The DAI team worked on documentation and refactored the incremental load of the data warehouse.

Table 4 - Daily Scrum Meetings

Appendix B - Daily scrum meetings, contains the details of all daily scrum meetings.

Sprint review: Sprint reviews were done at the end of the Sprint, here each team would explain the achievements and show to the rest of the group.

Date	24/05/2022
Status	Incomplete
Attendees	Adrian, Adriana, Agostina, George, Vlad, Khaled, Luis, Morten, Stefania, Tabita, Tomas
What was done	User story 25 has been removed and merged with 15, 16 and 17. User stories: 45, 46, 47, 48 were added in the Product Backlog and chosen in the current sprint. User stories 49, 50, 51, 52 were added to the Product Backlog and completed. The Android team focused on the “Manage environment” and “Manage personal profile” use cases as well as user friendly implementation. The DAI team implemented endpoints for retrieving threshold data and threshold logs, edit user, delete area. Additionally the incremental load for the data warehouse was refactored and worked on sending data to the hardware device. The IoT team added functionality for the servo to open and close the window if a measured temperature exceeds the thresholds set. The device also accepts changes to thresholds from the data team. User stories 7, 9, 15, 16, 17, 18, 19, 20, 21, 22, 31, 34, 43, 45, 46, 47, 48, 49, 50, 51, 52 are considered complete

Table 5 - Sprint Review

Appendix C - Sprint review meetings, contains the details of all our sprint review meetings.

Sprint retrospective: Here, the team discussed what went well, and what could be improved. The purpose of this meeting was to make room for all team members to voice their opinions about the process of the overall work or of a specific task.

Date-time	24/05/2022 15:00
Attendees	Adrian, Adriana, Agostina, George, Vlad, Luis, Khaled, Morten, Stefania, Tabita, Tomas
What to start	Completing documentation, focus more on implementation, read

doing	documentation regarding android implementation
What to stop doing	Spend time on unnecessary things
What to continue doing	Schedule meetings with supervisors

Table 6 - Sprint Retrospective Meeting

Appendix D - Sprint retrospective meetings, contains the details of all our sprint retrospective meetings.

All meetings were held online in our Discord group in order to accommodate everyone in the group, and apart from the aforementioned Scrum ceremonies, the following Scrum artefacts were also created: the Product Backlog (see Appendix G), the Sprint Backlogs (see Appendix E) and an overall Burndown Chart (see Appendix F) as well as the definition of done from both the Scrum Master and the Product Owner's point of view (see Appendix A).

5.4. Elaboration

At the start of the Elaboration phase, the team was asked to deliver a Proof of Concept. Therefore, the team started by focusing on the design and implementation of the core functionality of the system which related to viewing the latest measured temperature. The hand-in of the PoC implied having a functional system that was sending data from an IoT device to the server to be stored in a database and eventually be displayed in the Android application as per request.

The elaboration phase is also where the team started using the Scrum methodology and started sprints. Additionally, the team members were split up into the three different sub-teams by choice in order to focus on specific areas of the system.

5.5. Construction

The Construction phase was the most challenging for the team, as each team encountered many obstacles related to the implementation of various different tasks. During this phase, the team started their full-time work on the project. With the majority of the critical

requirements already implemented, the main focus was to complete the few critical requirements that were left unfinished from the previous sprints as well as start on the features with less priority.

5.6. Transition

The last week of the project and therefore also the last sprint was allocated for the transition phase. Here, the main objective was to complete the implementation of the system and refactor already-existing code where needed. This also included double checking the functionality of the system as a whole and ensuring that everything functioned as expected. The team also spent a lot of time writing documentation during this phase, as this aspect had been somewhat neglected prior to the transition phase.

6. Personal Reflection

6.1. Adrian - Gabriel Vaitis

This semester was different for me as, after 3 semesters, I changed the team. I knew the members because they are my classmates, but I have never worked with them. It turned out that I was on the same page with them and we had a great relationship during the process of work.

