The Smart Coat Rack

Design Project Report

Authors

Ryan Cooijmans	s2736314
Tijn Hassing	s3004619
Deniss Kornijenko	s2953846
Daan Wensink	s2617390
Niek Zandt	s3024946
Simon van Zeijts	s2850966

Supervisors

Max Pijnappel
Roland van Rijswijk-Deij

Figure 1: The IoT Lab

University of Twente

Abstract

We have designed and implemented a smart coat rack and smart key box for the IoT Lab. The coat rack and key box use the MQTT communication protocol to communicate with a smart space, based on the inputs they receive from their sensors. A user can connect coat racks and key boxes, and see their status on a website.
In this report, we aim to show the way in which we conducted our design process, prototypes, and supervisor/client interaction. We will also showcase the design of the system, the ways in which it was tested, and more. Lastly, we reflect on the project, the teamwork, our personal experience, and possible improvements that could be made to the product in the future.

Contents

Chapter 1

Introduction

1.1 Background

In Zilverling 2070 there is a room that functions as the IoT lab. IoT stands for internet-of-things. The core principle of the internet of things is to have a lot of small devices equipped with sensors and connected to the internet. The devices can then interact with the world and with the other devices to make peoples lives more convenient.
The IoT lab is then outfitted with as many smart devices as possible. A smart device is (typically) a device that connects to other smart devices in a network (possibly the whole internet) [1]. These smart devices all serve different functions in the lab.
The IoT lab is missing a coat rack. In the spirit of having as many smart devices as possible, this coat rack will then have to be a smart coat rack.

1.2 Objectives

The main goal of the project is to develop a functional coat rack. Next to this, a secondary goal is to have the coat rack be smart. This means the coat rack will be equipped with sensors and will be connected to the internet. The final goal of the coat rack is to make the clients life more convenient.
To achieve these objectives, the client will be included in as many design choices as possible. Having input from the client will ensure the client knows the design of the product and helps make choices that will benefit the client.

Chapter 2

Requirements

2.1 Stakeholder analysis

The first step in designing a system is to pinpoint the stakeholders for the system. Stakeholders are any people or groups that have an interest in the completion and functioning of the system. The smart coat rack has a few stakeholders that the development team has analyzed.

Venues with wardrobes Large commercial venues with wardrobes can have an interest in the development of a smart coat rack. The coat rack could help with organizing and managing all the coats in the wardrobe, and could work better than the current system of physical tags.

Smart home owners There are many people that have a smart home. A smart home is a home in which many devices are smart devices. People with smart homes do not have an option to replace their coat rack with a smart coat rack yet as smart coat racks do not exist. This is why people with smart homes are also stakeholders as they could be interested in replacing a new device.

Due to the amount of resources and time of the development team not all these stakeholders can be interviewed and managed. As the team already has access to a stakeholder of the smart home owner type, it is then decided to prioritize this stakeholder for the development. All features will thus be decided with smart home owners in mind.

2.2 Client Interviews

The second step in requirement analysis is to have an interview with the client to discuss their needs and ideas. Before the very first talk with the client, the development team got together and brainstormed a list of possible features for the system. During this first brainstorm, no focus was put on plausibility of the imagined features as this is not important yet. Then during the first talk with the client, the list of possible features was presented to the client to

get their opinion on these features. The client was also asked if they had any features that they had already thought of and wanted to see. If there was overlap in features imagined by both the client and the team, these features were highlighted as important features.
Then, a version of the system was sketched and imagined with all the features discussed in the first interview. This system was not yet built, but some diagrams were sketched and the features were separated into 'Must haves' which were deemed essential for the system to function, 'Should haves' which are not essential for function but essential for experience, and 'could haves' which are features that would make the system more extensive but are not a necessity.
During these first interviews, and every subsequent interview, notes were taken on the comments the client had and on the discussions the team had with the client. These notes can be found for every interview in appendix B.

2.3 Lo-Fi Prototype Interview

The goal of a Lo-Fi prototype is to give the user some form of the system to interact with without having sunk a large amount of resources into development yet. The Lo-Fi prototype will be very limited in functionality and require some manual input or function to fully display planned features. Having a physical prototype for the user to interact with will give the user an easier time with formulating their needs. But keeping the invested amount of resources low will also allow changes to be made easily if the client dislikes a feature.

2.3.1 Lo-Fi Interview 1: Coat Rack versus Coat Hangers

The first Lo-Fi interaction round with the client will be to determine if the system should be a single connected rack with a certain amount of hooks, or if the system should be a hanger based system where every hanger requires some hardware to function. During earlier interviews the client had shown preferences for different aspects of both systems, and thought of limitations for both systems. When interacting with Lo-Fi prototypes of both systems, the client can more easily point out features they like, and features they do not like.

Lo-Fi Rack

For the Lo-Fi rack, the main part of the prototype is 3D-printed. This also allows the prototype to be stress tested to show if 3D-printing parts of the system is viable. The model was made using Autodesk Fusion, which is a designing tool which can be easily learned to design simple products. Functionality wise the Lo-Fi prototype only has the NFC reader and stickers. Having only these functionalities will show the client how the sticker placement will look, which is essential to the functionality of the system.

Lo-Fi Hanger

The Lo-Fi hanger prototype basis is a wooden coat hanger. Attached to the hanger is an NFC reader which will allow the hanger to read an NFC tag attached to the coat. The reader is positioned in such a way that when a coat is hung on the hanger it is perfectly in range of the tag hung from the loop of the jacket. The Esp and the reader are powered by 2 1.5 V AA batteries. This way the hanger can be used without being connected to any outside power supply, which makes it far more useful and user friendly.

Results

After letting the client interact with both Lo-Fi prototypes, they showed a preference for the hanger version compared to the rack version. The client was able to support this preference with clear reasoning and already imagined a few scenarios in which they would have more use for the hanger compared to the rack. Based on this, the Lo-Fi hanger prototype was chosen for further development.
See also: Appendix B.2.

2.3.2 Lo-Fi Interview 2: Interaction with the Key Storage System

For a second Lo-Fi interview, the client was asked to interact with the design for the key storage system to get their opinion on its functionality. The current design of the system uses audio jacks which can be attached to the key chain and a box that has the ports for these audio jacks. The current design was chosen with some discussion with the client, so they are already aware of the design. However, it is still useful to let the client interact with a physical version of this system, as it will give them a better overview of how it will look and feel in their actual hands.

Results

Despite some problems with the keybox being too small, and the ground pins being bent in a strange way, the reaction was positive overall.
See also: Appendix B.3.

2.4 Hi-Fi Prototype Interviews

The goal of a Hi-Fi prototype is to let the client interact with a version of the system that is close to what the completed system will look like. When the user interacts with the Hi-Fi prototype, they can easily see how finished features feel and interact and comment on that, it also shows how all the finished features are integrated into a singular system. The Hi-Fi prototype will also attempt to look like the finished system so the user can comment on the appearance as well. For a Hi-Fi prototype interaction, the user will be instructed to interact with the prototype as they would interact with the completed product. This will be done by

instructing the user to imagine they are having a regular day, and act out an interaction with the system that would fit in such a regular day. The 'normal' interaction with the system will clearly show how the user enjoys the experience of the system, and can help find overlooked issues that can occur with regular use.

2.4.1 Hi-Fi Interview 1

For the first Hi-Fi interview a few requirements were determined to be the most important to have finished so the client can interact with them. These requirements were chosen as they form the main interaction point between the client and the system, and are therefore the most important to the feel of the system.

3D-Printed Hanger For the first Hi-Fi prototype it is important that the hanger is 3D printed and has all the electronics inside it. This will ensure the hanger for the Hi-Fi prototype looks and feels like the finalized hanger will look.

Website Frontend For the first Hi-Fi prototype the website frontend should be fully functional. At this point it does not need to have to be fully designed visually yet, as the client will be asked to comment on the way it should look which they can do best when they can see the full functionality.

Key Box The Hi-Fi prototype will also show the functionality of the key box. When the user interacts with the system, their opinions on the reminder system, and on if they have more ideas on ways to be reminded to take their keys next to the ways that are currently implemented.

Results

From the first Hi-Fi interview it became clear that the client was very enthusiastic about the look and functionality of the system. They particularly enjoyed the look of the 3D printed key box. After this Hi-Fi interview, the next step will be to finish the prototype, make a few additional prototypes, and sow some RFID tags into actual jackets to do further testing with. See also: Appendix B.4.

2.5 Analyzed Requirements

From the project description, it was clear that there is a lot of freedom for us to decide what can be implemented. Starting with that, we have derived from the provided project description the MoSCoW requirements and presented them to the client and the supervisor. We have divided our requirements into 2 main categories using agile requirement formulation. Since the requirements below were constructed from one centralized pool of requirements, some stakeholder requirements might be mentioning the same requirement as some system requirements or vice versa.

