NTNU
Kunnskap for en bedre verden

Department of Computer Science (IDI)

Faculty of Information Technology and Electrical Engineering (IE)

TDT4290 - Customer Driven Project

Lildog IoT Testing Framework

Authors:

Adele Felicia Ellingsen-Grønningsæther
Elin Haugum
Haakon Karstensen
Jacob Benjamin Alveberg
Olav Selnes Lorentzen
Timon Alexander Selnes

21 November 2024

Table of Contents

List of Figures

List of Tables

1 Executive Summary

This report details the work completed by our team for the Lildog IoT Testing Framework as part of the TDT4290 - Customer Driven Project course. The project ran from August 28th to November 21st, 2024, with the goal of creating a framework to simplify testing for Internet of Things (IoT) devices, focusing on compatibility, connectivity, and ease of integration.

The motivation behind this project was to provide startups and developers with a free, open-source solution for IoT testing, reducing the reliance on costly proprietary systems. By developing a custom backend solution, we ensured the flexibility and modifiability needed to meet Lildog's specific requirements while leaving room for future development.

Throughout the project, we followed an agile development process, combining elements of Scrum and Kanban. Bi-weekly sprints, regular team meetings, and tools like Figma and GitHub helped us stay on track and adapt to feedback from our client. We also placed a strong emphasis on sustainability and accessibility, ensuring the software was not only functional, but also inclusive and future-ready.

The final product is a Minimum Viable Product (MVP) that meets the essential requirements set by the client, offering a foundation for further development. While some advanced features, including machine learning (ML) to enhance testing, were left for future iterations, the current version delivers a practical and user-friendly framework that we hope will support the growing needs of the IoT community.

We are grateful to our client, our supervisor, and the Faculty of Information Technology and Electrical Engineering at NTNU for their valuable support and guidance throughout this project.

Links to Resources

1. **Figma:** https://www.figma.com/design/Z7pVMeKVYEedF5ZlFmmizW/Lildog-Design-Suggestion?node-id=0-1&t=0nSQInPbrdZQzIX1-1
2. **Code repository:** https://github.com/EliHaugu/liltest
3. **Presentation:** https://docs.google.com/presentation/d/1QdVlFQTIPiEyJG4AJJIr5gbRV0tWjzm_rIXAMHihW8k/edit#slide=id.p

2 Introduction

This project report details our efforts, reflections, and product for the course "TDT4290 - Customer Driven Project" at the Norwegian University of Science and Technology. This course assigns each group of students to an industry customer to create a product. Our six person group, with two Informatics B.Sc. students and four Computer Science M.Sc. students were assigned to the local Trondheim company Lildog AS.

2.1 Customer

Lildog AS is a Norwegian tech company, known for creating innovative GPS trackers and activity monitors for dogs and cats. Founded in 2016, Lildog's mission is to improve pet care by making it easier for owners to monitor their pets' health and activity. Their flagship product, the Lildog tracker, provides real-time location tracking, activity monitoring, and other practical features. With over 50,000 products sold and 4.8 million activities logged on their app, Lildog is dedicated to using technology to enhance the lives of both pets and their owners (Lildog 2024).

2.2 Motivation

The idea for this project came from a common challenge faced by startups in the IoT industry; Testing IoT devices is often expensive and time-consuming. Lildog wanted to address this issue not just for themselves but for other startups as well. They envisioned an open-source testing framework that would be simple to use, free, and flexible enough to meet the needs of growing businesses. This framework would help companies save time and resources while fostering innovation and collaboration in the IoT community.

2.3 Project

The main goal of the project was to create a user-friendly IoT testing framework tailored to Lildog's needs, while also being open and accessible to others. Our team focused on developing a solution that could test device connectivity and interaction, all while being easy to maintain and extend in the future. By building this framework, we aimed to give Lildog a practical tool for testing their devices while setting the stage for further growth and innovation in the IoT field.

2.4 Stakeholders

This project involved several key stakeholders, each playing an important role in its success:

- **Lildog AS:** The customer who provided the vision and requirements for the project. Their insights and feedback were critical to shaping the solution.
- **Project Team:** A group of students from NTNU responsible for designing, developing, and delivering the final product.
- **Supervisor:** A faculty member from NTNU who offered academic guidance, ensured the project aligned with the course objectives, and supported the team throughout the process.

2.5 Final product

In the end, we delivered an MVP that met Lildog's key requirements. The framework allows users to set up and run tests for IoT devices and track their progress in real-time. While advanced features like ML-based testing weren't included in this phase, the MVP lays a strong foundation for future development.

3 Pre-Study

3.1 Node-RED

Issue and Context

Our IoT testing software needed to support a variety of devices and protocols, as well as potentially image-based recognition using ML. Additionally, since it would be open source, it should be easy to modify and extend for future contributors.

After presenting their requirements for our software, the customer suggested we explore Node-RED, a graph-based, visual programming tool for connecting IoT devices and building automation flows. This tool has a strong community, widespread usage in the industry, an extensive library of nodes with various functionalities, and support for many IoT devices and protocols. This would mean a significant portion of the groundwork for device communication would already be handled, and as few of us had experience with IoT, we considered using Node-RED as a base for our project.

Options Considered

Our key considerations for our options were that they could meet the requirements of the customer, the program would be maintainable, and that it would be feasible within our timeframe.

We began by exploring the basics of Node-RED and using the tool with our own IoT devices to understand its strengths, weaknesses, and use cases. Thereafter, we looked into how and where Node-RED was being used, as well as other tools it could be integrated with, such as "Locust" (Pinker 2021) or being able to import and use any npm module (Pavitt and Steve McLaughlin 2023). While we did see similar tools that could aid our development, these were mostly designed for different purposes and use cases.

After learning about the use cases of Node-RED, we looked into using it as our backend through their API and leveraging its existing functionalities to handle basic IoT communication while building a custom layer on top for testing needs. Lastly, we researched tools that could aid us in creating a custom backend solution where we would create the entire system from the bottom-up without the use of Node-RED.

Based on our research into Node-RED, we landed on the following three options:

1. Build system as a support tool and custom nodes for Node-RED

This option entailed building a library or custom nodes for use in Node-RED.

One of the key benefits of this approach is that we could build upon the large, active community of Node-RED and existing support for various protocols and devices. The application covers basic functionality necessary for our application, such as multi-user editing, real-time logs, and flow management and export. In addition, it provides many pre-built nodes, e.g., for IoT, HTTP, web sockets, and more, as well as the possibility to add custom nodes, such as ML integration (Yokoi n.d.) and image processing (Stephen McLaughlin n.d.). This approach would allow us to potentially develop more advanced features for our customer as the groundwork would have been laid for us.

However, while investigating Node-RED, we noticed several issues. There was limited support and documentation for testing tools, and we had difficulties getting custom nodes and testing libraries to work as many were deprecated. The available tools, such as `node-red-node-test-helper` (Node-RED n.d.) and `node-red-contrib-flowtest` (Unknown n.d.), were not user-friendly or well-integrated into the flow management. Node-RED was primarily designed for automation and flow management rather than testing, making it less suitable for complex testing scenarios.

In addition, using Node-RED would give us a steeper learning curve as we would first have to learn the ins and outs of its structure and then figure out how to build upon that. This would

leave less time for core development and implementing customer requirements not already present in Node-RED. Since we would be dependent on it, we would also have limited flexibility if the customer requirements changed during the project as we have limited control over its core functionality. This would increase the risk of unforeseen issues as we delve deeper. In addition, the customer would be required to learn Node-RED to be able to use our application. Furthermore, it would be harder to customize the application and maintain the open source project due to the dependencies on Node-RED's core system and potential changes in its architecture. These findings were supported by other developer teams (Bettiche 2018).

2. Use Node-RED as backend and build our own system on top of it

This approach entailed using an API or other connection point to Node-RED to allow us to use its core functionality such as device, connection, protocol and custom node support, while building our own frontend and testing system on top of this.

This option had the same benefits and many of the disadvantages of the first solution, aside from the fact that the customer will not need to learn Node-RED. However, the API and support for building our own system on top of Node-RED was not well documented and lacked support. Customising it without native support could prove difficult, risking diverting too much time away from focusing on the customer's requirements.

3. Create a custom backend solution

This option involved creating the entire system from the bottom-up ourselves, including all communication between devices and custom support.

The main advantage of this approach was that it allowed for greater control over each aspect of the program's development and its architecture, from connections, devices, and protocols to user interface (UI) and testing flow management. This could make it easier to create an MVP tailored to the customer's specific needs, with testing as the primary focus of the application rather than as a secondary feature. Additionally, it posed less risk within the given timeframe, as we would avoid spending time learning external programs, giving us more time to focus on customer requirements and optimising performance for their specific use cases, such as multi-device usage. In addition, since we would not be dependent on an external program, it would make the system more flexible and easier to adapt to future changes or requirements. While we would still need to manage dependencies within our tech stack, these versions can be locked for stability, ensuring better control and maintainability over the long term.

Despite the customer encouraging investigating Node-RED, they also recommended building the application from scratch. From their experience it was often less work to have full control rather than to work around an external program. While we would not have to learn any external programs by choosing this approach, we would still have a steep learning curve ahead as we would need to learn how to implement the base functionality Node-RED already provides. There would be no existing resources or community to speed up development.

Without this, we would need to implement support for devices and protocols from scratch, which could be challenging given that none of the group members had prior experience with this type of application. This posed a significant risk, as it might prove more difficult than anticipated to develop an MVP that not only meets the customer's needs but also goes beyond the basic functionalities already available on the market. Given our timeframe, the number of devices and protocols we could realistically support would also be limited.

Decision

Based on the risk of depending on an external program, especially with our limited timeframe, and because Node-RED was not natively built for our customer's use case, we decided to go with option 3: Building our own custom backend solution.

This would allow us full control over our system, ensuring it could be tailored to our customer's specific needs, be more robust to change during development, and allow for easier maintenance in the future by avoiding external dependencies.

3.2 Backend Technology Decisions

Language

For the backend development of this project, the team opted to use Python. Known for its simplicity and readability, Python is easy to use, which allows for rapid development, making it a natural fit for this project. All members of the development team have prior experience with Python, which ensures a smooth and efficient workflow. Additionally, its extensive libraries offers tools that simplified working with IoT devices, which was particularly valuable given our limited prior experience in this area. Python also provides robust testing frameworks enabling seamless integration of automated tests to improve maintainability. Lastly, the customer also suggested using Python, and since our project is open source, this choice ensures that future contributors, including the customer, can easily maintain and modify the codebase.

Framework

With Python as the language, the team looked at the Python-based frameworks Flask and Django. While Flask was recommended by the customer, the team selected Django due to its "batteries included" approach, open-source nature, and wide usage across the industry, as well as previous experience with Django on the team.

Django provides built-in features such as the admin panel and ORM, which supports rapid development and allows the team to focus on implementing project-specific functionality. In addition, its modular Model-View-Template (MVT) architecture, particularly its separation of Models and Views, is highly fitting for our open source project as it aligns well with the quality attribute (QA) criteria of modifiability. This design simplifies view logic and allows changes to the data layer or business logic without affecting other parts of the system, ensuring maintainability and modifiability.

Lastly, Django's extensive community and its long-standing use in both small and large-scale applications means it is a well-tested and reliable framework that would likely be able to meet evolving needs of this project. It also offered good support for integration with the database we chose. Furthermore, as an open-source framework, future development teams can continue to contribute to the codebase with ease, ensuring its longevity and adaptability over time.

Database

The team chose PostgreSQL as the database system for this project, primarily due to its open-source nature, previous experience within the team, and the desire of some team members to learn the system. PostgreSQL's proven reliability, scalability, and compliance with industry standards make it a strong choice for handling the data storage needs of this project. The choice of PostgreSQL aligns with the future maintainability of the codebase, as it is widely used and supported within the open-source community, and it was also recommended by the customer. This ensures that future developers, including the customer, can easily adapt and extend the database structure, allowing the project to evolve over time without being locked into proprietary systems.

Automation and Continuous Integration Workflow

Early on, we discussed and decided on tools for our continuous integration (CI) pipeline to maintain high code quality and facilitate rapid development. Prioritizing this from the beginning not only minimized the need for refactoring later, but also enabled faster iterations. Clean, well-structured code made it easier for team members to understand, build upon, and review code they hadn't written. This clarity ensured that changes and new features could be added easily without introducing unnecessary complexity or technical debt. Especially since our project is an open source MVP intended to be extended in the future, maintainability was a key consideration. Our automated CI pipeline consisted of the following tools:

- `flake8`, `isort`, and `black` for enforcing a common code standard through linting and formatting to ensure clean code
- `mypy` for catching potential bugs early through static type checking and enforcing type consistency to enhance code clarity and reduce debugging time
- `pytest` for running automated tests to ensure new changes do not break other core functionality

3.3 Frontend Technology Decisions

Language

To enforce a maintainable codebase, we have chosen to employ Typescript for the client-side business logic and dynamic elements. This approach allows us to achieve static typing, in addition to introducing concepts such as inheritance and types, enabling easier maintenance after delivery and ensuring that errors are caught during compile time rather than in runtime environments.

Framework

The UI for the project is created using Vue.js, a versatile and approachable framework for developing web applications. Given the project's intent of being an open-source solution, it was important that the codebase was approachable. Using Vue.js allows for swift development cycles, with a highly maintainable component-based codebase that offers a high readability for new and future developers. Additionally this was the preferred framework requested by the customer for this project.

Styling

To expedite development cycles and ensure consistency, we adopted Tailwind CSS as our primary cascading style sheets (CSS) framework. By leveraging Tailwind's utility-first approach and prebuilt CSS classes, we were able to rapidly prototype and iterate on UIs. This approach also facilitated the creation of a well-structured, maintainable codebase by enforcing a design system. Furthermore, Tailwind's utility-first methodology eliminates the need for multiple CSS files, enabling us to directly style components and elements within their respective files, promoting code clarity and organisation.

Automation and Continuous Integration Workflow

To ensure a consistent code base we employed a CI pipeline for our code base. For the frontend part of the repository we implemented it with ESLint and Prettier to enforce code best practises guidelines and consistent formatting. ESLint is a static code analysis tool allowing us to uncover problematic code based on community guidelines and best practises, while Prettier allows us to set formatting rules for our code base to ensure consistency.

3.4 IoT devices

Since our project involved testing IoT devices, we also needed devices to use during development to ensure our project was working as intended. For this, the customer recommended and offered Nordic Semiconductor nRF development kits. The single board development kits supported both BLE (Bluetooth Low Energy) and UART (Universal asynchronous receiver-transmitter), and support for these protocols was one of the key concerns of the customer. The customer was also using these kit in their own development.