Working with a big group was nice, as I could listen to multiple opinions and learn how others work and I could learn with and from them. We did not have many problems regarding communication and meetings as we scheduled them so that everybody could attend.

Talking about specialisation, I chose to be in the IoT team as I wanted to learn more about what I wish to do in the future after graduation and also because this is what I am going to do during my internship.

Working with the IoT team was a bit challenging for me as the guys already had some experience with this technology and I had to keep up with them. When I was in need, they were patient and helped me understand what was unclear to me. Regarding the actual work, me and the guys worked together, one of us was sharing the screen and we were watching the screen and discussing what and how to do in a specific situation. While I was the one sharing the screen, I was working with the guys on the class diagram, the sequence diagram

for the main task and also on implementing the sound and CO2 sensors. I also worked on implementing the servo task and also on google tests.

Also, we had some meetings with our IoT supervisor, Ib, who advised us any time and showed us the path we had to follow to succeed with our project.

To conclude my experience from this semester, I am grateful for having the opportunity to work with multiple technologies and I will for sure use my knowledge gained in this period in my future career.

6.2. Adriana Grecea

As expected this semester project was entirely different from the previous ones, both in terms of process and team members, and in terms of requirements. One of the biggest changes for me to adjust to was the fact that now the group consisted of twelve people instead of four and several adjustments were needed to be made to the group contract in order to cover new scenarios that may occur. Additionally, I found communication between and within the subteams to be chaotic as sometimes it would happen that certain aspects that different teams depended on to be changed without prior discussions.

Using Google Docs and Excel proved to be challenging when paired together with Scrum methodology and a large team. While overall the meetings went fine, I would have liked to focus more on the communication between team members as there were times when clashes of opinions impacted important aspects of the project. Contrary to the previous semester, this one I was only part of the development team, however, I often found myself organising our meetings and managing communication between teams.

Since I chose the DAI specialisation for this project, I had to get used to working closely with people other than my previous teammates, which in turn meant getting accustomed to a different working environment and schedule. I mainly chose this specialisation due to my interest in the data engineering field. While I was familiar with the programming language that we used, Java, the majority of technologies and frameworks that the team decided to work with were new to me. Nevertheless, in our team, we did our best to deliver a good end product and I enjoyed working with them.

As for my own contribution to the project, I mainly worked on tasks regarding the source database in communication with the Android and IoT teams. I implemented the SentMeasurement that was retrieving custom data from the database and sending it through the API to the Android team. Similarly, I also implemented functionality for Thresholds, retrieving Thresholds modifications, and Exceeding thresholds. I designed several required native SQL queries and implemented the functionality to send data to the hardware device via a downLink. Lastly, one of my main contributions was to communicate with the Android team and make sure that the data provided by the API was fit to their requirements.

The materials my team required were similar to the topics we approached during class so applying the new knowledge we acquired was approachable. Whenever an issue or dilemma regarding Data Warehouse came up, our supervisor, Knud was helpful and gave constructive suggestions either during class, when one of us would ask him or through our general team meetings with the supervisors.

Overall I am satisfied with the project that we managed to develop and deliver. Working together in a big group is never easy and I would like to thank all my teammates' efforts and dedication for helping make this project. This semester project was certainly a new and enriching experience.

6.3. Agostina Mezzabotta

This semester's project was unlike any other in a number of ways. The size of the group is, of course, the first aspect to mention. I was scared since I had some difficulties working with smaller groups in prior semesters, and it was quite stressful most of the time. It was a nice surprise to discover that everyone was clearly invested in the project.

I chose the Data Engineering specialisation since I am interested in the field. Working with my three groupmates, I found that they had a deeper understanding of the technologies and concepts we used, and I became a little bit demotivated because I couldn't seem to get simple things done. It was, however, a fantastic opportunity for me to learn from them and enhance my skills. After a few days, I felt more confident participating in pair coding, asking for help when needed, and contributing in any way that was possible. .

