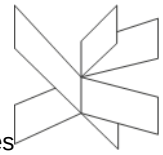


Greenhouse

SEP4 - Project Report

Students

Mark Vincze, 224478

Priyanka Shrestha, 299543

Maria Asenova, 239533

Constantina Tripon, 253085

Daniel Railean, 294241

Ilia Nikov, 297112

Tamas Peter, 299124

Florina Toldea, 299116

Michel Sofus Engelhardt Sommer, 273966

Viggo Petersen, 314203

Supervisors

Kasper Knop Rasmussen

Knud Erik Rasmussen

Ib Havn

[Number of characters: 67.393]

Software Technology Engineering

4th Semester

17/12/2021

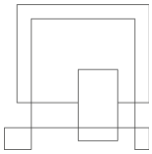
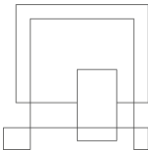
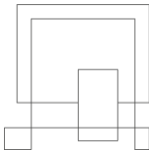


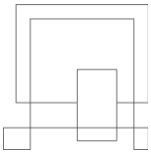
Table of content





List of figures and tables

Optional



Abstract

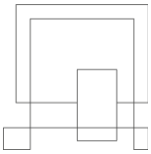
The focus of the project report is to create a sustainable solution for an automated remote greenhouse using wireless sensors connected to the Internet of Things. The system is capable of monitoring, analysing and visualising data.

The development process began with the analysis phase where the business requirements were stated, and the behavioural diagrams created using UML. The design phase started with creating the structural diagrams which defined the overall architecture of the system.

The system makes use of a microcontroller connected to a Loriot server by a LoRaWan antenna, communicating through a Web Socket API connection to a gateway server application. The server application is designed in a Spring Boot framework such that it supports a RESTful architecture with the Android client. Data analysis was done following the dimensional modelling process using Microsoft SQL Server as a tool to handle the ETL system development. The data visualisation was created using Microsoft PowerBI.

The result is a software with an architecture which allows the user to add multiple devices to the greenhouse, set and reset target and threshold values to the sensors, have periodic measurement reading for each sensor and an overview of the recorder measurements over a period. For the data analyst, the recorded data is displayed in an interactive dashboard.

Tests were conducted to ensure a performing system. The report ends with a brief discussion of future improvements.



1 Introduction

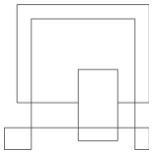
A research report of McKinsey&Company done in 2017, states that 40% of Danish working hours could be automated by usage current technologies. While automation will bring substantial improvements in prosperity, the transition will also face challenges (McKinsey&Company, 2018), but since Danish society has a high level of trust in the public institutions, Denmark is well positioned to become one of the world's leaders in digital front runners (Alfter, 2020).

Greenhouse remote automation refers to the software created to automate actions of the technical installations installed within the greenhouse. The greenhouse anticipates weather forecasts, optimizing the irrigation strategy accordingly, whilst keeping the user updated through a mobile application at any given place and time. (DutchGreenhouses, 2021)

Based on the background information, this project report describes a software system capable of monitoring, analysing, and displaying live data on demand, allowing the user to set up and manipulate the threshold values.

The system can monitor multiple devices installed in a Greenhouse. The microcontrollers support four types of sensors: humidity, temperature, CO_2, and light. The responsibility of the user is to set up the device and choose the target measurement for each sensor and set up the threshold values accordingly.

The system has been developed in relation with the user requirements and the overall process is depicted in the following chapters. The final solution can be found in the Result section which concludes this project report.



2 Analysis

The following section defines and analyses the requirements and business logic of the system. An introduction to the concept and description of the client will follow accordingly. A use case diagram will be created to group the user stories based on business goals. The behaviour of the different use cases will be analysed using an activity diagram. The findings from the analyses will be concluded in a Domain model.

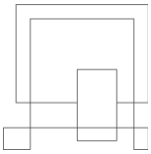
2.1 Introduction to the concept

The greenhouse system is created with the purpose of helping users optimize the growth process of plants in a remotely controlled environment. The system is built around multiple sensors responsible for measuring the temperature, humidity, and CO2 of the controlled environment, and will store all the measurements such that they will be available for access at any point in time, but it will also notify the user of any action that needs to be followed such that the greenhouse to be always in an optimum state.

The user will have full control of the settings and threshold values needed for the specific greenhouse, being able to control the system through an android application, reducing the labour hours and costs. Statistics on historic measurements an will also be available for the user in the Android application.

The system should support different features for handling common problems that arise when creating and maintaining a greenhouse automation. From the user's perspective, the system must support:

- a login and edit profile functionality
- specific sensors for environmental control displayed on a mobile application, capable of being remotely controlled
- the data should be updated with the latest value, live data
- the data gathered to be persisted such that it is available on demand



- the data of the outside temperature to compare with the one inside the greenhouse

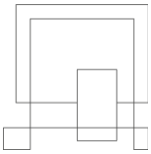
1.1. Description of the client

Martin Høgh is a Danish entrepreneur starting a greenhouse installation business. The small business Martin is building is aimed at both B2C and B2B customers. The company wants to help individuals and other businesses not only with their greenhouse installation but also with monitoring the environment inside the greenhouse.

To control climate, the greenhouse environmental system relies on information collected from sensors which monitor temperature, humidity, CO_2, and light. Martin wants to provide his customers with:

1) Humidity measurements over specific period
2) CO_2 measurements over specific period time
3) Light measurements over specific period time
4) Temperature measurements over specific period time
5) Daily update of average environment variables
6) Monthly median/average/mean of measurements
7) A log of when the data has crossed the threshold value
8) An option to control the environment remotely (windows, air conditioning etc.)
9) The option to add multiple devices with their sensors in the greenhouse

The client wants to make sure that the greenhouses using this system will have better environmental factors for the plants having a higher quality harvest time.

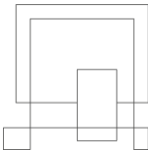


2.2 Actors' description

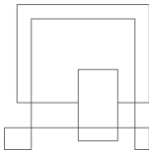
1. User - is responsible for setting up and controlling the threshold values of the sensors. Can be represented by any owner of a greenhouse that uses the system.
2. Data analyst of the business - is responsible with analysing the data gathered from all the sensors. Is represented by a data engineer hired in the business.
3. Time - on a certain amount of time a new measurement is made
4. Environment - exterior weather conditions

2.3 User Stories

1. As a user, I want to be able to register and log in so I can have access to the system.
2. As a user I want to gain access to the system to control the different environmental factors.
3. As a user, I want to be able to see data gathered by the temperature sensor so the temperature can be adjusted accordingly by setting up new threshold values.
4. As a user, I want to be able to see data gathered by the humidity sensor so the humidity can be adjusted accordingly if needed by setting up new threshold values.
5. As a user, I want to be able to see data gathered by the CO_2 sensor to get an overview of the CO_2 concentration in the greenhouse so I can adjust the level needed for the specific plant.
6. As a user I want to be able to see data gathered by the light sensor so I can make sure that the lights are functioning in a timely manner.
7. As a user I want to manage my device settings to the control system.



8. As a user I want to be able to set threshold values to devices to ease access to remote locations.
9. As a user I would like to select different time frames and compare the data (temperature, humidity, photoresistor, CO_2) collected of the specific device to monitor the state of the greenhouse.
10. As a user I want to get notified when the temperature sensor is out of the threshold value.
11. As a user I want to get notified when the humidity sensor is out of the threshold values.
12. As a user I want to be able to add multiple devices to my greenhouse such that I can monitor the data from different locations inside my greenhouse.
13. As a user I want to get notified when the CO_2 sensor is out of a threshold value.
14. As a user I would like to be able to display the local weather (temperature, humidity, CO_2) relevant to the greenhouse's environment such as to compare interior and exterior data.
15. As a data analyst of the business, I want to be able to see the temperature, humidity, CO2, and light data of all my devices in a single dashboard so that I can compare the data.
16. As a data analyst of the business, I want to get the minimum, maximum and the median temperature data of the greenhouse collected from the beginning to the latest recorded data.
17. As a user I want to be able to review the latest data in case of no internet connection.
18. As a user I want to be able to create multiple devices such that I can keep track of the devices installed in a greenhouse and where they are located.
19. As a user I want to be able to remove a device from my list of devices in a greenhouse when I no longer need to monitor data from the device installed in the greenhouse.
20. As a user I want to be able to edit the settings of a device located in a greenhouse such that I can change the targeted temperature, humidity, CO2 and light.



21. As a user I want the device to be able to operate windows, in relation to the greenhouse environment, to regulate the environment.

2.4 Non-Functional Requirements

1. The LoRaWan Greenhouse-node must send data to the server, called “gateway server”, via the Loriot LoRaWAN gateway every 5 minutes.
2. The Loriot gateway must be able to store data sent from the systems gateway server.
3. End users will interact with the system through a native Android app.

2.5 Use Case Diagram

The user stories with common business values are grouped into one use case, and the different actors are the ones that trigger the use cases, see Figure 1 displaying the Use Case diagram

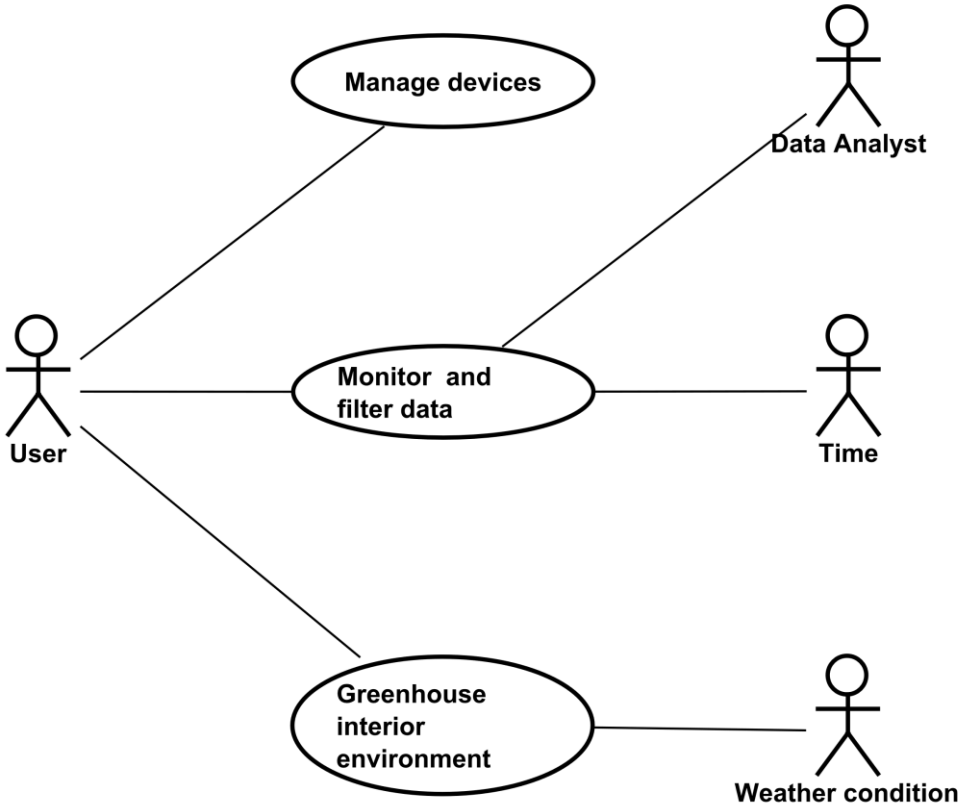
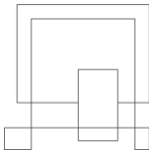


Figure 1. Use Case Diagram

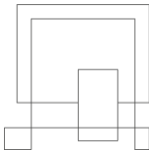
2.6 Use Case description

To further analyse the use case of Monitor and Filter Data, a use case description in fully dressed format was created to ensure the robustness of the system. To depict both sunny and rainy scenarios of the use case, the following use case description was made.

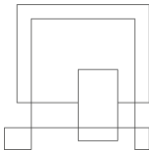


- Monitor and Filter Data

<table>
<tr><td>Use case name:</td><td>Monitor and Filter Data</td></tr>
<tr><td>Level:</td><td>View data recorded</td></tr>
<tr><td>Primary actor:</td><td>User</td></tr>
<tr><td>Stakeholders and interests:</td><td>The actor wants to view measurement data from different devices over time.</td></tr>
<tr><td>Precondition:</td><td>The actor must be logged in, connected to the server and have a device added.</td></tr>
<tr><td>Success guarantee:</td><td>The actor has been provided with a visual representation of the information of the specific sensor and a time frame on a chart.</td></tr>
<tr><td>Main success scenario:</td><td>
<table>
<tr><th>User</th><th>System</th></tr>
<tr><td>1. Selects a device by name (device card shows latest measurements)</td><td>1.1. The system opens the selected device showing the latest information from all the sensors (temperature, humidity, CO2, light).</td></tr>
<tr><td>2. Selects Humidity</td><td>2.1. The system opens a Humidity page populated with the latest recorded</td></tr>
</table>
</td></tr>
</table>



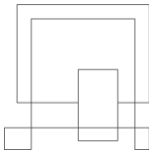
		data about humidity over time. 2.1.a. Selects a time frame - 15mins - 1hr - 24hrs - 1week - 1year	2.2.a The system displays the data according to the selected time frame
	3. Selects Light 3.1.a. Selects a time frame - 15mins - 1hr - 24hrs - 1week - 1year	3.1. The system opens Light page and displays if the light is off or on 3.2.a The system displays the data according to the selected time frame showing the pattern of on and off hours
	4. Selects Temperature 4.1.a. Selects a time frame	4.1. The system opens Temperature page populated with the latest recorded data 4.2.a The system displays the



<table>
<tr><td></td><td>
<table>
<tr><td>- 15mins
- 1hr
- 24hrs
- 1week
- 1year</td><td>data according to the selected time frame</td></tr>
<tr><td>5. Selects CO2

5.1.a. Selects a time frame
- 15mins
- 1hr
- 24hrs
- 1week
- 1year</td><td>5.1 The system opens CO2 page populated with the latest recorded data

5.2.a
The system displays the data according to the selected time frame</td></tr>
</table>
</td></tr>
<tr><td>PostCondition</td><td>The information requested is displayed in a line chart on the UI.</td></tr>
<tr><td>Extensions</td><td>*a. At any time the user uses internet connection
1. The system displays the latest data saved for humidity locally on the application
2. The system displays the latest data saved for temperature locally on the application
3. The system displays the latest data saved for CO2 locally on the application
4. The system displays the latest data saved for light</td></tr>
</table>



	locally on the application

2.7 Activity diagram

For a better illustration of the system's reaction regarding the user input, the following activity diagram (Figure:2) was made. The diagram covers the interaction of the system and user regarding monitoring the data.

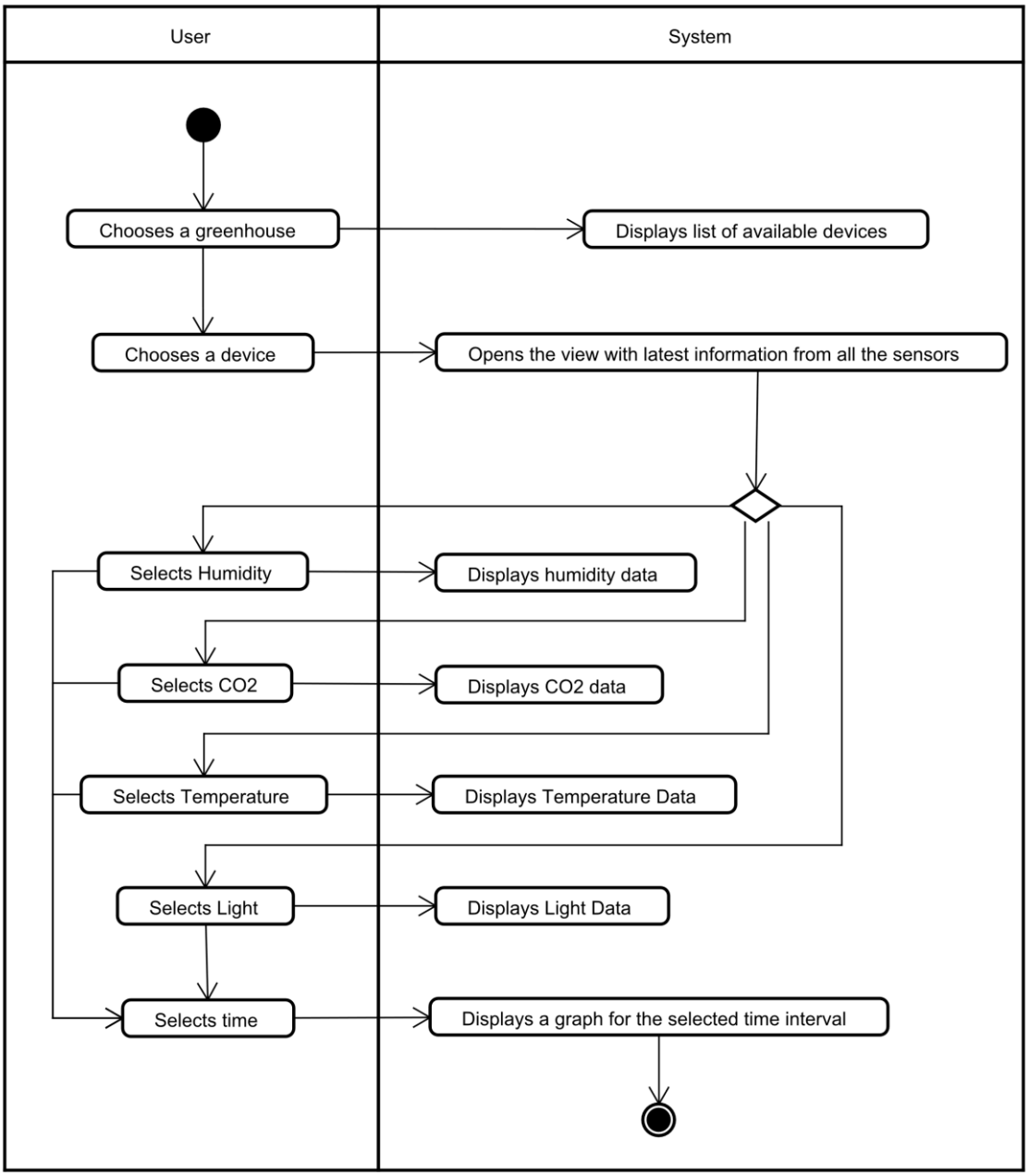


Figure 2: Activity Diagram

2.8 System sequence Diagram

To illustrate the actions taken by the user to view humidity measurements over a period and the corresponding reactions of the system, a system sequence diagram was created (Figure 3).

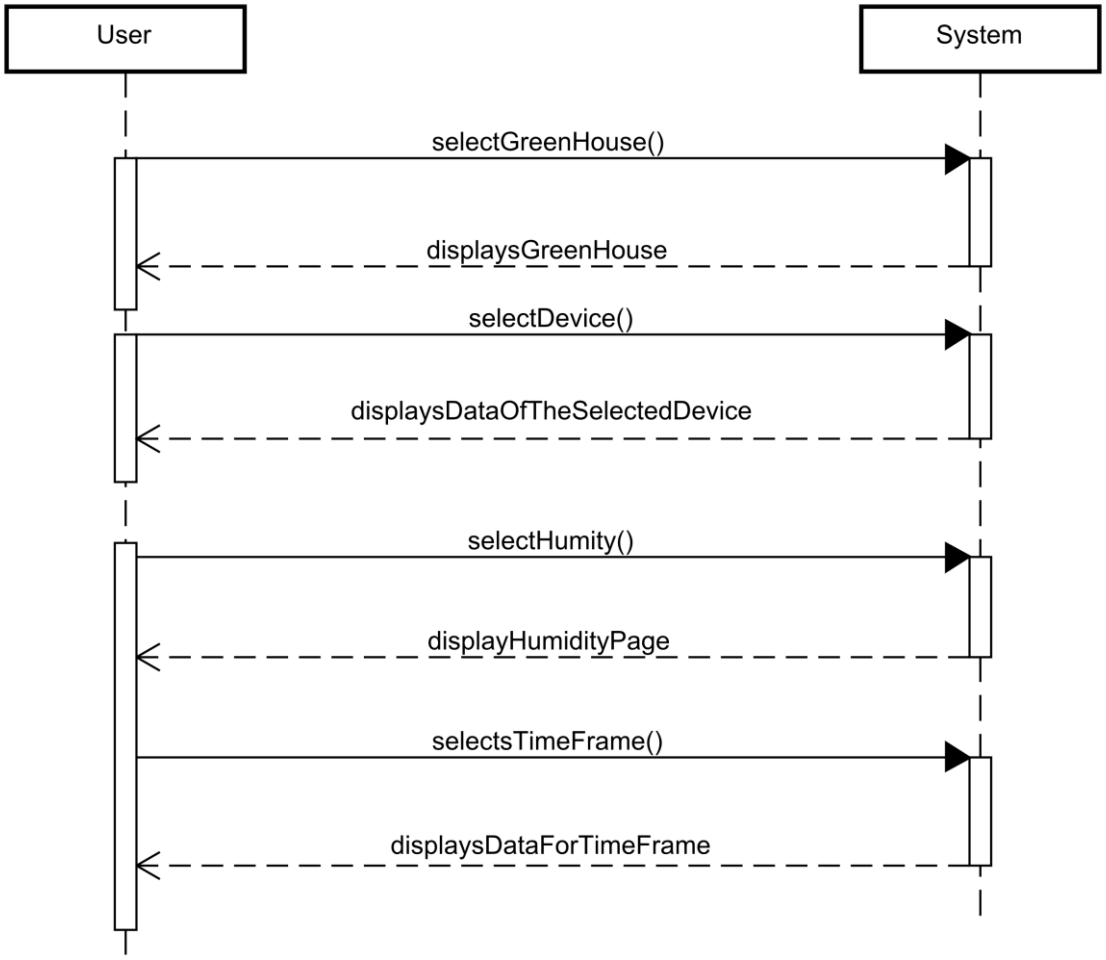
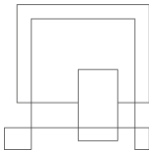


Figure 3: System Sequence Diagram

2.9 Test cases



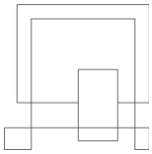
The following test cases have been built based on the created use case description. Both sunny and rainy scenarios have been covered. The test cases will serve as a specification for a manual functional testing in the test phase.