In addition, Nordic Semiconductor provided several helper applications and resources for development and testing with the nRF kits, which would help debug potential issues in the test flows and connections. One team member already had one of these kits, and for these reasons the nRF development kits became a natural choice for which devices to use during testing of our application.

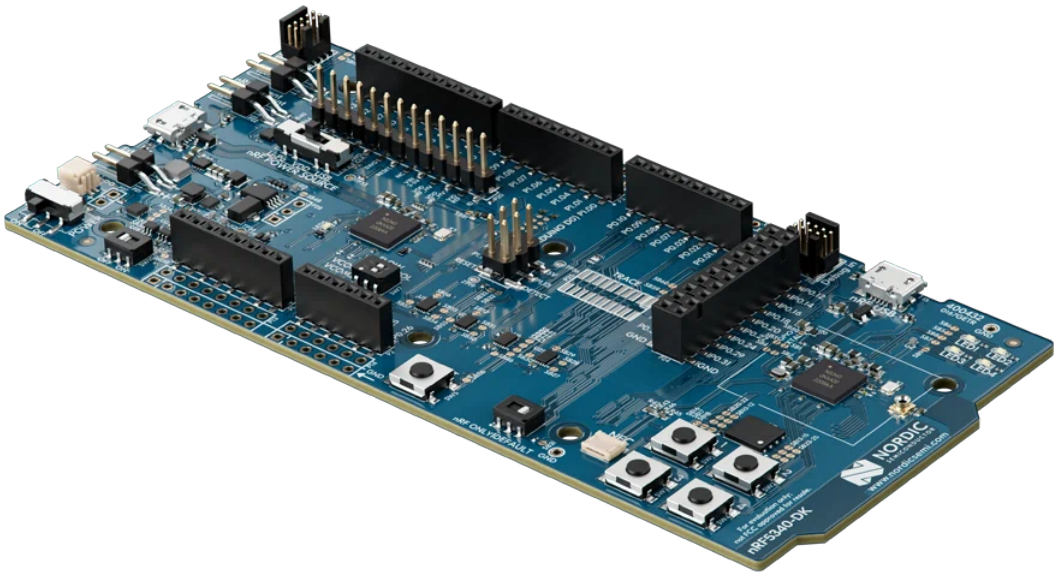

Figure 1: nRF5340 DK - One of the many nRF development kits

Since the customer wanted to test connections between nRF kits and Android phones specifically, our environment also required supporting tools to facilitate this. This included the `nRF Connect SDK`, `nRF Connect app`, `nRF Logger app`, and `Android Studio`.

3.5 Monitoring Device Communication

Issue and Context

Since the customer mentioned various communication protocols between devices, we needed a way to capture the data flow between them and their respective logs. This would be necessary both for asserting changes on devices and sending commands. We also potentially needed to assess whether the connection itself was working correctly, rather than just asserting if the results at each endpoint were correct.

Options Considered

Our key concern was being able to check the device states, either by actively polling or passively listening. We were unsure whether we would need to insert ourselves between the devices to assess their connection, so we presented multiple options to our customer.

- **Sniffing (Passive Listener)**: Passively listens to communication between devices, tools online required all communication to be public and unencrypted.
- **Broker (Active Participant)**: The host computer acts as a broker, managing connections and enabling explicit control over device data (EMQX Team 2024).
- **Man-in-the-middle (MitM)**: Combines sniffing and active participation by intercepting communication unnoticed, but tools are primarily designed for security testing.

For an in-depth analysis of the use of these approaches, see G.

Decision

We presented these options to the customers, who clarified that assessment of and direct access to the connection between the IoT devices was not necessary. Since the host computer would only

need to access the current states at each connection endpoint, we selected option 2, where the host computer acts as a broker between the devices.

4 Planning

4.1 Development Methodology

During our first team meeting, our top priority was to decide on a development methodology. The reason for this is that a set structure for roles, meetings, and planning would allow the team to more efficiently discuss during the planning and pre-study phase. The team's most highly valued factor when choosing a development methodology was previous experience and familiarity. Choosing one that we were already familiar with made it easier to quickly get structured. For this reason, the team started off with Scrum as all team members have used this in other courses and generally have positive experiences with it. There were, however, a few aspects of Scrum that most of the team had regarded as disadvantageous in previous projects, which will be discussed later in this section. The customer did not have a preference for what methodology the team used, but he relayed that the company used Scrumban, a combination of Scrum and Kanban. He recommended this approach if we felt that pure Scrum was too prescriptive. Based on these factors, the team landed on Scrum with Kanban influences as the project's development methodology.

4.1.1 Scrum Roles

Using Scrum, the team assigned three roles: Product owner, Scrum master and development team. The responsibility of the product owner is to set the product vision and priorities (Kniberg and Skarin 2010, p. 11). The natural choice for product owner was then the customer since he was best aware of what his company needed from the project. Traditionally, the product owner would be present at both the sprint demo and sprint planning, however, the customer did not have the time and expressed a disinterest in too many organized meetings. To solve this, we also nominated a proxy product owner that represented his interests to make sure the team prioritized according to his needs (Kniberg and Skarin 2015, p. 17).

The Scrum master, responsible for removing impediments and providing process leadership, was chosen based on interest in the role and experience in previous projects (Kniberg and Skarin 2010, p. 11). These factors were important as the team felt experience leads to a more efficient Scrum structuring, and interest leads to proactivity in organizing activities and driving meetings. All team members were set to be a part of the development team per the course's requirements.

Scrum prescribes small, cross-functional, self-organizing teams, while Kanban allows for specialized teams (ibid., p.50). We decided to divide the team into backend and frontend teams, adopting Kanban's approach to specialized team division. The main reason for this was that we deemed the project as technically complex due to integration with IoT devices, and it would be very time consuming for all team members to learn this technology well. While all Scrum activities were still completed together, each sub-team focused on its area of expertise. This allowed each team to determine decisions within their respective fields, delegating tasks within their teams to enhance the decision making process to relevant parties. This approach ensured that decisions were made by those most knowledgeable in each area, leading to better decision-making and smoother workflows. It also created a feedback loop where each sub-team implemented tasks based on their choices, further improving their expertise. Code reviews were more effective as well since team members reviewing the code were highly familiar with the standards and requirements of their specialized area.

4.1.2 Sprints

Scrum imposes division of development into time-boxed iterations called sprints that start off with sprint planning and end with a sprint demo and retrospective (Kniberg and Skarin 2010, p. 13). Such sprints allows for a set feedback loop from the customer to the development team, which the team thought was advantageous. The team decided on two-week sprints for three main reasons for this. Firstly, any sprint length shorter than two weeks would not be enough time to make significant progress to show the customer during the sprint demo, considering that the team was not working full time. Secondly, a shorter sprint would result in more overhead from additional planning, demo and retrospective meetings, which would slow down development. Lastly, the team felt sprints longer than two weeks could lead to too much development without any feedback from the product owner potentially resulting in wasted work.

4.1.3 Scrum Activities

Sprint Planning

After establishing a goal, the team would figure out which tasks would need to be completed on the frontend and backend to meet said goal. In Scrum, one would prioritize user stories or functionality as a whole and divide them into atomic tasks (ibid., p. 37). This did not suit the project because the frontend had an iterative development, and the backend had a more linear development (see section 8), which meant that user stories were not completed as a whole during each sprint. To address this, we decided on a hybrid approach, combining Kanban's flexibility with Scrum's structured feedback loop. Sprints provided a regular interval for customer feedback through the demos, but the team did not rigidly commit to specific tasks within a sprint. Instead, sprint goals served as guiding objectives, and tasks could be adjusted or re-prioritized during the sprint as needed (ibid., p. 50).

Another modification made to sprint planning was task estimation. Scrum typically uses a form of estimation for each task to figure out the effort required to complete it, and chooses tasks from the backlog based on the team's velocity (ibid., p. 31). The group's consensus, based on experience in previous projects, was that task estimation is time consuming and often leads to inaccurate estimates. In addition, since the team was not working on the project full-time, the amount of time available for each person to work on the project varied somewhat from week to week because other courses could have weeks with varying workloads. Kanban, as opposed to Scrum, makes estimation optional. We adopted a mix of Scrum and Kanban's approach by using deadlines instead of estimating effort for each task. During each planning session, the team members evaluated how much time they could spend on the project in the upcoming sprint. Based on this availability, each task was assigned a deadline set by the assignee. When each person had enough tasks assigned to them such that their last deadline reached the end of the sprint, the sprint backlog was set.

Daily Standup

When using Scrum, standup is performed daily (ibid., p. 38), but with a part-time project, this would be inefficient. As the course estimated we should use three days a week on average for the project, the team decided initially on three standups a week to mimic the implementation of daily standup for full-time projects. However, after two weeks of this arrangement, the team felt that three meetings were too many, causing unnecessary overhead. Thus, we reduced to two standups a week. These were led by the Scrum master, who asked each team member in turn to explain shortly what they had done, what they were planning to do next, and if they had any obstacles. The main objective of standups are to uncover any blockers that are hindering the development. By encouraging each team member to concretely state what they are working on and planning to work on, it was easier to reflect on what had been causing issues. By revealing obstacles, the team could figure out how to best resolve them.

Sprint Demo

At the end of every sprint, the team had a sprint demo with the customer. During this meeting

we would provide a progress update and demo any new functionality to get feedback from the customer. After that followed a discussion about priorities for the next sprint. The feedback and priorities during this meeting would serve as the basis of the product owner proxy's input during sprint planning as mentioned in section 4.1.1. These meetings were also instrumental in uncovering any misunderstandings between the team's and the customer's understanding of the scope and requirements.

Sprint Retrospective

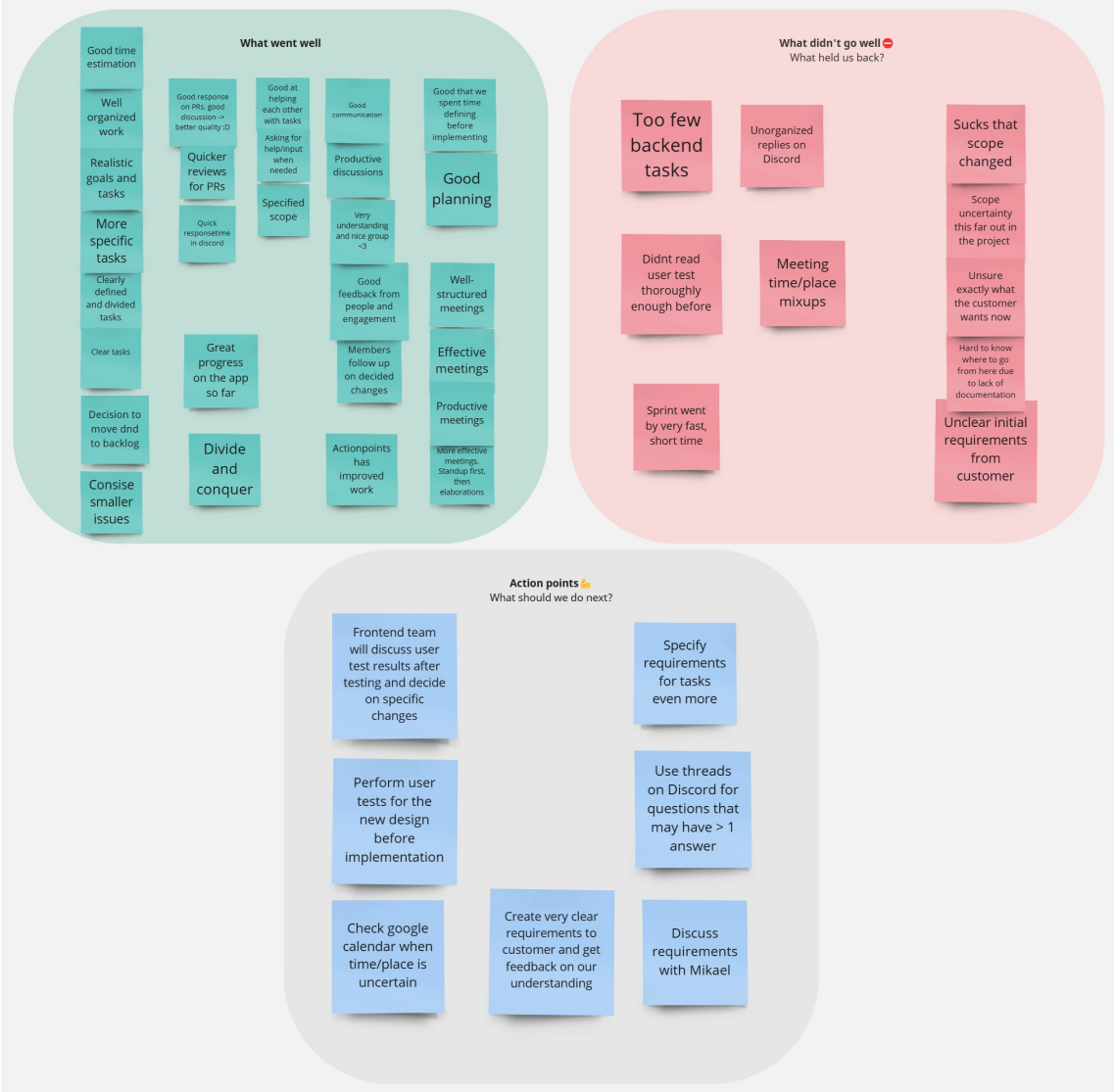

Figure 2: Miro board for retrospective

A sprint retrospective would be held directly after the sprint demo. It was divided into two parts, starting with the Scrum master reminding the team of the sprint goal and whether it was accomplished or not. This served as a reminder for the team of the events that occurred during the sprint, helping them to better recall and consider these details when evaluating its high and low points later. After sprint 1, we decided to add another section to the retrospectives where we reflected over the action points from last retrospective. This was implemented because the team wanted to reflect on whether the previous action points were completed and if they had the intended effect.

After this we moved on to Miro where the Scrum master had created a sprint retrospective board divided into three squares: What went well, what could have gone better, and action points as

shown in figure 2. The team had about 10 minutes to write their reflections on digital post-it notes for the two former points, which were then discussed. The Scrum master would read out the emerging themes and ask the team for any additional input or reflections for the positives and the negatives. Once finished, we would discuss if there were any actions that could be taken to combat the negative themes.

Traditionally, dot voting would be used to decide which improvements would be chosen as action points (Kniberg and Skarin 2015, p. 38). We decided not to use this during the project for two reasons. Firstly, we noted that while there were generally about seven improvements suggested, they were small changes and often related to only to a specific role or subsection of the group. For instance, asking the group leader to give a summary of the group leader meetings. Thus, we felt that we had the capacity to accomplish all action points. Secondly, we had experienced from previous projects that with real-time, public dot voting, team members seemed to often be influenced by whomever voted first, leading to one or two people deciding the action points. However, after having completed the project, we have found that for two out of five development sprints only half of the action points were implemented. In future projects, we would therefore like to try a form of voting again to better prioritise the action points. The issue of voting influence is still relevant, so we would attempt voting all at once by show of hands to eliminate this issue.