Two of my key contributions to the project were data analysis and PowerBi visualisations. Aside from that, I assisted in the overall testing of the API. In addition, I implemented endpoints, services, and daos for areas and temperature. Finally, I collaborated in the creation of conceptual models. I also actively contributed to scrum documentation, process documentation, and project report documentation.

During this semester, I believe that supervisions were more specific. Especially with Knud, who was in charge of the Data Engineering teams. He promptly responded to all of our emails and scheduled lengthy sessions with us, during which he answered all of our inquiries. This provided us with a clear picture of our progress and increased our confidence in the project's execution.

This project gave me many takeaways. I'm happy with the final product, my new technical skills, and the ability to experience this process with such an inspiring group of people.

6.4. George Eduard Andronache

As I expected, this semester project was a real challenge for me due to the lack of experience that I've had in the past. Gratefully I got back on track with the help of my team which guided me and helped me every time I got stuck with some tasks that needed to be done.

I choose to work in the AND team because I think it suits me the most, I always liked the idea of mobile development. I had an amazing team because the app in the end looks very good and it handles as it should. My teammates always helped me when I had troubles getting a task done and even if the development team had problems sometimes in communication, in the end a solution was found for everything.

The shock that the team almost tripled from the last SEP project was not really meaningful because the development team managed to part our work correctly so each of us knows exactly who and what is working on. Probably the issues that the development team, as a group, encountered were the method used to divide and split those tasks which was Google Sheets, kind of a primitive way to monitor the sprints. I know there are modern ways to monitor and manage tasks using software like Monday or Jira, but there were not any impediments that could have obstructed our work in any way.

Hours estimation for me was kind of a big deal because I didn't really knew how many hours it will take me to take a task done considering that I might have gotten stuck at some point and asking for help which left me with the need to add some extra hours, that means that the sprint was incomplete and my task was assigned again to me to continue it in the next sprint.

In reality I was feeling like I was dragging the team down because my lack of experience in coding from the previous semester due to some personal problems that were distracting me from my learning path, but my team helped me with the questions that I've had and in the end we've made the things working out either by solving the problems or by assigning me tasks that I can complete.

My main contributions in this project were providing the translations for the different regions (mainly the countries that my teammates come from), coming up with the app UI idea and colour rules that will suit the most this kind of app by creating a wireframe design and mock up in Figma, completing different tasks like MVVM for different user stories for example implementing "Delete Employee", performing Black-Box testing for different scenarios, creating Espresso testing for UI, writing documentation in the process and project report and creating a User Guide which can be found in the Appendix section.

Speaking about the project itself, it was an idea that excited me from the start because It was the first time where the development could have merged hardware with software, so a more practical project than usual.

A problem that I've had during the development process was working with Github because even if I have used it before in the previous projects in other semesters, for me it was always a bit harder for merging because the OS that I was using was different than my teammate's. I was recommended to use Github Desktop on MacOS which helped me to see the history and merging or reverting to commits easier by one of my AND team members which looked like the perfect solution for my problems.

In conclusion, the project overall was a nice experience where I could fill my gaps in knowledge, where I found how to work organised in a bigger and team and also where I found how to correctly implement the requirements that I was lacking experience with, all these things helping me to model a better Software Engineer and Android Developer and to use my skills to pursue my goals in my future career.

6.5. Georgiana Ion

This semester project was the most exciting one for me. I got to work with some friends and gained even more. It was my first time working in a group with girls, and I saw a significant difference because I felt more involved and listened to. I opted to switch classes before the start of the semester to be closer to my pals. I recall being worried about not getting along with my friends and compromising our friendship, but thanks to understanding and empathy, we had a great time together and had no arguments since we learned how to communicate and support each other.