As a user, I want to be able to see data gathered by the temperature sensor so the temperature can be adjusted accordingly by setting up new threshold values.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	
2.	- Selects Temperature from the dashboard	- System displays the temperature view for the greenhouse	
3.	- Selects the period of time for data to be displayed	- System displays the temperature values over the period of time	

As a user, I want to be able to see data gathered by the humidity sensor so the humidity can be adjusted accordingly if needed by setting up new threshold values.

SN	Action	Reaction	Status

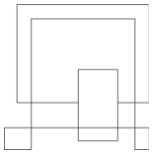


1.	- Selects the greenhouse	- System opens the specific greenhouse view	
2.	- Selects the humidity from the dashboard	- System displays the humidity view for the greenhouse	
3.	- Selects the period of time for data to be displayed	- System displays the humidity values over a period of time	

As a user, I want to be able to see data gathered by the CO_2 sensor to get an overview of the CO_2 concentration in the greenhouse so I can adjust the level needed for the specific plant.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	
2.	- Selects the CO2 from the dashboard	- System displays the CO2 view for the greenhouse	
3.	- Selects the period for data to be displayed	- System displays the CO2 values over a period	

As a user I want to be able to see data gathered by the light sensor so I can make sure that the lights are functioning in a timely manner.



SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	
2.	- Selects the Light from the dashboard	- System displays the current state of the Light sensor	

As a user I would like to select different time frames and compare the data (temperature, humidity, photoresistor, CO_2) collected to monitor the state of the greenhouse.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- The system opens the selected greenhouse and displays the latest recorded data for Temperature, Humidity, CO2 and Light.	
2.	- Selects the environmental variables (Temperature, Humidity, CO2 or Light) from the dashboard	- The system opens the view for the selected variable and displays a graph with the recorded data	

3.	- Selects the timeframe available in the view (15min, 1 week, 1 month, 1 year)	- The system displays the data recorded for the selected time frame in a graph	

2.10 Domain Model

The domain model shown in Figure 4. is built after the analysis of the user stories. It's a visualisation tool to highlight the actors involved in the interaction with the system and the flow of events that needs to be followed. The domain model below shows the classes and their relationships within the domain of a remote greenhouse automation.

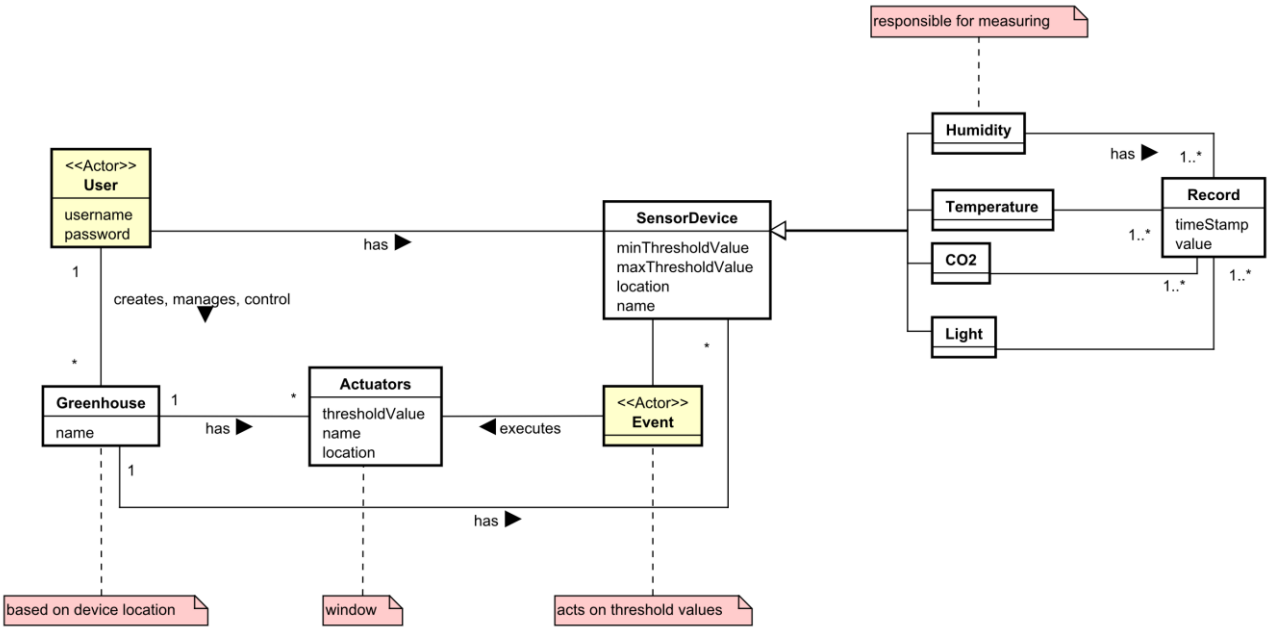
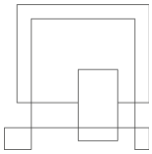
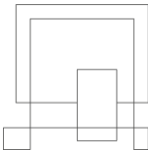


Figure 4: Domain Model



The domain model concludes the findings from the analysis phase and serves as input for the design phase.



3 Design

(Constantina Tripon, Maria Asenova, Mark Vincze, Priyanka Shrestha)

The following section makes usage of the findings from the analysis phase which serves as a base of the design process. The creation of a deployment/architecture diagram follows accordingly after the Domain model. Structural diagrams such as class and package diagrams were created to design the core architecture and divide the responsibilities between the different parts of the system. Interaction diagrams such as sequence diagrams are included to depict the behaviour of the different parts of the system.

The design section combines the functionality of an IoT system capable of retrieving, analysing, and visualising sensor data into one system. An overview of the choice of technologies and communication protocols for the entire system are introduced together with a detailed presentation from each specialisation.

The deployment diagram below (Figure 5) illustrates the overall architecture of the system and its structure in terms of specified components and their interrelationships.

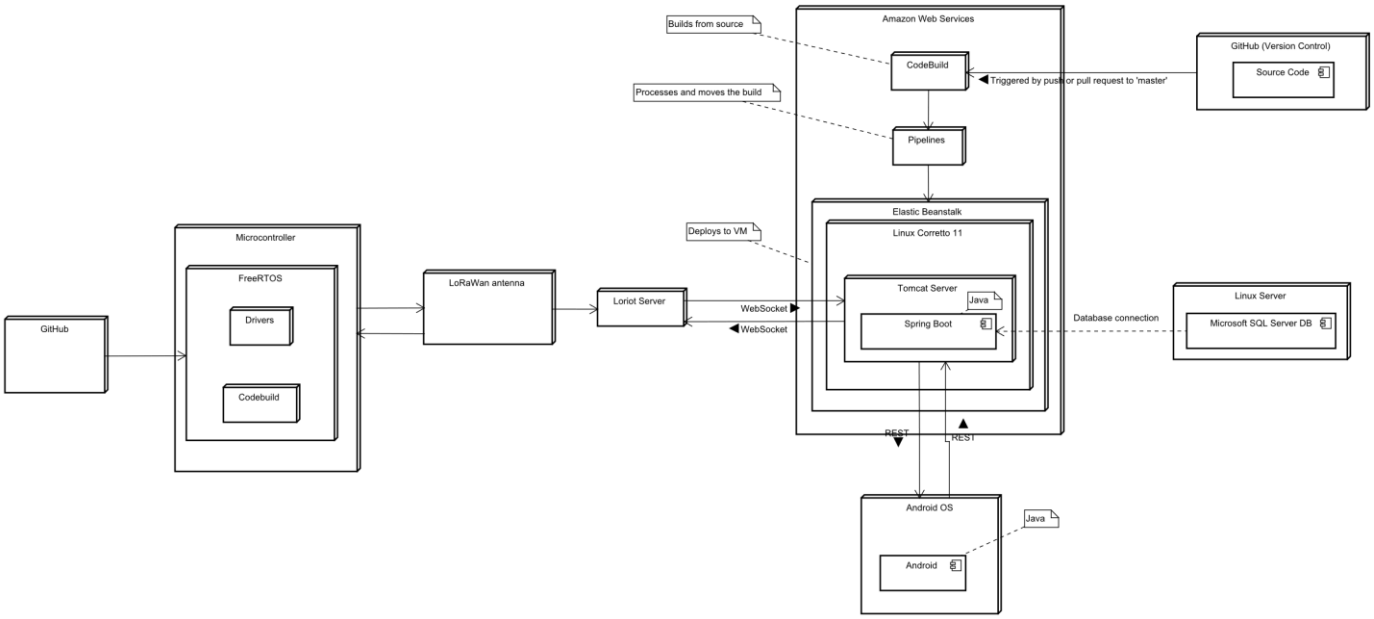


Figure 5: Deployment Diagram

The package diagram below (Figure 6) depicts the dependencies between the packages (folders in IoT) in each part of the system, LoRaWan Greenhouse-node, Loriot gateway and the Android app.

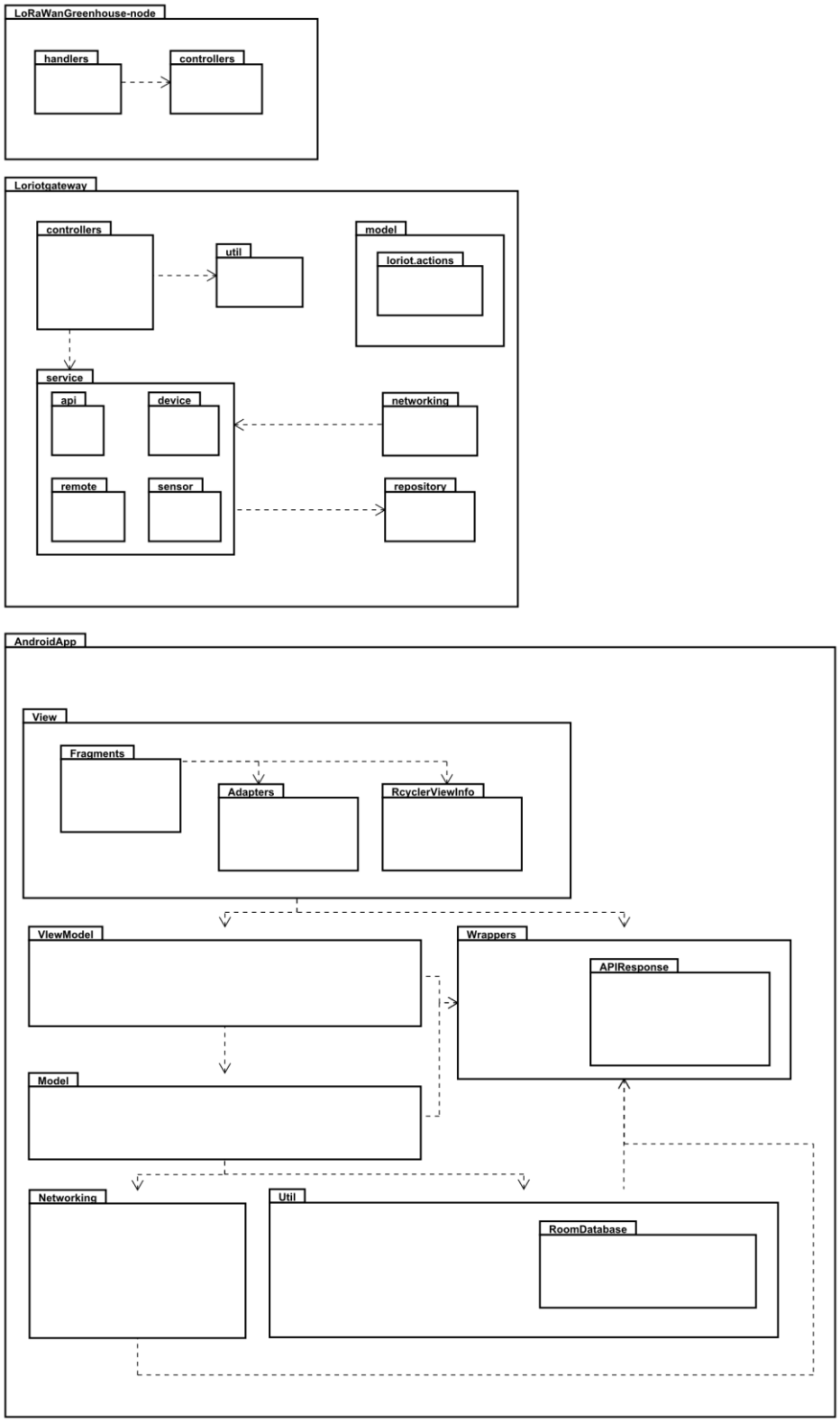

LoRaWanGreenhouse-node
handlers
controllers
Loriotgateway
controllers
util
model
loriot.actions
service
api
device
networking
remote
sensor
repository
AndroidApp
View
Fragments
Adapters
RcyclerViewInfo
VlewModel
Wrappers
APIResponse
Model
Networking
Util
RoomDatabase

Figure 6:
Package Diagram

The sequence diagrams below (Figure 7 and 8) illustrate the flow of data from the user to the IoT. The diagrams show the behaviour of the system as the user requests the latest measurements recorded.

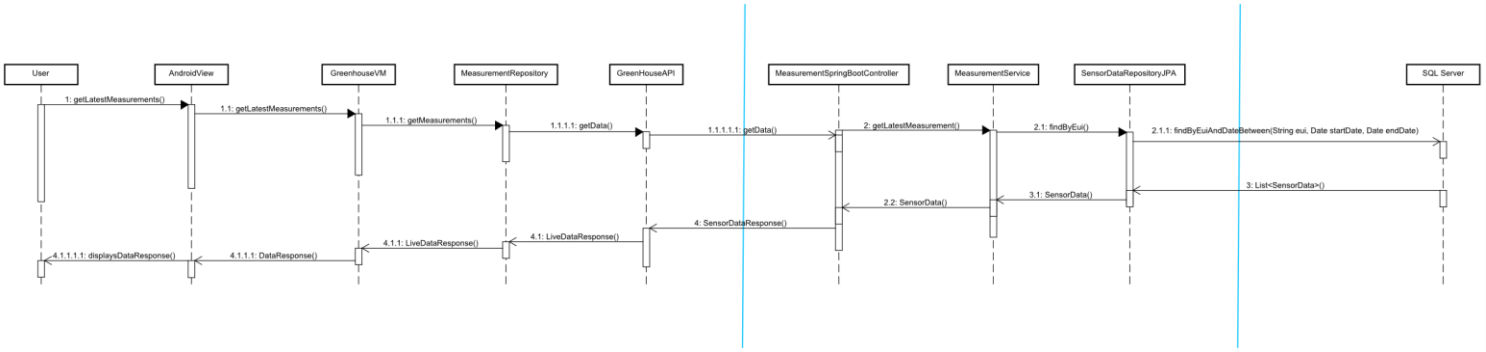


Figure 7: Sequence Diagram

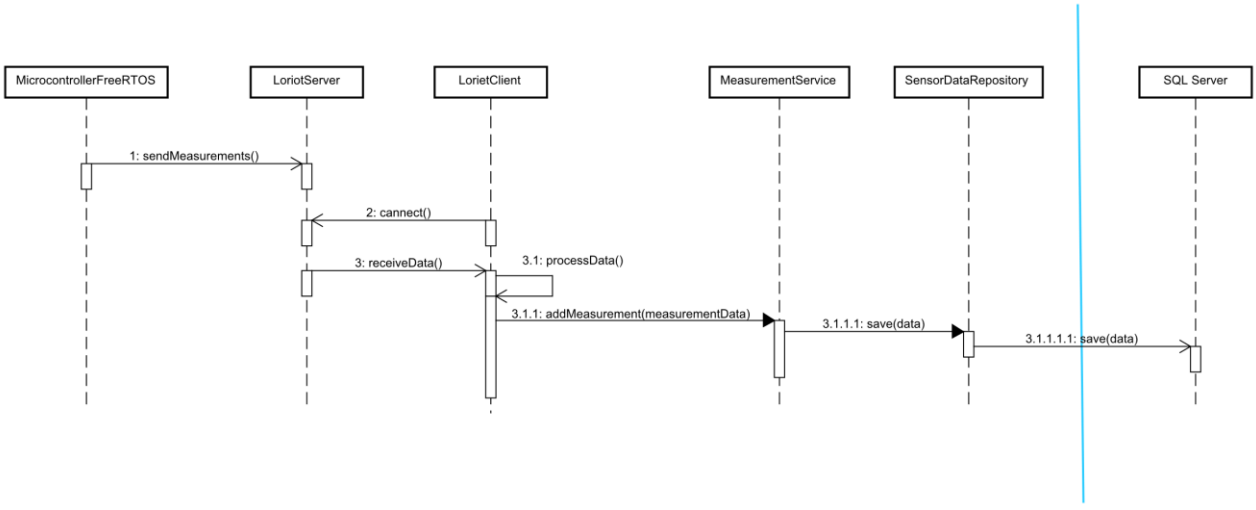
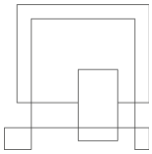


Figure 8: Sequence Diagram



In the next sections of the design chapter, each part of the design system will be documented.

3.1 IoT

(Michel Sommer, Florina Toldea, Viggo Petersen)

The previous analysis shows the need for sensors that measure temperature, humidity, co2 and sensor to track the plants growth. to help her regulate the environment a servo motor to open and close a window is chosen. A valve for regulating water, to watering plants, or some control for regulating light could also have been options.

To increase productivity, the design of data to be exchanged between the LoRa node and database lies first in the design process, to ensure agreement on the format exchanged through the gateway.

The environmental values are handled by a MCU and transmitted via LoRaWAN to a gateway, via the Loriot network which has a good coverage, in test setup. this provider can be changed depending on the provider that is available on the location.

Hardware for this is:

- MCU (ATMega 2560),
- VIA MEGA 2560 Shield
- Temperature and Humidity sensor (HIH-8120)
- CO2 sensor (MH-Z19)
- Light sensor (TSL2591)
- LoRa Module (RN2483)
- Servo motor (generic)

3.1.1 Data packet:

(Michel Sommer)

Sensors Data Types are the start point for the design of the data packet to and from the LoRa Node. Temperature and Humidity sensor (HIH-8120) can return values in float or temperature in int16_t and humidity in uint16_t. CO2 sensor (MH-Z19) returns a ppm value in a uint16_t*. Light sensor (TSL2591) returns lux as a float*. Other values as visible, infrared, and full spectrum light as uint16_t*. Servo motor (generic) receives servo id as a uint8_t and position as an int8_t (-100 to 100).

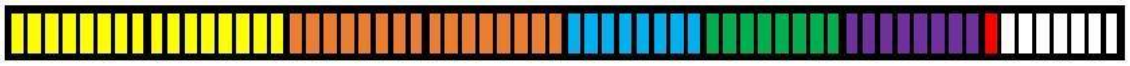
With the data load limited to 20 bytes, available to communicate on the LoRaWAN via available drivers for LoRa Module (RN2483).

There were two options, one dividing the combined payload to different payloads using different transmission ports. That approach would give a delay in data that has to be analysed later in the process. Therefore, the option of using the huge payload was chosen. To save payload, **temperature** is sent in int16_t as 10X the measured value. The **humidity** was in uint16_t as 10X the measured value, but if divided by 10 again the values can be stored in a uint8_t, it gives a lower resolution in the received data. If we look at the environment in the Greenhouse, then decimals have a minor impact on the environment. **CO2 sensor** data is kept in uint16_t, but since the maximal sensor data is unlikely to extend 32767ppm it gives a free bit, used to show if the **window** is open or closed. There is only need of lux value from the **light sensor**; it is changed from float to uint32_t, but as the sensor delivers values in the range from 0 to 130000, and there is no need for the lowest values, that describes the level of moonlight. It gives room for dropping the highest significant byte.