2.5.1 Stakeholder Requirements

Must-Have Requirements

1. As a user, I want to be able to hang my coat on either a hanger or a coat rack, with the total number of available hangers or coat racks being at least four.
2. As a user, I want to be able to see what coats I have linked to a certain person.
3. As a user, I want to be reminded to not forget my keys when I am leaving the household.
4. As a user, I want only people registered to be able to access all the implemented features.

Should-Have Requirements

1. As a user, I want to know the current weather conditions at the provided location.
2. As a user, I want to get recommendations on which coat to wear in certain weather conditions.
3. As a user, I want to have a home presence list to be able to track who has entered or left the household.
4. As a user, I want to have the smart goat rack integrated with home assistant to connect to other smart devices.

Could-Have Requirements

1. As a user, I want to receive push notifications from the web platform to mobile devices.
2. As a user, I want to be able to preheat or dry my clothes.

2.5.2 System Requirements

Must-Have Requirements

1. As a hanger or a coat hanger unit, the 3D design of such should fit all the electronics and sensors that are required for all other implemented features.
2. As a hanger or a coat hanger system, it needs to be modular such that it can be easily added to the system to remain as a scalable system.
3. As a system, it must differentiate individual users to allow various features to provide personalized output.
4. As a system, it must recognize a hanged coat via sensor to further use that coat's information for various implemented features.

5. As a system, it must provide a web portal through which a user is able to access all implemented features regarding the household information.
6. As a system, it must securely store user data in case the web portal is on the world wide web to prevent any data being accessed or used by an unauthorized user of world wide web.
7. As a system, it must allow users to be registered to access all the implemented features.
8. As a system, it must allow to assign many different coats to a certain user.
9. As a system, it must show all the coats that are linked to a certain person.
10. As a system, it must detect when someone is leaving the household by tracking when a person took his jacket.
11. As a system, it must have a key holder unit to provide an option of storing/holding a key of a person once that person arrives to the household.
12. As a system, it must remind persons to not forget to take stored keys from the key holder unit.

Should-Have Requirements

1. As a system, it should provide the current weather conditions at the provided location via the weather API.
2. As a system, it should use the data from the weather API and the knowledge of all available coats, to deduce the best suitable peace of clothing.
3. As a system, it should visualize the clothing recommendation via the web platform or lighten up LEDs.
4. As a system, it should visualize the home presence list on the web platform using the information of coats of different people arriving and leaving the household.
5. As a system, it should disable all appliances once the last person has left the household and turn them back on when the first person enters the household.

Could-Have Requirements

1. As a system, it could be able to send a mobile push notification to a user.
2. As a system, it could be able to heat up the clothing of choice with the help of the heating element based on the weather or heating plan provided by the user.
3. As a system, it could have a storage compartment for various clothing of people in the household.

4. As a system, it could have a mechanical arm or rail that could provide an individual user with the coat of his selection.

Chapter 3

System Design

In this chapter, we will go over the design of the system, and its various components. For instructions on deploying the system, see Appendix A.
The system design consists of: Hardware like the micro-controllers and sensors, a database for information storage, the back-end of the website, the front end of the website and the CI/CD pipeline. Each of these component fulfills a specific task in the system, and all the components combined produce the final product.

3.1 Hardware

The hardware of the system will handle every interaction with the 'real' world. There is hardware for sensing changes in the world, and hardware for notifying the user of changes from the system. To connect the hardware to the system there is also a controller that handles incoming and outgoing data.

3.1.1 ESP32

Both the coat rack and the key box use an ESP32-C6 from Seeed Studio. This controller was chosen because it is very compact. The controller should be compact as it needs to fit within the components of the system, which have very limited space. The ESP32-C6 has 14 pins for input and output, of which two are power and one is ground. This amount of pins is sufficient for the amount of sensors the hanger and key box require to function.
To write code for the ESP the Arduino IDE was used. This IDE is specifically designed for writing code for micro controllers, and has some features that make development for the ESP easier, like recognizing the pin layout of the specific board. The Arduino IDE also allows easy code uploading to the ESP, which streamlines the development process.

3.1.2 RFID Reader

During our design phase we tested multiple NFC readers, the two main choices we ended up with are the PN532 NFC RFID and the RFID-RC522. After extensively testing both of them,

we decided to continue with the PN532 NFC RFID. We chose this reader because the reading distance is slightly bigger than the RFID-RC522. This does come at the cost of a slightly bigger power consumption, but the increased detection ranged made up for the need of a slightly bigger battery.

Figure 3.1: PN532 Reader

Figure 3.2: RC522 Reader

The reader is connected to a small XIAO ESP32-C6, which contains the scripts. It sends MQTT messages on the arrival and departure of every coat. These messages are then picked up by in our backend and updated on the frontend via this.
The ESP is powered by a thin battery that outputs 3.7 V.

3.1.3 Key Box Jacks

The keybox works using audio jacks. We use female jacks inside the keybox that are wired using a presence pin, a data pin, a power pin and a ground pin. From the presence pin we get an interrupt, that tells us a male jack is entered or taken out of the female jack, since it then completes a circuit. The data pin sends a signal after the presence is detected, runs through the one wire chip in male jack and reads the id. The jacks are powered with 3 V over a 1 kΩ resistor, and each female jack also has a ground connection.
Using his setup we connected 4 jack holders side by side to form our key box. It is all connected to another XIAO ESP32-C6 which sends the arrival and departure messages to the server via MQTT. The server then updates the frontend.

Figure 3.3: female jack

3.1.4 LED's

To allow the system to interact with the user, there are LED's added to different components. The LED's can light up when specific criteria are met, informing the user of changes in the system.
One addition of LED's is the addition of a NeoPixel LED strip in the key box. If the user takes their coat without collecting their keys, the key box will receive a message from the server. The LED strip will then start lighting up, reminding the user to not forget their keys. There are multiple LED's in the LED strip which are all controlled from a single data pin. When the key box receives the message that a coat is taken with the IDs of the keys associated with that coat, it will first check if any of the keys are in the key box. If a key is detected, the LED strip will display a colorful animation.
There is also a LED in the coat hanger itself. The goal of the LED in the coat hanger is to give the user more info about the state of the coat hanger. There are two colors the LED can turn based on what the coat hanger is currently doing.

Red The LED will turn red if there is some problem when connecting the hanger to either MQTT or WiFi.

Green The LED will turn green if the jacket on the hanger is detected, indicating the hanger is functioning as intended.

3.1.5 Battery

For powering the ESP in the hanger we use a simple 3.7 V battery. The main focus in choosing a battery was the size and weight, which had to fit our hanger design. The battery is also rechargeable, with a port on the hanger allowing for easy charger access.

3.1.6 Hardware and software integration

The hardware works together with the web application by sending MQTT messages to the server. First the ESP connects to the Wi-Fi, and afterwards the ESP follows MQTT protocol

to send all the required info to the website, which can then display this information to the user and reply with any further messages back. This allows for a seamless connection between the hardware and the frontend, and this whole relation can be seen in this diagram.