With this project, I tried to push myself to the maximum in order to get the most out of it. I felt that I was not as involved in the first three semesters because I was "lowering the team", but the reality was that I was just as capable as my teammates, and this new team allowed me to see it. I'm pleased that I was able to learn more than I had previously, and that I was able to learn from my mistakes with the assistance of my teammates, for which I am grateful. I believe that, even if I outdid myself, I did not live up to my colleagues' expectations, and I regret that I was unable to offer more; yet, I did my best, and I surely improved.

I saw how confusing it may be to estimate numbers with twelve people at the same time. It's a procedure that might work better with a smaller group of individuals and effective communication. I believe we all agreed that we are moving too slowly when it comes to estimating, but it took us much too long to do something about it, which was separating the estimations per team, which sped things up a little. Also, utilising Excel for our Scrum process was a little perplexing at first, but we got the hang of it. If I were to do it again, I would place a greater emphasis on clearly dividing the tasks, as we occasionally mixed up the tasks.

Implementing validation for login and registering users were among *my contributions* to this project (Also offered some refactoring for registering to actually work). I created the drawer, which included the header and menu, as well as the app theme. In terms of measures, I created the functionality for obtaining sound measurements. I also implemented the personal account functionality and made the list of employees(recycler view). Aside from adding functionality, I helped refactor the code and improve the style in places like the app's singing-out feature and various layouts. I also worked on the Scrum documentation and performed multiple black-box tests for completed requirements. The last implemented thing

by me was the onboarding intro for the new users; however, I spent the majority of my time on the reports and class diagram in the end.

My conclusion after this semester project is that I should not feel worried about working with friends, because if we are mature and close enough, we can work simply together, and grow together. And that making mistakes is better than not doing much, since it brings a lot of growth. Being part of the Android team was the best decision I could make, since it helped me develop new skills in the direction that I am enjoying the most. I personally am really satisfied with this project, it turned out better than expected for me. I am very grateful for my colleagues, and I feel lucky that I got to have this experience with them. I also appreciate our supervisors, which helped us a lot in our process.

6.6. Ioan Vlad Dirlea

Being aware how difficult it can be to form a proper team for a semester project, I am grateful we decided to work together as a group from the last semester. Being unfamiliar with each others' workflow, we ended up with a more rigorous group contract compared to the former semesters, in which I had the same team. Despite it not being fully followed, I am thankful we didn't end up with a situation that could put our development on the line.

The development process occurred on Discord for the most of the time, which I think was the best given the fact we could easily share our screen and switch between voice channels to have separate conversations. A thing we could have avoided was to keep our scrum artefacts on drive files as it became hard to maintain with 12 team members. Also switching from story points to remaining hours was slightly off and I feel we spent a bit too much time getting used to how to estimate and how to calculate our burndown chart.

Feeling more comfortable with Frontend Development I chose to be part of the Android team. The resources from the class were easy to follow and implement in our project. I think it's unfortunate we couldn't go deeper into the studies as I found myself stuck for days on the internet looking how to implement a notification system, which can be seen in most of the applications on android

My contribution to this project consisted in designing and creating new views, linking them to the view models and setting up the connection between the repositories and the server API

in order to retrieve the data. I have also implemented adapters for the recycler views and the adapter for the Measurement API. My tasks covered as well the notification and settings requirements, as well as blackbox testing through the entire development. A large amount of time was also dedicated to fixing bugs and merging conflicts. Regarding documentation, I've contributed to both Project and Process Reports as well as setting up appendices and designing class diagrams.

Getting accustomed to a new team was a real challenge given the fact I had the same team for three semesters. Not being aware of the others' competences and motivation made it hard to build a level of trust and a convenient workflow for everyone but I am glad we found a satisfying way to work together.

As for supervision, I feel we received positive feedback in a decent amount of time. The assigned teachers were available when needed and we could easily set up meetings or send emails whenever we encountered problems. The answers were precise and helped a lot with the development of the project.

In the end, I think this project really improved my skills as an Android developer and Software Engineer. I feel satisfied with the project we managed to deliver and with the fact that we were able to work as a whole despite the conflicts we encountered as a big group.