Send data, shown in bits, grouped in bytes

Data packet sent to the node, for setting threshold values is chosen to cover temperature in int16_t as the **High** and **Low** value. Humidity is in uint8_t as the **High** and **Low** value. The window's **opening percentage** is int8_t and lasts a bit to actively as a Boolean value to show if the window has to **open or close**.

Receive data, shown in bits, grouped in bytes



3.1.2 Program structure:

(Michel Sommer)

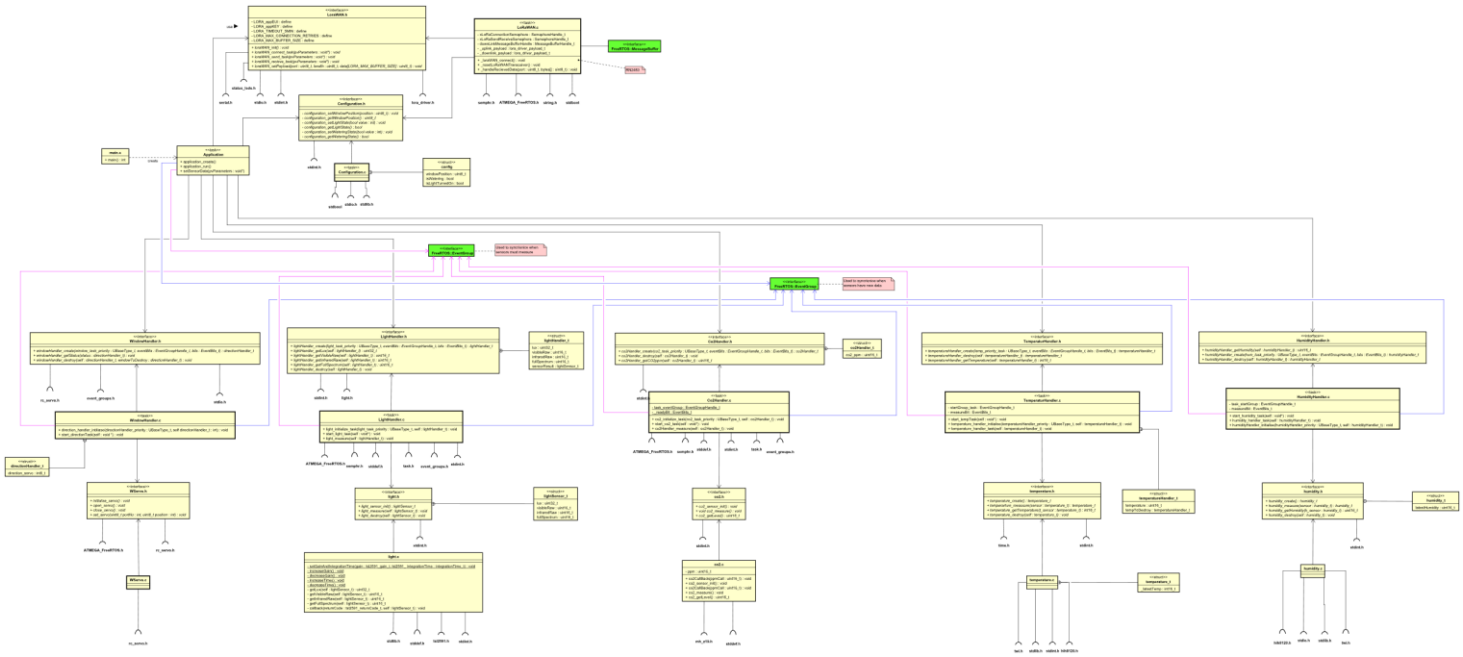


Figure 9: Class diagram - IoT

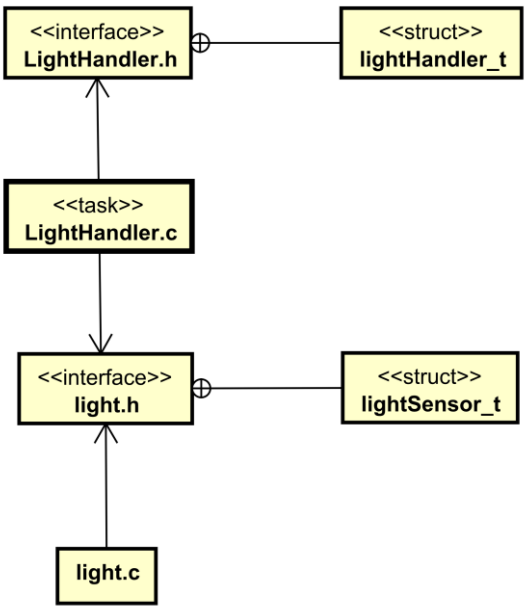


Figure 10: Solid Example

The Class diagram for the IoT part, shown in Figure 10, can be found in a larger version in Appendix C. The upper part shows some core functions, the FreeRTOS application itself, and the Task responsible for handling the LoRaWAN connection, and data from sensors.

The lower part of the class diagram shows a flat structure to follow the Single-responsibility principle and the Open-closed Principle, in the SOLID principles, without dependencies on other tasks. It would also make it possible to include more functionality without changing other tasks e.g., adding previously mentioned functions for regulating water or light. Figure 10. uses the Light sensor as an example, on the design of the lover part of the class diagram from Figure 9. The task manager in FreeRTOS implements the header file “LightHandler.h” according to the SOLID Interface segregation principle.

The Task is located in “LightHandler.c”, and according to the Single-responsibility principle, it only handles the task itself. Instead, it calls functions in the header file “light.h” according to the

Dependency inversion principle. In the bottom, “light.c” handles the communication to the driver, so according to the Liskov Substitution Principle, the only file that needs to be changed, if the sensor is changed to another type, using another driver, is the “light.c” itself.

3.2.3 Program loop (Michel Sommer)

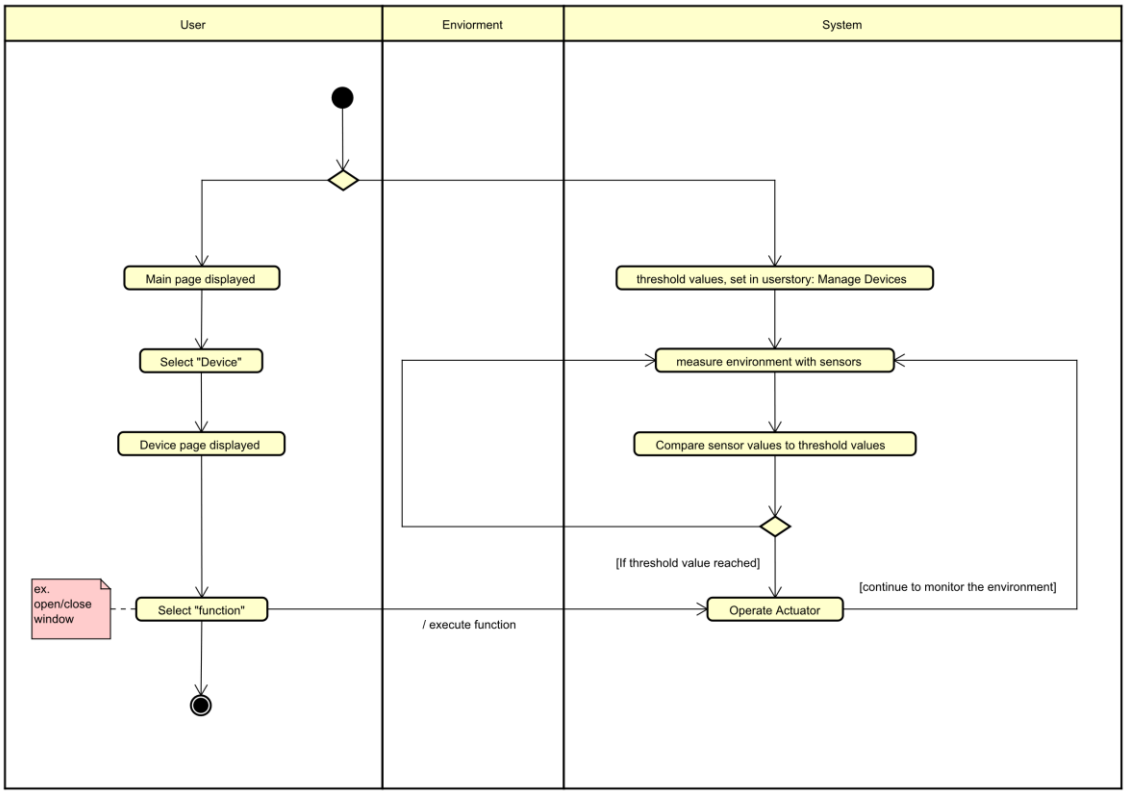
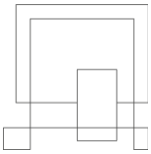


Figure 11: Activity Diagram from user story: Regulate Environment



A step back in the design to the Activity Diagram for the user story: “Regulate Environment” in
Figure 11. can be found in full size in Appendix C. The diagram describes that the task manager in FreeRTOS continuously measures sensor values, and compares the result, with a threshold value, set by the user. If the threshold value is reached, the Taskmanager will call “WindowHandler.h” to open or close the window, according to the reached value.

Each 5 minute, according to the LoRa protocol, the LoRa Module sends latest sensor values, to a gateway, in the previously mentioned data packet, from where it can be collected via the network providers API. In the same connection, the LoRa Module will receive an update to stored settings or an action to open or close the window.

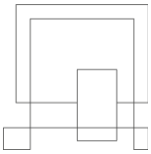
3.2 Data engineering

(Mark Vincze, Pryianka Shrestha, Maria Asenova, Constantina Tripon)

The data engineering design section will describe the choices of technologies and communication protocols used for developing the part of the system responsible for retrieving and storing data, but also with analysing and visualising it from a business perspective.

3.2.1 Amazon Web Services

For the development of the web service, AWS was used as software as a service (SaaS) because of its on-demand delivery of resources over the Internet. (amazon, 2021)



Three services were used, CodeBuild, Pipeline and Elastic Beanstalk (ES), these require a way of setting up the design in a manner that can utilise these.
Amongst these, ES is the most interesting design wise, and the rest will come in handy with the implementation part only - as of now the ES is responsible to hold the deployment of the build in the correctly configured environment.

3.2.2 Loriot Network server connection

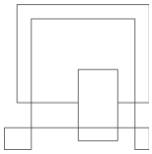
By its role as a middleware, there are two ways of connections on the “sides”. The Loriot server connection is established through a WebSocket API to listen for new data, also ready to send commands to the Device to start an actuator. This connection is responsible for the sending of data to the Device as well, by sending the data to the Loriot and queueing it for reception in the microcontroller.

3.2.3 Data Gateway Server Application

The responsibility of the gateway server application is to maintain an open connection to the LoRaWan server, to store the received data and to send it, upon request, to the android client. In this way, it acts like a client for the LoRaWan server and as a server for the android.

The application is built using a Spring Boot framework tool which helped improve and simplify the development of the web service. By using dependency injection, the components of the system became loosely coupled.

The JPA Repository module allowed for easy access and persistent data between the java object and relational database. Beyond that, it activates persistent exceptions. (spring, 2021)



The RESTful methods are handled by the Spring Boot framework allowing for a better integration of the system functionalities based on the requirements, built on the HTTP protocol optimising the communication with the android client by creating uniquely identifiable resources. (spring, 2021)

3.2.3.1 Design Patterns

Taking advantage of the Spring Boot framework, the Dependency Injection pattern was used such that the Inversion of Control principle to be implemented. One of the advantages of using this architecture is a decoupled system, where the concrete implementation is hidden from its execution. In this way, the SOLID principles were also followed during the development phase.

For a better understanding of the implementation of the design patterns, a low fidelity class diagram was developed (Figure 12).

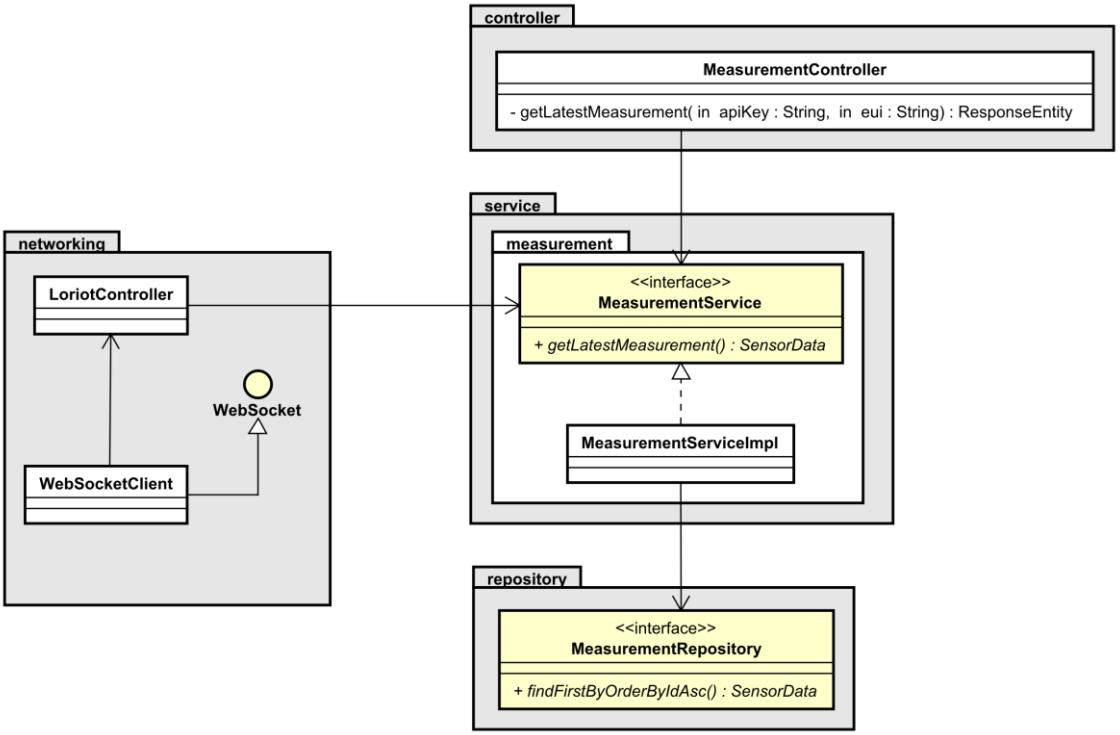


Figure 12: Low Fidelity Class Diagram

The package diagram shown below, describes the structure of the software and the separation of concerns inside the Spring Boot framework.

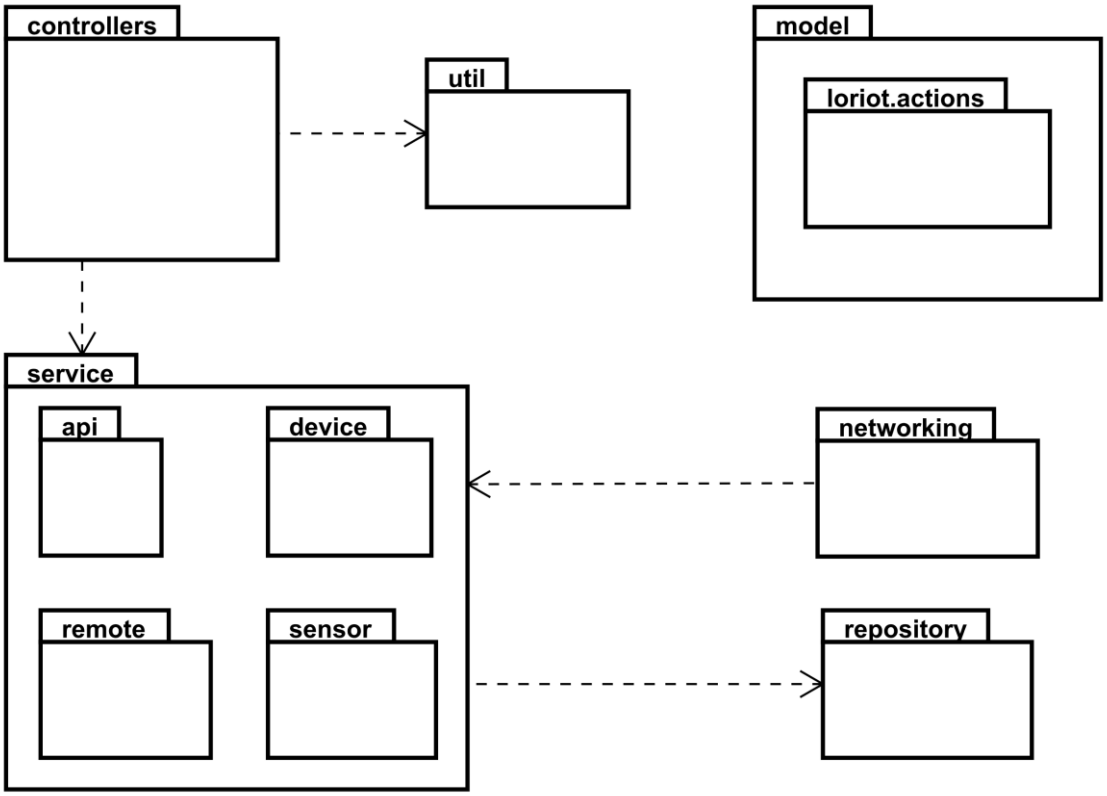
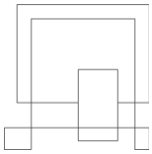


Figure 13: Package diagram Gateway Server Application

3.2.4 Microsoft SQL Server Database

The main function of the database server is to store and retrieve data as requested by other software applications, which may run either locally or across the network. (Microsoft , 2021)

The development environment to support dimensional modelling was also supported by the server database.



3.2.5 Dimensional modelling

A dimensional model is developed based on the business requirements of the user in connection to the data recorded.

3.2.5.1 Identify the business Process

Martin Høgh wants to focus on helping owners of greenhouses keep track of environmental variables for further development of their plants. Martin's greenhouse system sends measurements from each sensor every 5 minutes. He wants to show that data over a specific period to his customers, as statistics for them to keep track of the climate inside the greenhouse.

Martin would also like to be able to see the minimum, the maximum and the average values recorded for each sensor belonging to one device and compare the values to other devices.

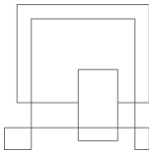
He would also like to be able to see the target values for each sensor, and the minimum and the maximum threshold values chosen by the customer and compare them to the recorded values of each sensor such that he can see how many times the recorded values of the sensor have reached the threshold values. In this way he can analyse the performance of each sensor.

3.2.5.2 Description of the grain

A single payload of measurements is the lowest granularity in the system. The grain is the data received from the server, recorded measurement from each sensor with a timestamp.

3.2.5.3 Identify the dimensions

The dimensions chosen are Dim_Measurement, Dim_Device, Dim_Time and Dim_Date.



These dimensions are derived from the description of the client as Martin needs to provide analysis of measurements over time to his customers. As the measurements need to be compared over time Date dimension is needed as well. Dim_Measurement is created based on the sensor_data table and Dim_Device is derived from the device table.

3.2.5.4 Identify the facts

The facts have been derived from the business requirements and identified as the individual measurement of each sensor with a timestamp.