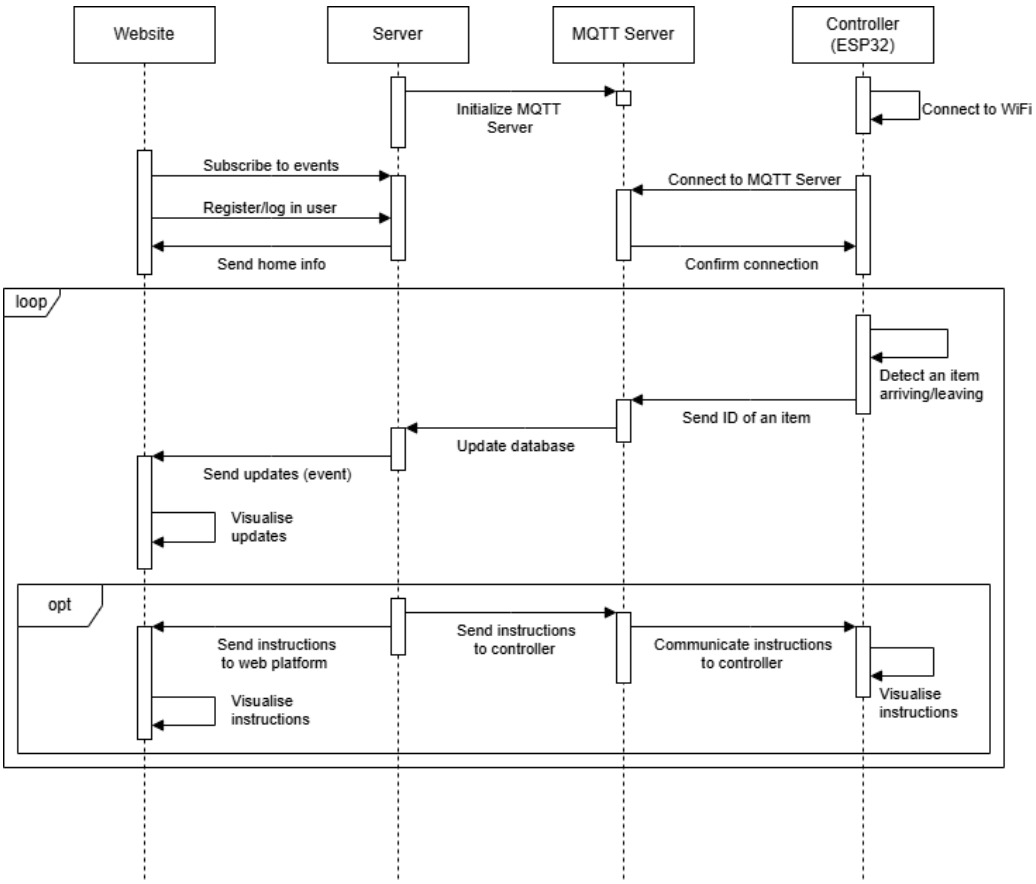

Figure 3.4: Diagram of system interaction

3.2 Database

Our application uses a PostgreSQL database to store all our relevant data including our user accounts, the persons these accounts create, the jackets and keys these people keep and the hangers and key racks linked to the user. For updating the database throughout the project we have been using a migration system. Every time the backend starts, it automatically runs any new migration files that have been added since the last time the application has been run. The migration files themselves are simple SQL files that create or modify tables. This keeps the development flow clear and makes the database easy to work with.

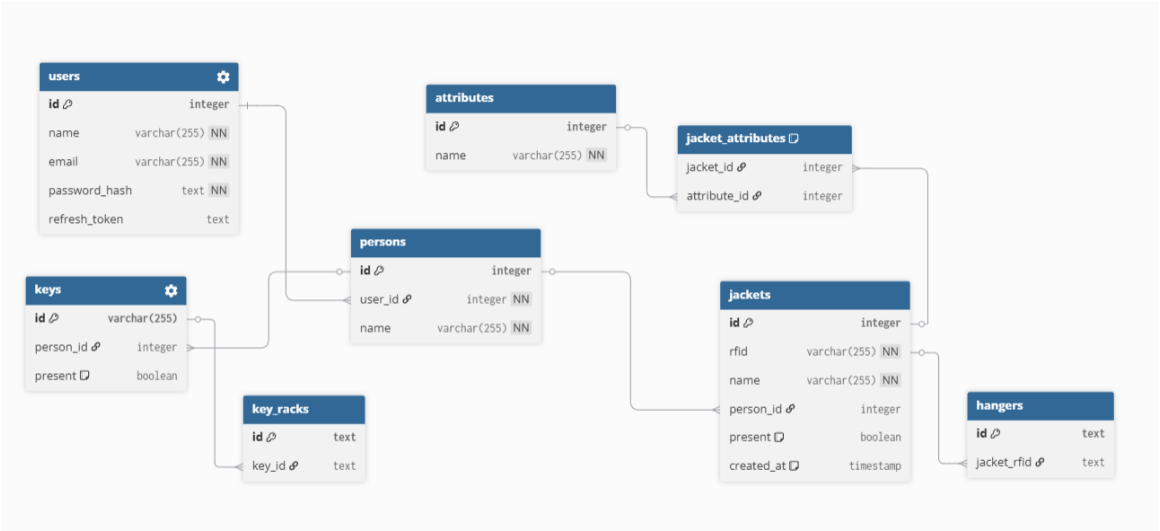

Figure 3.5: Diagram of the database

Each table in the database uses a primary key to uniquely identify its records. For example, hangers, jackets, persons and users each have their own ID. Relationships between tables are established using foreign keys. For instance, a jacket references a person through a foreign key, indicating ownership, and a hanger references a jacket to indicate which item is currently placed on it. This structure allows us to efficiently query related data.

3.3 Backend

The backend is a REST API written in *NodeJS* using the *Express* library. It consists of many controllers that perform transactions with the database to retrieve and update data.
A backend controller can only access the database through a database pool client, which it obtains by being wrapped in a transaction in the routing file. This avoids mistakes where a controller might have originally relied on the auto-commit of a single query, and adding another query while forgetting to place them into a transaction together would have broken the controller's atomicity. Additionally, queries that require inputs known only at runtime, do not use string interpolation. Rather, they use parameterized queries, which avoids (most) SQL injections.
JWT Tokens are used for user authentication with an expired token being the flag to redirect a user back to the login page. The system was designed to be able to handle both token expiration check and renewal, however, due to complications and inability of solving the issue with older tokens being used by the client side, it was decided to revert the implementation of token renewal. Without tokens being renewed, the current system users will be forcefully logged out once token reaches its expiration time. The backend also contains a web socket server that pushes live jacket data to clients listening through the frontend application. In the following diagram, the life cycle of a JWT access token can be seen, including the reverted implementation of token renewal (being a showcase of how the system could function once completed).

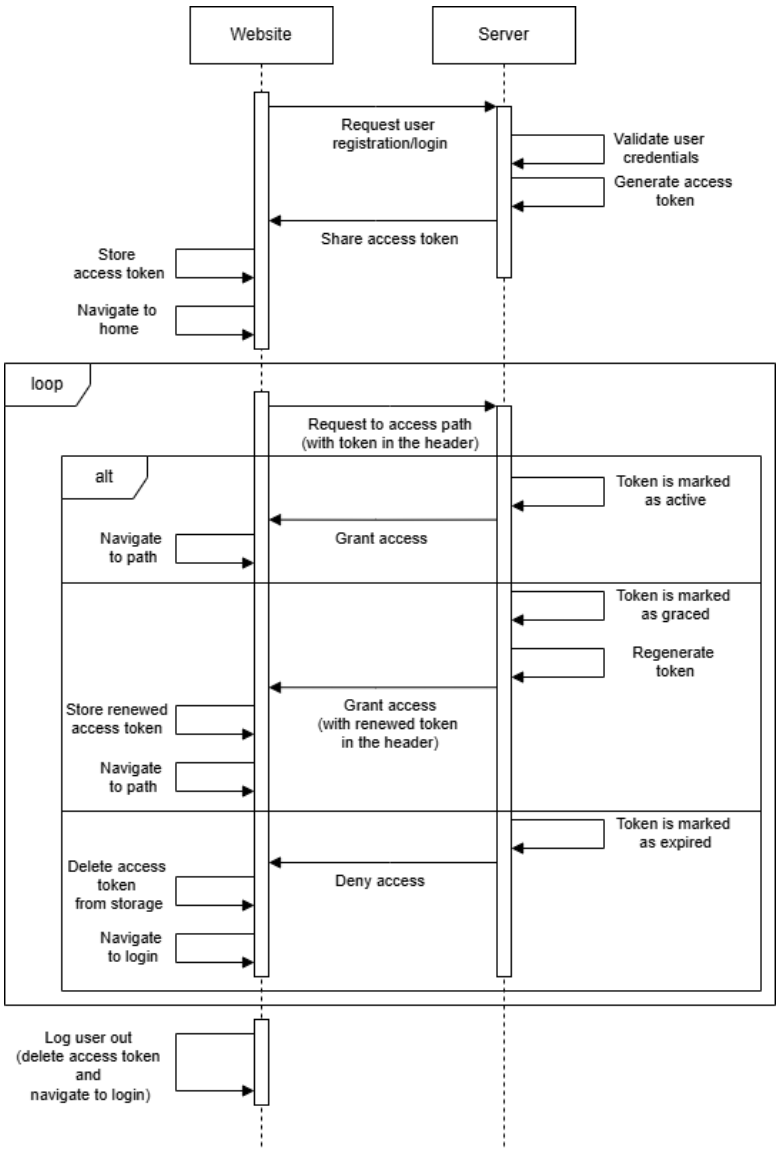

Figure 3.6: Diagram of the JWT Token usage

3.4 Frontend

The frontend of our smart goat rack application is implemented using *Angular* and provides a live visual of the status of the hangers and keybox. It shows which hangers and keybox slots are occupied, which coats are on which hangers, and which keys are stored in the keybox.
To achieve this, the frontend is continuously updated via WebSocket connections. Whenever the backend receives a message from a hanger or keybox, this data is forwarded through the WebSocket to the frontend, where the corresponding jacket, key, owner, and name are displayed in real time.
In addition to WebSockets, the system uses REST API's to communicate between the frontend and backend. These API's are used for operations such as retrieving the current state of hangers and registering new jackets or keys.
If a new RFID tag or key ID is not recognized, the system triggers a registration overlay. This allows users to assign the item to a person. For jackets, users can also add specific attributes,

which are later used by the jacket recommendation feature to suggest appropriate clothing based on the current weather conditions.
Since the current design of the smart coat rack does not provide any visible signs of a low or empty battery, the main information source regarding that is the web platform. It will provide information regarding the connectivity of all devices, and in case one got disconnected, the status bar of the connected devices will present this information to the user. Since the website can be considered as the complete overview of all the information presented to users, the following sections below describe the purpose and functionality of each one of them.