6.7. Khaled Hammoun

Being part of a group full of motivated people makes the stress of SEP negligible. When I found out the fourth-semester groups should consist of around 12 people, I got worried. I was used to working in groups of 3-4 people where we knew each other well and we knew our motivation level. Nonetheless, we were lucky to form a group, where most of us took responsibility for our tasks.

As a Scrum master, I am proud to say that my job was never easier. I got the support of my whole team and all of us managed to make this project successful. The times when I could not attend some of the meetings were not lost, because my teammates took the responsibility of performing and documenting the meetings.

As I come from a mechanical background, I have a strong pull toward software that is related to machinery. That's why when I heard about the embedded part of the project, I knew that

this would be the part of the project I would like to participate in. Luckily there were not many “C” enthusiasts, so there was a place for me. The software part of the project was extremely challenging and highly rewarding. It was not easy to make the hardware perform the way we wanted it to, but when we finally got a grasp at it, we felt the immediate satisfaction of success.

I got to contribute in analysing the requirements, creating the design of the IoT system, both the class and sequence diagram. The IoT sub-team that I was part of was following the pair programming, where I got my fair share of coding the system and later on making the Google test (Unit tests) for part of the system.

If I had to change anything regarding the embedded part of the project, I would allow the students to have a look at the drivers for the different sensors and actuators. To start playing around with them and make the sensors and actuators work the way that the group needs them to. Another thing is, that the required choice of sensors is limiting the ideas to very few. I would have liked to have a broader choice of sensors, so we can create a product that provides data that is truly valuable for analysis.

In my SEP3 reflection, I had set myself a few goals that I have accomplished. We managed to divide the user stories into more meaningful tasks so that our team knew exactly what needed to be done for the sprint. Unfortunately, we did not agree as a group on using Jira for tracking our progress, which in my opinion was a missed opportunity. Because of that, we had to spend many hours trying to figure out how to use the spreadsheets. Remark - I might have changed something in them that messed them up. I am fortunate enough to use Jira in my student job, so in the sense of learning Jira, I have accomplished that goal.

I am glad that this semester’s project had this format, as I think it will benefit me greatly for my upcoming internship. Thank you to the supervisors for their valuable inputs and guidance. I would think that after 4 semesters I had a pretty good understanding of how software should work, but I was proven wrong. I love this field because it never gets boring and there is always something new to learn.

6.8. Luis Fernandez Ponton

SEP4 has been a very positive experience for me. I feel that the project as a whole is a great idea and very likely closer to a real life setting than previous semester projects. At the beginning, I thought that it would be a complete disaster, because it had already been difficult at times to collaborate with four to five people in previous semester projects, so I couldn't even imagine how bad it would be with twelve people. However, even though we have had some issues with the process management, I feel that it ended up working out well. And I am thankful to have had the opportunity to work with some great people.

I chose to work with Data Engineering. This was what I wanted from the beginning, so I am glad that I had the chance to do this. Therefore my main contributions were within data engineering, especially building the data warehouse. I participated in the whole process from analysis to implementation, and helped my colleagues with building endpoints and building the gateway. In general, working on this project has helped me to learn a lot about implementing web services, about what infrastructure can be implemented, how to work with cloud services and build a data warehouse. The experience also gave me a glimpse (I think) into what it means to work with data in a real life IoT project. We were the link between IoT and Android and this meant that we had to interact and have an interfacing contract with both of them. I liked this because it meant that I had to understand both parts of the project, at least to the extent where we were able to help both teams. For example, we had to know what type of packages we would receive from the devices and how to build an API that Android could consume.

Due to the long commute I have and other personal circumstances, it was important for me to be able to work remotely and I appreciate that we were able to figure it out. It can be challenging to work remotely as a team of twelve. The Internet connection is not always optimal, it can be hard to hear others, and people will talk over each other. Things become a bit impersonal and it can be easy to lose motivation. But I think that it can also be good experience, seeing how normal this has become in many companies. It is also more convenient when it comes to pair programming and demos. And I think that it helps when most of the team members go to class and you can see them in person on a weekly basis and discuss things from the project.