3.2.5.5 Star Schema

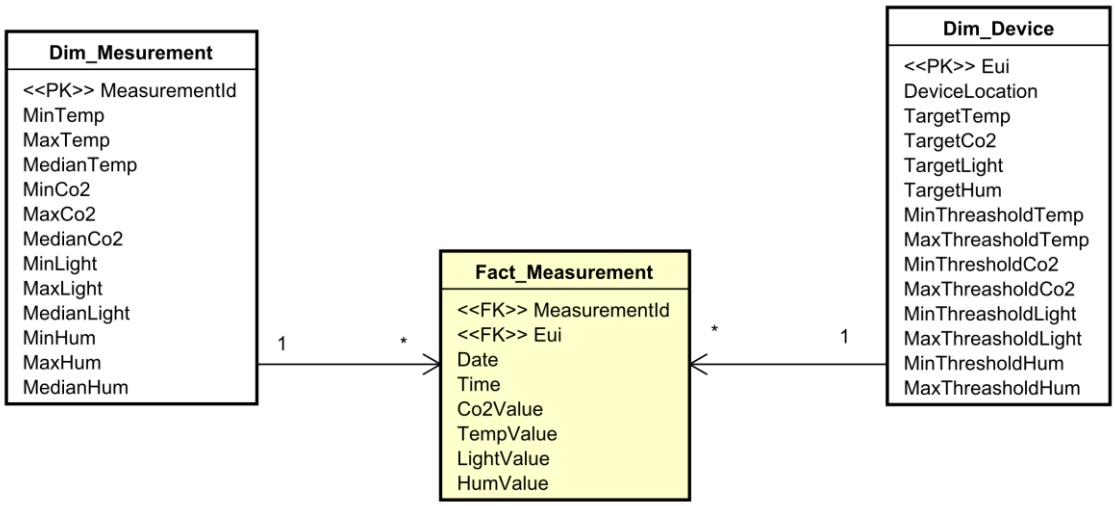


Figure 14: Star Schema - Initial Load

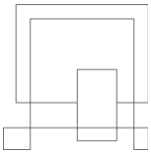
3.2.5.6 Source target mapping

A detailed analysis of the data process between source database and data warehouse is pictured below (Figure 16) for the fact table. The other dimensions can be found in the appendix section, Appendix: XX

Table Name	Fact_Measurement
Table Type	Dimension
Display Name	Dim_Sensor
Comment	
Size	25k

Column Name	Description	Unknown Member	Example values	Comments	Target							Source					
					Datatype	Size	Precision	Key?	FK To	NULL?	Default value	Column Name	Source System	Source Schema	Source Table	Source Field Name	Source Datatype
MeasurementId	Business key fom source system (aka natural key)	-1	1,2,3 ..		int	4 bytes		PK		N		id	GreenHouseDB	dbo	sensor_data		int
Eui	Business key fom source system (aka natural key)	-1	1,2,3 ..		nvarchar	16		PK	dim_location	N							
DateId	Business key fom source system (aka natural key)	-1	1,2,3 ..		int	4 bytes		PK	dim_date	N							
TimeId	Business key fom source system (aka natural key)	-1	1,2,3 ..		time	7		PK	dim_time	N							
Co2Value	The value recorder from the CO2 sensor	N/A	826		int	4 bytes						co2	GreenHouseDB	dbo	sensor_data		int
TempValue	The value recorder from the temperature sensor	N/A	29		int	4 bytes						temperature	GreenHouseDB	dbo	sensor_data		int
LightValue	The value recorder from the light sensor	N/A	9352		int	4 bytes						light	GreenHouseDB	dbo	sensor_data		int
HumValue	The value recorder from the humidity sensor	N/A	74		int	4 bytes						humidity	GreenHouseDB	dbo	sensor_data		int

Figure 16: Source to target mapping



3.2.5.7 ETL

The next section will describe the ETL architecture of the data warehouse and will end with an activity diagram which will highlight the steps taken during this process.

3.2.5.7.1 Extract

In this stage of the process, a data profiling analysis has been done, where all the data recorded in the source database has been loaded into the stage schema of the data warehouse. At this point no change data capture has been considered. (Kimball Group, 2009)

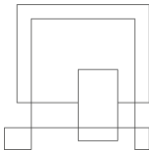
3.2.5.7.2 Transform

During the transformation stage, the data cleansing process occurs in the staging area, where all NULL value entries are updated with a default value such that upon analysis, the data quality is consistent and objective. (Pipino, u.d.)

3.2.5.7.3 Load

Surrogate keys have been added to individually identify each row of the dimension tables of the data warehouse. Surrogate keys offer support for type 2 slowly changing dimensions, are embedded in the fact tables as foreign keys, and improve indexing and query performance. (Group, 2006)

Persistent static date and time dimensions have been created such that the data is analysed as accurately and efficiently as possible. Static dimensions are less error



prone, easy to customise and because of the persistence, there is no need for a look up in the source database.

3.2.5.7.4 Star Schema for Data Warehouse

The diagram below depicts the changes done in the staging area such that the data is loaded in the data warehouse.

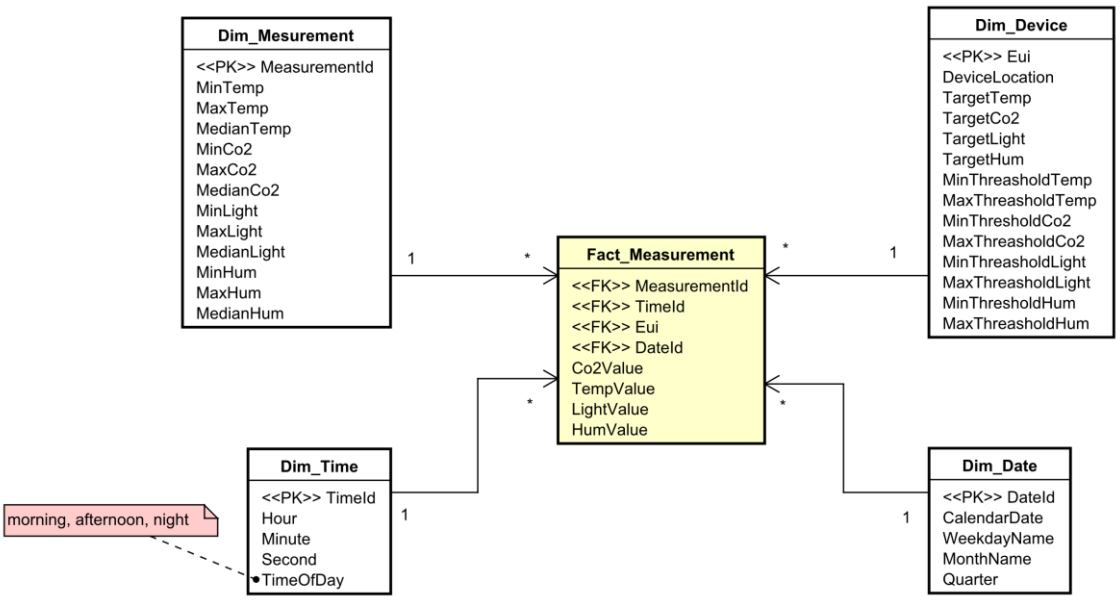
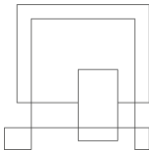


Figure 17: Data Warehouse Star Schema



3.2.5.7.5 Initial Load Activity Diagram

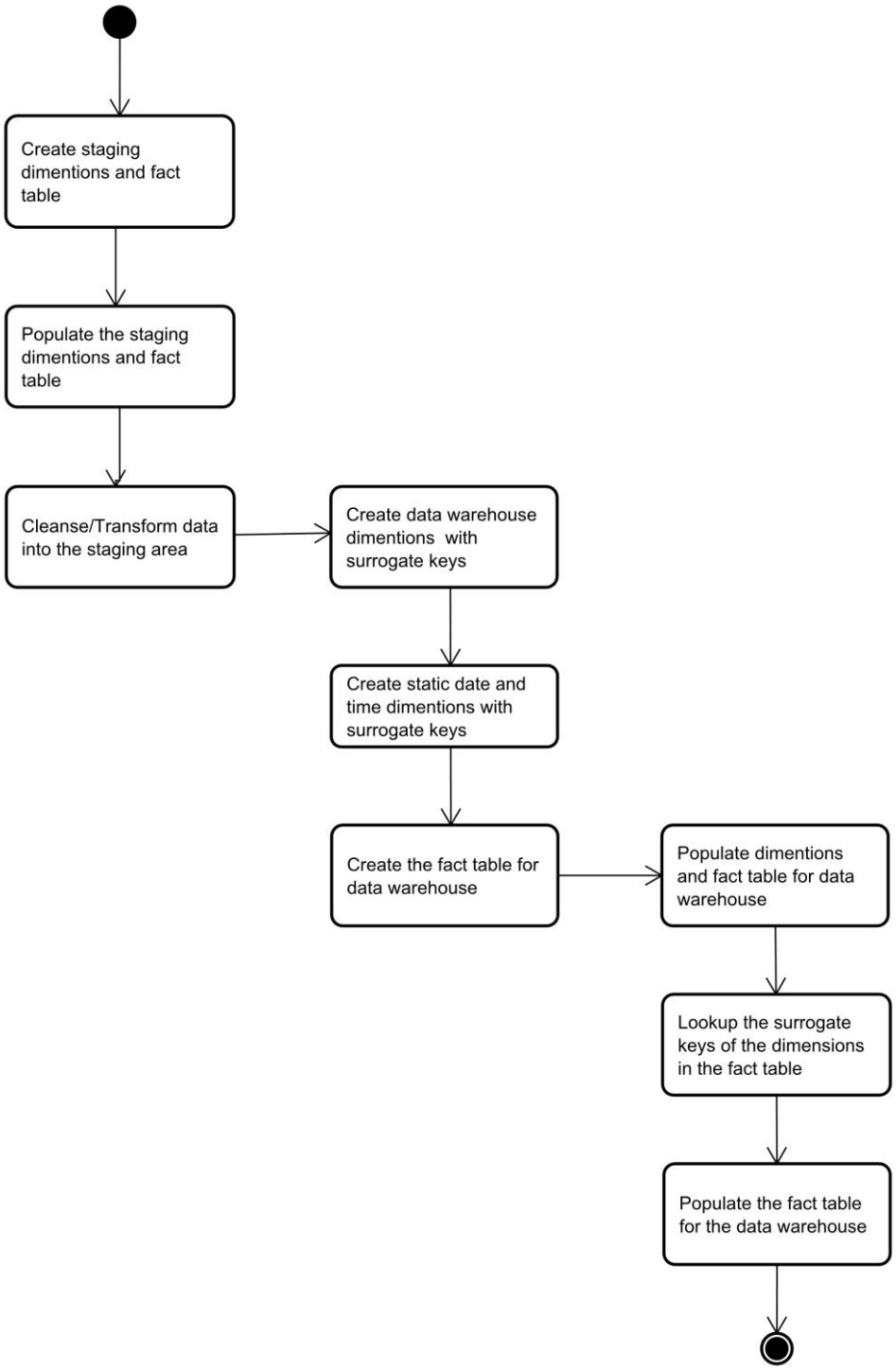
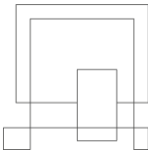


Figure 18: Activity Diagram - Initial Load



3.2.5.8 Type 2 changes

After loading the data from the staging area into the enterprise data warehouse, a change data capture system has been performed such that the data warehouse will always be updated with the latest new entries from the source database instead of performing a new loading every time a new entry is recorded.

A log mining approach was followed such that all changes in the database to be recorded. This approach creates a decoupling of the operational system and the data warehouse.

For a better overview of the incremental load process an activity diagram has been created, covering the cases of insertion, deletion and updates recorded inside the source database and processed such that they are also stored inside the data warehouse.

3.2.5.8.1 Incremental Load Activity Diagram

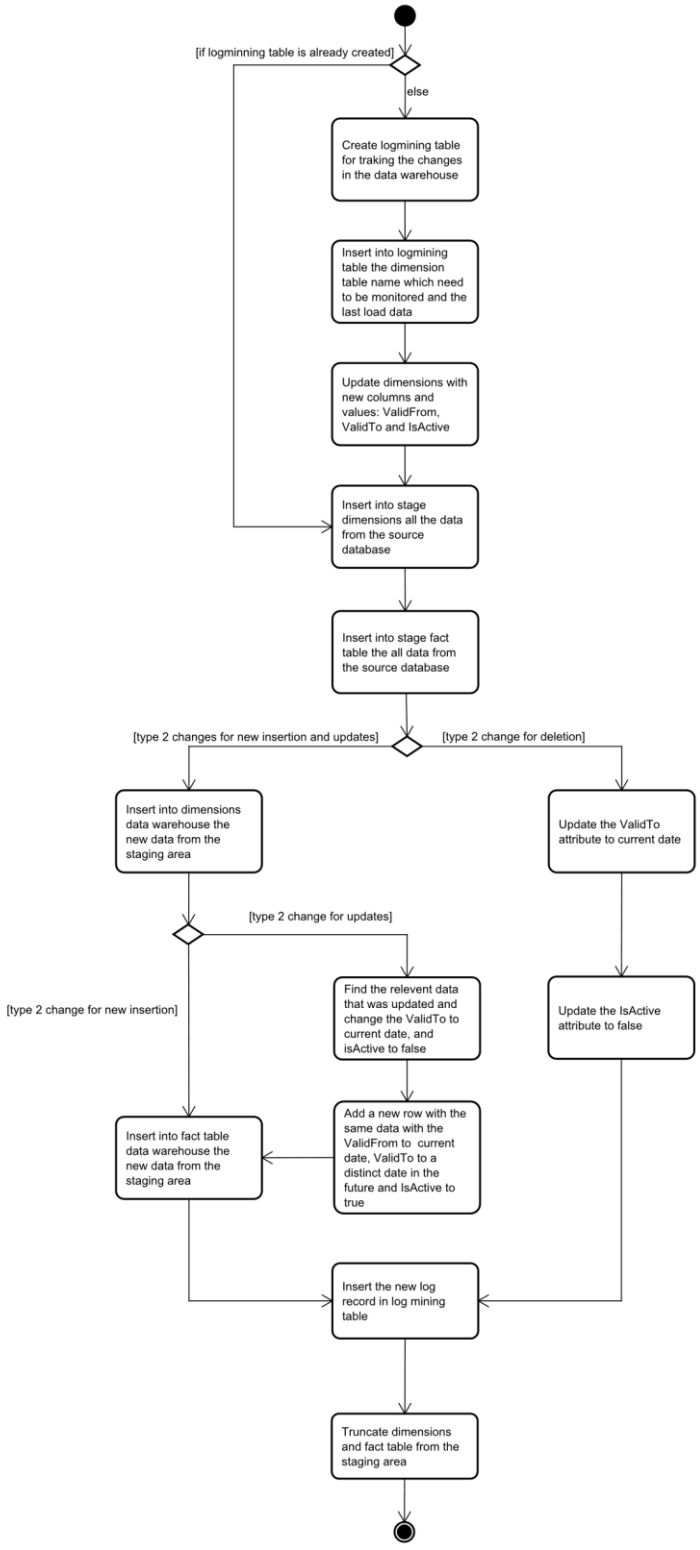
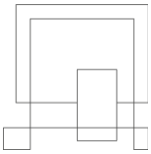


Figure 19: Activity Diagram- Incremental Load



3.2.6 Visualisation

After the dimensional modelling process and having in mind the business requirements, a visualisation analysis will help the data analyst have a better overview of the data collected. Visual Analytics can be perceived as an integrated approach that combines visualisation, human factors, and data analysis. (Digital Vidya, 2021)

Because the overall business requirement aims to achieve automation by threshold and environment values comparison graphs that change over time were the primary selection for data visualisation. For the same reason, a mean approach was followed such that the data set to be symmetrically distributed.

To be able to include all target audiences, the choice of colour consists of a colour-blind theme, also the cognitive bias was avoided by respecting the theory behind the SOR model. (Microsoft, 2021)

3.2.7 Installation guide

The installation guide for the data warehouse can be found in Appendix:XX
The installation guide for the Gateway Server can be found in Appendix:XX

3.3 Android

(Ilia Nikov, Tamás Péter, Daniel Railean)

This part of the report will focus on the design decisions made by the android team package and a general architecture diagram will be presented to give an introduction on the general application structure.

3.3.1 Architecture

The architectural pattern used for project development was MVVM architecture, used to separate the development of the user interface from the development of the back-end logic. (MVVM (Model View ViewModel) Architecture Pattern in Android - GeeksforGeeks, 2021). The MVVM was chosen because it is the official architecture with first party Android libraries and for the ease of understanding reading and maintaining.

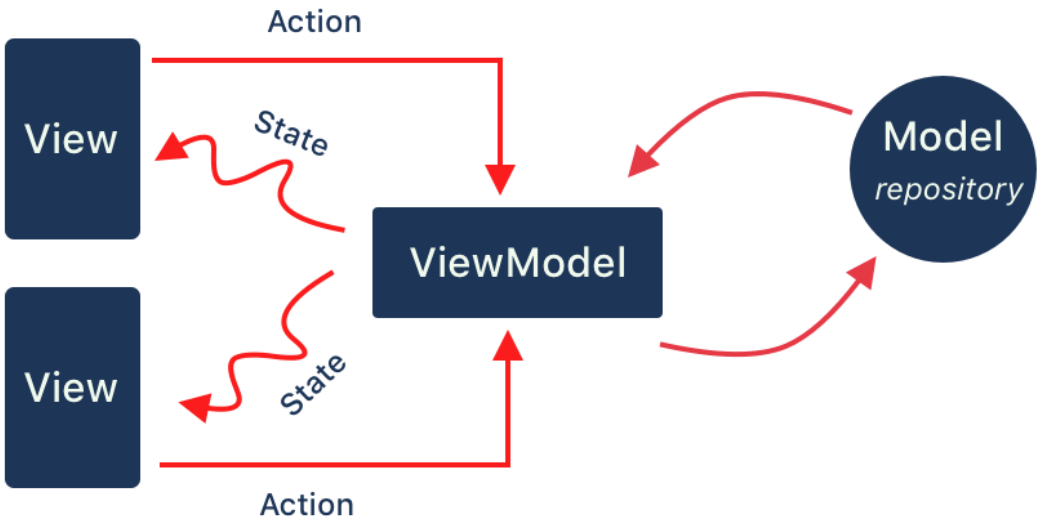
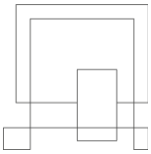


Figure 20: Architecture Diagram - Android

3.3.2 Technologies

The framework chosen for developing the client application was Android as per course requirements and to provide a native application performance and user experience.

For the communication between the client and the RESTful web service HTTP protocol was used as the client is not state dependent and a stateless architecture is well suited. The exchange of information was done through JSON objects that are being deserialized into Java Objects



The project requirements specify a caching procedure to allow use in offline mode, for that, the Room ORM together with SQLite database were used for the ease of implementation and a great support on android devices.

3.3.3 Design Patterns

During the development design patterns were use, the most notable being:

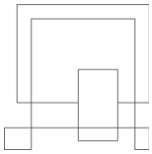
Builder design pattern was used for the creation of the Client WebAPI classes passing the URL, the converter factory and the http interceptor for passing the api-key for the data web service as well as passing the JWT for the android one

```
private static Retrofit retrofit = retrofitBuilder.client(client).build();
private static Retrofit retrofitAuth = authBuilder.build();
private static Retrofit retrofitAndroidData = authBuilder.client(clientJWT).build();
```

.

Adapter design pattern was used to transform Java Objects into UI elements such as DeviceListAdapter which transforms a list of devices into a scrollable list of UI cardboards, adapter pattern was also used to display items in a spinner.

```
@NonNull
@Override
public DeviceListAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.greenhouse, parent,  attachToRoot: false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    Device current = ghList.get(position);
    holder.gHItemTitle.setText(current.getLocation());
    holder.ghHum.setText(current.getLatest().humidity+" %");
    holder.ghTemp.setText(current.getLatest().temperature+" °C");
    holder.ghLight.setText(current.getLatest().light+" LUM");
    holder.ghCO2.setText(current.getLatest().co2+" PPM");
}
```



Repository pattern is found in the Data Access Objects which provides the access to the user data, as if it was in a local list, while in the back side, the repository changes between local and remote storage as well as makes all the required web-service calls.

```
private UserRepository(Application application){
    userAPI = ServiceGenerator.getUserAPI();
    authAPI = ServiceGenerator.getAuthAPI();
    loggedUser = new MutableLiveData<>();
    apiResponse = new MutableLiveData<>();
    userDevices = new MutableLiveData<>();

    loggedUser.setValue(new LoggedUser(LocalStorage.getInstance().get("access_token")));
    apiResponse.observeForever(s -> Log.e( tag: "API response user:" ,s));

    getLoggedUser();
}
```

Observer is the pattern used to trigger the rendering of UI elements on the data change providing a callback with the modified data source that can be used for update or reinitialization of the UI components.

```
devicesViewModel.getAll().observe(getViewLifecycleOwner(), adapter::updateData);
```

The package diagram for the greenhouse app is shown below. It shows the MVVM architecture and the dependencies between the packages.