4.2 Meetings

The meetings the team used for internal and external coordination described in section 4.1.3 were not the only ones. We also had weekly meetings with our supervisor to get feedback on the process and report, and to get advice regarding customer misunderstandings. Figure 3 shows the meeting schedule for odd weeks where the sprint would start on Wednesday, followed by the meetings in even weeks illustrated by figure 4. The sprint would end on Tuesday of the subsequent odd week.

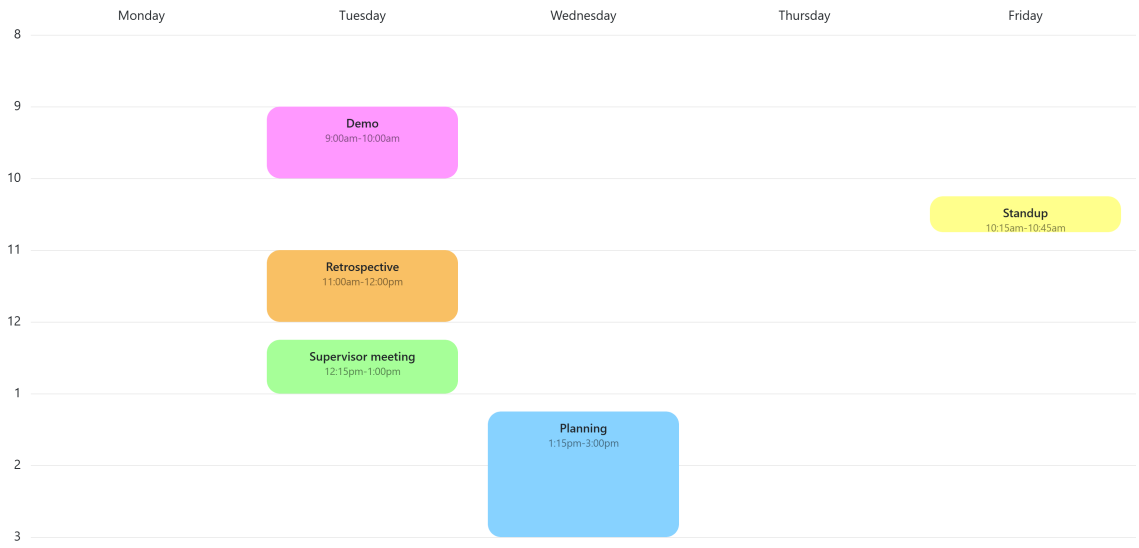

Figure 3: Meeting schedule odd weeks

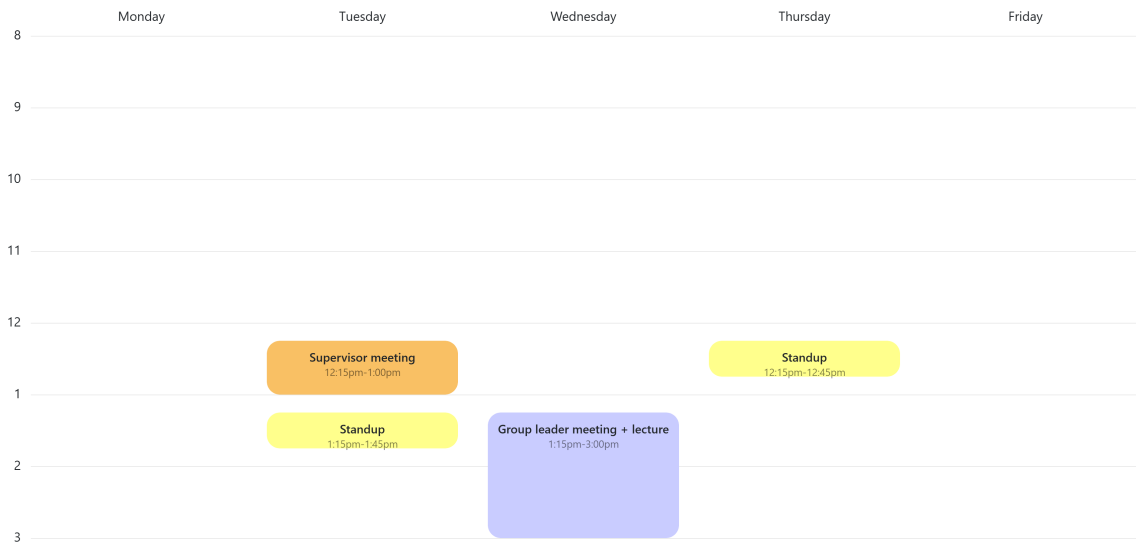

Figure 4: Meeting schedule even weeks

4.3 Roles

To ensure a clear division of responsibilities, the team assigned certain roles to some team members. The course required one team member to be the group leader to give updates to other groups during group leader meetings. We decided to broaden this role's responsibilities by making the group leader the proxy product owner (see section 4.1.1). The team also chose to assign two team members as managers of testing and security. These managers were not solely responsible for all tasks pertaining to these topics, but rather they were responsible for making the overarching plans during the pre-study phase, and making sure they were appropriately prioritised during sprint planning. We felt this was important as we had experienced in previous projects that testing and security tended to be forgotten until the end when no one was specifically responsible for making sure it was done. Moreover, we decided that all team members would take turns as secretaries for meetings. The team chose this over a set secretary because we agreed it was harder to contribute to the meeting while taking notes. Rotating this role allowed every team member the opportunity to be fully present during meetings in equal amounts. Table 1 gives an overview of all roles and their respective responsibilities.

In retrospect, the team expressed interest in combining the Scrum master role with the group leader's responsibilities. We felt this could have streamlined leadership by consolidating the "leader" role for a single person to avoid overlap or misalignment between the two roles. Since the group leader was already responsible for providing progress updates to other teams, combining the roles could have also improved communication and ensured that team activities and updates were always aligned with the overall project objectives.

Role	Responsibilities	Role type	Assignee
Developer	Develop application	Primary role	Everyone
Group leader	Represent group in group leader meetings, proxy product owner	Fixed role	Haakon
Testing manager	Overall responsibility for testing, follow up testing in subteams	Fixed role	Jacob
Security manager	Overall responsibility for security, follow up security in subteams	Fixed role	Olav
Scrum master	Facilitate Scrum activities	Fixed role	Adele
Secretary	Meeting minutes, getting approval on finished minutes from external parties	Rotating role	Everyone

Table 1: Roles and responsibilities

4.4 Tools

To facilitate the project's development, the team decided on using various tools.

4.4.1 Discord

The first tool chosen was Discord for communication. Discord is a digital messaging and voice chat platform. It was suggested by the customer for communication with them as they were already using it for internal communication. Most team members and the supervisor were already familiar with the platform, hence it made sense to also use it for all communication. Another advantage is that Discord allows communication to be split into different "channels". This division made discussions more organized. Lastly, we valued that the platform also features voice and video chat which was useful for remote meetings. Having all this functionality in one platform made Discord a well-suited choice for our project.

4.4.2 Git and GitHub

To simplify and organise concurrent development, Git was a natural choice as the whole team has multiple years of experience with it. The team also chose GitHub as the primary code management and storage platform. The customer stated that they themselves use GitLab, but that they had no preference on what platform we choose. We still decided on GitHub over GitLab, because the team was more familiar with it.

4.4.3 Google Workspace

Google Workspace is the collection of cloud based collaboration and productivity tools including, among others, Google Drive, Google Docs, and Google Calendar.

A common Google Drive was created for the team to store documentation and meeting minutes written using Google Docs. Google Workspace also made it easy to share specific documents with the supervisor and customer. It allowed them to comment and make suggestions which we thought would be helpful when finalising meeting minutes. Furthermore, the team decided to create a shared Google Calendar to organize meetings. This allowed us to easily send meeting invitations to the customer and have a single source of truth for meeting times and places internally.

4.4.4 Linear

For project management, the customer recommended their platform of choice, Linear. Given that many project management applications are not free, we decided to try it out during the pre-study phase to see how we liked it. After using it for a few weeks, the team agreed that Linear was a suitable choice as it is organised and easy to use.

4.4.5 Figma

Figma is a collaborative web application for interface design. The frontend team used Figma to create mock-ups of designs to show the customer and perform usability testing with before implementation. We decided that creating prototypes would be beneficial so new designs could be evaluated before implementation (see section 8.1). Figma specifically was chosen because all team members have some experience with it from other university courses.

4.4.6 Miro

Miro is a digital collaboration platform. The team used it for retrospectives as shown in figure 2. It was chosen because it is free, easy to use, and has templates for retrospective boards.

4.5 Process Risk Management

During the planning phase of our project, we conducted a thorough process risk assessment to identify and address potential challenges that could affect our workflow and outcomes (see appendix E). We systematically evaluated risks related to communication, team commitment, technology selection, health issues, and other factors. For each identified risk, we assessed its likelihood and impact, categorizing them as either medium or high. Based on this, we developed tailored strategies to mitigate these risks. For example, we promoted open communication through frequent stand-up meetings, assigned research tasks to familiarize ourselves with new technologies, and planned ahead to account for potential resource shortages during critical phases. We ensured each strategy was actionable by assigning clear responsibilities and deadlines. This proactive approach allowed us to stay flexible and aligned with our Scrum-based development methodology, ensuring that we could adapt to challenges as they arose.

4.6 Project Plan

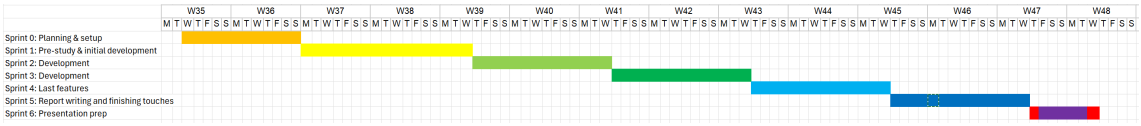

Figure 5: Project plan outline

At the start of the project, our supervisor advised us to create a project plan to map how much time we could afford to spend on each phase of the project. We decided to create the plan with a high level of abstraction by outlining the number of sprints we had time for given the project deadlines and our chosen sprint length of two weeks. Then we worked backwards from the deadlines to figure out how much time should be allocated to the pre-study and development phases. Figure 5 shows the six sprints that would fit in the timeframe in a Gantt chart with the deadlines coloured as red.

The presentation is on November 27th, thus sprint 6 would need to be dedicated to preparing for the presentation and creating a demo of the product. The report deadline is November 21st, which lead us to decide that sprint 5 would be dedicated to report writing and small finishes to the

product. Following this, sprint 4 would need to be devoted to the last features. This project plan was created during what is described as sprint 0. Based on the progress at the time of creation, the team decided that the planning and setup sprint should end by the end of week 36.

This approach enabled a longer first sprint, which we deemed necessary due to the initial development tasks. These included setting up the frontend, backend, database, GitHub pipelines, and conducting the pre-study — an effort that was both time-intensive and critical, as it established the foundation for the rest of the project. Sprints 2 and 3 were then left to be sprints with the main purpose of developing the product. In the end, the project plan consisted of one sprint for planning and setup, one for pre-study and initial development, three for the main product development, one for report writing and finishing touches, and one for presentation preparation.

4.7 Effort registration

To register the efforts of each team member, we used Google Sheets. Each team member had one sheet with columns for hours spent, the week number, and a short description of the task. This allowed us to assure an even workload for each team member. We have attached the graph for our time efforts in appendix A, with some dips in the graph when team members were absent. Overall we are satisfied with each member's efforts during the development and planning of this product.

5 Requirements

The following sections dives into the requirements set for the product and aims to provide insights into the the process of developing and finalising the requirements. The majority of the requirements were established during the pre-study, but expanded and altered throughout the project in accordance with customer communication.

5.1 Business goals and requirements

The company, Lildog, had few business goals and requirements in regards to the product. Their main motivation for this product was to ease the implementation of new devices and software, without using capital to acquire expensive existing solutions or resources by developing it themselves. Another key part of the product is the open source aspect, as Lildog wanted to contribute to the start-up community. This way, several start-ups could potentially benefit by not having to use already limited capital and resources themselves.

In addition to the ones stated by the customer, several other goals and related requirements were identified. In the following table the complete set is listed.

Goal ID	Goal	Req. ID	Requirement
BG1	Ensure reliability	BReq1	Ease the process of implementing new devices and software, and test that they work as intended in tandem with the other parts of the system by implementing the testing application.
BG2	Secure development and use	BReq2	Protect the integrity of the codebase, especially given its open-source nature, by scanning third party dependencies, code-reviews and push-protection.
BG3	Guarantee ease-of-use	BReq3	Ensure easy integration and implementation for both Lildog and others who want to utilise this application, by implementing standards and open frameworks.
BG4	Gain open-source community support	BReq4	Facilitate growth in the start-up community by creating a free-to-use open-source testing framework that would otherwise take up substantial capital or efforts.
BG5	Provide flexibility	BReq5	Allow testing for a wide range of devices by applying general solutions open to modifications.
BG6	Implement universal design	BReq6	Ensure accessibility for audiences with diverse technical expertise and abilities, by implementing usability tactics.

Table 2: Business Goals and Requirements

5.2 Functional requirements

The functional requirements are meant to provide clarity to what the product should deliver in regards to functionality. Each requirement is formulated in a unique and well-defined way, making sure they are understandable to the stakeholders. The requirements are meant to describe what a system should do, but avoid overspecification (NASA 2017, p.70).

5.2.1 Initial functional requirements

The first draft of the functional requirements were derived from the customer's project proposal and a brief meeting at the very beginning of the project. These requirements built upon our understanding of the project and were communicated to our customer to establish a common understanding of the project's requirement and scope.

During a meeting with the customer in sprint 2 we discovered that some of the functional requirements were either obsolete or should have been de-prioritized, resulting in a change of the functional requirements. FR8 was misunderstood in terms of what was defined as "support" various connection types. The team had prior to the aforementioned meeting the understanding that this encompassed the software's ability to handle and pick up interactions between IoT-devices using different communication types. This is further elaborated on in section 8.2.3. However, since the change was in terms of a definition, not the actual functional requirement, it was not changed. FR18 was altered from "Must have", to "Won't have", as the product would not simulate interactions between the IoT-devices, but rather function as a mediator between devices.

Besides the changes already discussed, FR19 related to the implementation of ML should be mentioned. Section 3.1 described how we ultimately decided to build the entire application from scratch, and since this introduced a larger scope, the implementation of ML was de-prioritised to "Won't have".

5.2.2 Finalized functional requirements

During the development process, both our understanding of the technologies and discussions with the customer resulted in expansion or alterations to the functional requirements. In the table below, the final functional requirements are listed. They have been prioritized following the MoSCoW method (Clegg and Barker 1994):

Must have - Essential to the product and required to have a working product.