3.4.1 Home Page

After successfully logging in (or registering), users are presented with an overview of all the main information. This page allows users to connect new hangers or key boxes to their account and select their location. The displayed information is:

Connected Devices A number of connected devices (both coat hangers and key boxes).

Weather A weather forecast in the chosen location.

Present Coats The coats currently present, as well as special highlighting for the recommended coat.

Presence List The people currently at home, versus the people currently away. This is determined based on coat presence.

Present Keys The keys currently present.

3.4.2 Persons Page

Additionally, users are presented with a detailed overview of people that were added to the account as it implies that a user is a collection of multiple persons in one household. Users are able to create new persons, delete them, or modify information about them. On the respective tile of each person there is a list of all jackets linked to that person, which can also be disconnected, deleted from a person.

3.5 Home Assistant

Home Assistant is open-source software used to enable centralized home automation. It can be used to connect and automate IoT devices with a variety of protocols. Using Home Assistant a user can easily design flows; for example, turning on smart lights automatically when someone hangs up their jacket. The Smart Coat Rack uses the MQTT protocol to communicate with all hardware devices. The MQTT server can be repurposed to also communicate with Home Assistant, as MQTT is one of the protocols it supports. In order to

use Home Assistant with the Smart Coat Rack, one can simple connect Home Assistant to the MQTT server and tap into a topic in order to automate tasks.

3.6 CI/CD Pipeline

Our project also uses a CI/CD pipeline to automate the deployment process. Whenever a change is pushed to the main branch of the git repository, the pipeline automatically builds the Docker images for the frontend and the backend, and deploys the new containers to the server. This removes the need for manual deployment and ensures we always have the most up to date version of our application running.
It still prevents failed builds from destroying our application since only completed builds are deployed, ensuring a stable application.

3.7 Coat Hanger Design

3.7.1 Prototypes

For the design of the coat rack, we went through multiple iterations. The first major decision that had to be made was to choose between implementing a smart coat rack or smart coat hangers. In order to choose one of the two designs, prototypes for both were made so the product owner could interact with both and decide which one suited their needs best.
The main advantages of implementing a smart coat rack are the ease of use of hanging jackets onto a coat rack and that the coat rack does not have to be wireless and can be powered by a normal socket. On the other hand, the downside of a coat rack would be that detection of a jacket using NFC chips and readers would be harder, since their range is small and the jacket does not always hang the same way on a coat rack. Making a coat rack would also make it harder to scale the project when there are more jackets than expected, since an entirely new coat rack would need to be made.

Figure 3.7: The prototype for the coat rack

The benefits of making individual coat hangers are that the same jacket always hangs in the same way on a coat hanger, which makes the range of NFC chips and readers less of a problem. Individual coat hangers are also easily scalable, because an extra jacket only requires one additional coat hanger. The challenges of making a coat hanger are that it needs to be powered wireless and communicate wireless. The lack of space in a coat hanger would also mean limited room for hardware in the coat hanger.

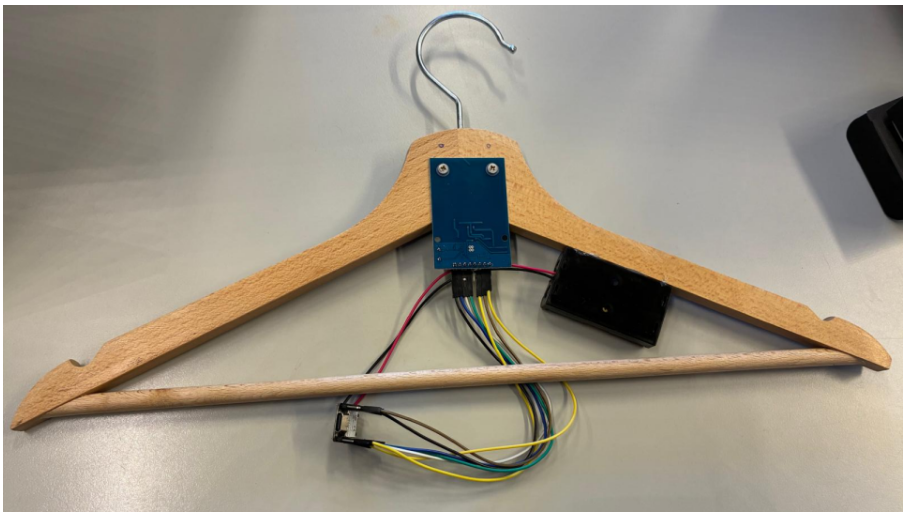

Figure 3.8: The prototype for the coat hanger

After careful consideration, the coat hanger was chosen as the definite prototype to expand on, as it would yield fewer problems with the NFC chips and readers and because it is easily scalable.

3.7.2 Final Design

For the final product, the coat hanger looks like this.

The coat hanger is 3D printed with five different components. It consists of a hook, two backplates, and two lids. The hook and back plates are assembled together, and the lids can be screwed on. The coat hanger is hollow, so the hardware needed for detecting coats can be put inside the coat hanger. The coat hanger has two holes, one for cables going to the NFC reader and one that could be used for a charging port in the coat hanger. It also has four screw holes to screw the back and front plates together. The hook is held into place by the back and front plates.