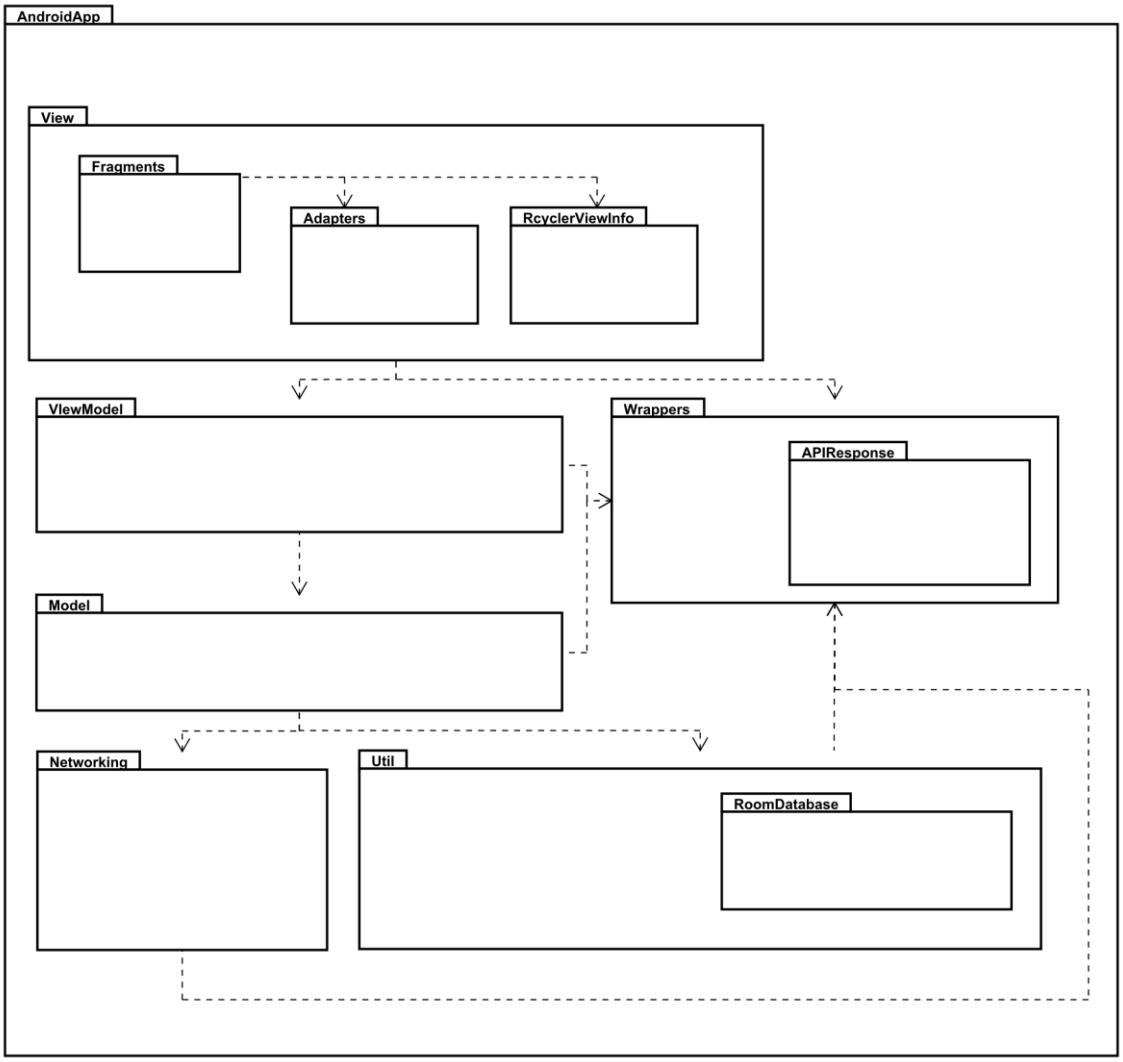


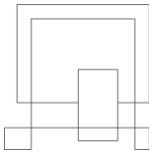
Figure 21: Package Diagram - Android

For more details the class diagram can be found in Appendix X

3.3.4 User Interface Sketches

User interface sketches were done in Figma, a free tool for designing user interfaces,

The team was then following the agreed upon design when implementing the native android layout files in xml format.



Below the sketches for the All devices page and add a new device page are shown, the interested reader can find the remaining sketches in the Appendix X.

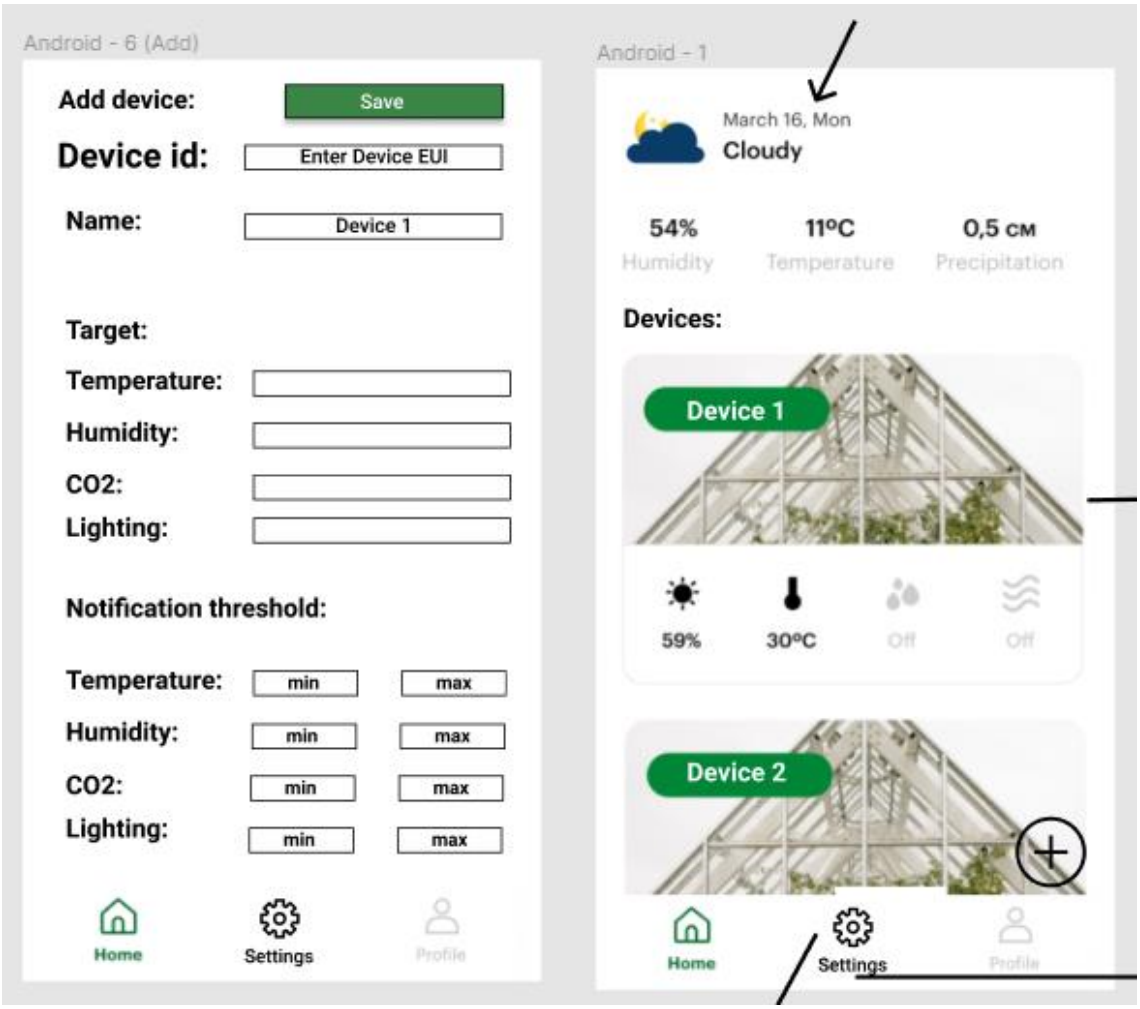
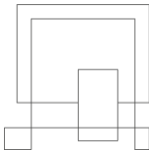


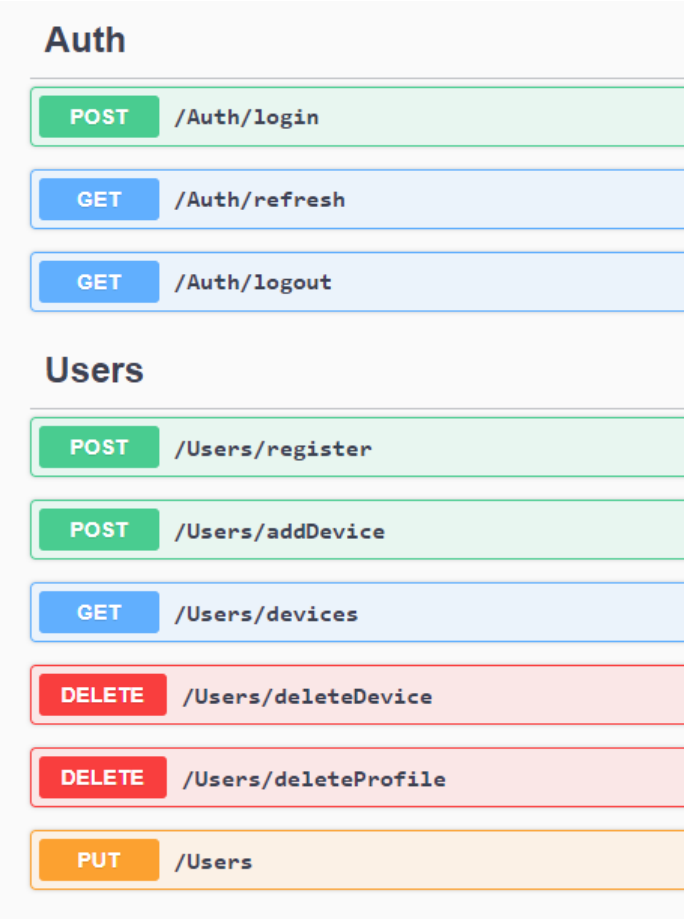
Figure 22 : Figma Prototype

3.3.5 Data models, persistence

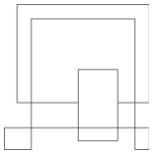
Two data models were used to persist user data, one for the User and one for the IOT devices. All user data was saved in a database and accessed through RESTful API calls over HTTP, the client is also caching the last response in an local SQLite database for use in case of connection loss. The data model responsibility is to provide



access to that data depending on current client conditions, through the repository pattern mentioned before.
The android client is making use of two web services, one of which was built by the data team, and one by the android team. A Brief overview of the android server can be seen below as well as in the OpenAPI specification present at greenauth.ddlele.com/swagger



The android app is using it when logging in and registering as well as adding and removing devices saving only the device eui. Environment and more detailed device information such as threshold and target values are saved in the data web service.



3.3.6 User Manual, Installation Guide

Interested readers can find the User Manual containing the interaction between the user and the app to achieve desired use-case as well as the Installation Guide containing the installation details for the app as well as for the android server in the Appendix X

4 Implementation

The overall implementation section will highlight, in sequential order, the functionality of the system in relation to software development. Each specialisation will describe the flow of events for the Monitor Data use case.

4.1 IoT

(Michel Sommer, Florina Toldea, Viggo Petersen)

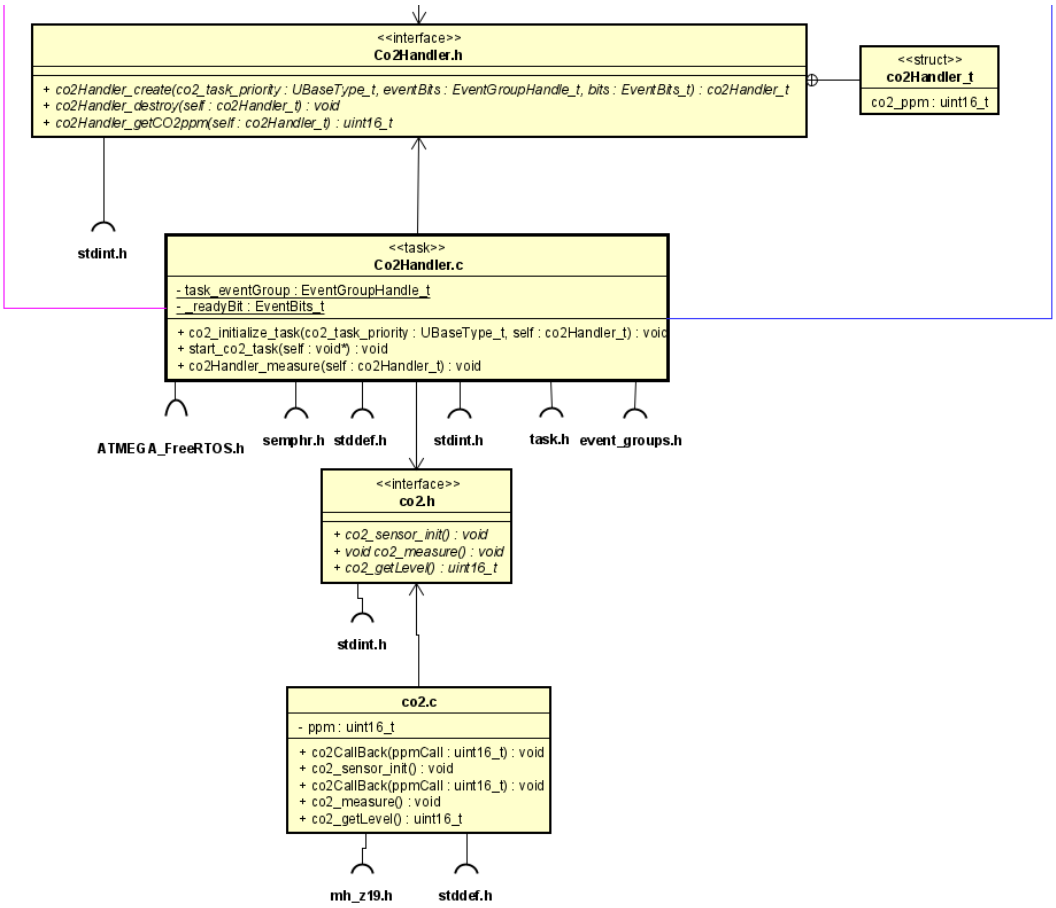
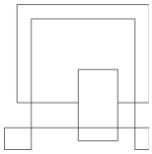


Figure 23: Class Diagram -IoT

To better understand the IoT implementation, the flow from the CO2 sensor to the LoRaWAN uplink will be explored. All the sensors have two parts, a controller named *co2.h* and a handler named *Co2Handler.h*. The controller is responsible for directly implementing the sensor in a testable state. These classes will have an initialization function, one or more measure functions for sensors and functions for getting the result in a proper format. Below is a snippet of the *co2.h* file.

```
11  #include <stdint.h>
12
13  void co2_sensor_init();
14  void co2_measure();
15  uint16_t co2_getLevel();
```



The handler classes are made as an abstraction layer for the sensors to initialize the FreeRTOS Tasks and contain the EventGroups along with any ADTs (Abstract Data Types) that may contain sensor results.

```
24 co2Handler_t co2Handler_create(UBaseType_t co2_task_priority, EventGroupHandle_t eventBits, EventBits_t bits)
25 {
26     co2Handler_t newSensor = calloc(1, sizeof(co2Handler));
27     if (newSensor == NULL) {
28         return NULL;
29     }
30
31     co2_sensor_init();
32     _readyBit = bits;
33     task_eventGroup = eventBits;
34     newSensor->co2_ppm = 0;
35     return newSensor;
36 }
```

Instead of using multiple semaphores we decided to try out the EventGroup from FreeRTOS because they seemed to fit in nicely with our design. They are used to indicate when an event occurs. In our case the event would be an update from the sensors. If we look at the measurement function below, the readyBits are awaited and then checked to see if the sensor is free, if so the co2 levels are measured and saved in the handler passed in. This could have also been done with semaphores.

```
71 void co2Handler_measure(co2Handler_t self) {
72
73     EventBits_t readyBits = xEventGroupWaitBits(task_eventGroup,
74     _readyBit,
75     pdFALSE,
76     pdTRUE,
77     portMAX_DELAY);
78
79     if ((readyBits & _readyBit) == _readyBit) {
80         co2_measure();
81         vTaskDelay(300);
82         self->co2_ppm = co2_getLevel();
83         xEventGroupSetBits(task_eventGroup, _readyBit);
84     }
85 }
```

If we move up one level in our diagram to the *application.h* class, where both the handlers and the LoRaWAN implementation are connected. The application is

considered the main connection point for the system, where handlers, LoRaWAN and the configuration are all connected and controlled.

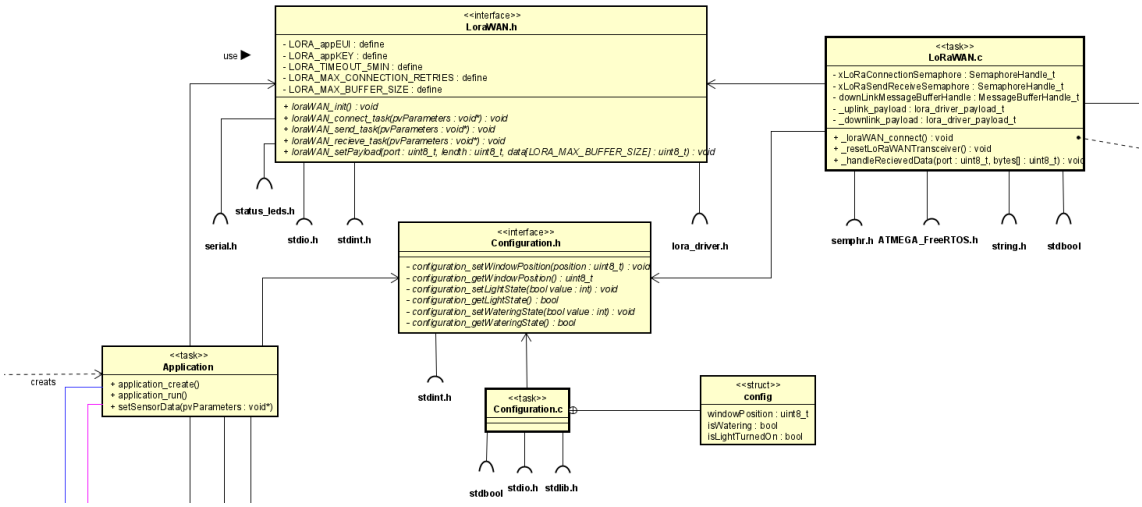
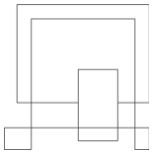


Figure 24: Class Diagram – IoT

A problem was encountered here when trying to initialize the handler tasks, that whenever 5 or more tasks were created the hardware would reset as soon as it started. The handlers' tasks worked whenever only 4 tasks were running so to test them the LoRaWAN tasks had to be removed. Below is a screenshot from HTerm of the sensor tasks measuring data, however the CO2 sensor seemed to always return 0 on #dev1, despite the driver returning OK.

```
co2 level: 0\n
hum level: 42\n
tem level: 23\n
0:23:42:0:0\n
MEASURE CO2 0\n
```

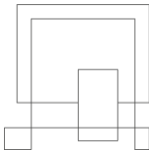
To get around the limitation of 4 tasks, all of the sensors were moved into a function called *_setSensorData* which measured data from all sensors, formatted it, and sent it



to the LoRaWAN implementation payload. It also took the last received payload from the configuration class and sent it to the servo for opening the window. This function ended up doing quite a lot of things because of this problem. Below is a picture of the sensor task and the out-commented sensors of how they should have been implemented.

```
64      xTaskCreate(
65          _setSensorData,
66          "Sensor_update_task",
67          configMINIMAL_STACK_SIZE + 200,
68          NULL,
69          SENSOR_TASK_PRIORITY, // priority 3
70          NULL);
71
72
73      /*
74      * BUG: Whenever we create 5 tasks no matter the stack size, the application halts
75      *
76      * The hardware seems to be running out of space on the stack, so because of the
77      * large stack size required by FreeRTOS. Therefore the sensors cannot have a
78      * task for them selves.
79      */
80      // co2_initialize_task(SENSOR_TASK_PRIORITY, _co2Sensor);
81      // light_initialize_task(SENSOR_TASK_PRIORITY, _co2Sensor);
82      // temperature_handler_initialise(SENSOR_TASK_PRIORITY, _co2Sensor);
83      // humidity_handler_initialise(SENSOR_TASK_PRIORITY, _co2Sensor);
```

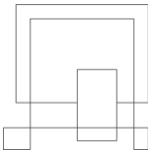
Moving on to the LoRaWAN implementation, as can also be seen in the class diagram picture above. This was implemented using semaphores as was required in the project. There are three main parts to the implementation, the connection part, the sending and the receiver. The connection part is mostly based on the example and is therefore not interesting. The sender however uses a combination of FreeRTOS tick counters and semaphores as shown below. There are two semaphores: connection semaphore and sendRecieve semaphore. When it is connecting to the LoRa network both semaphores are taken, so it cannot try to send or receive data. Once the hardware has connected successfully it will give up both semaphores and the other tasks can run. The sending task has the highest priority because packets are only received right after sending.



```
97  void loraWAN_send_task(void* pvParameters) {
98      // Setup timeout
99      TickType_t xLastWakeTime;
100     const TickType_t xFrequency = pdMS_TO_TICKS(LORA_TIMEOUT_5MIN);
101     xLastWakeTime = xTaskGetTickCount();
102
103     while(1) {
104         if (xSemaphoreTake(xLoRaSendReceiveSemaphore, 100) == pdTRUE) {
105             status_leds_shortPuls(led_ST4);
106             printf("Upload Message >%s<\n", lora_driver_mapReturnCodeToText(lora_driver_sendUploadMessage(false, &_uplink_payload)));
107             xSemaphoreGive(xLoRaSendReceiveSemaphore);
108             xTaskDelayUntil( &xLastWakeTime, xFrequency );
109         }
110     }
111 }
```

The receiver is very similar to the sender, except it sets the payload data in the configuration class, so it can be picked up by the application and sent on to the corresponding servo. In this case only the window servo has been implemented. The received data is based on which port it is sent to and a certain configuration value will be set. For the window servo a single byte will be sent on port 8, with a percentage of window position.

```
127 void loraWAN_recieve_task(void* pvParameters) {
128     TickType_t xLastWakeTime;
129     const TickType_t xFrequency = pdMS_TO_TICKS(500UL);
130     xLastWakeTime = xTaskGetTickCount();
131     while (1) {
132         if (xSemaphoreTake(xLoRaSendReceiveSemaphore, 100) == pdTRUE) {
133             xMessageBufferReceive(downLinkMessageBufferHandle, &_downlink_payload, sizeof(lora_driver_payload_t), 2000);
134             /* For debugging purposes only */
135             printf("DOWN LINK: from port: %s with %d bytes received!\n", _downlink_payload.portNo, _downlink_payload.len);
136             for (int i = 0; i < _downlink_payload.len; i++)
137             {
138                 if (i > 0) printf(":");
139                 printf("%02X", _downlink_payload.bytes[i]);
140             }
141             printf("\n");
142
143             _handleRecievedData(_downlink_payload.portNo, _downlink_payload.bytes);
144             xSemaphoreGive(xLoRaSendReceiveSemaphore);
145             xTaskDelayUntil( &xLastWakeTime, xFrequency );
146         }
147     }
148
149 }
```



4.2 Data Engineering

(Mark Vincze, Pryianka Shrestha, Maria Asenova, Constantina Tripon)

The following section, for the data engineering specialisation, will start with the implementation of the Loriot Network server connection and the steps to persist data in the data warehouse. The ETL process will also be documented together with the analysis of the visualisation required by the data analyst in relation to the *Monitor and Filter Data* use case description. The key points from the RESTful architecture as well as the web socket implementation will be discussed

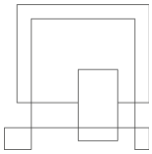
4.2.1 Data Gateway Server Application

The implementation of the server application is explained sequentially. The following section shows the implementation of the connection with the LoRaWan server and the process of persisting sent data and sending it to the Android client.