Should have - Important to achieve the desired product, but are not vital and can be included at a later time if necessary.

Could have - Nice to have features that would improve the product, but have a smaller impact.

Won't have - Fall short of being prioritized either due to out of scope or time constraints.

ID	Description	Priority
FR1	Users should be able to add new categories	Must have
FR2	Users should be able to add new devices	Must have
FR3	Users should be able to create flows	Must have
FR4	Users should be able to create new nodes	Must have
FR5	Users should be able to add created nodes to a flow	Must have
FR6	Users should be able to execute tests for IoT devices	Must have
FR7	The system should trigger interactions between IoT devices when nodes are executed	Must have
FR8*	Software should support various connection types including BLE, UART, Wi-Fi, API and ADB for communication between IoT devices and mobile phones.	Must have
FR9	Users should be able to implement and test interactions involving multiple IoT devices and mobile phones within a single test setup.	Must have
FR10	Software should be able to integrate with other testing tools and systems successfully.	Must have
FR11	The system should provide reports and analytics on test results, including pass/fail metrics and performance indicators.	Must have
FR12	Users should be able to track execution of test scenarios and troubleshoot issues using the app's real-time logs.	Must have
FR13	Users should be able to save, load, edit, and delete test scenarios for reuse and version control.	Should have
FR14	Users should be able to export and import test configurations and results in standard formats (e.g., JSON, XML, CSV).	Should have
FR15	Users should be able to differentiate between different devices	Should have
FR16	Users should be able to see what connection type is utilized by devices	Should have
FR17	Users should be able to see multiple logs at the same time	Should have
FR18*	Software should be able to simulate interactions between IoT devices and mobile phones.	Won't have
FR19	The software should be capable of using ML algorithms to implement image-based testing and validation.	Won't have

Table 3: Functional requirements. ** implies change after resolving misunderstanding*

5.3 Quality attributes

For this project there were three main quality attributes that the team had focus on: modifiability, usability and interoperability. These attributes aligned with the essentials of the product.

5.3.1 Modifiability

As previously mentioned, the open source nature of the application and the intent to allow the start-up community to utilize it requires the application to be modifiable. This is to allow users and developers to add new features that fit their specific needs, and to allow the open source

community to improve and update existing code with the need to refactor or scrap larger parts of the software.

ID	M1
Source	New developer of the application
Stimulus	Developer would like to implement new feature to fit their needs
Environment	In development
Artefact	Code, components
Response	System is modular with low coupling and high cohesion
Response measure	Developer is able to implement new feature without needing to alter other parts of the application

Table 4: Add new feature

ID	M2
Source	New developer of the application
Stimulus	Developer would like to update one part of existing code
Environment	In development
Artefact	Code, components
Response	System is modular with low coupling and high cohesion
Response measure	Developer is able to update the part of code without affecting any other part

Table 5: Update existing code

5.3.2 Usability

The application is supposed to be available to use for everyone wanting to utilize it. Hence, to ensure that people with a wide range of technical skills and capabilities are able to use the application, usability is a key quality attribute.

ID	U1
Source	New user of the application
Stimulus	User wants to add new category and device
Environment	Normal operation
Artefact	UI
Response	The application provides consistent UI design and behaviour
Response measure	The user understands how to add a category and device within 1 minute

Table 6: Add new category and device

ID	U2
Source	New user of the application
Stimulus	User wants to run flow and verify device functionality
Environment	Normal operation
Artefact	UI
Response	The application provides feedback to user on result of device testing
Response measure	The user detects errors in real time during application runtime

Table 7: Run flow

5.3.3 Interoperability

The software must be able to integrate with other testing tools and systems to simplify the testing cycle, thus interoperability is imperative. By facilitating this integration, the application will meet the goal of making testing a wide range of devices easier.

ID	I1
Source	New user
Stimulus	User wants to connect device utilizing a different communication type
Environment	Runtime
Artefact	Backend
Response	The system utilize open standards and protocols
Response measure	The system adheres to protocols and is able to successfully connect to device

Table 8: Connect device with different communication type

6 Security and Risk Management

6.1 Risk Management Framework

The Risk Management Framework (RMF) was applied systematically to identify, evaluate, and mitigate risks associated with our project (Meland 2024). The process began with identifying business assets, followed by defining business goals and assessing risks based on their likelihood and impact. The results of this process laid the foundation for mitigation strategies and specific security requirements. Below, we provide a detailed walkthrough of how we applied the RMF, along with the associated artefacts.

6.1.1 Business Assets

Business assets refer to the valuable resources and components of the system that must be protected. These assets are critical to the operation and success of the project and include both tangible and intangible elements (ibid.). Table 9 outlines the identified business assets for the Lildog IoT testing framework.

ID	Asset	Description
BA1	IoT device logs	Logs that contain information about the status of the IoT devices in the chain.
BA2	Testing framework	The software that is being built.
BA3	Source code repository	The open-source code base of the project.
BA4	User base	Any developers and system integrators that are using the software.
BA5	User documentation	Guidance material to help users operate the software.

Table 9: Identified Business Assets

6.1.2 Business Goals

Business goals define the desired outcomes and priorities for the project, as outlined in section 5.1. These goals ensure that the system's development aligns with the overarching objectives and motivations of the company, including reliability, security, ease of use, and community support (Meland 2024). We use the business goals to effectively tailor the risk management efforts to address the critical aspects of the project.

6.1.3 Risk Likelihood and Impact Dimensions

To evaluate risks, we used a matrix combining likelihood and impact dimensions. Likelihood represents how often a risk might occur, and impact reflects the severity of its consequences (ibid.). Tables 10 and 11 define these dimensions.

Likelihood	Description
Low	Once a decade
Medium	Once a year
High	Once a month
Extreme	Once a week

Table 10: Likelihood Dimensions

Dimension	Low	Medium	High	Extreme
Modifiability	Minor modifications require little effort.	Some parts of the system can be modified easily, but core components require considerable effort for updates.	Significant effort is needed to modify or update the system, leading to costly and time-consuming changes.	The system is rigid and cannot be modified without a complete overhaul, making it quickly outdated and costly to maintain.
Interoperability	The system integrates with most tools with minor manual adjustments or configurations.	Integration with some tools requires significant configuration or custom development effort.	Many tools cannot be integrated without major changes or workarounds, disrupting workflows.	The system is unable to integrate with other tools, severely limiting its usefulness in streamlining the testing cycle.
Usability	Minimal changes required for users to complete tasks with little to no confusion.	Some users experience difficulty or require additional steps to complete tasks.	A significant number of users struggle to complete tasks or avoid using the system due to frustration.	Most users are unable to use the system effectively, leading to complete abandonment or heavy reliance on support.

Table 11: Impact Dimensions

The impact dimensions (modifiability, usability, and interoperability) are directly aligned with the quality attributes (see Section 5.3) defined for this project. This alignment emphasizes the strong connection between the core qualities of the system and its associated risks. By using the same set of attributes, we ensure that the risk evaluation process remains focused on the most critical aspects of the product's functionality and success. This approach also highlights the dual role of these attributes: as benchmarks for product quality and as indicators of potential risk impact.

6.1.4 Business Risks

Business risks represent potential threats to achieving the identified business goals (Meland 2024). Table 12 outlines the business risks, their likelihood, impact, and mitigation strategies.

ID	Related Goal(s)	Description	Likelihood	Impact	Risk Level	Mitigation Strategy
BR1	BG1	Failure to detect faulty devices due to bugs in the system.	Medium - Bugs are common in software projects.	High - Failure to detect faulty devices compromises the system's purpose.	High	Automated testing and log monitoring.
BR2	BG1	Logs are inaccessible due to system downtime or unavailability.	Low - System runs locally and is controlled by internal personnel.	High - Unavailability during device failures impacts reliability.	Medium	Backup logs regularly and redundancy in logging mechanisms.
BR3	BG2, BG4	Malicious code contributions or security vulnerabilities in open-source libraries.	Medium - Open-source contributions can introduce vulnerabilities.	Extreme - A single vulnerability can compromise the entire system.	High	Implement strict code review processes and automated security scans on every contribution.
BR4	BG2	Tampering or spoofing of IoT device logs, resulting in incorrect conclusions about device failures.	Low - Software runs on their own servers and not exposed externally.	Medium - Tampered logs can lead to false diagnoses.	Low	Ensure cryptographic signing of logs for integrity verification.
BR5	BG3	Complexity in configuring the framework leads to user frustration.	Medium - Some users may struggle if configurations are not user-friendly.	High - If the framework is difficult to configure, users may abandon or misconfigure it.	High	Provide clear documentation and offer default, out-of-the-box configurations to reduce setup complexity.
BR6	BG3	Users struggle to interpret the logs and identify the faulty device.	Medium - Logs may be hard to interpret or be visually hard to read.	Medium - Confusion may delay fault detection and resolution.	Medium	Include log analysis tools or visual aids that make the information more accessible to all users.
BR7	BG5	Inflexibility in the testing framework leads to difficulty integrating new IoT devices.	Medium - Different IoT devices may have varying communication protocols and requirements.	High - Failure to integrate with diverse devices reduces the framework's utility.	High	Design the framework with modularity in mind, allowing for easy addition of new device interfaces and protocols.
BR8	BG6	The software is not accessible to users with varying levels of technical expertise or disabilities.	Medium - Software systems often overlook universal accessibility.	High - A lack of universal design could reduce adoption or usability among different user groups.	High	Ensure adherence to accessibility standards (e.g., WCAG) and offer simple, intuitive interfaces.
BR9	BG6	Language and localisation barriers prevent some users from using the framework effectively.	Medium - If localisation is not considered during development.	Medium - Failure to support multiple languages could limit the tool's usability for non-English speakers.	Medium	Build support for multiple languages and localisation into the framework from the outset.

Table 12: Identified Business Risks

6.1.5 Technical Risks

Technical risks are derived from business risks and focus on specific vulnerabilities in the system (ibid.). These risks are categorized using the STRIDE model, which identifies six threat categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (OWASP Foundation 2024). These categories are detailed in Table 13.

ID	Related Business Risks	Description	STRIDE Category
TR1	BR1	Undetected software bugs in the code logic could lead to incorrect detection of faulty IoT devices.	Denial of Service
TR2	BR2	Database/file system failures, single point of failure, inefficient storage.	Denial of Service
TR3	BR3	Unpatched dependencies, insecure contributions, weak validation of code.	Tampering
TR4	BR4	Weak authentication, lack of integrity checks, improper access control.	Tampering
TR5	BR5	Overly complex configurations, lack of validation, poor documentation.	Information Disclosure
TR6	BR6	Unstructured logs, insufficient log verbosity, lack of log visualisation tools.	Information Disclosure
TR7	BR7	Hard-coded assumptions, lack of modularity, incompatibility with new protocols.	Denial of Service
TR8	BR8	Lack of accessibility features, complex UI, no internationalisation.	Information Disclosure
TR9	BR9	Hardcoded strings, lack of multilingual support, encoding issues.	Information Disclosure

Table 13: Identified Technical Risks

6.1.6 Security Requirements

Based on technical risks, we defined security requirements. These requirements focus on practical and testable measures to address risks (Meland 2024). Table 14 summarizes these requirements.

ID	Technical Risk	Requirement	Test	Expected Result
SR1	TR1	The system must implement comprehensive testing mechanisms (unit, integration, and system-level tests) to detect bugs in IoT device detection logic.	Run unit, integration, and system tests to validate device detection logic under various scenarios.	All tests pass, ensuring device faults are correctly identified without crashes or false positives.
SR2	TR2	The system must implement redundant log storage, with backups and automatic failover to ensure log availability in case of a storage failure.	Simulate a log storage failure and verify that backups and failover mechanisms take over seamlessly.	Logs remain accessible during and after the failure, with no data loss or downtime.
SR3	TR3	The system must scan all third-party dependencies for vulnerabilities and ensure security patches are applied regularly.	Run automated vulnerability scans using tools like Dependabot on third-party dependencies.	No critical vulnerabilities are detected, or patches are successfully applied where vulnerabilities exist.
SR4	TR4	The system must enforce strong authentication and role-based access control (RBAC) for log access and integrity checks (e.g., digital signatures).	Attempt unauthorised access and tampering with logs, and verify that authentication and signatures block access.	Unauthorised users are denied access, and any tampered logs fail integrity checks.
SR5	TR7	The system must enforce version control, require code reviews, and run automated security and functional tests on all code contributions.	Submit a code contribution and run tests through the version control pipeline.	Code fails to merge without passing security and functional tests, and reviews are completed before merging.

Table 14: Security Requirements

6.1.7 Applications of RMF

Failure to Detect Faulty Devices (SR1)

One of the primary risks is the potential failure to detect faulty IoT devices due to bugs in the system (BR1). This risk is associated with the goal of **ensuring reliability** (BG1). If undetected, these bugs could compromise the framework's core purpose. To mitigate this risk, we implemented tests for the business logic of the application.

Security Vulnerabilities from Open-Source Contributions (SR3)

Given that our framework is open-source, there is a risk of malicious code contributions or vulnerabilities in third-party libraries (BR3). This risk impacts the goal of **securing development and use** (BG2) as a single vulnerability could compromise the entire system. To mitigate this, we enforced a strict code review processes on all contributions to the project. We also set up Dependabot with GitHub Actions to automatically keep the project's dependencies up-to-date and secure (see appendix C). Dependabot is a a feature of GitHub that automating the dependency update process by identifying outdated or vulnerable dependencies and proposing updates to address them.

Redundant Log Storage with Backups and Automatic Failover (SR2)

Due to time constraints, we were unable to implement SR2, which involves setting up redundant log storage with backups and automatic failover mechanisms. This requirement is closely tied to the business goal of **ensuring reliability** (BG1), as it ensures logs are accessible even in cases of system downtime, addressing **Business Risk 2 (BR2)**. Though the risk of logs being unavailable is low due to local, internally controlled operation, its impact is high if it coincides with device failures, as it would compromise the framework's reliability. In discussion with the customer, we agreed to leave the backup responsibilities to the users themselves, but enhancing this aspect remains a potential area for future work to improve system resilience and log accessibility.

6.1.8 Conclusion

The RMF provided a systematic approach to understanding risks, prioritizing mitigation strategies, and defining security requirements. While the scope of the project limited the implementation of all requirements, key areas such as SR1 and SR3 were prioritized to ensure the system's security and reliability.

6.2 Misuse case diagram

The misuse case diagram (figure 6) highlights the critical risks and threats associated with the system, as identified through the RMF. It visually connects actors (e.g., user), misactors (e.g., external attacker), misuse cases (e.g., tampering with IoT logs), and mitigations (strict code review) to give a more visual representation of the potential threats our application faces.