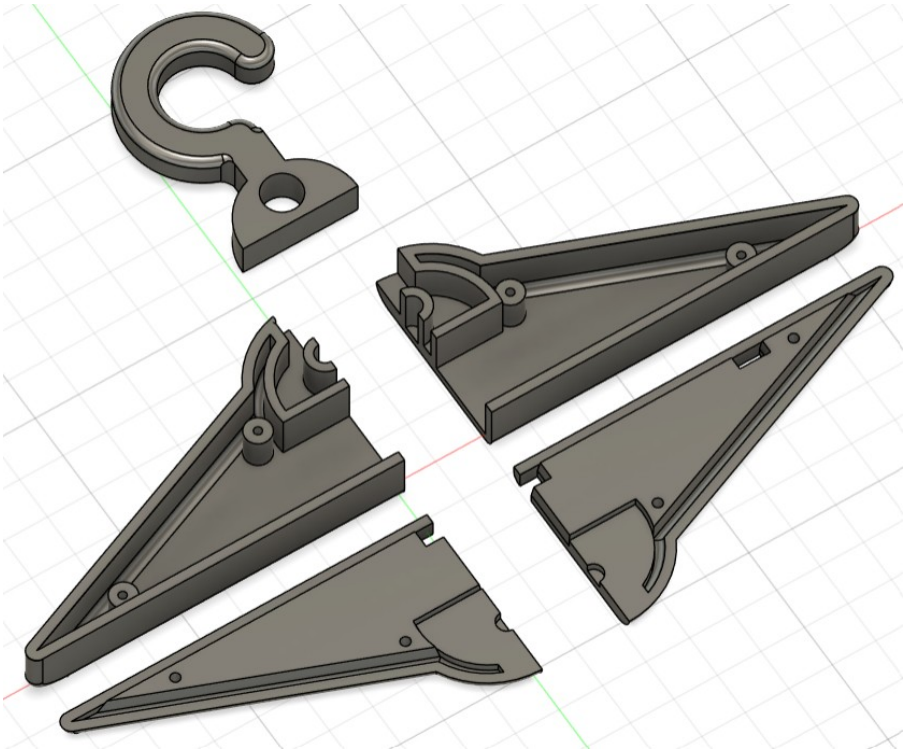

Figure 3.9: The design for the coat hanger with all different parts

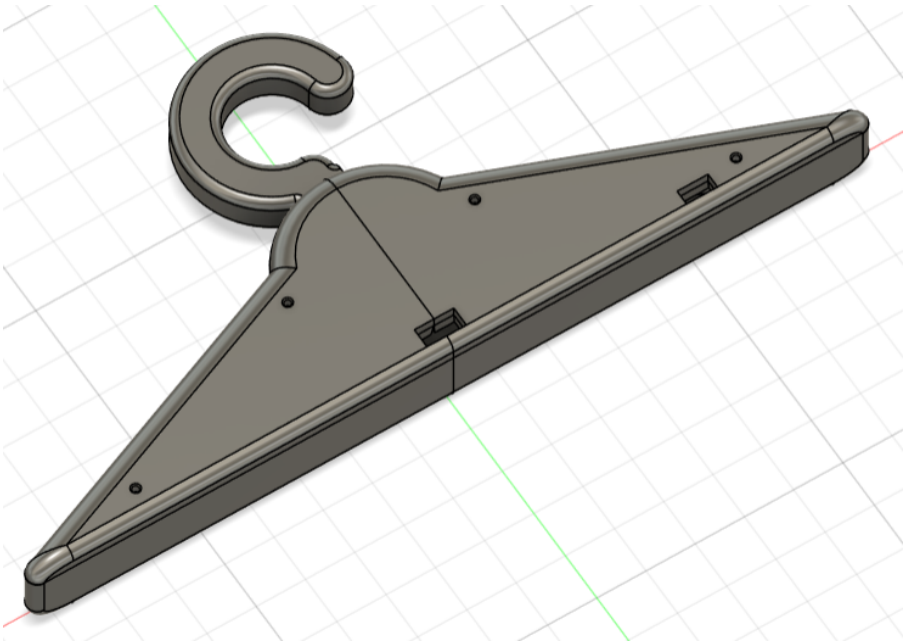

Figure 3.10: The design of the coat hanger assembled

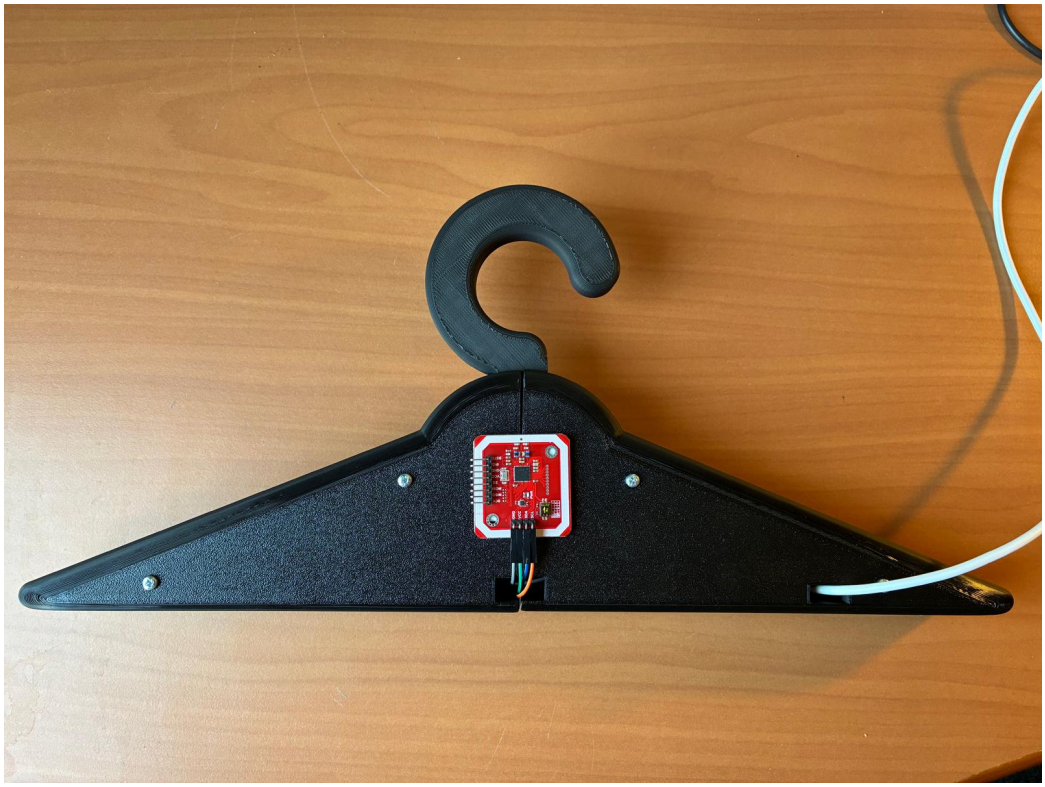

Figure 3.11: The design of the final printed hanger

3.8 Keybox

When we started our keybox design process we immediately thought of a simple wall mounted box with 4 holes inside of it that the 4 audio jacks could poke through. So we started on a simple laser cutting design which consisted of 5 panels that used box joints to fit together. The design had square holes in it. However we quickly found some issues with this design, if a user pushed the key in too forcefully the audio jack would sometimes fall out of the square holes, there was no backplate that could be used for mounting, and the box was to small to fit all of the wires that were needed. When we showed our prototype to the client he also mentioned that he preferred a 3D printable design over a laser cut design.

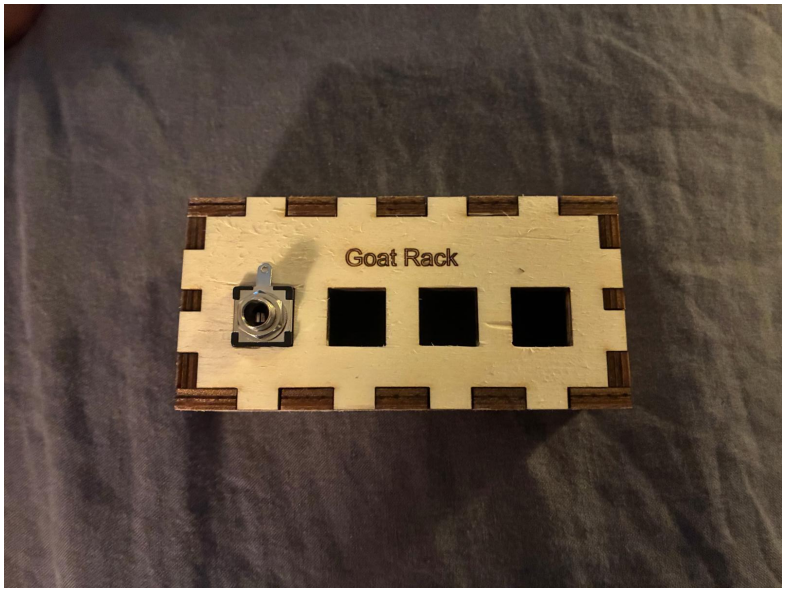

Figure 3.12: Prototype of our keybox

So we got to work on our finalized keybox model. It consists of a box and an removable backplate. The backplate has holes in it that can be used to mount it to a wall, and the box has circular holes instead of square ones. There are also 2 small openings in the box, one for a power cable and one for the led strip.
The backplate can then be attached using screws and heat inserts.

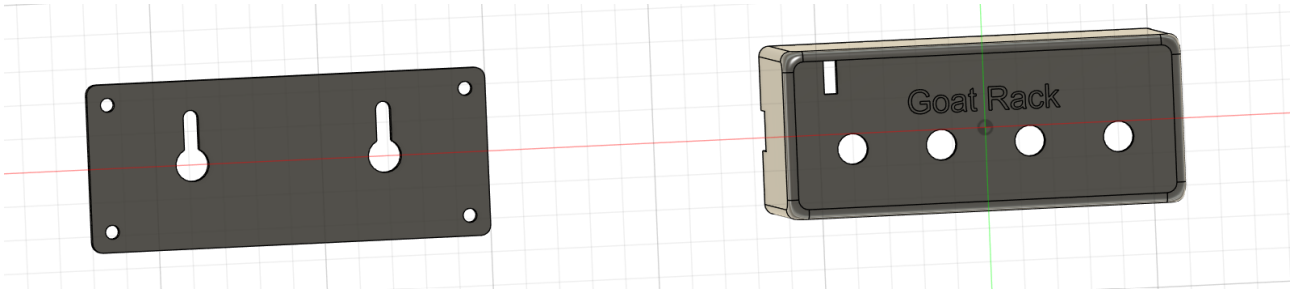

Figure 3.13: Our 3D model

Figure 3.14: The final keybox model

Chapter 4

Testing

In this chapter we will show the full test plan for the Smart Coat Rack system, covering both automated unit tests and structured manual tests. The goal of the test plan is to verify that all must-have and should-have requirements are met, and that edge cases are handled gracefully.
Our test plan is mainly divided into 2 types of tests, unit tests, which mainly apply for testing the backend of our product, and manual testing which is necessary to ensure proper functionality of our hardware and frontend and our overall system interactions.

4.1 Unit Tests

The unit tests mostly focus on the backend and the database, for this we used Vitest to be able to run these tests automatically. This is because the frontend framework we used, Angular, already comes included with Vitest. In our case the frontend was not complicated enough that it specifically required unit tests. If that were to change, we could use the same testing framework for both ends, and possibly reuse common functionality as well.
For instructions on how to run the unit tests, see Appendix A.2.

4.1.1 Backend

For the backend, we test that the controllers work correctly. In particular, we test that they perform a transformation on the database correctly (if applicable), and that they do not leave the database in an incorrect state upon failure.
Each controller has at least one code path with a `2xx` HTTP code, which is the successful path. The tests assert that the resulting object matches a particular TypeScript type, with the values being the ones entered in the request body (if applicable). Additionally, some tests have code paths for client errors (`4xx`).
Since tests run on controllers that modify the database, we have to do some setup to alleviate issues like modifications being visible between different tests running in parallel, or the database being in an unexpected form.