4.2.1.1 Loriot Network server connection

The implementation of the gateway server application begins with establishing the connection with the LoRaWan server. The communication uses Web Sockets API and the class implementation in Java was provided. The only addition to the original design is the implementation of the PropertyChangeSubject interface, provided by the Java beans so the server application is always listening for new messages.

```
@Override
public void addPropertyChangeListener(String eventName, PropertyChangeListener listener) {
    support.addPropertyChangeListener(eventName, listener);
}
```

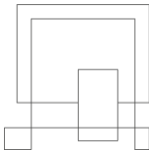


When a new measurement is received by the web socket listener, the data is passed to the LoriotController class.

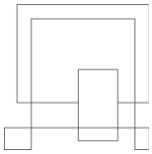
```
public void receiveData(PropertyChangeEvent event) {
    String receivedString = event.getNewValue().toString();
    logger.info("Received data {}", receivedString);
    UpLink message = gson.fromJson(receivedString, UpLink.class);
    if (message.getCmd().equals(Constants.RECEIVE_COMMAND))
        receiveMessage(message);
}
```

The data is first extracted into a String, and deserialized using the Gson library developed by google into an UpLink class model object.

After the data String was processed, the individual measurements were separated into relevant data and encapsulated into a SensorData model object.



```
private SensorData processData(UpLink message) {
    SensorData data = new SensorData();
    String[] parts = new String[5];
    Matcher matcher = Pattern.compile(Constants.MESSAGE_REGEX)
            .matcher(message.getData());
    if (matcher.matches()) {
        for (int i = 0; i < matcher.groupCount(); i++) {
            parts[i] = matcher.group(i + 1);
        }
    }
    int temp = 0;
    int hum = 0;
    int co2 = 0;
    int light = 0;

    for (int i = 0; i < parts.length - 1; i++) {
        if (i == 0) {
            temp = Integer.parseInt(parts[i], radix: 16);
        } else if (i == 1) {
            hum = Integer.parseInt(parts[i], radix: 16);
        } else if (i == 2) {
            co2 = Integer.parseInt(parts[i], radix: 16) * 2;
        } else {
            light = Integer.parseInt(parts[i], radix: 16) * 4;
        }
    }

    data.setHumidity(hum);
    data.setCo2(co2);
    data.setLight(light);
    data.setTemperature(temp);
    data.setDate(processTimestamp(message));
    data.setEui(message.getEui());
    return data;
}
```



The encapsulated object is sent to the measurement service.

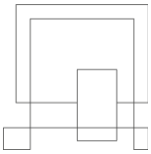
```
private void receiveMessage(UpLink message) {
    SensorData data = processData(message);
    logger.info("Received message: {}", data);
    try {
        measurementService.addMeasurement(data);
    } catch (Exception e) {
        logger.info("Data could not be processed");
        e.printStackTrace();
    }
}
```

The service acts as a mediator between the controller and the repository with the only purpose of holding the logic of the system.

```
@Override
public SensorData addMeasurement(SensorData data) throws NotFoundException {
    try {
        return measurementRepository.save(data);
    } catch (Exception e) {
        throw new NotFoundException("Data could not be saved");
    }
}
```

The repository makes use of all the APIs for basic CRUD operations by extending the JpaRepository.

```
@Repository
public interface MeasurementRepository extends JpaRepository<SensorData, Integer>
```



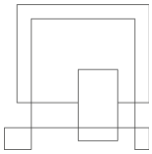
4.2.1.2 RESTful Web Server

This section documents the flow of events for displaying the data persisted as well as saving the threshold and target values set by the user of the Android application.

The rest controller is responsible for the execution of the HTTP protocol. By consuming the API's, the Android client is triggering the resource needed, in the current example, it targets the latest measurement.

```
@RestController
@RequestMapping(path = "/measurements")
public class MeasurementController {
    @Autowired
    private MeasurementService measurementService;

    @Autowired
    private ApiKeyUtil util;

    @Operation(summary = "Get the latest measured data")
    @GetMapping(path ="/{eui}/latest", produces = "application/json")
    public ResponseEntity getLatestMeasurement(@RequestHeader("api-key") String apiKey,
                                               @PathVariable String eui) {
        try {
            util.checkApi(apiKey);
            return new ResponseEntity<>(measurementService.getLatestMeasurement(eui),
                    HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT_FOUND);
        }
    }
```

The service's only responsibility is to act as a middleman between the controller and the repository.



```
@Override
public SensorData getLatestMeasurement(String eui) throws NotFoundException {
    try {
        return measurementRepository.findFirstByEuiOrderByIdDesc(eui);
    } catch (Exception e) {
        e.printStackTrace();
        throw new NotFoundException("Data could not be found ");
    }
}
```

The getLatestMeasurement() method in the MeasurementserviceImpl throws a NotFoundException exception which is catched in the MeasurementController which sends a NOT_FOUND http status code to the client.

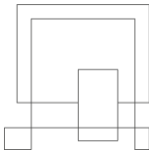
4.2.2 Microsoft SQL Server Database

The persisted data is saved in the remote source database. For the project, two tables were created in the source database, namely device and sensor data. The device contains the threshold values and the target values which are set by the user whereas the sensor data consists of the actual data that was collected from the sensors located in the greenhouse. The following process explains how the data is handled and persisted to the data warehouse.

4.2.2.1 ETL - Initial Load

The Extract, Transform and Load implementation started by creation of the database GreenhouseDataWarehouse where two schemas were created, stage and edw.

The first step was the extraction of the data which was done by creating the dimension tables in the stage schema such as Dim_Device and Dim_Measurement. The data was then loaded from the source database into the stage tables. Taking into consideration

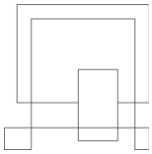


the single grain of the business process, the Fact_Measurement table was designed and loaded.

```
TRUNCATE TABLE [stage].[Fact_Measurement]
INSERT INTO [stage].[Fact_Measurement]
           ([DateId]
           ,[MeasurementId]
           ,[Eui]
           ,[TimeId]
           ,[Co2Value]
           ,[TempValue]
           ,[LightValue]
           ,[HumValue])
SELECT  CAST(s.date AS DATE),
           s.[id],
           s.[Eui],
           CONVERT(NVARCHAR(10), s.date, 108),
           s.[co2],
           s.[temperature],
           s.[light],
           s.[humidity]
 FROM [GreenHouseDB].[dbo].[device] d
 INNER JOIN [GreenHouseDB].[dbo].[sensor_data] s
 ON s.[Eui] = d.[Eui]
```

For the second phase of the process that is the Transformation phase the data was cleansed. The cleansing of the data was done by updating all the NULL values that might have been updated in the stage tables from the source database. The application has a method / function where the data was put to some dummy value (-4444) if the data was out of bound upon receiving from the LoRaWAN server. Hence the cleansing of data was done by updating the values to an average value of the respected data/sensor/column. Therefore, the data was restrained from affecting other values such as the minimum, maximum and average of the achieved result.

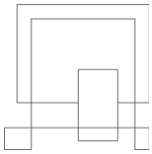
```
UPDATE [stage].[Fact_Measurement] SET Co2Value = '4999' WHERE Co2Value = '-4444'
UPDATE [stage].[Fact_Measurement] SET TempValue = '0' WHERE Co2Value = '-4444'
UPDATE [stage].[Fact_Measurement] SET LightValue = '65000' WHERE Co2Value = '-4444'
UPDATE [stage].[Fact_Measurement] SET HumValue = '49' WHERE Co2Value = '-4444'
```



After transformation, the loading of the data in the edw schema was set up. The edw tables were created such that the tables contained the surrogate keys as the primary key of the table. When loading the Fact_Measurement table in the edw schema, the key lookup feature/process was used.

```
INSERT INTO [edw].[Fact_Measurement]
    ([D_ID],
     [M_ID],
     [DV_ID],
     [T_ID],
     [Co2Value],
     [TempValue],
     [LightValue],
     [HumValue])
SELECT d.[D_ID],
     m.[M_ID],
     dv.[DV_ID],
     t.[T_ID],
     f.[Co2Value],
     f.[TempValue],
     f.[LightValue],
     f.[HumValue]
FROM [stage].[Fact_Measurement] AS f
    INNER JOIN [edw].[Dim_Date] AS d
 ON d.[DateId] = f.[DateId]
     INNER JOIN [edw].[Dim_Device] AS dv
 ON dv.[Eui] = f.[Eui]
    INNER JOIN [edw].[Dim_Measurement] AS m
 ON m.[MeasurementId] = f.[MeasurementId]
    INNER JOIN [edw].[Dim_Time] AS t
 ON t.[TimeId] = f.[TimeId]
```

There are two dimensions which are made static, Dim_Date and Dim_Time. The loading of the time dimension can be seen below, and the date dimension can be found in Appendix:XX

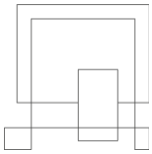


```
DECLARE @Time AS [TIME];
SET @Time = '00:00:00';

DECLARE @counter AS int;
SET @counter = 0;


WHILE @counter < 86400
BEGIN
    INSERT INTO  [edw].[Dim_Time] (
                          TimeId
                        , Hour
                        , Minute
                        , SECOND
                        , TimeOfDay)
                VALUES (@Time
                        , DATEPART(HOUR, @Time)
                        , DATEPART(MINUTE, @Time)
                        , DATEPART(SECOND, @Time)
                        ,CASE WHEN (@Time >= '00:00:00' AND @Time < '06:00:00') THEN 'Night'
                          WHEN (@Time >= '06:00:00' AND @Time < '12:00:00') THEN 'Morning'
                          WHEN (@Time >= '12:00:00' AND @Time < '18:00:00') THEN 'Afternoon'
                          ELSE 'Evening'
                          END
                    );

    SET @Time = DATEADD(SECOND, 1, @Time);

    set @counter = @counter + 1;
END
```

4.2.2.2 ETL - Incremental Load

After the initial load was successfully completed, the incremental load of the ETL process started.

The strategy of incremental load was taken into consideration to handle the data change in the source database. There were three major changes that have been implemented: new insertion, deletion and changed data.

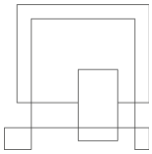


While a new data is added, the stage tables were loaded with all the data from source and then the edw tables were loaded with only the newly loaded data from the stage tables.

```
INSERT INTO [edw].[Fact_Measurement]
    ([D_ID],
     [M_ID],
     [DV_ID],
     [T_ID],
     [Co2Value],
     [TempValue],
     [LightValue],
     [HumValue])
SELECT d.[D_ID],
     m.[M_ID],
     dv.[DV_ID],
     t.[T_ID],
     f.[Co2Value],
     f.[TempValue],
     f.[LightValue],
     f.[HumValue]
FROM [stage].[Fact_Measurement] AS f
    INNER JOIN [edw].[Dim_Date] AS d
 ON d.[DateId] = f.[dateId]
     INNER JOIN [edw].[Dim_Device] AS dv
 ON dv.[eui] = f.[eui]
    INNER JOIN [edw].[Dim_Measurement] AS m
 ON m.[measurementId] = f.[measurementId]
    INNER JOIN [edw].[Dim_Time] AS t
 ON t.[timeId] = f.[timeId]
 WHERE m.[M_ID] NOT IN (SELECT [M_ID] FROM [edw].[Fact_Measurement])
```

After which the log mining table was updated with the updated table name and current date to keep track of the last update made in the data warehouse.

```
DECLARE @NewLoadDate [DATE]
SET @NewLoadDate = CONVERT([DATE], GETDATE())
INSERT INTO [etl].[LogUpdate] ([Table], [LastLoadDate])
VALUES ('Dim_Device', @NewLoadDate)
```



The deletion of the data in source was handled in a way that the data remained in the data warehouse but as historical data. For that the validity of the deleted data was changed to the current date and the isActive status was marked to inactive in the respected edw table.

```
DECLARE @NewLoadDate date
SET @NewLoadDate = CONVERT(date, GETDATE())
DECLARE @RowIndicator BIT
SET @RowIndicator = 0
UPDATE [edw].[Dim_Device] SET ValidTo = @NewLoadDate, IsActive = @RowIndicator
WHERE eui NOT IN (SELECT eui FROM stage.Dim_Device)
```

When the data was changed in the source database, the query was made to search for the specific data in the edw table and the validity of the row was changed to current date as well as the isActive status was marked to inactive so that the data remains in the data warehouse as historical data.

```
DECLARE @NewLoadDate [DATE]
SET @NewLoadDate = CONVERT([DATE], GETDATE())
DECLARE @FutureDate [DATE]
SET @FutureDate = '2023-12-31'
DECLARE @RowIndicator [BIT]
SET @RowIndicator = 1
UPDATE [edw].[Dim_Device] SET [ValidFrom] = @NewLoadDate, [ValidTo] = @FutureDate, [IsActive] = @RowIndicator
WHERE [ValidFrom] IS NULL
```

And the same data with the new changes was treated as if the data was newly added data. Hence the new row was added in the edw dimension table following the loading in the fact table. The use of surrogate key as the primary key was particularly very useful in this scenario.

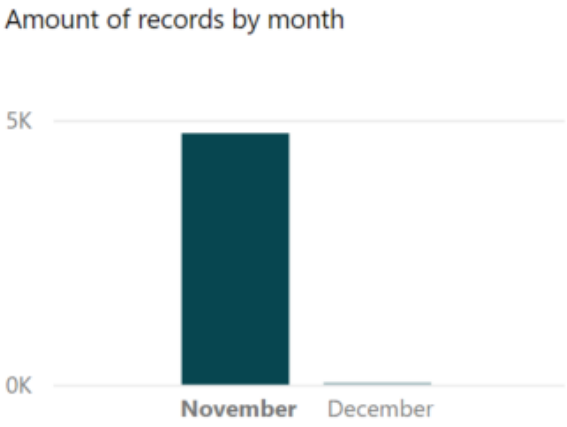
```
DECLARE @NewLoadDate [DATE]
SET @NewLoadDate = CONVERT([DATE], GETDATE())
INSERT INTO [etl].[LogUpdate] ([Table], [LastLoadDate])
VALUES ('Dim_Device', @NewLoadDate)
```

4.2.3 Visualisation

This section will describe the interactive data visualisation in relation to customer requirements.

The data can be filtered by months using the bar chart shown below (Figure 24) It also shows how many records can be found in the specific month. If clicked, on one of the months, it will only show data on the dashboard for that month.

The same is implemented for the matrix that holds the active devices and to be used as a filter for only to show the information on the dashboard by selected device.



Devices by EUI

Eui	Number of active devices
0004A30B00259D	1
0004A30B00259D2	1
0004A30B00259D2C	1
2021-15-12T02:16	1
aoe	2
DUMMO30B00259D2B	1
DUMMP30B00259D2B	1
DUMMY30B00259D2B	1
Total	**9**

Figure 24: Dashboard PowerBI

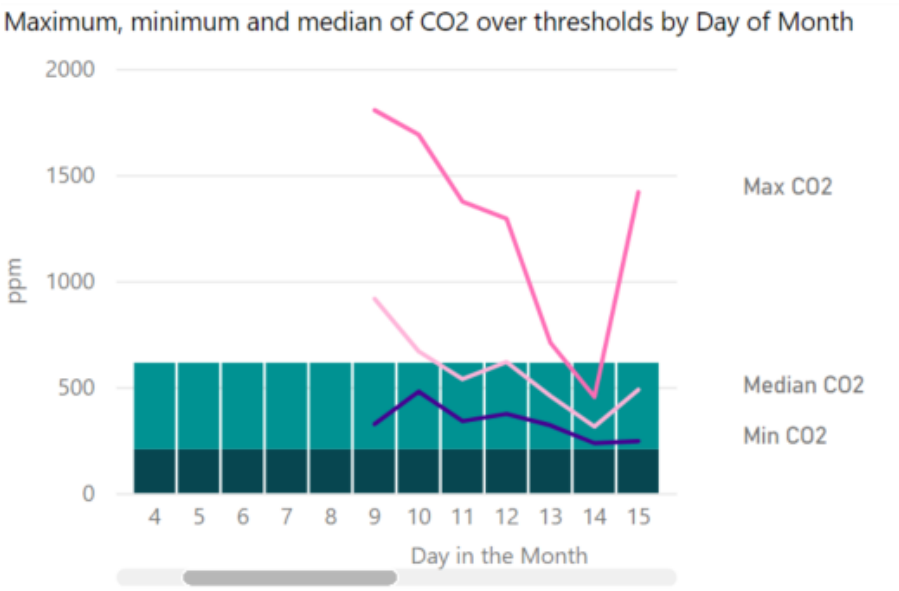


Figure 25: Dashboard PowerBI

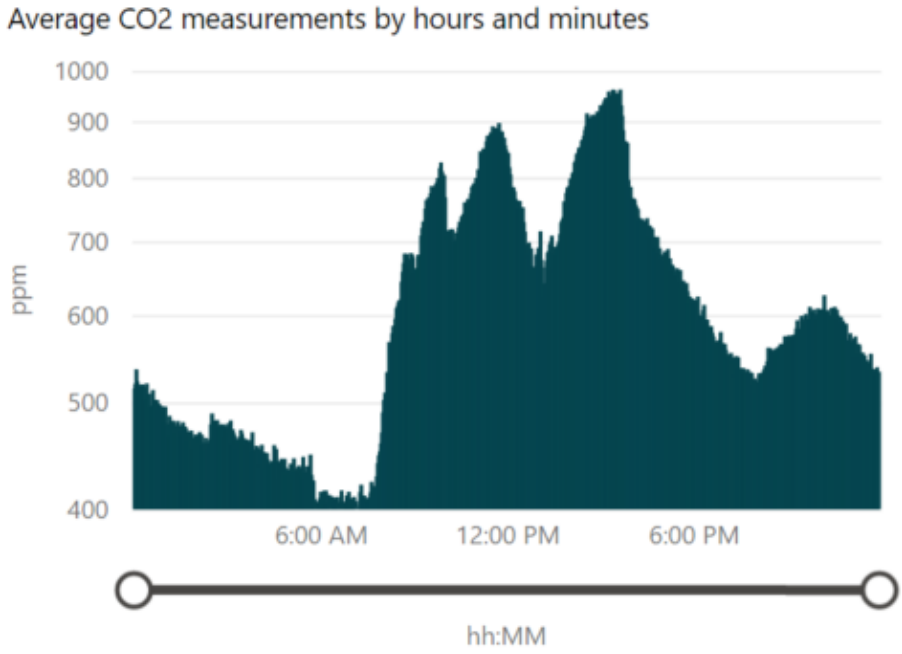
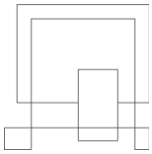


Figure 26: Dashboard PowerBI



4.3 Android

(Ilia Nikov, Tamás Péter, Daniel Railean)

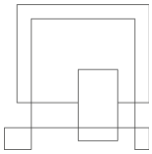
Main connection with the device list:

The first step for the Greenhouse app was to establish connection with the database. A RESTful web service and a HTTP protocol was used to make this connection.

```
public interface DeviceAPI {
    @GET("measurements/{deviceId}/latest")
    Call<GreenData> getLastData(@Path("deviceId")String deviceId);

    @GET("measurements/{deviceId}/periodic")
    Call<List<GreenData>> getIntervalData(@Path("deviceId")String eui,
                                          @Query("start") String start, @Query("end") String end);

    @GET("devices/{deviceId}")
    Call<Device> get(@Path("deviceId")String deviceId);

    @DELETE("devices/{deviceId}")
    Call<Device> delete(@Path("deviceId")String deviceId);

    @POST("devices")
    Call<Device> create(@Body Device created);

    @PUT("devices")
    Call<Device> update(@Body Device created);

    @POST("remote/{deviceId}/window")
    Call<String> windowPosition(@Path("deviceId")String eui,@Query("commandPercentage")int value);


    @POST("remote/{deviceId}/water")
    Call<String> waterControl(@Path("deviceId")String eui,@Query("waterValue")int value);


    @POST("remote/{deviceId}/light")
    Call<String> lightControl(@Path("deviceId")String eui,@Query("lightValue")int value);


}
```