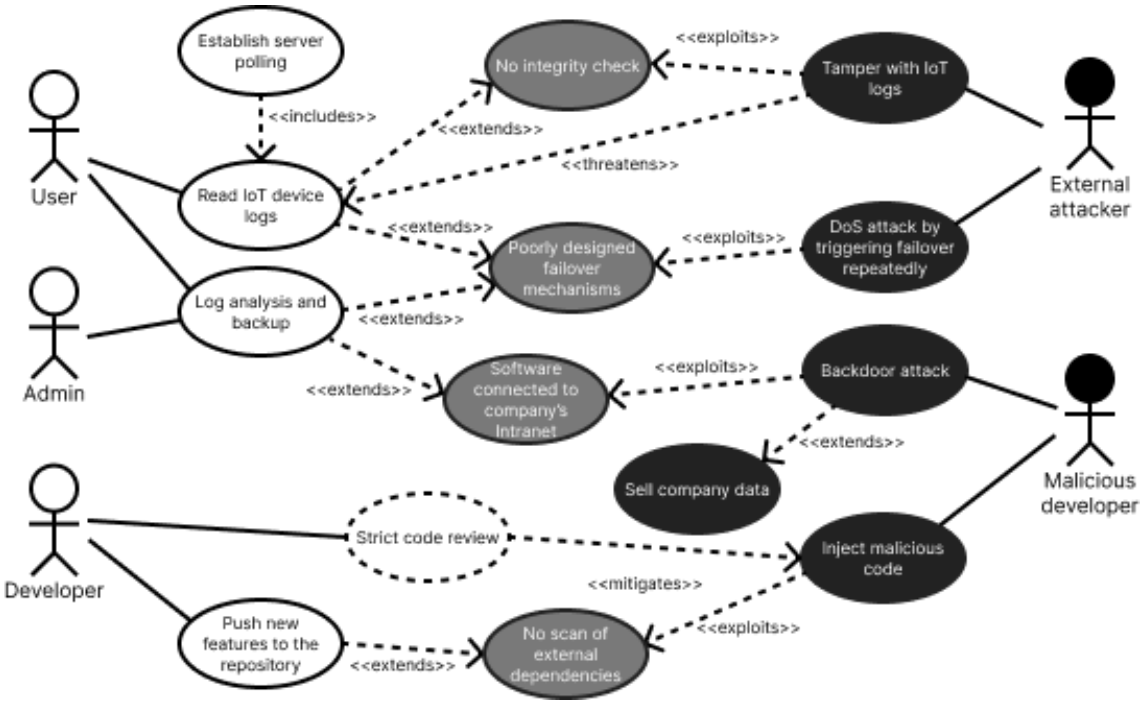

Figure 6: A misuse case diagram highlighting potential threats

6.3 Risk management - Tech stack

In this section, we identify and assess potential risks associated with each component of our tech stack. Mitigation strategies for each identified risk are also provided. For detailed synthesized risk of tech stack, see appendix D.

6.3.1 Vue.js

- **External Dependencies**: Vue.js projects often rely on npm packages or external libraries, which may contain vulnerabilities. Outdated dependencies can increase exposure to known vulnerability attacks.
- **Mitigation**: Regularly update libraries and use tools like `npm audit` to detect vulnerabilities early.

6.3.2 TypeScript

TypeScript introduces minimal risk due to its type safety. However, using outdated libraries alongside TypeScript could still expose the project to potential vulnerabilities. This is mitigated by dependabot performing weekly checks to keep libraries up to date.

6.3.3 Tailwind CSS

- **Complexity**: Overuse of custom classes may make the project harder to maintain.
- **Mitigation**: Enforce design guidelines and establish style consistency to reduce complexity. Use components where applicable to ensure a consistent UI across the whole application.

6.3.4 Django Python

- **OWASP Top 10 Vulnerabilities**: Django may be susceptible to common web application attacks.
- **Mitigation**: Focus on secure development practices, utilize Django's built-in security features, and monitor updates for security patches.

The customer will run the application locally on their Intranet, and will therefore not be vulnerable to these attacks as long as their Intranet is secure. That said, we chose to focus on preventing these attacks by applying Django's security middleware, as this was low effort and improved the application security considerably, should the application be exposed to the Internet.

6.3.5 PostgreSQL

- **SQL Injection**: Malicious SQL injection attacks can compromise data.
- **Backup and Recovery Failures**: Failures during backup or recovery processes can result in data loss.
- **Mitigation**: Use the Django ORM to provide a strong foundation against SQL injection attacks. Django's ORM automatically escapes queries, which helps prevent SQL injection.

In discussion with the customer, and as previously mentioned in section 6.1.7, we concluded that handling backup of data will be left for the customer.

6.3.6 Conclusion

After assessing the risks, we conclude that most of the identified risks can be effectively mitigated through regular updates and secure development practices. While backup strategies could further enhance security and resilience, these were not implemented within the scope of this project and remain a consideration for future work.

7 Testing

7.1 Test Plan

During the pre-study phase we created an initial test plan, as shown in figure 15, involving several types of tests. The tests represent quadrant one and three in the agile testing quadrants and will help support the team from a technological angle, as well as techniques that critique the product from a business angle (Crisping and Gregory 2009). As work progressed, we were unable to fully adhere to the initial test plan. Given that the goal was to create an MVP, further testing such as performance testing, was not planned.

Test Method	Description	Implementation	Framework
Unit test	Testing individual components like methods, classes, and objects to ensure they function correctly	Developed after important features using the Pytest framework and run automatically in the pipeline.	Pytest
End-to-end test	Simulating real-world user scenarios to ensure that all parts of the application work together seamlessly. Meant to catch system wide issues	Mimic a user interaction made in Cypress and are run in the pipeline automatically. Should be implemented when a user story is finished	Cypress
Usability tests	Testing with actual end-users to gather feedback on usability, functionality, and overall experience	Tests on diverse users including the customer. Must be done after the prototype is finished, and at the beginning of sprint 4. Each test needs detailed documentation	
Acceptance test	Verifying that the product meets the customer's business requirements and acceptance criteria	Test around the time we do retrospectives with thorough documentation and detailed specifications. Should be done with the customer	

Table 15: Initial test plan

8 Development Process

Due to the different nature of both the teams' development processes, the following two sections separately cover both the frontend and backend teams' development journeys for this project.

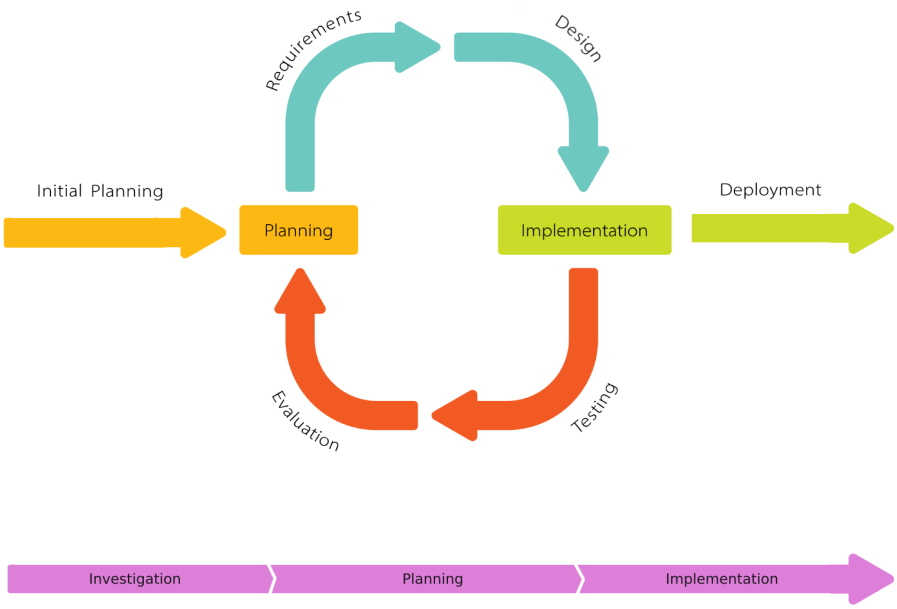

Figure 7: Frontend development process (top) vs. backend development process (bottom)

8.1 Frontend Development

Working with the frontend we adopted an iterative process as illustrated in figure 7. This meant that we created, tested, and revised the frontend several times until we were happy with the product. Using usability tests and customer meetings we were able to receive feedback that helped us design a better version of the product. This circular process was instrumental in shaping the frontend into the final product.

8.1.1 Sprint 0: August 28th – September 6th

We started the planning and setup phase by making a Figma prototype. This resulted in our first prototype shown in figure 8. The flow chart board was inspired by a programming game made for children called Scratch. This included stacking the "devices" on top of each other, having the "signal" run vertically from top to bottom. Using a drag and drop function, the user would easily be able to intuitively build flows for testing different devices.

During our initial formal customer meeting on the 6th, we presented the prototype and received positive feedback on colour scheme and UI. While the customer was satisfied with the overall design, they identified some potential issues. Specifically, they noted that the functional requirement FR9 may not be fully met, as the vertical stacking did not allow parallel blocks stemming from the source. Additionally, fulfilling functional requirement FR12 may be challenging; the customer highlighted the need for logs from all devices and raised concerns regarding failure scenarios. They questioned whether a failure in the top device would prevent logs from being generated for other devices.

The customer also expressed a preference for more responsive visual feedback from the test blocks in application. This would enable them to quickly identify which device failed when multiple devices are chained and running, rather than having to manually search through extensive logs to locate errors. Furthermore, the customer requested an option to filter the logs, as the current setup displays all logs in a single, continuous list (see Figure 9).

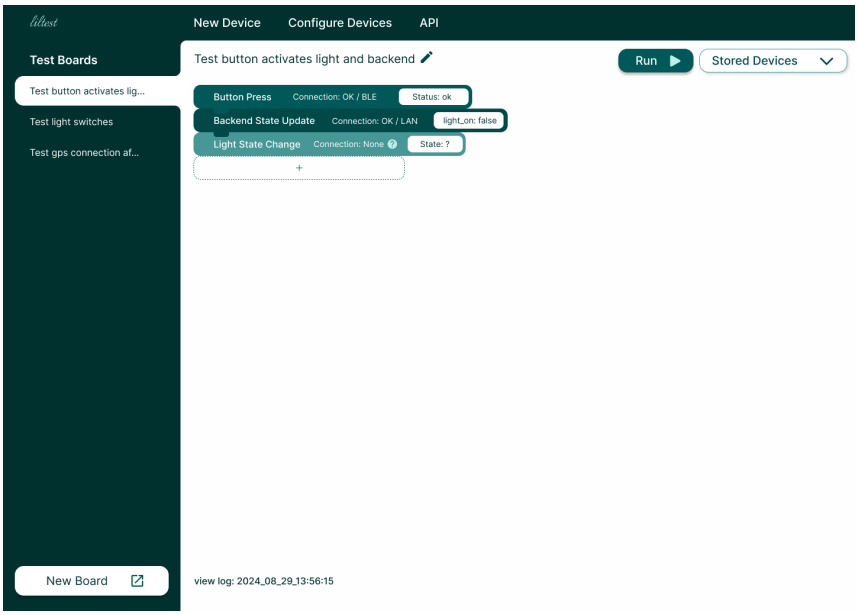

Figure 8: First prototype

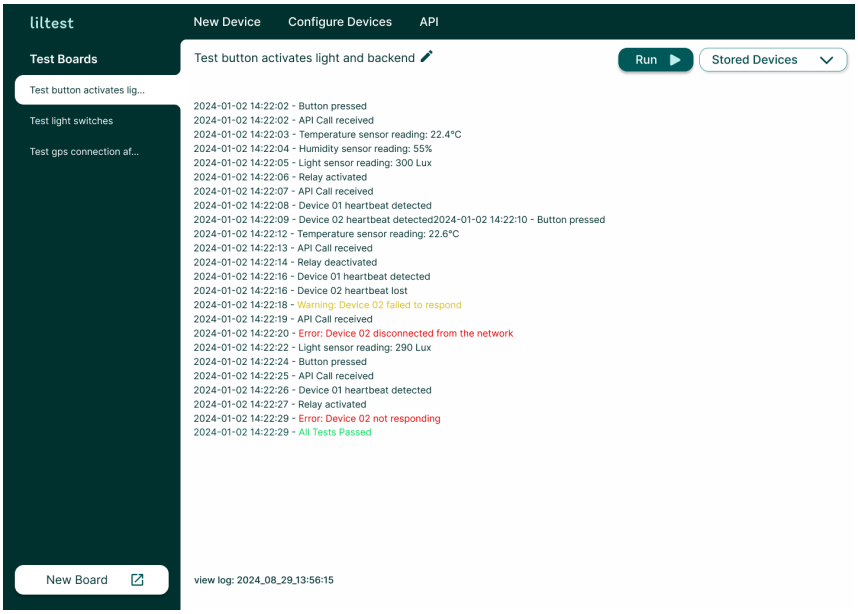

Figure 9: First implementation of logs

8.1.2 Sprint 1: September 9th – September 24th

Taking into account the feedback from Sprint 0, we revised the Figma prototypes to address the issues the customer highlighted in the previous demo, such as incorporating filters for devices (see Figure 10) and adding nodes with connecting edges (see figure 11). These substantial modifications would enable the user to navigate the logs more easily and quickly identify points of failure in the flow view. This solution also utilizes a drag-and-drop system; however, instead of stacking the ”devices”, we implemented a design where lines are drawn between one ”node” and another ”node”. This adjustment allows each node to connect to several other nodes.

After the sprint planning on the 9th we also decided to do conduct a usability test, because we were uncertain whether the user interface was intuitive enough (see appendix B for an example). The usability tests were conducted with three students, all of whom had technical knowledge.

The test participants found it unclear what was happening on the page. Specifically, it was not immediately obvious that selecting a flow, which we called "board" at the time, would directly open it or how the application was intended to function overall. Additionally, there was confusion about the purpose of the "API" button and the meaning of a "board".

A significant oversight on our part was not addressing this feedback promptly. Nevertheless, the feedback provided invaluable insights, clearly highlighting the issues with the interface. Unfortunately, we only realised the value of the results in the retrospective. They were added as action points and implemented in sprint 2.