I had a good experience with the supervision. Especially with Knud, as I worked in Data Engineering. He has been really helpful both during our DAI class and also during SEP feedback sessions. He has been always available for us if we needed to meet for questions and ready to reply to our emails.

All in all, it has been a very positive experience, where I have gained technical knowledge that I was interested in, experienced what it takes to work in a larger group with teams having different responsibilities, and got to meet some nice people along the way.

6.9. Morten Hansen

As anticipated, this semester was very different from previous semester projects. The contrast was of course mostly due to the size of the group, as the group now consisted of a total of twelve people compared to the four we had been previously. The fact that we were split into different sub-teams also played a big part in this. Having twelve people in the group was definitely an interesting change of pace as it brought more unique ideas to the table, though at the same time, it was at points also a disadvantage because of the same reason. It meant that we all had to agree on how to do certain things, and even though we had been studying the same courses, we each had different ways of doing certain things regarding both project and process documentation. This complicated things but was ultimately something that could be discussed and sorted out.

In regard to the sub-teams, I decided to work in IoT together with Adrian and Khaled. I already had some exposure to the C programming language, as I had briefly studied it before starting at VIA, so this naturally gave me a bit of an advantage and made the process a bit easier, thus at the same time allowing me to occasionally discuss design and implementation approaches with group members from the Android team as well as provide assistance when problems arose. I originally thought about joining the Android team, but as it was the most desirable technology to work with, and IoT the least, I figured I would just work with IoT instead, and in the end it turned out to be perfectly fine.

In the IoT team, we decided to do everything from analysis and design to implementation together, taking a pair-programming sort of approach. This allowed us to bounce ideas off one another and it also meant that all three of us had a complete understanding of the entire codebase and how specific classes functioned and worked together. Therefore, I contributed

to the entire IoT application and the IoT documentation, certain other parts of the overall documentation for the project- and process report as well as setting up appendices and other miscellaneous tasks like creating diagrams and black-box test cases.

I feel like we had a good chemistry and working environment despite being such a big group of people and I am pleased with working together with all colleagues. That being said, I think we at times could have been better at communicating between the different sub-teams, as the communication occasionally was a bit lacking, thus leading to issues that could have probably been prevented otherwise. Nevertheless, I am satisfied with the final result and I appreciate everyone's efforts to the group as a whole. Working together in a big group like this is not easy and was not something we were used to or had really tried before, so I think we managed pretty well considering. This semester project was a different but good experience, and despite the struggles along the way, I think we managed to deliver a satisfactory end product.

6.10. Stefania-Gabriela Tomuta

This semester project turned out to be a challenge for me and my team mates. Especially at the beginning were we had to take in another team member and rediscuss the group contract. Working with so many people that had different ways of doing things was a little bit difficult, but I believe that in the end we managed to align our differences and make the best of the resources we had.

I chose to be part of the Android team as I am interested in Android development as well as to improve my frontend skills and diversify my expertise. Compared to other semesters, were the semester group was more minimalistic, I had experienced a lot of various approaches to both writing implementation and documentation. I think this process was helpful in order to see different perspectives, to learn from others and discover that sometimes your approach may not be always the best one.

As I previously worked with Scrum, I was very surprised to see that writing the documentation for it in Excel files by 12 people was not the best approach. I think Jira would have been a better choice to keep things more organised and at ease. My role as a Product Owner turned out to be a lot easier for the Android team as the application design choices were fast and easily discussed, no need to ask other people from other specialisations. I

think everyone did their best, even though we had many sprints “Incomplete”. I would have liked Android team to be a little more focused and spend more time on trying to deliver the tasks and users stories.