First of all, we use the `globalSetup` config option to run migrations. The server would normally do this, but we do not run it for the tests. Additionally, we use test fixtures to automatically wrap all tests in a database transaction, which is always rolled back once the test is done. This works similarly to the transaction wrapper function in the server routes, where a database client can be obtained through a mechanism that starts a transaction automatically, thus making it safe(r).
The way the Vitest test files in the project are structured is by using the `suite` function to group tests of the same controller, and then using the `test` function to run a particular test case. For example, there is a `suite` for `'login'`, which has three `test`s: logging in with valid credentials (`200`), logging in with a non-existent user (`401`), and logging in with the wrong password (`401`).

4.2 Manual Testing

Manual testing is mainly used for the frontend and hardware, since unit test do not suffice for testing on these parts of the product. For our manual testing we will go over the following main logic sections.

- User authentication (registration, login, token renewal)
- Coat-hanger RFID detection and MQTT messaging
- Key-box jack detection and MQTT messaging
- Jacket registration and person assignment
- Home-presence tracking
- Weather-based jacket recommendation
- LED feedback on the coat hanger
- Live frontend updates via WebSocket
- Device connectivity monitoring

Our manual testing has been quite in depth, since hardware and the frontend are difficult to test in any other way, so we moved the specific manual tests we did to the appendix in order for our testing session to not take up to much space in our report. For this you can see Appendix C.

Chapter 5

Future Work

In this chapter, we will go over the work we would have done or considered doing if we had a larger time budget.

5.1 Possible Improvements

5.1.1 Coat Detection

The RFID scanner is still not very strong, despite us switching to a different model of scanner that was stronger than the one we used initially. In the future, we could choose to implement an even stronger scanner in the hardware (e.g. an antenna), or switch to a different method entirely. A problem with this would be interference of multiple coat hangers' scanners, though one can argue this problem already exists with the current version of the coat hanger, but it is simply hidden by the low detection range.

5.1.2 Device setup

Currently, all devices are set up using Bluetooth. This allows a user to connect to a device over Bluetooth in the web interface, which then allows them to set a WiFi SSID and password. This feature works well, but unfortunately Bluetooth functionalities are only accessible using chromium browsers. This means that you will not be able to set up the devices using, for example, Firefox. This is a limitation that cannot be resolved if the interface is a web application, because other platforms simply do not allow it. One solution would be to convert the web application into a downloadable application. Frameworks exist that can convert Angular web apps to mobile applications, which can be used to overcome this problem.

5.1.3 Guest Coats

The system is currently is only able to handle interactions with coats that have a RFID tag in them. This means that a coat will need to be altered to be able to be used with the system. If a user has other people coming over to their house, their coats will have none of the

features of the smart coat rack. A possible improvement to the system would be to have a way of detecting coats different than the RFID. This could be achieved by for example using weight sensors in the hooks of the hangers.

5.1.4 JWT Token renewal

As was mentioned in the system design section, our current design supports only JWT token creation and redirection of a user back to the login page. An attempt of adding the token renewal section to the system was made, however, inability to solve the issue of past token being picked up by the system led us to reverting this addition. This limits users interaction with the system as they will have to enter their login credentials much more often. Once this issue is resolved, the JWT token will be used up to the proper level.

5.2 Possible Additions

During the stage of requirement elicitation, the requirements of MoSCoW format were gathered. Our group has successfully implemented all the *must* requirements and all the *should* requirements, leaving out the could requirements completely. Primary reason for not having those features was time constraint and some minor complications.

5.2.1 Mobile Push Notifications

Our current design of the smart coat rack system is focusing on users having the web platform open on their computer/phone. However, in case a user does not put their attention on the screen, they might miss some possibly important information that is specifically for them. That is easily solved with mobile push notifications, with the only constraint left being the web platform transformed into the (mobile) application. Of course, it is only the example of our thought process, meaning that the actual design and final implementation of mobile push notifications may vary from our vision.

5.2.2 Screen Display

The system uses mostly LED's to communicate with the user. However, a possible improvement would be to have a designated screen display that is hanged on the wall somewhere close to the coat hangers. The display could show the website of the system. With the screen, the user could easily check which coat to take, what the weather will be like, and where their keys are. The screen was not added in the current version of the system as the development team does not have the resources to permanently dedicate a device to the system.

5.2.3 Smart Coat Heating

One of the nice to have features for the coat rack is to have an option to preheat or dry clothes when requested by user. This would require a heating element integrated into the coat rack or its other supporting parts (like gloves hanger) in the most efficient way. Another nice feature on top of this would be that it could be based on a person's typical schedule, such that the heating could be turned on shortly before they leave.

5.2.4 Gloves Hanger

Aside from having a key box, that allows users to put their keys in it and later be reminded to take them when leaving, the same could be done for gloves. Having gloves hanger would allow users to, for example, dry their gloves after the rain, or, in case there is a heating element integrated into the gloves hanger, preheat them or dry them even faster.

5.2.5 Robotic Coat Rack

This addition is heavily robotics-oriented. This feature can be understood and implemented in many different ways. The vision our group had is to have a mechanism for storing and retrieving a desired coat from a storage compartment. A user would not have to search through the coat rack, the coat rack would be able to mechanically adjust itself so the coat would be presented to the user.

Chapter 6

Evaluation

6.1 Planning

For the planning of the project, we used Trello to have a good overview of what needed to be done and when the deadlines for those tasks were. This meant that every deadline was met and everything was finished on time. Trello also allowed a clear overview of the progress of the tasks. Therefore, it was instantly clear when a feature was ready to be tested and which features still needed testing. This made the design process smoother and easier.

6.2 Responsibilities

Name	Tasks
Ryan Cooijmans	Design prototype coat rack, Design final coat hanger 3D model, Report writing.
Tijn Hassing	Readers & audio jack, keybox 3D model, weather API, presence detection + registration (key & jacket), report writing.
Deniss Kornijenko	Jacket suggestion (Frontend + Backend), Weather info (Frontend), Report writing.
Daan Wensink	Neopixel and LED software, final coat hanger design, report writing.
Niek Zandt	Design and implement Frontend and Backend architecture. Soldering and wiring of the coat hanger and key box. Frontend UI. CI/CD Pipeline. Key box & Coat hanger setup protocol. Design Report writing.
Simon van Zeijts	Testing, Backend (+ unit tests), Frontend, Report writing.

6.3 Conclusion

Overall, we are quite happy with the result of the project, as we made a lot of progress in the 10 weeks we were given. We were able to deliver a working product, which implemented all our must and should have requirements.
Throughout our project we were given a lot of creative freedom, we just had to deliver a smart coat rack. We started off with gathering requirements, to find out what our clients actually wanted from their smart coatrack and we found that the ideas that we had brainstormed mostly aligned with our clients. Since we had started the requirements process quite early and since we met up daily, we already had a finished prototype around week 2. This quick start to the project and our frequent meetings, meant we experienced little time pressure even if we encountered some difficulty with our weak points, working with hardware and interface design (since none of us had a lot of experience with this).
If there is one area of weakness which we would have liked to improve more it would have to be the RFID detection reliability. This is mostly limited by our hardware and budget, but right now the coat detection only works in quite a small range on the hanger. If we had had more time we would have liked to look into alternatives to RFID scanning to detect jackets.
In conclusion this project was both a very valuable learning experience and quite an enjoyable process. Overall we are quite satisfied with the system we developed and the experience and skills we gained during this process.

6.4 Personal Reflection

Ryan Looking back at the project I had a lot of fun working on it. I really liked the creativity of coming up with our own version of what a smart coat rack entails and working that out. The planning was realistic and the responsibilities were distributed well along our team members, ensuring a nice workflow and achievable goals.

Tijn Before this project started I had not done much with IoT other than some not very in depth mandatory courses. This project has definitely opened my eyes a bit to how much fun it can be. I also really enjoyed using a bit more hardware than computer science students generally use during projects, and I feel like I have learned a lot of new skills.

We were also able to start quite quickly into the module, since we formed our team early and we were able to almost immediately have our design proposal finished, since we were able to get into contact with our supervisors quite easily. This quick start allowed for a mostly stress free project cycle.

If I were to redo the project I would start testing the system earlier. Now we found out quite late that RFID detection on the jackets only works with more expensive stronger readers. If we had found this out earlier (during our prototyping phase). Then we could have changed our design to make this more robust.

Deniss This project was once again about a complete new thing for me to discover and learn about. By being surrounded by smart devices, different people in various places talking about smart home, I was finally able to be on the developer side of a smart device, which in the end, I will say, can correspond to the term.

This project allowed us to approach the design of the required system with almost complete freedom, which I found really enjoyable as that uncovered many new ways of approaching and implementing certain features.

As the one without much of experience, I enjoyed and appreciated guidance and knowledge that was shared through-out this module.