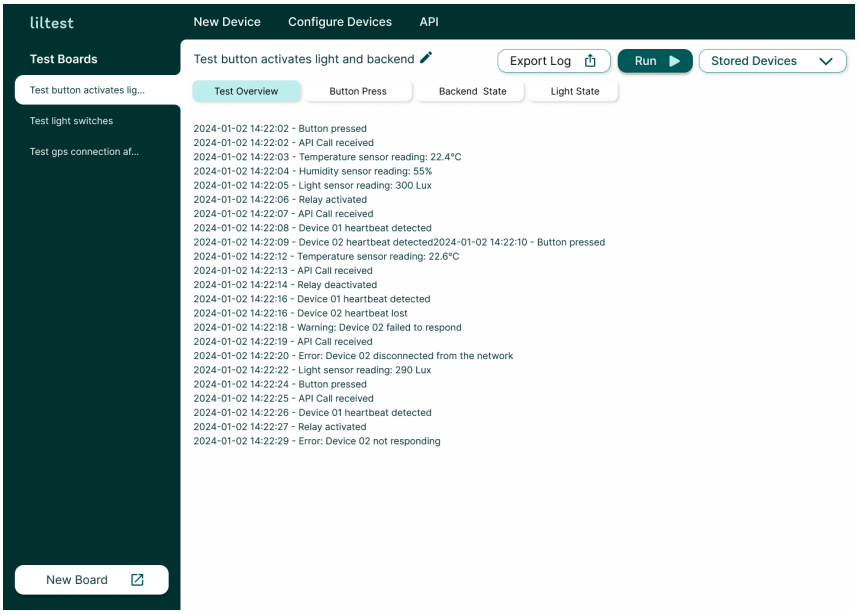

Figure 10: Improvements on the envisioned logs

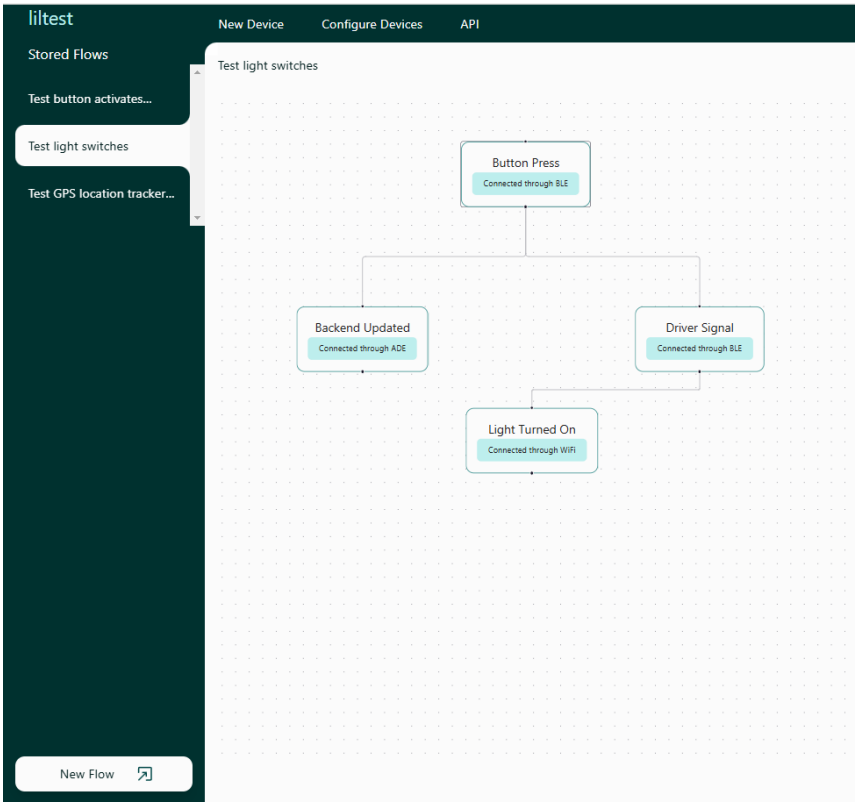

Figure 11: Second prototype

8.1.3 Sprint 2: September 25th – October 8th

Entering the second sprint we had a lot of tasks at hand. During the customer meeting we had on the 24th we got some valuable feedback and suggestions for improvement. The customer requested the use of distinct colours to represent different connections on the devices to simplify debugging within the flow. For example, if an error occurred during a flow execution, the affected connection should turn red, accompanied by a symbol indicating the issue. Additionally, the customer emphasized the need to think on a larger scale. The system needed to support scenarios where users might add up to 100 similar devices, as well as allow for multiple log windows to be open simultaneously.

The feedback from both the customer and our usability tests in the previous sprint, presented a significant workload. We pushed the multiple log windows to the next sprint, and to manage the rest of the tasks effectively, we prioritized the most critical tasks. First, we implemented a system to assign devices to categories, facilitating the handling of multiple similar devices. Furthermore, we finished a darkmode option, witch was a feature that we started on in sprint 1 (see figure 19 for an example of a flow in darkmode).

We redesigned the flow nodes (see Figure 12) to incorporate colours based on the type of connection. For example, Bluetooth devices were assigned various shades of blue, while Wi-Fi devices were assigned shades of purple. When a flow was executed, nodes provided clear feedback by displaying symbols within the node and a coloured ring around it to indicate success or failure. These symbols were especially helpful for users with low vision who might struggle to distinguish colours.

In response to usability test feedback, we streamlined the UI. The top navigation bar was removed and replaced with two buttons: ”Configure Devices” and ”Test Flows” (see figure 13). This reorganization simplified the design, better categorized devices, and improved flow management. We also created an overview page for flows, which included a dropdown menu to quickly access the devices in each flow (see Figure 14). These updates collectively addressed critical feedback and laid the groundwork for a more scalable and user-friendly system.

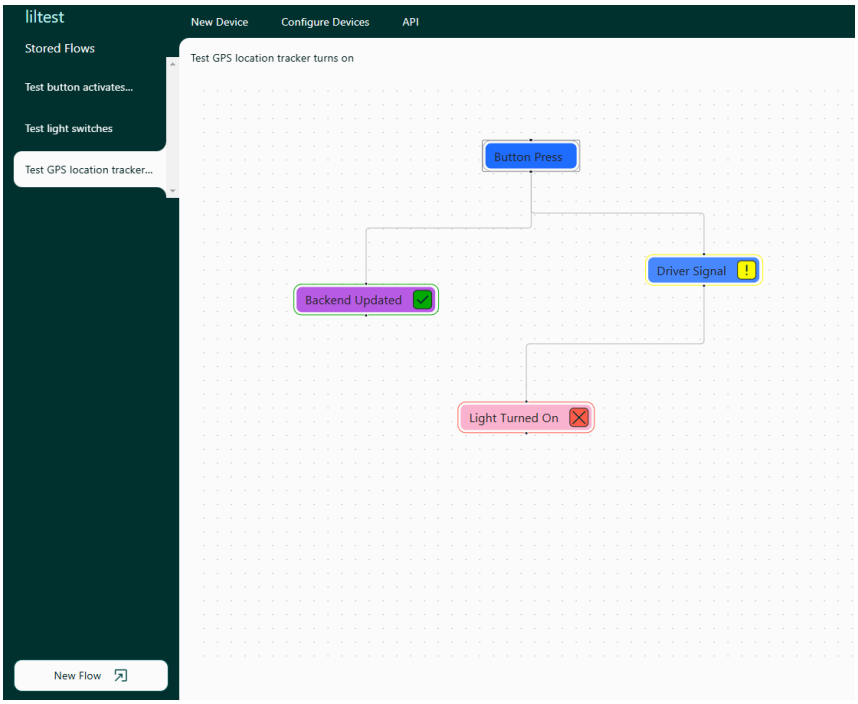

Figure 12: Responsive flow nodes

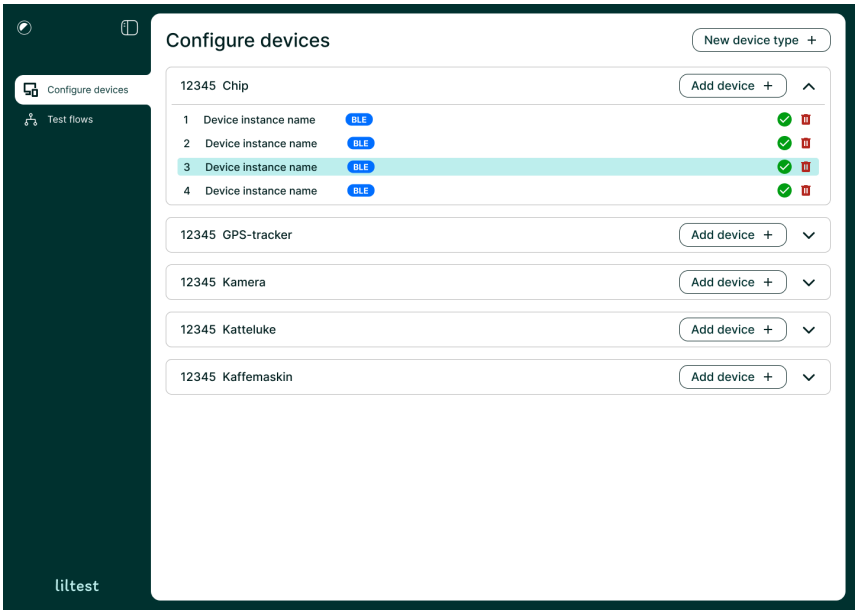

Figure 13: Devices stored in categories with new UI

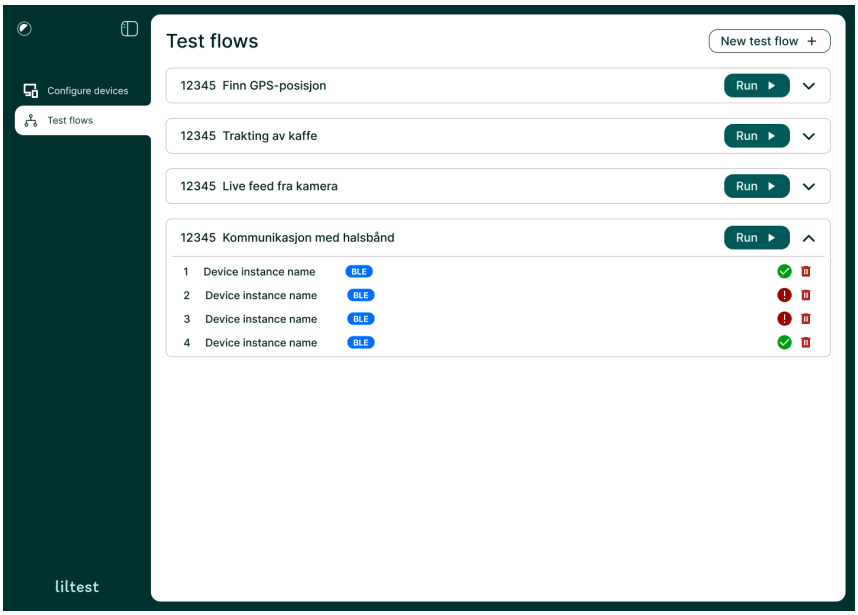

Figure 14: Flow overview

8.1.4 Sprint 3: October 9th – October 22nd

Learning from our previous mistakes, we decided to analyze the usability test immediately and add action points in the sprint planning. The results revealed that there was too much information on one page, and the test subjects did not like the accordion. We decided to do a complete overhaul of the "test flows" page and the "configure devices" page. We implemented a grid with "cards" instead. We felt this would give the user a simpler and clearer overview of all the devices categories and flows as shown in figures 15 and 16). The new design was more intuitive for device creation. If the user wants to edit or add a specific device within a category, they have to enter a new page, thereby dividing device categories and specific device instances into separate pages (see figure 17).

We dedicated time in this sprint to completing the customers request to have several tabs of logs open at the same time. This allows the user to be able to see the logs from different devices at the same time, allowing for simpler debugging and a much better overview when running flows (see figure 18.