Regarding the Android team, in the beginning our progress was slowed due to the dependency on the endpoints. In truth, data was mocked as much as possible, but the endpoints changed various times and I found myself trying to refactor what was already done. On the other hand, I had some difficulties getting accustomed to working with other people, except Vlad, with whom I have worked the past 3 semesters and consider ourselves being on the same page. I think both of us tried to make this team balanced, even though sometimes we felt overwhelmed, but in the end we got accustomed to each other.

My contribution to the project comprised the documentation: Scrum documentation, Project Description, Project and Process Report. I implemented MVVM architecture, page adapters for tabs and adapter recyclers. I spent time on implementing the Farmerama database that comprised refactoring models, writing implementation in the repository to store data locally as well as remove local storage at the choice of the user. I wrote manual queries and figured out a way to see if the user is online or not. Also, I designed and implemented patterns and document them in order to make the implementation easier on the go. I worked on the networking side, reading the error messages from the web service as well as implementing utility classes for converters. On the other hand, I dedicated time to performing black box testing, unit testing and bug fixes. Moreover, as for the requirements, I implemented the user stories related to thresholds, measurements and spent time on contributing to the rest of the requirements whenever refactoring was needed.

As for the meetings with our supervisors, I have enjoyed hearing different opinions and being up to date with what the other teams are going through. I would say the most helpful meetings were the ones with our team supervisor, Kasper, who helped and guided us to implement what the application lacked and provided good advice regarding documentation.

In the end, I would like to thank everybody for the efforts of making this team a team and delivering a good project in the end. Therefore, this project was a good experience, from which I have learned a lot and I am looking forward to apply my new skills in future projects.

6.11. Tabita Roxana Varlan

This semester was different from the previous one considering the transition from a team of 3 people to one of 12 but I think we managed to be a united team and get used to each other's way of working. I've noticed from the previous semester that the team members are very motivated and willing to have the best team possible.

Working with SCRUM was not something new but in such a big team was a bit difficult in the beginning and we spent a lot of time trying to accommodate each other but the more meetings we had the easier it was. It is hard to gather and choose between all ideas coming from 12 people. All the meetings were online but there were no disadvantages from this point of view, on the contrary when someone could not attend, they could at least listen to what was discussed in the meeting.

I chose to be part of the Android Team and I learned a lot from the teacher during the classes but also from the team. During the classes we have learned the basics but working on SEP we confronted more difficult situations that required hours of watching tutorials. I received guidance when I asked, and I saw people willing to help at any moment when I faced a situation of not knowing how to accomplish a certain task. In fact, even when we were splitting our tasks, this was done so that everyone could do it. All opinions were considered, including the last ideas proposed by the teacher during the last meetings.

In this semester project I contributed to the documentation section (part of use case description, class diagram), writing in the reports, fulfilling tasks to which I was assigned in Product Backlog (login and register view-view model, edit area, exceeded measurement, view and edit account with uploading images to firebase) and performing Black Box testing.

One of the things I had issues with during the development process was GitHub (merging issues, committing from 2 accounts), even though it's the third semester when I use it, it seems like I still have to improve my skills. Another team member suggested that I use GitHub Desktop and it was much easier.

In conclusion, this experience gained working in a larger team will definitely help me in the future professionally and I won't hold back when it comes to working in a big team.

6.12. Tomas Ondrejka

Each semester project brings a different set of challenges and learning opportunities. This semester was interesting in that it was very similar to working on a real project. Because there were 12 people in the group, strong collaboration was required to satisfy the product owner's expectations. The start of SEP4 was difficult, as it is with every new project. Small disagreements arose as a result of the team's large number of members. We needed to discuss the technologies we would utilise because the technologies that the team members had previously used were diverse. It was also critical to get to know one another in order to find a way on how to collaborate. This was no longer an issue after a few weeks when we got to know each other, and the cooperation was great.

Data engineering is the specialisation I've selected. Back-end development of the data service that received and exposed data and database setup were the majority of my tasks. This speciality has given me a lot of expertise in back-end development, which is something I'd like to explore further in my career.