Daan It was really interesting to design a system with almost complete freedom to decide on features ourselves. I learned a lot about interacting with a client and designing a full system from scratch. The mix between hardware and software was also interesting, as we usually do not work with hardware often.

This project I learned a lot about time management skills from my teammates. This is normally one of my weaknesses but with the motivation of the team and regular meetings it went really well for this project.

Niek I personally enjoyed the Smart Coat Rack project a lot. It allowed us to start the design and development process directly from the start without any prerequisites. We started prototyping and testing our ideas right from week one, which allowed us to quickly gather feedback on the system and helped us understand the scope of the project. I think that thanks to having a very short cycle we were able to complete almost all of the requirements that we established at the start of the project and I am very happy with the final result.

If I were to redo the project, I would make sure that the development was split more into better defined issues. Because at times I found that we could lose the overview on all the different tasks that we had. Luckily, because we met almost daily, we were able to overcome these issues easily.

Simon A fun project, more personal responsibility than most modules (much like the Research Project). The work division was not an afterthought, so we did not have to spend the whole module chasing deadlines. I would say the final result(s) are quite good, though we could have improved the communication by centralizing it more. For instance, only having a Discord server and not also a WhatsApp group, as well as tracking not some, but all, issues/tasks on our Trello board. What would sometimes happen is that we would send lists of things on both the Discord server and WhatsApp group, instead of referring to the Trello board.

Bibliography

[1] ARM. *ARM smart device glossary*. URL: https://www.arm.com/glossary/smart-devices.

Appendix A

Deployment

A.1 Main

A.1.1 Prerequisites

The prerequisites to run the software are:

- Node + NPM
- Docker (Desktop)

In order to test the software, the program **MQTTX** could be of use as well. In this program you can see all messages sent on specific streams, which is very useful for testing and debugging.
Furthermore, before running the software, you must:

1. Copy the `.env.example` file into a file with the name `.env`.
2. Fill in valid addresses and/or API keys into placeholder fields, for which the example file should provide instructions.

A.1.2 Setup

Before any deployment can happen, do the following:

`docker desktop start` to start Docker Desktop (it only needs to be in the background for Docker Engine to be running).

A.1.3 Deploy

1. Container cleanup might be necessary if the project has previously been built.

   `docker compose down` to remove containers (if any), or

   `docker compose down -v` to also remove the database volume.

2. Building can be done with (one of) the following commands:

   `docker compose build` to build all containers, or

   `docker compose build db backend` to skip the frontend container.

3. Deploying can be done with (one of) the following commands:

   `docker compose up` to start all containers, or

   `docker compose up db backend` to skip the frontend container.

   The `-d` option can be used (before the service names) to automatically detach as well. The `--build` option can be used (before the service names) to automatically run the build step as well.

The frontend can be started separately by executing `npm run start` from the `frontend/` directory. This is useful for development for two reasons:

- It compiles faster.
- It can reload on file changes.

The second aspect could also be achieved using Docker file watching, though we did not implement this.

A.2 Unit Tests

A.2.1 Prerequisites

The prerequisites are the same as in subsection A.1.1.

A.2.2 Setup

The setup is the same as in subsection A.1.2.

A.2.3 Deploy

To deploy tests, use the 'docker-compose.test.yml' file, along with specifying the service (e.g. 'backend-test'). For simplicity, we define '`prefix`' to mean '`docker compose -f docker-compose.test.yml`'.

1. Container cleanup might be necessary if the project has previously been built.

   `prefix down` to remove containers (if any), or

   `prefix down -v` to also remove the database volume(s) (`jacket_data` and `test_data`), or

`docker volume rm smart-coatrack-test_data` to only remove the `test_data` volume.

2. Building can be done with (one of) the following commands:

   `prefix build backend-test` to build the backend-test container.

3. Running tests can be done with (one of) the following commands:

   `prefix run --rm backend-test` to run the backend tests.

   The `--rm` flag is to remove temporary containers that are created, after running the tests. Additionally, the `--build` flag can be used here as well.

Appendix B

Meetings with Supervisors

In this chapter, we summarize what was discussed in meetings with supervisors.
In the first week, we arranged an initial meeting with Max on Monday week 2 (Roland could not be there). All meetings with supervisors were held in the IoT-Lab (Zilverling 2070).

B.1 Meeting 1

Meeting with Max on 2026-02-09.
We talked about how to get started, with respect to setting up the project, and what to do for the Design Proposal.
In particular, for the project, we went over:

- Getting credentials for the MQTT server through which the smart coat rack communicates.
- Getting hardware (Max agreed to provide ESP32's for us to use).
- Thinking about which API's to integrate.
- The privacy aspect of the project.

For the Design Proposal, the most important part was formulating the requirements (MoSCoW). The result of this ended up in Chapter 2.
Finally, we planned biweekly meetings on Wednesdays at 15:00, starting on 2026-03-04 (due to a peer review session followed by a holiday week).

B.2 Meeting 2

Meeting with Max and Roland on 2026-03-04.
We talked about the Design Proposal, the prototypes, and how to continue from here. Overall the Design Proposal was well-received, so we kept the requirements in this report mostly the same.

We attempted to give a demo for both prototypes, unfortunately there was a problem with the hanger. Nevertheless, we conveyed the pros and cons of each prototype, so that the client can decide which one they want. Namely, the coat rack prototype makes it easier to hang a coat, assuming the coat has a loop or hood, but makes scanning the tag less consistent. The coat hanger prototype makes it relatively more difficult to hang a coat, especially if it has wiring, but scanning is more consistent (as long as multiple hangers are not too close to each other. Ultimately, they chose the coat hanger prototype, which means we will now spend the remainder of the project developing the coat hanger.
The next steps include figuring out how to improve tag detection, and user feedback. For the tag detection, Max provided different scanners. For the user feedback, we discussed kinds of lighting on the coat hanger that could indicate certain actions like scanning a coat to the user. Other concerns included making user logins, key reminder functionality, and how to store keys (one or multiple boxes, key-chain system?).

B.3 Meeting 3

Meeting with Max on 2026-03-18.
We talked about our progress on the Reflection Report deadline, using the new reader, and making the keybox. We showcased the reader distance improvement, as well as how the keybox works. The keybox pins grounds' are positioned strangely, and we forgot to use a resistor, though the components should not be damaged as they can withstand the voltage. We then showcased the state of the website, including login, dashboard, key registration and presence. Max also asked if we group jackets in any way, and we responded affirmatively that an account can have many users (jackets). Then, we showed the current 3D design for the coat hanger. We asked for Max to send the dimensions of the battery, to ensure that there will be enough space in the coat hanger. We then talked about what properties the 3D model should have to be printed correctly, and potentially making test parts as well. We also received some LED's to use for signaling things to the user of a coat hanger.
We continued by saying what we will be working on, which is jacket recommendations and indicating to the user whether the jacket is registered. Max suggested using effects rather than many colors to represent certain coat rack states. We should also consider adding some kind of protection to the registration, so we do not give the entire internet the ability to create an account on an arbitrary network with a smart coat rack and MQTT.

B.4 Meeting 4

Meeting with Max and Roland on 2026-04-01.
We did a demo of the 3D printed hanger and keybox, as well as the new website design. We did a walkthrough of the registration/login/usage flow, as well as showcasing the Bluetooth hanger registration functionality. We also showed the people page, along with the modification

options. The frontend also has jacket recommendations now, which are always on.
As for things we have worked on and will continue to work on, we mentioned that we are trying to figure out how to do the lighting well, i.e., the lights give useful indications, and making it so the jacket does not cover the lights and hide these indications. As for the question on whether we should give each key its own light, we argued that a single indicator for all keys would suffice, as everyone hopefully knows which key is theirs. Furthermore, we mentioned that there has been work on setting up automated testing, as well as on writing the report.
We received some feedback, namely that the hanger looks much more complete now. Additionally, we were advised to test the hanger with more coats, and especially coats that have NFC tags attached to them in some way (e.g. sowing or with stickers). Roland also offered to bring an old coat for us to test with in this way.
We mentioned that we were done with most of the requirements. There was a common agreement among us that we should largely stop adding features, and move to mostly testing from now on. For instance, we might want to test the battery life of our current setup.
As for planning, the final presentation is on the 14th of April. Because of this, the meeting on the 15th of April has been scrapped, as there would not be much to do on that meeting. The demo market is on the 16th of April, Roland will be there with some colleagues.

Appendix C

Test Plan

This Chapter will show all the manual tests we have performed in order to ensure a working product.