The Database specialization provided the path to the server.

```
private static Retrofit.Builder retrofitBuilder = new Retrofit.Builder()
        .baseUrl("http://gatewayserver-env.eba-jv7rk7pv.eu-central-1.elasticbeanstalk.com/")
        .addConverterFactory(GsonConverterFactory.create(gson));
```



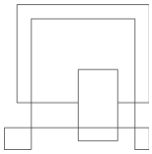
This is in the getAll method which returns a live data object containing either data cached in the database or fresh data from the API.

```
public LiveData<List<Device>> getAll(){
    fetchDevices();
    return allDevices;
}
```

The fetchDevices() method contains two chained API-requests and is used to retrieve the devices one by one then retrieve the data for each one, the device Id of a specific user are saved in the same web service that handles the authentication. A freshly initialized ArrayList is used to save retrieved Objects from the API, as the endpoint to get all devices is not exposed in the data web service now.
The current devices list is updated from the deviceAPI and later the updated values are cached. In the event of missing Environment data, the displayed values would be -1.

```
if (response.isSuccessful()) {
    current[0].setLatest(response.body());
    currentAll.add(current[0]);
    allDevices.setValue(currentAll);
    cacheDevices(currentAll);
    apiResponse.setValue("Device found!");
    cacheDevices(currentAll);
    Log.e( tag: "green response green-data", msg: "call: "+response.body());
    Log.e( tag: "localStorage: ",  msg: "update: " + currentAll);
    Log.e( tag: "deviceAPI response", msg: "call: "+current[0]);
    Log.e( tag: "deviceAPI all devices: ",  msg: allDevices.getValue().size()+"");
} else {
    current[0].setLatest(new GreenData());
    currentAll.add(current[0]);
    allDevices.setValue(currentAll);
    cacheDevices(currentAll);
    apiResponse.setValue("Unable to get data for "+ userDevices.get(finalI).getEui());
    Log.e( tag: "green response", msg: "call: "+response.code()+" "+response.message());
    Log.e( tag: "green response", msg: "call: "+response.raw().toString());
}
```

The DevicesViewModel has the instance of the DeviceRepository so it can provide the Views with the proper data.



```
public class DevicesViewModel extends AndroidViewModel {
    DeviceRepository repository;
```

In the start activity a bottom navigation bar is created with three fragments:

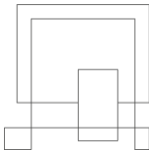
- The Home Fragment which is the first view after the login/registration process. This fragment has the list of the devices and adding a new device.
- The Settings Fragment where the threshold values can be set up for each device.
- The Profile Fragment where the user can change their password.
-

```
BottomNavigationView navView = findViewById(R.id.nav_view);
// Passing each menu ID as a set of Ids because each
// menu should be considered as top level destinations.
AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(
        R.id.navigation_home, R.id.navigation_settings, R.id.navigation_profile)
        .build();
NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment_activity_main_page);
NavigationUI.setupActionBarWithNavController( activity: this, navController, appBarConfiguration);
NavigationUI.setupWithNavController(binding.navView, navController);
```

In the Home Fragment the DeviceViewModel instance observes the changes and fetches the data for the recycler list according to it.

```
devicesViewModel.getAll().observe(getViewLifecycleOwner(), new Observer<List<Device>>() {
    @Override
    public void onChanged(List<Device> devices) {
        DeviceListAdapter adapter = new DeviceListAdapter(devices);
        recyclerViewMainPage.setAdapter(adapter);
    }
});
```

This made possible by the DeviceListAdapter what can bind the data with the list using the ViewHolder inner class



```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    Device current = ghList.get(position);
    holder.gHItemTitle.setText(current.getLocation());
    holder.ghHum.setText(current.getLatest().humidity+" %");
    holder.ghTemp.setText(current.getLatest().temperature+" °C");
    holder.ghLight.setText(current.getLatest().light+" LUM");
    holder.ghCO2.setText(current.getLatest().co2+" PPM");
}
```

Local Data Storage

The local data storage was made using the Room persistence library from Jetpack. First the notations for the device entity were added to tell Room which the entity is to work with and how.

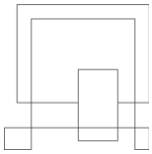
```
@Entity(tableName = "device_table")
public class Device {
    @PrimaryKey @NonNull
    public String eui;
    public String location;
```

The GreenData object was a nested or embedded object so it was notated as well.

```
@Embedded
public GreenData lastData;
```

Next the Data Access Object was made to provide an abstract access to the local database made by room.

```
@Dao
public interface DeviceDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insert(Device device);
```

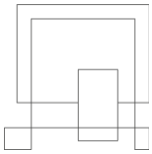


```
    @Query("DELETE FROM device_table")
    void deleteAll();

    @Query("SELECT * FROM device_table")
    LiveData<List<Device>> getAllLocal();

}
```

Next the Database was created. A singleton class was used for the simplicity and to make sure there is only one database but later this can be swapped to a class using dependency injection.

```
@Database(entities = {Device.class}, version = 2)
public abstract class GreenHouseDatabase extends RoomDatabase {
    private static GreenHouseDatabase instance;
    public abstract DeviceDao getDeviceDao();

    public static synchronized GreenHouseDatabase getInstance(Context context){
        if(instance == null){
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    GreenHouseDatabase.class, "greenhouse_database")
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return instance;
    }
}
```

At the current phase of the app, the local database rewrites the last deviceData that the user successfully retrieved and then sets it to the Observable Data in the case that the internet connection went down.



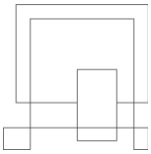
```
if (userDevices.size() == 0) {
    deviceCache.getAllLocal().observeForever(devices -> {
        if (!androidClient.isNetworkAvailable()) {
            allDevices.setValue(devices);
        }
    });
    return;
}
```

```
public void cacheDevices(List<Device> devices){
    clearCache();
    for(int i = 0; i < devices.size(); i++){
        int finalI = i;
        executorService.execute(() -> deviceCache.insert(devices.get(finalI)));
    }
}
```

To optimize the local database a better approach would be updating instead rewriting, but for a small number of devices, performance should not be an issue, as it is not handled in the main thread and the User Experience is maintained.

Device Info with Chart

The DeviceInfoFragment shows the latest measurements of a single device. The measurements are shown in a grid recycler view where each measurement and its information are represented in a small green box called indicator.



The information that the indicators display is retrieved by sending a bundle from the Home Fragment which contains all the latest measurements of the device chosen from the list.

```
Bundle bundle = new Bundle();
bundle.putString("eui", adapter.ghList.get(clickedItemIndex).getEui());
bundle.putString("location", adapter.ghList.get(clickedItemIndex).getLocation());
bundle.putInt("temperature", adapter.ghList.get(clickedItemIndex).getLatest().getTemperature());
bundle.putInt("light", adapter.ghList.get(clickedItemIndex).getLatest().getLight());
bundle.putInt("co2", adapter.ghList.get(clickedItemIndex).getLatest().getCo2());
bundle.putInt("humidity", adapter.ghList.get(clickedItemIndex).getLatest().getHumidity());

Navigation.findNavController(getActivity(), R.id.nav_host_fragment_activity_main_page)
        .navigate(R.id.navigation_greenhouse_show, bundle);
```

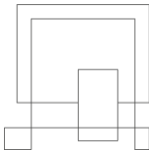
By clicking one of the indicators, the chart fragment will be shown for the chosen indicator, containing the measurements up to a year ago.

To display data measured by the sensors in a chart, an external library called MPAndroidChart was used. To import the library into the project the following dependency had to be added in the module level Gradle file and the maven repository to the settings Gradle file.

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

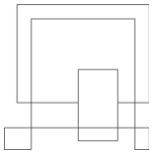
```
maven { url 'https://jitpack.io' }
```

In the ChartFragment it can be found how the chart is being initialized. Also, there is an option to choose a time interval for which data should be shown and the default is set to one week.



```
minutes.setOnClickListener(v -> {
    if (!loading) {
        selectInterval(minutes);
        timeChosen = available_times.minutes15;
        getData();
    }
});
hour.setOnClickListener(v -> {
    if(!loading){
        selectInterval(hour);
        timeChosen = available_times.hours4;
        getData();
    }
});
```

To show the updated data when opening the chart or a different period is being chosen a method is called to update the data.



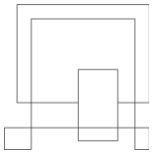
```
public void updateData(List<GreenData> newData)
{
    LineDataSet dataSet = new LineDataSet(setDataValues(newData), key);
    dataSet.setDrawFilled(true);
    if (Utils.getSDKInt() >= 18) {
        // fill drawable only supported on api level 18 and above
        Drawable drawable = ContextCompat.getDrawable(getContext(), R.drawable.fade_green);
        dataSet.setFillDrawable(drawable);
    }
    else {
        dataSet.setFillColor(Color.BLACK);
    }
    ArrayList<ILineDataSet> dataSets = new ArrayList<>();
    dataSets.add(dataSet);
    LineData data = new LineData(dataSets);
    reportChart.setData(data);
    reportChart.notifyDataSetChanged();
    reportChart.invalidate(); // Everytime data is changed this refreshes the chart
    toast.cancel();
    toast = Toast.makeText(getContext(), text: "Loading finished!",Toast.LENGTH_SHORT);
    toast.show();
    loading = false;
}
```

To retrieve the data from the API a method in the view model is being observed which returns a list of data objects from the repository.

```
viewModel.getChartData(eui,now, timeChosen,noOfPoints).observe(getViewLifecycleOwner(), this::updateData);

public LiveData<List<GreenData>> getChartData(String id,Date end ,available_times interval,int noOfPoints)
{
    return repository.getChartData(id,end,interval,noOfPoints);
}
```

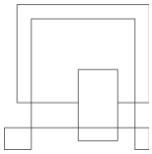
In the repository there are a couple of methods that work together to retrieve the data and then average and transform it according to the interval selected by the user, showing 0 when the data was not recorded. To accomplish that we split the final wanted result into smaller subtasks:

We have a method to calculate the start of data that we ask for based on the number of points displayed on the graph, the end of the data being the request time:



```
public static Date calculateStart(Date end, available_times interval, int noOfPoints) {
    Calendar c = Calendar.getInstance();
    c.setTime(end);
    switch (interval) {
        case minutes15:
            c.add(Calendar.MINUTE, i1: -noOfPoints*15);
            break;
        case hours4:
            c.add(Calendar.HOUR, i1: -noOfPoints*4);
            break;
        case days1:
            c.add(Calendar.DAY_OF_MONTH,-noOfPoints);
            break;
        case days7:
            c.add(Calendar.DAY_OF_MONTH, i1: -noOfPoints*7);
            break;
        case months1:
            c.add(Calendar.MONTH,-noOfPoints);
            break;
        case months6:
            c.add(Calendar.MONTH, i1: -noOfPoints*6);
            break;
        case years1:
            c.add(Calendar.YEAR,-noOfPoints);
            break;
    }

    return c.getTime();
}
```

After that, we ask for the data passing the device Id and start and end of interval. We get the data, which can contain at most 84 data points when the 15 minutes interval is selected and 2.6 million data points when the yearly interval is selected. TransformData method is then run with interpolates the data points to the amount passed as parameter.



```
private void TransformData(List<GreenData> all,Date end ,available_times interval,int noOfPoints) {

    List<GreenData> returned = new ArrayList<>();

    Date start = DateUtil.calculateStart(end,interval,noOfPoints);
    Date localEnd = DateUtil.calculateEnd(start,interval, noOfPoints: 1);

    for (int i = 0; i < noOfPoints; i++) {

        List<GreenData> lastInterval = extractInterval(all,start,localEnd);
        GreenData averaged = averageArray(lastInterval);
        returned.add(i,averaged);

        start = localEnd;
        localEnd = DateUtil.calculateEnd(start,interval, noOfPoints: 1);
    }

    chartData.setValue(returned);
}
```

5 Test

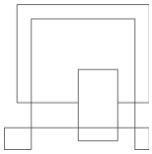
In the following section different types of testing will be documented according to specialization.

5.1 Test Specifications

5.1.1 IoT

(Michel Sommer, Florina Toldea, Viggo Petersen)

The IoT application has been tested using unit tests and Interface Testing**.** Unit testing was used to test parts of the program, interacting with drivers, as discussed 3.1.2 under description of the program loop shown in Figure 10. in which it does not depend on FreeRTOS components.



Amazon offers “AWS IoT Device Tester (IDT) for FreeRTOS (amazon, 2021) as a test suite for FreeRTOS. As the built-in does not support LoRa and in the absence of technical knowledge, the use of this framework has been opted out.

Fake Function Framework (fff) a micro-framework for creating fake C functions for tests (meekeosoft, 2019) that includes gTest from Google.

The IoT application GreenhouseNode, and corresponding tests can be found in Appendix F.

5.1.2 Workflow Testing (Grey Box Testing)

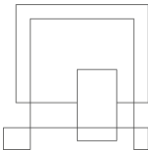
(Michel Sommer, Florina Toldea, Viggo Petersen)

LoRaWAN connection has been tested for a throughput from Node to Gateway, by sending hardcoded data, shown in figure 26 from node to receiving server. figure 26 shows the data is being received at the gateway. The test is passed, as shown in the table in figure 26

Date	Message sent	Message expected	PASS/FAIL
16/12/2021	00 1D 4A 19 D0 92 23	00 1D 4A 19 D0 92 23	PASS

Figure 26: Test Schema

```
62    /* Will eventually be replaced with an actual implementation */
63    void _setTestSensorData(void* pvParameters) {
64        const TickType_t xFrequency = pdMS_TO_TICKS(5000UL); // 5 sec
65        TickType_t xLastWakeTime = xTaskGetTickCount();
66        // sample data 00 1D 4A 19 D0 92 23
67        // temp=29, hum=74, co2=826, lum=9352
68        while (1) {
69            uint8_t data[7] = {0, 29, 74, 25, 208, 146, 35};
70            loraWAN_setPayload(4, 7, data);
71            xTaskDelayUntil( &xLastWakeTime, xFrequency );
72        }
73    }
```



0004A30B00259D2C	16/12/2021, 12.39.09	867.900	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	16/12/2021, 12.35.44	867.100	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	16/12/2021, 11.16.59	867.900	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	16/12/2021, 11.16.07	867.100	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	16/12/2021, 11.15.09	868.300	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	16/12/2021, 11.10.41	868.300	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.36.51	867.900	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.33.56	867.900	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.26.50	867.300	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.23.46	867.700	SF12 BW125 4/5	3	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.23.42	867.300	SF12 BW125 4/5	2	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	14/12/2021, 12.23.02	868.300	SF12 BW125 4/5	0	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	13/12/2021, 13.44.39	867.900	SF12 BW125 4/5	6	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	13/12/2021, 13.44.35	868.500	SF12 BW125 4/5	5	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	13/12/2021, 13.43.59	867.100	SF12 BW125 4/5	4	4	00 1d 4a 19 d0 92 23
0004A30B00259D2C	13/12/2021, 13.43.18	868.300	SF12 BW125 4/5	2	4	00 1d 4a 19 d0 92 23

5.1.3 Unit tests (White box testing)

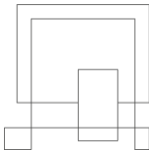
(Michel Sommer, Florina Toldea, Viggo Petersen)

The application has been white box tested using Fake Function Framework (fff)

In the code snippet, shown below the external C file dependencies is shown, as it can be seen, the drivers' library interfaces are dependencies of the test module together with the module interfaces.

```
1   #include "../fff-master/gtest/gtest.h"
2   #include "../fff-master/fff.h"
3
4    DEFINE_FFF_GLOBALS  //  Initialize the framework
5
6   extern "C"
7   {
8   #include "../GreenhouseNode/driver/mh_z19.h"
9   #include "co2.h"
10
11  #include <stdbool.h>
12  }
13
```

The code snippet shows some fake functions (mocks to simulate the actual function), needed for the mh_z19 sensor to make a custom return value from the driver.



```
15      //  Fake function to take a measuring from the CO2 Sensor
16      FAKE_VALUE_FUNC(mh_z19_returnCode_t, mh_z19_takeMeassuring);
17
18      //  Fake function to get the CO2 value from the CO2 Sensor
19      FAKE_VALUE_FUNC(mh_z19_returnCode_t, mh_z19_getCo2Ppm, uint16_t*);
```

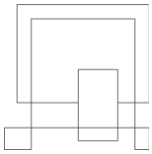
The C++ class shows the setup and tear down of the individual tests, resetting the fake methods from figure above.

```
21      class CO2_Test : public ::testing::Test
22      {
23      protected:
24          void SetUp() override {
25              RESET_FAKE(mh_z19_takeMeassuring);
26              RESET_FAKE(mh_z19_getCo2Ppm);
27              FFF_RESET_HISTORY();
28          }
29          void TearDown() override {
30
31          }
32
33      };
```

code snippet of the test class.

The test shown in figure YZ. uses the CO2_Test class shown in figure ZZ. as an argument, with the name of the test.

1. Arrange: adds the pointer for the callback
2. Act: calling the measuring function
3. Assert: Validate the results, by counting test calls, then testing return code, then check the returned value, if it is as expected.



```
62  TEST_F(CO2_Test, Test_CO2_set_data_is_called) {
63
64      //  Arrange
65      uint16_t value = co2_getLastCO2ppm();
66
67      //  Act
68      co2_getLatestMeasurement();
69
70      //  Assert
71      ASSERT_TRUE(mh_z19_getCo2Ppm_fake.call_count == 1);
72      EXPECT_TRUE(MHZ19_OK == mh_z19_getCo2Ppm_fake.return_val);
73      ASSERT_TRUE(1500 == co2_getLastCO2ppm());
74      ASSERT_FALSE(value == co2_getLastCO2ppm());
75  }
```

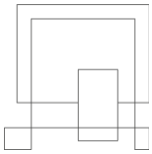
code snippet of a test

5.1.4 Data Engineering

(Mark Vincze, Priyanka Shrestha, Maria Asenova, Constantina Tripon)

REST API Black Box test

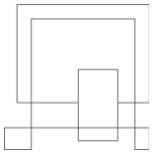
Endpoint	Request	Result	Fail/Pass
POST '/devices'	New device: { "eui": "DUMMYTEST", "location": "TestLocation", "minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865, "minLight": 1000,	HTTP 201 { "eui": "DUMMYTEST", "location": "TestLocation", "minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865,	PASS



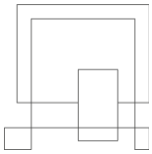
	"maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40, "targetLight": 4000, "targetCO2": 600 }	"minLight": 1000, "maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40, "targetLight": 4000, "targetCO2": 600 }	

	Same device:
	{
	"eui": "DUMMYTEST",
	"location": "TestLocation",
	"minTemperature": 12,
	"maxTemperature": 23,
	"minHumidity": 43,
	"maxHumidity": 70,
	"minCO2": 564,
	"maxCO2": 865,
	"minLight": 1000,
	"maxLight": 2000,
	"targetTemperature": 34,
	"targetHumidity": 40,
	"targetLight": 4000,
	"targetCO2": 600
	}
		HTTP 403
		Device is already registered!
			PASS

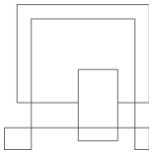
	Without ‘eui’: { "location": "TestLocation",	HTTP 201 { "location":	FAIL it should have



	"minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865, "minLight": 1000, "maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40, "targetLight": 4000, "targetCO2": 600 }	"TestLocation", "minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865, "minLight": 1000, "maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40, "targetLight": 4000, "targetCO2": 600 }	returned 403 for missing ‘eui’
GET ‘/device/{eui}’	Added device. eui = ‘DUMMYTEST’	HTTP 400 Device eui needs to be 16 digits	PASS
	Existing device. eui = ‘DUMMY30B00259D2B’	HTTP 200 { "eui": "DUMMY30B00259D2B", "location": "skagen", "minTemperature": 0, "maxTemperature": 0, "minHumidity": 0, "maxHumidity": 0,	PASS



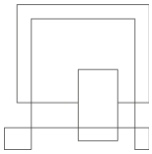
		"minCO2": 0, "maxCO2": 0, "minLight": 0, "maxLight": 0, "targetTemperature": 0, "targetHumidity": 0, "targetLight": 0, "targetCO2": 0 }	
	Not an existing device. eui = 'DUMMY30AS0259D2B'	HTTP 404 Device is not found	PASS
PUT '/devices'	Update existing device. { "eui": "DUMMYTEST", "location": "TestLocation12", "minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865, "minLight": 1000, "maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40, "targetLight": 4000,	HTTP 200 { "eui": "DUMMYTEST", "location": "TestLocation12", "minTemperature": 12, "maxTemperature": 23, "minHumidity": 43, "maxHumidity": 70, "minCO2": 564, "maxCO2": 865, "minLight": 1000, "maxLight": 2000, "targetTemperature": 34, "targetHumidity": 40,	PASS