Outside of the data engineering specialisation, I also have a responsibility in the DevOps domain. DevOps is a topic that has grabbed my curiosity for some time. I reasoned that by using my past skills, I might make development and integration easier for SEP4 and the teams. It was very satisfying to see how deployment automation can make work easier and faster. In addition, I gained a lot of knowledge and insight in this field.

Following the scrum and unified process was one of the prerequisites for this semester's project. Because we were in a small team the previous semester, it was difficult for us to fully understand scrum and all of its procedures. This semester, I feel we followed scrum as accurately as possible, and I have gotten a better understanding of the methodology. Our meetings took place online so that everyone in the team could participate, even if they were not physically on campus.

Tasks related to the Data Service API and infrastructure were the majority of my contributions. I was involved in the design, implementation, and testing of several classes such as controllers, services, dao, and jpa. To improve my Java skills, I also took on responsibility for exception handling, building "dev" and "prod" spring profiles with various configurations, receiving and processing data from LoraWan network, and scheduling DW

incremental load script. I have also contributed by creating database hosting, data service hosting, and a pipeline for continuous delivery on the AWS infrastructure.

Overall, I am pleased with the results of this semester's project. I believe that as a team, we have grown tremendously, and that we will benefit from the SEP4 learnings in future projects.

7. Supervision

The supervisors assigned for this project were Ib, Kasper and Knud. Supervision had two modalities:

General supervision:

- Fixed online meetings with our three teachers after each hand-in

Supervisions depending on specialisation:

- Online meetings when requested to the specialisation supervisor
- Onsite meetings when requested
- Email Q&As every time we needed

The supervisors' guidance was key to developing the project and finishing on time.

The DAI team was supervised specifically by Knud. He helped the team with advising regarding the modelling and ETL design process, as well as visualisation choices. The communication was mainly by email, which he always answered quickly. Additionally, during scheduled meetings Knud always gave helpful and constructive feedback.

The Android team was supervised by Kasper who advised us how to improve the application and documentation. He recommended that we should implement notifications, have data stored locally on the device and display the last session, use graphics and structure the navigation. We received fast feedback through email or meetings when we were confronted with issues regarding our work.

The IoT team was supervised by Ib who provided great and useful resources for both documentation of the various drivers as well as examples of how an IoT application could be structured. He was quick to respond to emails in order to set up physical meetings in which he provided good feedback and input as well as support regarding the Google tests that the team at first struggled to properly set up.

To conclude, we would like to thank all three supervisors for their useful feedback, input and flexible schedules throughout the entire semester project.

8. Conclusion

Overall, an aspect that worked really well in this semester's project was working remotely throughout the entire development period. The majority of the team was motivated and present. While in the beginning, the communication between team members was lacking due to different ways of working, it was remediated and improved early into the semester. Naturally, this also aided to the team working online, making it easier to coordinate tasks and functionalities between subteams.

Despite all group members having worked with Scrum before, the fact that we merged different groups into one caused some problems because everyone had their own style of utilising it. The time spent looking for an agreement paid off. This is because we were able to come to an agreement and create a productive approach to collaborate. We would definitely switch from spreadsheets to a work management system in the future so that everyone can understand the rules and layout.

All of the technical obstacles have allowed us to thrive and develop as future professionals. Working in a large group has given us a more accurate sense of how it would be to participate in a real working environment and we feel satisfied with the process and the way we have handled being in a group of twelve individuals as well as with the end product as it covers all the proposed requirements. That being said, we are naturally also aware that with every project, there is always room for improvement.

9. Sources of Information

Country comparison. Hofstede Insights. (2021, June 21). Retrieved May 31, 2022, from https://www.hofstede-insights.com/country-comparison/

The 2020 scrum GUIDETM. Scrum Guide | Scrum Guides. (n.d.). Retrieved June 1, 2022, from https://scrumguides.org/scrum-guide.html

What is Scrum? Scrum.org. (n.d.). Retrieved June 1, 2022, from https://www.scrum.org/resources/what-is-scrum