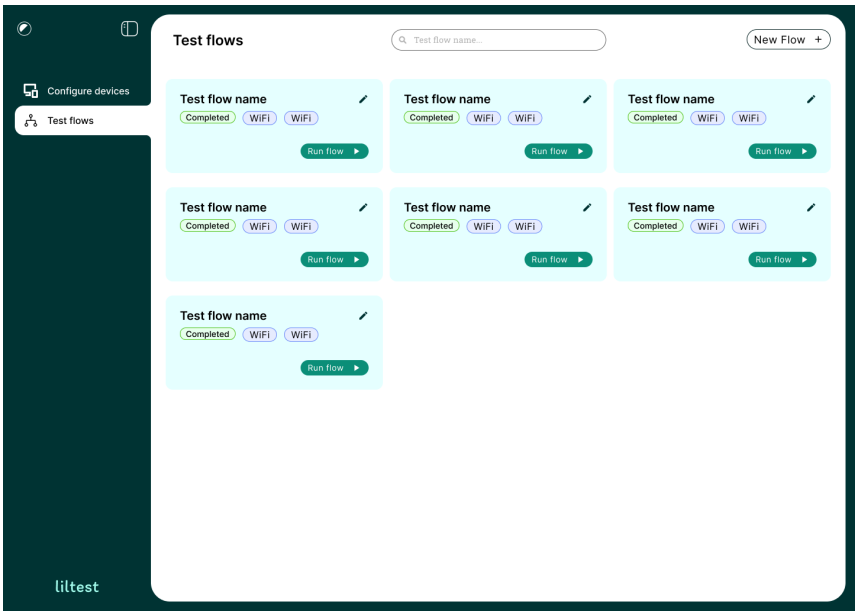

Figure 15: Flows organized in cards from Figma

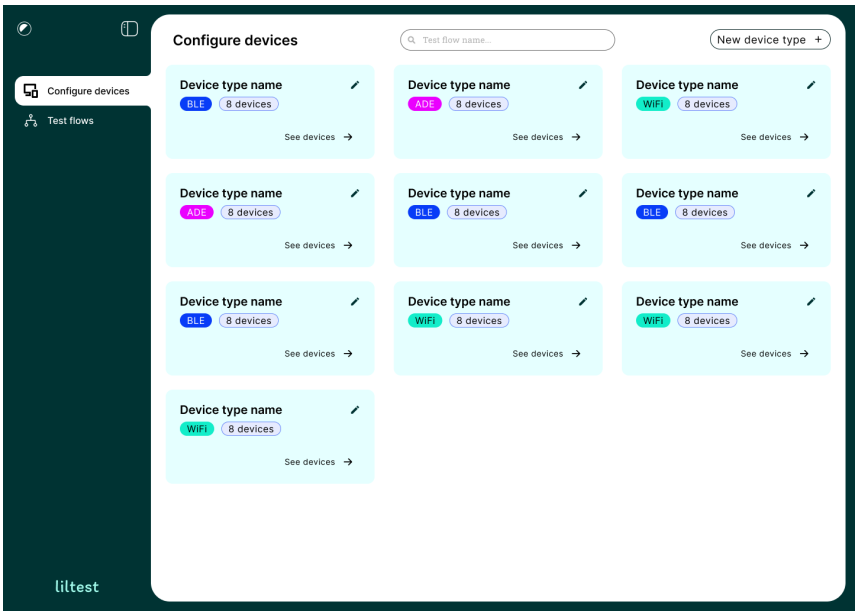

Figure 16: Configure device landing page from Figma

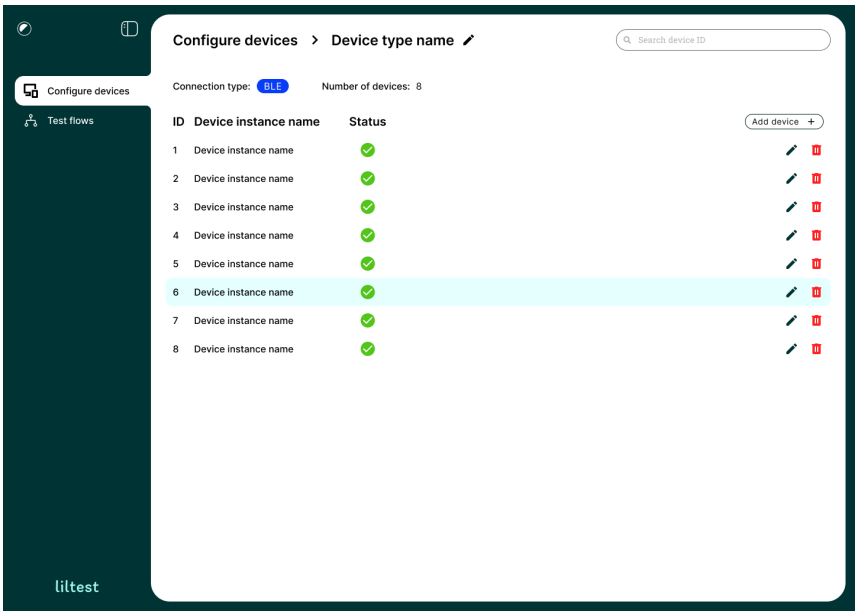

Figure 17: Devices in a device category from Figma

Figure 18: Log opened with three extra tabs

8.1.5 Sprint 4: October 23rd – November 5th

During the customer meeting held on the 22nd, we received positive feedback, which enabled us to concentrate fully on integrating the frontend with the backend. The process of connecting the entire system presented several challenges, requiring adjustments to multiple aspects of the frontend where the communication in the group had not been good enough. This is further elaborated in section 11.2. One such aspect involved finalizing the functionality of adding nodes to a flow. Previously, we had only one menu for all saved devices, but while connecting to the backend we realized this did not work. Ultimately, we implemented a menu allowing users to select between assertion nodes and action nodes (see figure 19). Additionally, we improved the input fields for functions within the nodes to better align with the backend's requirements (see figure 20).

Facing unforeseen challenges when connecting to the backend, we did not manage to finish integrating all of the backend functionality onto the frontend in sprint 4. This meant we had to spend time well into sprint 5 on these tasks, leading to de-prioritising of requirements such as exporting the logs (FR9).

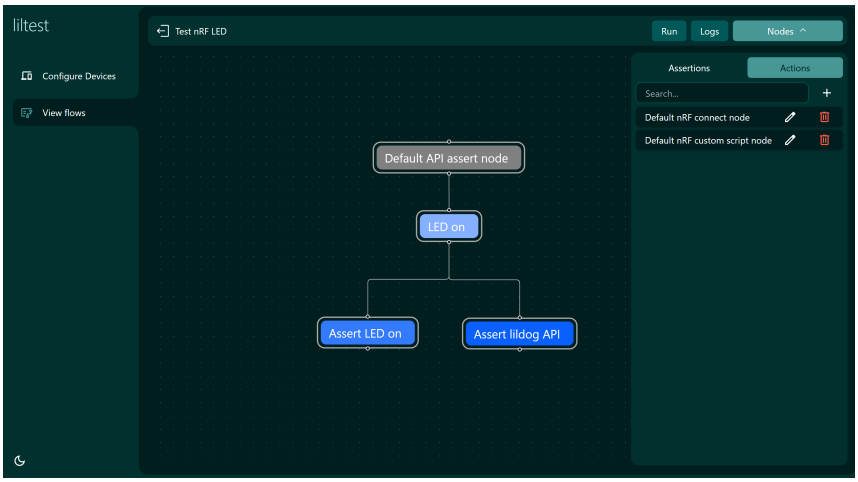

Figure 19: A flow in dark-mode with the menu open, ready to add other nodes to the flow

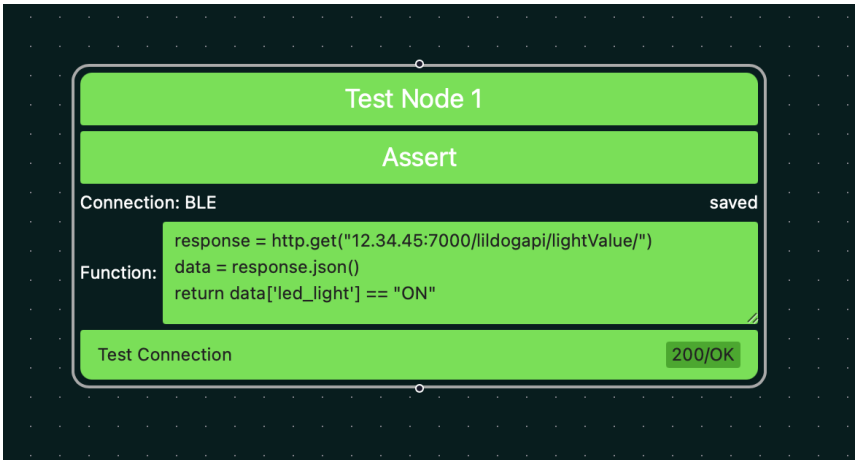

Figure 20: Node adapted to the backend architecture

8.1.6 Sprint 5: November 6th – November 19th

During sprint 5, we finished integrating the backend and connecting all API endpoints. We also focused on improving the accessibility on the product. This included better contrast ratios throughout the interface, in addition to updating our current ui to follow WCAG standards with appropriate ARIA labelling to improve the interface navigation when using keyboard navigation or a screen reader; these accessibility efforts are further detailed in section 10.2.2.

8.2 Backend Development

The backend team used a more linear development process than frontend as illustrated in figure 7. An iterative process of implementing and revising based on feedback was still used. However, because of the backend's technical complexity we ended up with a more linear approach as we had to spend more time understanding the technology before implementing functionality.

8.2.1 Sprint 0: August 28th - September 6th

Sprint 0's primary goal was planning and setup. During this sprint, the backend team focused on choosing an appropriate tech stack, creating a skeleton for the backend in the project repository,

and setting up pipelines in GitHub. These tasks were instrumental as the team had decided initial development would start next sprint. As stated in section 3.1, the customer recommended we look into using Node-RED in the project. The choice to start the backend pre-study early was intentional. It was important to start this as early as possible, because it created the foundation by which subsequent choices would be affected.

Before the first customer meeting on September 6th, the backend team asked the customer for a specific use case to work from. The intention was that it would be easier for the backend team to work with the unfamiliar technologies when getting a more concrete view of what is required by the backend flow. Some of the technical requirements by the customer were quite abstract, like FR8, so getting a specific use case helped demystify these concepts for us. Lastly, the team shared what they had learned about Node-RED so far and asked the customer how extensively it could be incorporated into the project if chosen. Clarifying this was important because if Node-RED proved to be a decent starting point, the team could use it as the foundation for the backend which would significantly influence the backend's architecture and further development.

8.2.2 Sprint 1: September 9th - September 24th

During sprint 1, the backend team focused on making a decision on whether to use Node-RED or not. The team spent time trying out the product themselves and investigating pros and cons as described in section 3.1. Based on this investigation, we decided on not using Node-RED and making the backend from scratch. Making this decision allowed us to properly start developing the backend. During this sprint we also setup the WebSocket communication between the backend and the frontend used to send real-time logs from the devices.

Another important part of the backend team's work this sprint was to investigate the IoT device communication logging further. We were unsure of exactly how to implement this as there are a lot of possibilities and our knowledge was limited. The customer had previously said that we could contact another employee, here on out "technical contact", for any technical questions. A team member reached out to the technical contact to get some guidance on how to acquire the logs from the IoT devices. The technical contact suggested multiple solutions like the "man in the middle" approach, UART, and sniffing as described in section 3.5.

8.2.3 Sprint 2: September 25th - October 8th

Now that we had decided not to use Node-RED, we could start working on the backend. This started with a setup of the database, creating the models for the database, as well as creating an API for managing the data in the database from the frontend. The backend team spent a decent amount of time designing the database models to make sure they would include all necessary information required to create a test flow.

During the sprint demo, a misunderstanding came up. The customer expressed that we did not have to tap into the communication between the devices as we previously thought. It was further uncovered that the customer did not relay the decided scope to the technical contact. This resulted in the technical contact giving us information that led us out of the customer's desired scope. Furthermore, as stated in section 5.2, we did not specify FR8 enough which aided the misunderstanding as we were not aware that the customer did not want inter-device tapping. In the following retrospective, the backend team suggested an action point for an additional meeting with the customer and the technical contact to get a clear, common view on how they wanted the devices to communicate.

8.2.4 Sprint 3: October 9th - October 22nd

As previously stated, the retrospective of sprint 2 resulted in a request for an additional meeting to get a better understanding of the desired device communication from the customer's side. At the start of sprint 3, before sprint planning, the backend team had a meeting with the customer

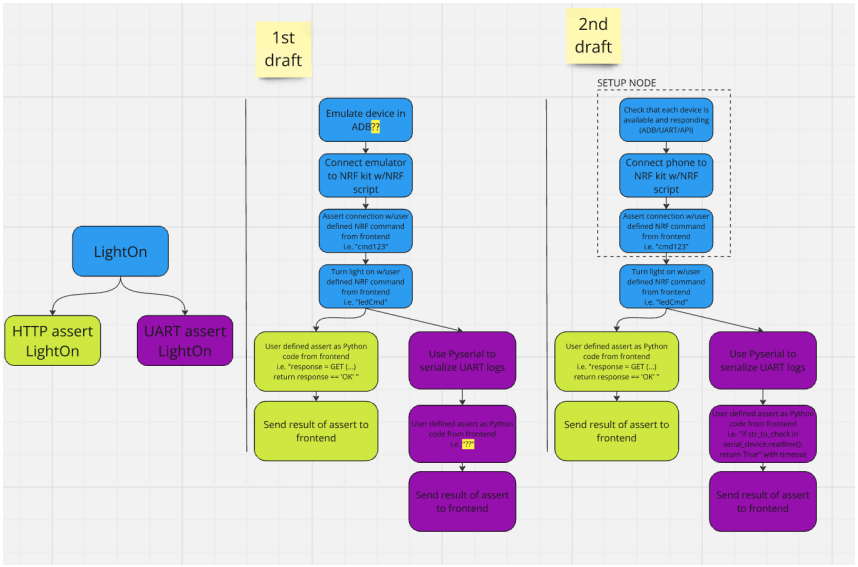

Figure 21: UI vs. corresponding backend flow (two drafts)

and technical contact to create a common understanding. Based on the discussion, we created a flow outlining our understanding of the backend device interactions and sent it to the customer for feedback, on which a second draft was based. Figure 21 shows these flows with a simplified representation frontend UI on the left-hand side, and the two drafts for the backend's corresponding flow on the right.

Based on this meeting, we updated the database models to work with the new required behaviour for the device. Clearing up this issue also allowed us to start the development on the different nodes for devices. While working with the devices, specifically the nRF kits, we encountered a lot of struggles. Using these kits was not as plug-and-play as other tools and frameworks of the project, especially given the limited experience with IoT within the team. Despite one of the members already having a kit and some experience using these kits beforehand, the team still had to spend significant time getting to know the kits and setting up the environment. However, Nordic Semiconductor provided several sample projects on their GitHub that could be used as test scripts once the environment was up, which expedited development.

8.2.5 Sprint 4: October 23rd - November 5th

Per the project plan, sprint 4 was used for the last features of the application. During this sprint, all backend node functionality was completed, as well as feeding the logs from the devices to websockets so they could be displayed on the frontend.

8.2.6 Sprint 5: November 6th - November 19th

The last sprint was dedicated to finishing touches on the development side. To improve usability, we added default nodes to the database so that the app would be initialized with common nodes that the customer stated he would need often. Furthermore, we created documentation for the APIs using Swagger which was also requested by the customer. Additionally, to simplify running of the app for users, a docker containerisation of the application was implemented by the backend team.

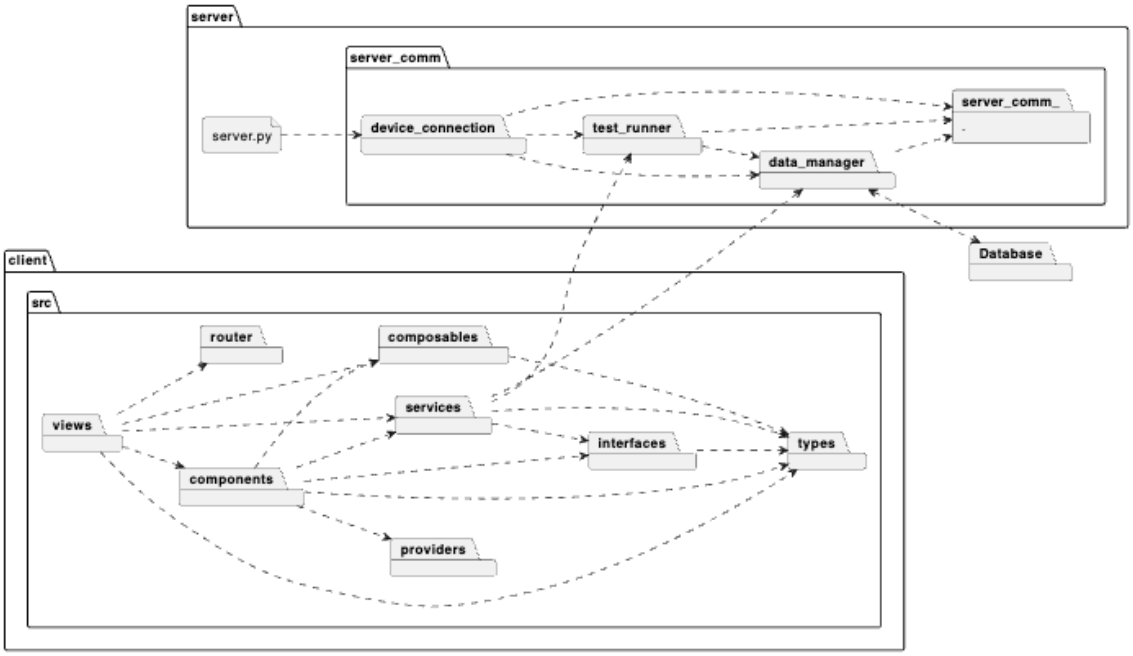

Figure 22: Enter Caption

9 Final Product

9.1 Architecture

The following sections will describe the architecture of the finalized product using the 4+1 views model (Krutchen 1995). The aim is to provide different perspectives on the architecture and give understanding of the product itself. Logical, process, development and physical views are depicted below.