C.1 User Authentication

Test	**Precondition**	**Steps**	**Expected result**	**Pass**
Successful registration and login.	No account exists for the test email.	Open the registration page. Enter a unique email and password and submit. Then log in with the same credentials.	Registration succeeds. The user is redirected to the dashboard and a valid JWT is stored in the browser.	✓
Login with incorrect password.	An account exists for the test email.	Attempt to log in using the correct email but a wrong password.	A 401 error is shown. The user stays on the login page and no token is issued.	✓
Login with nonexistent email.	No account exists for the email being used.	Attempt to log in with an email address that has never been registered.	A 401 error is shown. The user stays on the login page.	✓

Test	Precondition	Steps	Expected result	Pass
Registration with an empty required field.	The registration page is open.	Submit the registration form with the password field left empty.	The form is not submitted. A validation error is shown indicating the field is required.	✓
Duplicate email registration is rejected.	An account already exists for a given email.	Attempt to register a new account using the same email address.	A 409 conflict error is displayed. No duplicate account is created in the database.	✓
Session persists after page reload.	The user is logged in.	Reload the browser tab.	The user remains logged in and is not redirected to the login page.	✓
Logging out clears the session.	The user is logged in.	Click the logout button, then attempt to navigate to the dashboard directly via the URL.	The stored token is removed. The user is redirected to the login page and cannot access protected pages.	✓

C.2 Coat-Hanger RFID Detection

Test	Precondition	Steps	Expected result	Pass
Known jacket is detected on the hanger.	A jacket with a registered RFID tag is available. The hanger is powered and connected to the network.	Hang the jacket on the hanger so the RFID tag comes within range of the reader.	An `arrive` message containing the jacket ID is published. The dashboard shows the jacket as present.	✓

Test	Precondition	Steps	Expected result	Pass
Jacket removal triggers a leave event.	A jacket is currently detected on the hanger.	Remove the jacket from the hanger.	After the configured absence timeout a `leave` message is published. The dashboard marks the jacket as absent.	✓
Unregistered tag triggers the registration overlay.	An RFID tag that has not been registered is available.	Hold the unregistered tag in front of the reader.	An arrive message is published with the raw tag ID. The frontend displays a registration overlay prompting the user to assign the tag to a person.	✓
Two hangers in close proximity do not interfere.	Two hangers, each with a different registered jacket, are available.	Place the two hangers directly adjacent to each other and hang both jackets at the same time.	Each hanger detects only its own jacket.	✓
Hanger recovers after network disconnection.	A jacket is on the hanger and the system is running normally.	Disconnect the hanger from the network, wait 30 seconds, then reconnect.	After reconnection the hanger re-initializes the MQTT connection and an alive heartbeat is published immediately. Subsequent jacket events are transmitted normally.	✓

C.3 Key-Box Detection

Test	Precondition	Steps	Expected result	Pass
Registered key is detected on insertion.	A registered key is available. The key box is powered.	Plug the key's audio jack into a slot in the key box.	The presence pin triggers and the ESP reads the key ID. An arrive message is published. The dashboard shows the key as present in the correct slot.	✓
Key removal is detected.	A key is currently inserted in the key box.	Remove the key from its slot.	A leave message is published. The dashboard updates to show the slot as empty.	✓
Coat removed without collecting keys triggers reminder.	A person's jacket is on a hanger and their associated key is in the key box.	Remove the jacket from the hanger without first taking the key out of the box. Also try with multiple keys.	The backend detects the person is leaving while their key is still present. The LED strip on the key box lights up.	✓
No reminder when keys are collected before leaving.	A person's jacket is on a hanger and their associated key is in the key box.	Remove the key from the box first, then remove the jacket from the hanger. Also try with multiple keys.	The backend detects the key is already absent when the jacket leave event arrives. No LED lights up.	✓

Test	Precondition	Steps	Expected result	Pass
Unregistered key triggers registration overlay.	The key box is running. A key whose ID is not registered is available.	Insert the unregistered key into a slot.	The frontend displays a registration overlay for the new key ID. The key is not linked to any person until the user completes registration.	✓

C.4 Jacket Registration and Person Management

Test	Precondition	Steps	Expected result	Pass
Register a new jacket via the overlay.	An unregistered RFID tag has been scanned and the registration overlay is open.	Enter a jacket name, select a person, add at least one attribute, and confirm.	The jacket is saved to the database. The overlay closes and the dashboard shows the jacket as present and assigned to the selected person.	✓
Register a jacket without all data.	The registration overlay is open.	Complete the registration form without either a name, owner or attribute.	The registration overlay prompts the user to enter the required data.	✓
Delete a jacket.	A jacket is registered and not currently on any hanger.	Delete the jacket from the people page.	The jacket is removed from the database. If the same RFID tag is scanned afterwards, the registration overlay appears again.	✓

Test	Precondition	Steps	Expected result	Pass
Delete a person.	A person exists and has at least one present jacket registered under their name, another exists who has one absent jacket registered to them.	Delete both people from the people page.	The system prevents deletion with an informative error for person 1, and accepts deletion for person 2	✓

C.5 Home Presence Tracking

Test	Precondition	Steps	Expected result	Pass
Person is marked as home when their jacket arrives.	A person's jacket is absent from all hangers.	Hang the person's jacket on a hanger.	The presence list on the dashboard marks the person as home.	✓
Person is marked as away when their jacket is removed.	A person is currently marked as home.	Remove their jacket from the hanger.	After the absence timeout the person is marked as away on the dashboard.	✓
Multiple persons are tracked independently.	Two persons each have a registered jacket. Neither is currently home.	Hang both jackets on separate hangers at the same time.	Both persons appear as home in the presence list independently and simultaneously.	✓

Test	Precondition	Steps	Expected result	Pass
Last person leaving the household.	Only one person is currently home.	Remove that person's jacket from the hanger.	The presence list shows no one home. If the home-assistant integration is active, connected appliances are disabled as per the should-have requirement.	✓
Presence list is consistent after a system restart.	One or more persons are marked as home. The backend is then restarted.	Restart the backend container and reload the dashboard.	The presence list loads the last known state from the database correctly. No persons are incorrectly shown as home or away.	✓

C.6 Weather-Based Jacket Recommendation

Test	Precondition	Steps	Expected result	Pass
A suitable jacket is recommended based on current weather.	At least one jacket with weather-relevant attributes is registered. A valid weather API key is configured.	Open the dashboard.	The current weather conditions are displayed and the most suitable jacket is highlighted in the recommendation section.	✓

Test	Precondition	Steps	Expected result	Pass
One jacket is recommended when multiple jackets match.	Two jackets are registered with attributes that both match the current weather.	Open the dashboard.	Exactly one jacket is highlighted as the recommendation. No error or duplicate recommendation is shown.	✓

C.7 LED Feedback on the Coat Hanger

Test	Precondition	Steps	Expected result	Pass
LED turns red during setup.	The hanger has power.	Turning on the hanger and checking the LED color.	The LED will be red from the start of setup till setup is complete.	✓
LED stays red if there is setup issues.	The hanger has power, but no access to Wifi or MQTT.	Turning off the hotspot the hanger is connected to before starting the hanger.	The LED will stay red as setup cannot be completed.	✓
LED turns green for a registered jacket.	A registered jacket is available. The hanger is powered.	Hang the registered jacket on the hanger.	The hanger LED turns green, indicating the jacket is recognized and registered.	✓
LED resets when the jacket is removed.	A registered jacket is on the hanger and the LED is green.	Remove the jacket from the hanger.	After the absence timeout the LED turns off or returns to its idle state, indicating no jacket is present.	✓

Test	Precondition	Steps	Expected result	Pass
LED returns to correct state after reconnection.	The hanger LED is red due to a lost connection. A jacket is on the hanger.	Restore the WiFi or MQTT connection.	The hanger reconnects, and the LED returns to green since a registered jacket is present.	✓

C.8 Device Connectivity Monitoring

Test	Precondition	Steps	Expected result	Pass
Connected device is shown as online.	A hanger is powered and connected to the network.	Open the device status section of the dashboard.	The device is shown as online. The last-seen timestamp reflects a recent heartbeat.	✓
Device is marked offline after extended absence.	A hanger is shown as online on the dashboard.	Power off the hanger and wait longer than 60 seconds.	The dashboard marks the device as offline.	✓
Device returns online immediately after power-on.	A hanger is currently marked as offline.	Power the hanger back on.	The hanger publishes an alive heartbeat immediately on boot. The dashboard marks the device as online without waiting for the next scheduled heartbeat interval.	✓

Test	**Precondition**	**Steps**	**Expected result**	**Pass**
Brief network interruption does not cause a false offline state.	A hanger is online and shown as such on the dashboard.	Disconnect and reconnect the network within 30 seconds.	The device remains shown as online throughout. No offline state flashes on the dashboard.	✓
Multiple devices are tracked independently.	Two hangers are both online.	Power off one hanger and wait longer than 60 seconds.	Only the powered-off hanger is marked as offline. The other remains shown as online and is unaffected.	✓
Alive heartbeat is published immediately on first boot.	A hanger has just been powered on for the first time.	Observe the MQTT traffic using MQTTX immediately after the hanger boots.	An alive message is published within a few seconds of boot.	✓