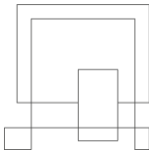
	"targetCO2": 600 }	"targetLight": 4000, "targetCO2": 600 }	

	Update non existing device.
	{
	"eui": "DUMEST",
	"location":
	"TestLocation12",
	"minTemperature": 12,
	"maxTemperature": 23,
	"minHumidity": 43,
	"maxHumidity": 70,
	"minCO2": 564,
	"maxCO2": 865,
	"minLight": 1000,
	"maxLight": 2000,
	"targetTemperature": 34,
	"targetHumidity": 40,
	"targetLight": 4000,
	"targetCO2": 600
	}
		HTTP 404
		Cannot find device with eui
			PASS

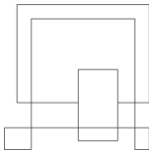
DELETE '/device/{eui}'	Delete existing device. eui = 'DUMMYTEST'	HTTP 200 { "eui" : "DUMMYTEST ", "location" : "TestLocation12", "minTemperature" : 12,	PASS



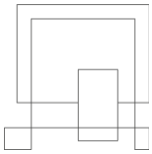
		"maxTemperature" : 23, "minHumidity" : 43, "maxHumidity" : 70, "minCO2" : 564, "maxCO2" : 865, "minLight" : 1000, "maxLight" : 2000, "targetTemperature" : 34, "targetHumidity" : 40, "targetLight" : 4000, "targetCO2" : 600 }	
	Delete non existing device. eui = 'DUMTEST'	HTTP 404 Cannot find device with eui	PASS
POST '/remote/{eui}/window'	Sending remote to existing device. eui = '0004A30B00259D2C' commandPercentage = 45	HTTP 200 Successful	PASS
	Sending out of boundary command to existing device. eui = '0004A30B00259D2C' commandPercentage = 200	HTTP 200 Successful	FAIL should not take commands outside of 0-100 boundary



	Sending commands to non existing device. eui = ‘0004A30123259D2C’ commandPercentage = 45	HTTP 404	PASS
GET ‘/measurements/{eui}/periodic’	Get data from existing device. eui = ‘0004A30B00259D2C’ start = 2021-11-10 16:21:00 end = 2021-11-10 17:21:00	HTTP 200 [{ "id": 5517, "eui": "0004A30B00259D2C", "date": "2021-11-10T16:21:13.000+00:00", "humidity": 43, "temperature": 68, "light": 45, "co2": 872 }, . . . { "id": 5546, "eui": "0004A30B00259D2C", "date": "2021-11-10T17:19:13.000+00:00", "humidity": 43, "temperature": 67,	PASS



		"light": 44, "co2": 846 }]	
	Get data with wrong formatted parameters. eui = ‘0004A30B00259D2C’ start = 2021-11-10 16:21 end = 2021-11-10 17:21	HTTP 404 Unparseable date: "2021-11-10 16:21"	PASS
	Get data from non existing device. eui = ‘TESTE0B00259D2C’ start = 2021-11-10 16:21:00 end = 2021-11-10 17:21:00	HTTP 200 []	FAIL should have returned 404 for device
	Get data with empty query. eui = ‘0004A30B00259D2C’ start = end =	HTTP 404 Unparseable date: " "	FAIL should have returned bad request
GET ‘/measurements/{eui}/latest’	Get data from existing device. eui = ‘0004A30B00259D2C’	HTTP 200 { "id": 9450, "eui": "0004A30B00259D2C", "date": "2021-11-	PASS



		16T13:27:13.000+00:00", "humidity": 44, "temperature": 68, "light": 46, "co2": 977 }	
	Get data from non existing device. eui = ‘TESTE0B00259D2C’	HTTP 200	FAIL Should have returned 404 for device not found

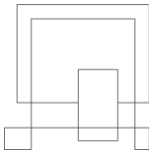
*Wrong API key always results in a NotFoundException.

5.1.4.1 Data Warehouse and ETL Test:

The data warehouse and the log mining table were tested by creating the test source database and test data warehouse namely TestDB and TestDW respectively. The test was done using Kimball's suggestion.

First of all, the script was generated from the source database and data warehouse. Afterward, the fraction of data was loaded to TestDB. For the ease of the testing only one device and the measurement for only the same device was loaded to TestDB. Then the TestDW, stage and edw tables were loaded the same way as the actual data warehouse was loaded but from the TestDB.

The actual tests were done by counting and comparing the number of rows in a table, the minimum, maximum and the average values of the measurements from the TestDB and TestDW and the test was approved when the result of these two queries matched.



```
SELECT COUNT(*) AS CountFromTestDB
FROM TestDB.dbo.sensor_data

SELECT COUNT(*) AS CountFromTestDW
FROM TestDW.edw.Fact_Measurement
```

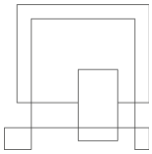
Testing have been done for all of the dimensions and can be found in Appendix:XX

5.1.5 Android

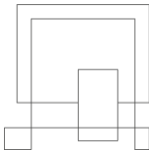
The android application was tested by creating UI tests and scenario tests according to use cases. The scenario tests are considered as blackbox testing and the UI tests as whitebox.

Scenario tests (blackbox tests):

Use Case	Test	Actor	Steps	Expected	Result
Managing devices	Add and see list of devices	User	1. Run the application 2. Log in the application successfully. 3. Press the add device floating button. 4. Enter device id and name, target and threshold values. 5. Click the “Save” button.	The adds a device for the user and shows it in the list of devices.	Passed



Monitoring and filtering data	See device latest and previous measurements	User	1. Run the application. 2. Log in the application successfully. 3. Choose a device from the device list and click on it. 4. Choose a measurement to see previous data. 5. Choose a period of time to display data.	The device shows a page that displays current measurements for all the sensors. The device displays a line chart with the measurements for the previously chosen period.	Passed
Regulate environment	Change environment factors in the greenhouse.	User	1. Run the application 2. Log in the application successfully. 3. Choose a device from the device list and click on it. 4. Click the button for manipulating the environment. 5. Choose a state for window, water and light.	The device should start changing measurements according to the changes, once the changes are done.	Passed



UI Tests (whitebox tests):

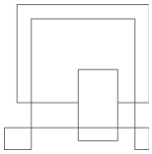
The UI tests are a type of whitebox testing for Android where part of the application is being tested. In the example below we test the Login functionality by entering the login credentials in the fields after which the login button is pressed. Since if the login is successful the page that opens is the home page of the application, the test is successful if the home page is opened after the login.

```
@Rule
public ActivityScenarioRule<LoginActivity> mActivityTestRule = new ActivityScenarioRule<>(LoginActivity.class);

@Rule
public ActivityScenarioRule<StartActivity> startRule = new ActivityScenarioRule<~>(StartActivity.class);

@Test
public void loginActivityTest() {
    onView(withId(R.id.text_login_email)).perform(replaceText( stringToBeSet: "iliya.nikov@gmail.com"), closeSoftKeyboard());

    onView(withId(R.id.text_login_password)).perform(replaceText( stringToBeSet: "1234"), closeSoftKeyboard());

    onView(withId(R.id.loginButton)).perform(click());

    onView(withId(R.id.home_fragment)).check(matches(isDisplayed()));
}
```

6 Results and Discussion

(Mark Vincze, Priyanka Shrestha, Maria Asenova, Constantina Tripon)

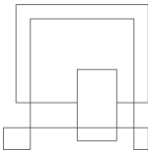
As defined in the project description, the problem statement consists of one main problem and 10 sub questions. The main problem statement of the project aims to optimize plant growth while saving resources. For the time frame of the project the team has successfully managed to provide a solution for optimizing the growth of plants by providing an optimal solution for monitoring the environment inside the greenhouse and providing statistics for the different measurements over time. 4 out of 9 sub questions to the main problem statement have been answered. Sub questions number 1,2,3 and 9 have been answered accordingly. The implementation of the system will contribute to the reduction of labour hours, as well as help plants to get the ideal amount of nutrition's. Data has been also gathered in a sortable manner.



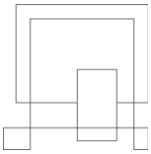
To fully answer the problem statement and sub statements the following functionality should be implemented/researched:

1) The different types of plants for which the system is suitable for
2) Set up a tool for finding the optimal conditions for a plant
3) Prioritization of the specifications of a plant
4) Measurements of the improvement of a plant's life
5) Measurements of a plant life cycle

	User Story	**Result**
1	As a user, I want to be able to register and log in so I can have access to the system.	Completed
2	As a user I want to gain access to the system in order to control the different environmental factors.	Completed
3	As a user, I want to be able to see data gathered by the temperature sensor so the temperature can be adjusted accordingly by setting up new threshold values.	Completed
4	As a user, I want to be able to see data gathered by the humidity sensor so the humidity can be adjusted accordingly if needed by setting up new threshold values.	Completed
5	As a user, I want to be able to see data gathered by the CO_2 sensor to get an overview of the CO_2 concentration in the	Completed



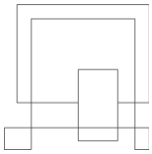
	greenhouse so I can adjust the level needed for the specific plant.	
6	As a user I want to be able to see data gathered by the light sensor so I can make sure that the lights are functioning in a timely manner.	Completed
7	As a user I want to manage my device settings to the control system.	Completed
8	As a user I want to be able to set settings/threshold values to devices to ease access to remote locations.	Completed
9	As a user I would like to select different time frames and compare the data (temperature, humidity, photoresistor, CO_2) collected of the specific device to monitor the state of the greenhouse.	Completed
10	As a user I want to get notified when the temperature sensor is out of the threshold value.	Not implemented
11	As a user I want to get notified when the humidity sensor is out of the threshold values.	Not implemented
12	As a user I want to be able to add multiple devices to my greenhouse such that I can monitor the data from different locations inside my greenhouse.	Completed
13	As a user I want to get notified when the CO_2 sensor is out of a threshold value.	Not implemented
14	As a user I would like to be able to display the local weather (temperature, humidity, CO_2) relevant to the greenhouse's environment such as to compare interior and exterior data.	Completed



15	As a data analyst of the business, I want to be able to see the temperature, humidity, CO2 and light data of all my devices in a single dashboard so that I can compare the data.	Completed
16	As a data analyst of the business, I want to get the minimum, maximum and the median temperature data of the greenhouse collected from the beginning to the latest recorded data.	Completed
17	As a user I want to be able to review the latest data in case of no internet connection.	Completed
18	As a user I want to be able to create multiple devices such that I can keep track of the devices installed in a greenhouse and where they are located.	Completed
19	As a user I want to be able to remove a device from my list of devices in a greenhouse when I no longer need to monitor data from the device installed in the greenhouse.	Completed
20	As a user I want to be able to edit the settings of a device located in a greenhouse such that I can change the targeted temperature, humidity, CO2, and light.	Completed
21	As a user I want the device to be able to operate windows, in relation to the greenhouse environment, to regulate the environment.	Not implemented

6.1 Data Engineering

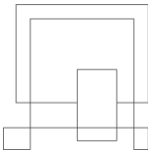
(Mark Vincze, Priyanka Shrestha, Maria Asenova, Constantina Tripon)



As requested from our client Martin Høgh, the measurements of humidity, CO2, temperature, and light over time have been visualized in an interactive dashboard. 7 out of 9 requirements defined by the client have been met.

	Requirement	Status
1.	Humidity measurements over specific period of time	Completed
2.	CO_2 measurements over specific period time	Completed
3.	Light measurements over specific period time	Completed
4.	Temperature measurements over specific period time	Completed
5.	Daily update of average environment variables	Not Completed
6.	Monthly median/average/mean of measurements	Completed
7.	A log of when the data has crossed the threshold value	Completed
8.	An option to control the environment remotely (windows, air conditioning etc.)	Not Completed
9.	The option to add multiple devices with their sensors in the greenhouse	Completed

To fully meet the requirements of the client, a daily average update for each environmental factor must be provided.



7 Test Cases

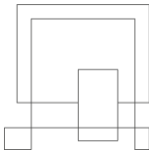
(Mark Vincze, Priyanka Shrestha, Maria Asenova, Constantina Tripon)

In the following section the test cases written in the analysis chapter will be revisited and updated with the actual result. Each user story involved in the use case *Monitor and Filter Data* will be tested in a separate table.

As a user I want to gain access to the system to control the different environmental factors.

SN	Action	Reaction	Status
1.	Login with valid inputs - Insert email and password - Press Login button	- System takes the input For correct inputs - Access is given and the dashboard is displayed	Passed
2.	Login with invalid inputs - Insert email and password - Press Login button	- System takes the input For incorrect or empty inputs - Access is denied and an error message is displayed	Passed

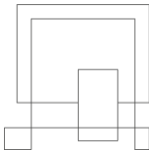
As a user, I want to be able to see data gathered by the temperature sensor so the temperature can be adjusted accordingly by setting up new threshold values.



SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	Passed
2.	- Selects Temperature from the dashboard	- System displays the temperature view for the greenhouse	Passed
3.	- Selects the period of time for data to be displayed	- System displays the temperature values over the period of time	Passed

As a user, I want to be able to see data gathered by the humidity sensor so the humidity can be adjusted accordingly if needed by setting up new threshold values.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	Passed
2.	- Selects the humidity from the dashboard	- System displays the humidity view for the greenhouse	Passed
3.	- Selects the period of time for data to be displayed	- System displays the humidity values over a period of time	Passed

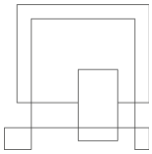


As a user, I want to be able to see data gathered by the CO_2 sensor to get an overview of the CO_2 concentration in the greenhouse so I can adjust the level needed for the specific plant.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	Passed
2.	- Selects the CO2 from the dashboard	- System displays the CO2 view for the greenhouse	Passed
3.	- Selects the period of time for data to be displayed	- System displays the CO2 values over a period of time	Passed

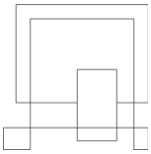
As a user I want to be able to see data gathered by the light sensor so I can make sure that the lights are functioning in a timely manner.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- System opens the specific greenhouse view	Passed
2.	- Selects the Light from the dashboard	- System displays the current state of the Light sensor	Passed



As a user I would like to select different time frames and compare the data (temperature, humidity, photoresistor, CO_2) collected to monitor the state of the greenhouse.

SN	Action	Reaction	Status
1.	- Selects the greenhouse	- The system opens the selected greenhouse and displays the latest recorded data for Temperature, Humidity, CO2 and Light.	Passed
2.	- Selects the environmental variables (Temperature, Humidity, CO2 or Light) from the dashboard	- The system opens the view for the selected variable and displays a graph with the recorded data	Passed
3.	- Selects the timeframe available in the view (15min, 1 week, 1 month, 1 year)	- The system displays the data recorded for the selected time frame in a graph	Passed

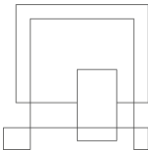


8 Conclusions

It is not an easy job to manage a greenhouse, especially remotely, but today's technologies allow that to be implemented. This solution can provide help for companies or individuals to maintain many greenhouses from a distance (with LoRaWan tower nearby) by planting these devices and registering them on their client. The section of the analysis concludes the user stories, use case diagram and following through one use case description, the Monitor and filter data, as respect for the size of the documentation.

From the Design to the Test chapter, three subchapters represent the different parts of the system, the IoT, the Data Engineering and the Android. With consideration to the main sections each of these subchapters explain the parts in detail.

Currently, the system is capable of monitoring, analysing and visualising data. The result of the project is a system implementing most of the user stories as discussed in the results section. All the test cases for monitor and filtering data are passed and the user can monitor and analyse the data. The system needs to be extended to fully cover the requirements and there are still some improvements to be made.



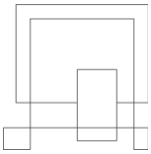
9 Project future

The project has a lot of functionalities implemented and is ready to be released. Due to time constraints, the low priority use cases were not implemented. Implementing unit tests and automated tests would provide good support for future implementation.

Now the app only supports one greenhouse, but in a real-life situation one user can control multiple greenhouses, as shown in the domain model. This update would make this system more valuable with less limits. It could also be compatible with other farming buildings such as stables and farm warehouses by adding the appropriate entities in the design.

To handle multiple greenhouses the team has to add a greenhouse entity in the database and update the android application accordingly.

For the system to support a complete concept of an automated remote greenhouse system more functionality needs to be implemented. This should include the user being able to control the internal environment from the android app. Such a functionality would include setting the value of an open window and setting the value of light and watering.



10 Sources of information

-

amazon, 2021. *amazon.* [Online]
Available at: https://docs.aws.amazon.com/freertos/latest/userguide/device-tester-for-freertos-ug.html
[Accessed 14 December 2021].

Anon., 2009. *Kimball Group.* [Online]
Available at: https://www.kimballgroup.com/2009/10/six-key-decisions-for-etl-architectures/
[Accessed 23 November 2021].

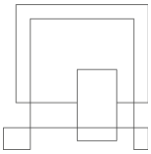
Anon., 2021. *amazon.* [Online]
Available at: https://www.aboutamazon.com/what-we-do/amazon-web-services
[Accessed 4 December 2021].

Anon., 2021. *Digital Vidya.* [Online]
Available at: https://www.digitalvidya.com/blog/visual-analytics/
[Accessed 4 December 2021].

Anon., 2021. *DutchGreenhouses.* [Online]
Available at: https://www.dutchgreenhouses.com/en/technology/greenhouse-automation/
[Accessed 7 December 2021].

Anon., 2021. *Microsoft.* [Online]
Available at: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15
[Accessed 6 December 2021].

Anon., 2021. *spring.* [Online]
Available at: https://docs.spring.io/spring-data/jpa/docs/1.3.0.RELEASE/reference/html/jpa.repositories.html
[Accessed 5 December 2021].



Anon., 2021. *spring.* [Online]
Available at: https://spring.io/guides/tutorials/rest/
[Accessed 7 December 2021].

Group, K., 2006. *Kimball Group.* [Online]
Available at: https://www.kimballgroup.com/2006/07/design-tip-81-fact-table-surrogate-key/
[Accessed 4 December 2021].

meekeosoft, 2019. *github.* [Online]
Available at: https://github.com/meekrosoft/fff
[Accessed 12 December 2021].

Microsoft, 2021. *Microsoft.* [Online]
Available at: https://community.powerbi.com/t5/Themes-Gallery/Color-Blind-Friendly/m-p/140597
[Accessed 10 December 2021].

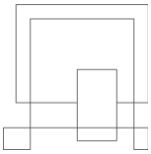
Pipino, L. L., n.d. [Online]
[Accessed December 2021].

YourCoach, S.M.A.R.T. goal setting | SMART | Coaching tools | YourCoach Gent. Available at: http://www.yourcoach.be/en/coaching-tools/smart-goal-setting.php [Accessed August 19, 2017].

GeeksforGeeks. 2021. *MVVM (Model View ViewModel) Architecture Pattern in Android - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/> [Accessed 13 December 2021].

FreeRTOS, Developer docs.
Available at: https://www.freertos.org/features.html
[accessed November 24. 2021]

Alfter, B., 2020. *Automating Society Report ,* s.l.: Algorithm Watch . [Online] Available at: https://automatingsociety.algorithmwatch.org/report2020/denmark/



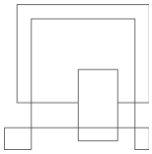
Anon., 2020. *Agriculture.* [Online] Available at: https://www.agritecture.com/blog/2020/11/12/europes-biggest-vertical-farms-to-be-established-in-denmark [Accessed 29 September 2021].

oregonstate, 2021. *Oregon State University.* [Online] Available at: https://extension.oregonstate.edu/gardening/techniques/environmental-factors-affecting-plant-growth [Accessed 29 September 2021].

Bitesize, 2021. *Bitesize.* [Online] Available at: https://www.bbc.co.uk/bitesize/topics/zxfrwmn/articles/zkf2mfr [Accessed 29 September 2021].

dti, 2021. *dti.dk.* [Online] Available at: https://www.dti.dk/specialists/optimising-plant-growth/38026 [Accessed 29 September 2021].

McKinsey&Company, 2018. *A Future That Works,* Aarhus: Economic Institut Aarhus University. [Online] Available at: https://innovationsfonden.dk/sites/default/files/2018-07/a-future-that-works-the-impact-of-automation-in-denmark.pdf



11 Appendices

Appendix A - Project Description

Appendix B - Scrum Sprint Tables

Appendix C - Diagrams

Appendix D - Dimensional Modelling

Appendix E - UI flow Mockups

Appendix F - Source Code

Appendix G - User Guides

Appendix H - Demonstrational Video

Appendix I - Group Contract