9.1.1 Development View

The development view is illustrated with a package diagram in figure 22. It aims to show the structure of the application, and the relationships between them. The scope of this diagram is limited to the actual development the team has done, not including the external devices and systems, and depicts only the internal relationships.

9.1.2 Logical View

The scope of the logical view is contained to the Django part of the application. This is due to the fact that Django is the most central part in terms of functionality. The right-most part of the diagram in figure 23 includes the vital parts of the actual process of testing, such as flow, category, device and nodes.

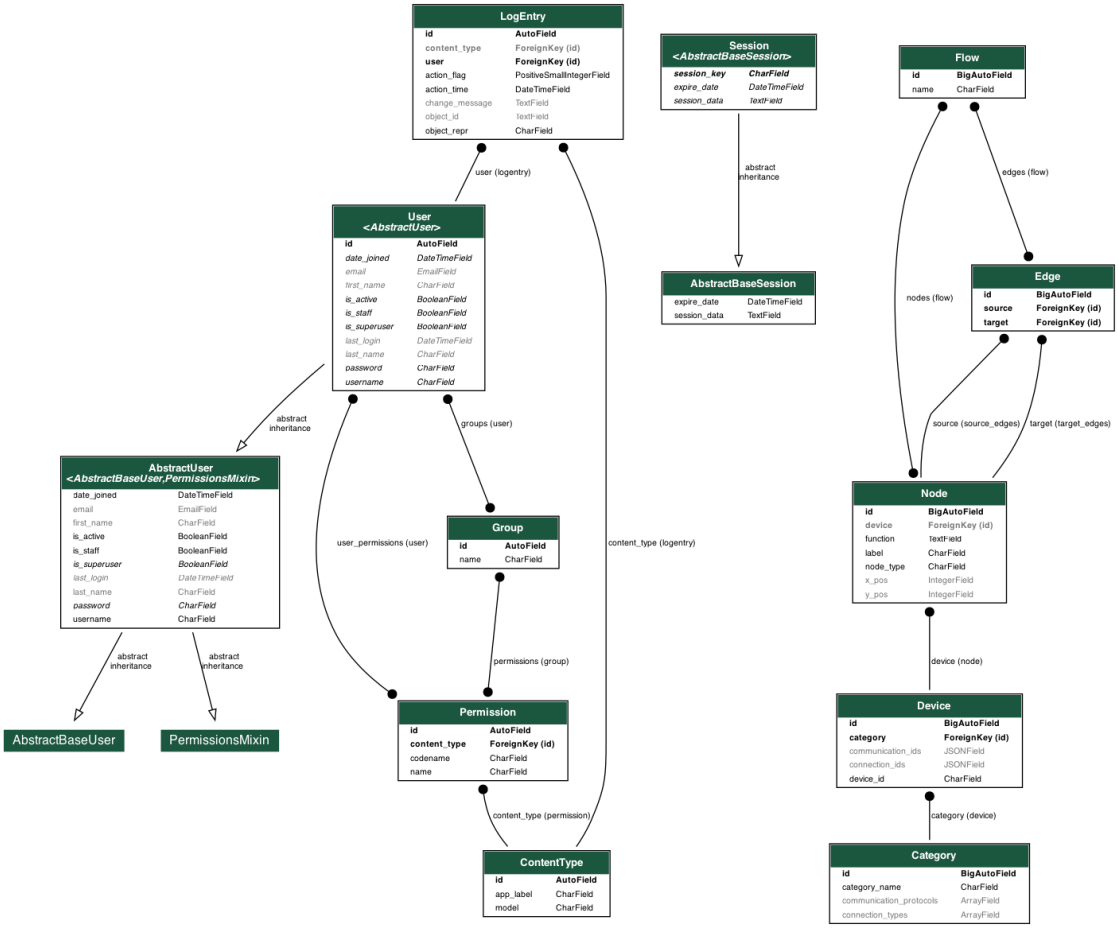

Figure 23: Logical view: Class diagram

9.1.3 Physical View

The physical view is intended to show the deployment of the application on hardware. In figure 24 the physical view is depicted. The main part of the illustration revolves around the local machine of which a user has installed and runs the application. Since Docker is implemented as a container, it handles all the dependencies and sets up the correct environment. Inside the container are Docker images for the server and client, and a volume for the database.

The connections show how the different parts interact. The Django server interacts with the database, and exposes a REST API to the frontend. The web socket server handles certain input form the frontend, and provides this to the Django server.

Because the application is intended to be used for testing external devices, not the local machine running the application, "User Mobile Device" and "Connected Device" are added to the deployment view to highlight this fact. As of now, the application only accepts Android as the operative system for the "User Mobile Device". Furthermore, the application does not interact directly with the "Connected Device", but forwards the instructions for execution via the "User Mobile Device". Both of these restrictions are illustrated in the diagram.

9.1.4 Process View

The process view is illustrated with an activity diagram in figure 25. It depicts a simplified overview of the runtime process of the application. The starting point assumes that the user has already installed the application and has it running, with a "User Mobile Device" and "Connected Device"

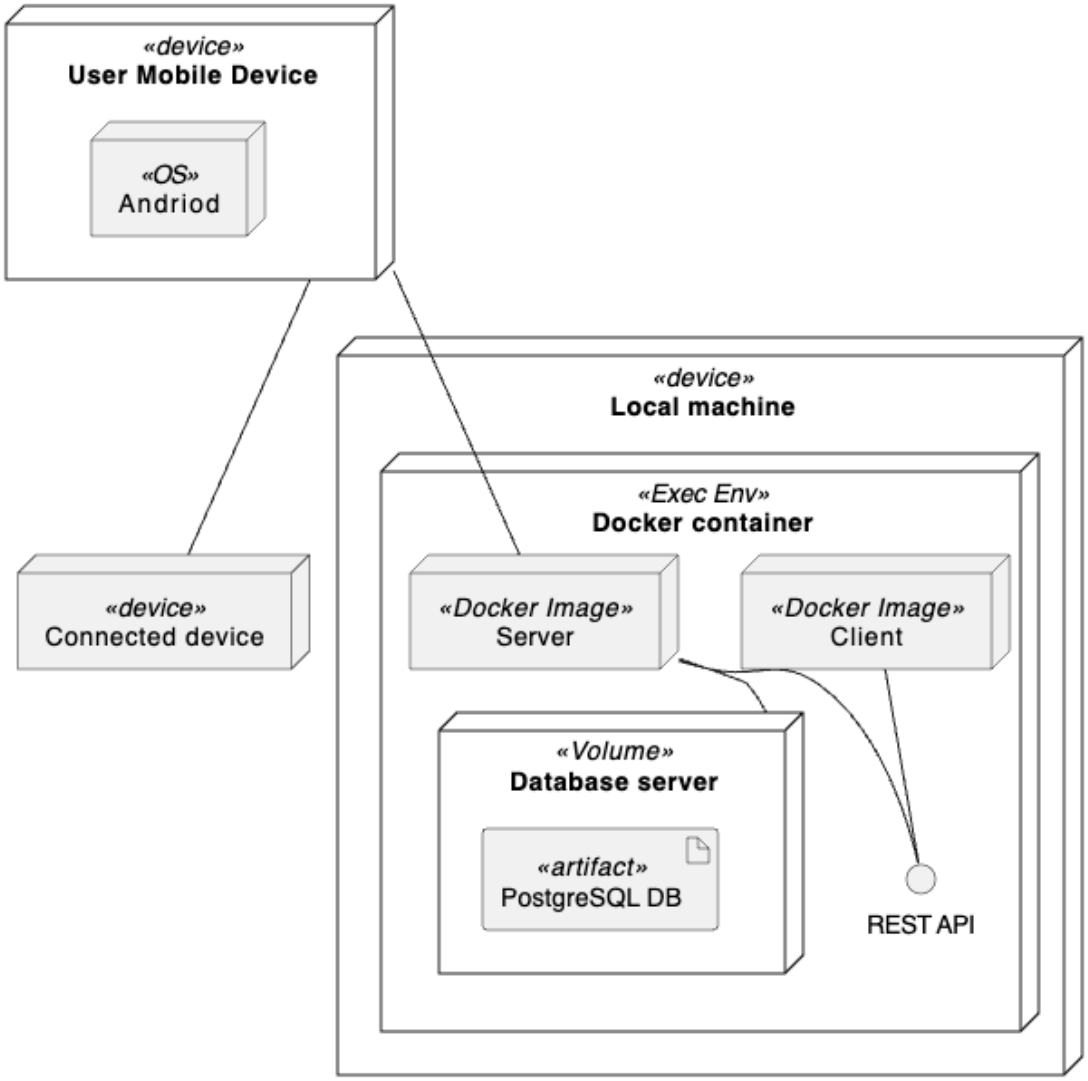

Figure 24: Physical view: Deployment diagram

connected.

The first part of the flow is for the user to create categories, devices, flow and nodes. It must happen in this order, as devices require a category id, and nodes require a device id to properly link them. When the user runs the flow, the backend parses the existing nodes in the flow, and then executes in order. Action nodes contain inputted instructions from the user on how and what to execute. The "User Mobile Device" receives the instructions, executes them, and sends instructions to the "Connected device". The assert nodes contains inputted instructions on how and what to assert.

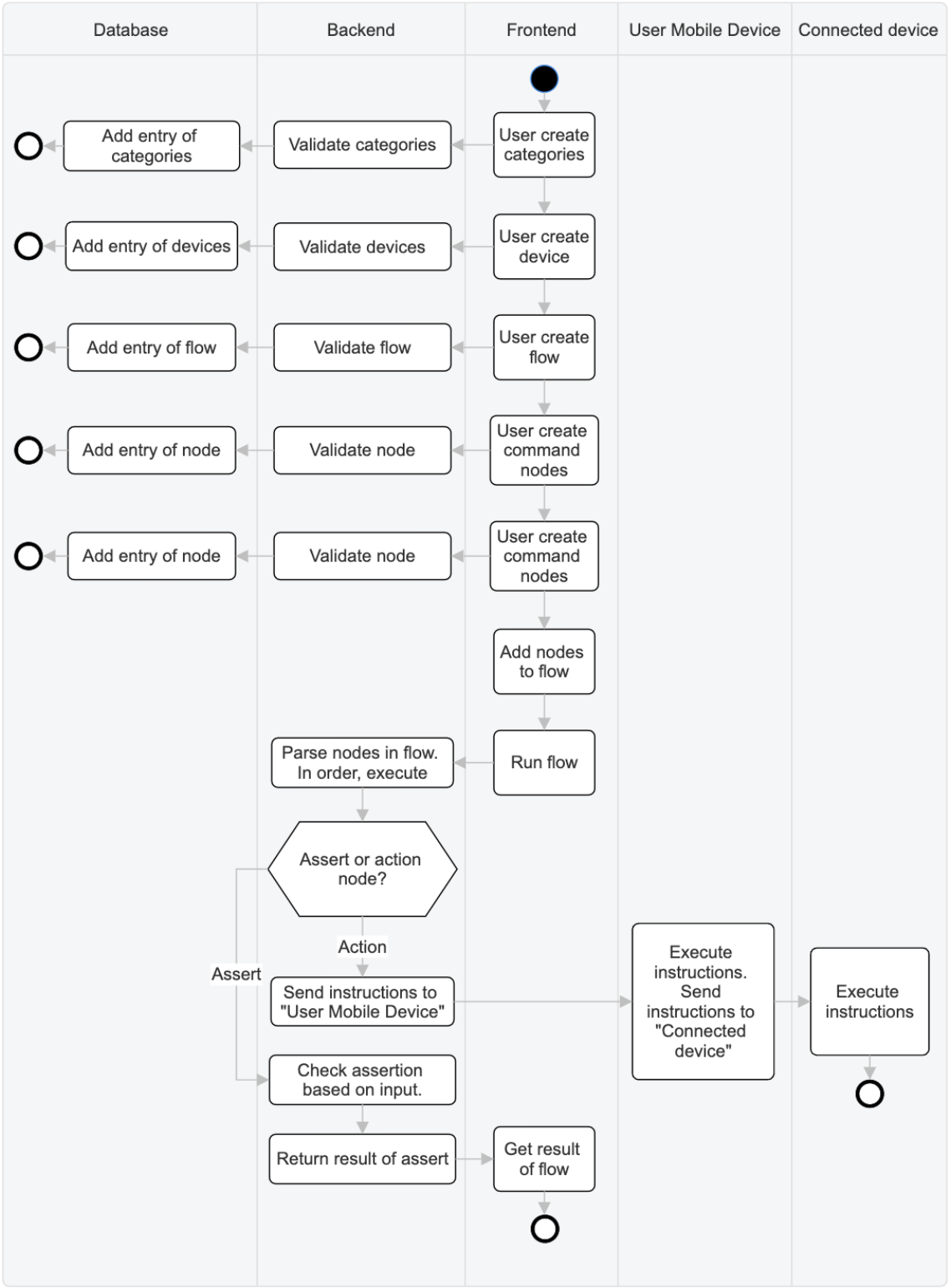

Figure 25: Process view: Activity diagram

9.1.5 Architectural tactics

Tactics, in the context of software architecture, are concrete actions taken to achieve quality attributes (Bass et al. 2021, p. 46-47).

Modifiability tactics:

Reduce coupling aims to ensure that different modules or parts of the code does not have high dependencies of each other (Bass et al. 2021, p. 106). By doing so, one part of the code can be changed without it affecting other parts of the code that it is utilized by.

Increase cohesion aims to ensure that the elements within different modules serve to fulfil the same purpose. In other words, a single module has a well-defined purpose, and every element within contributes to that purpose. Poor cohesion makes understanding, maintaining, and modifying the system more difficult, increasing costs. The team has employed tactics like splitting modules and reallocating responsibilities to enhance cohesion (ibid., p. 122-123).

Usability tactics:

Consistent UI design is, as stated by the name, implementing and using familiar and known design and system behaviour across the frontend part of the product. This should provide better understanding and intuitive actions for the user, and reducing the cognitive load.

Another usability tactic implemented is facilitate user initiative. Enhancing usability requires providing users with clear feedback on system actions and enabling them to respond effectively. It is essential for the system to promptly handle user-initiated cancel commands. For instance, when a user cancels an operation, such as creating a new device, all components involved in the operation must be quickly notified to take the necessary actions (ibid., p. 200).

Interoperability tactics:

Use of open standards and frameworks, and standardized communication protocols. This allows external systems to easily connected and utilize the application.

9.1.6 Architectural patterns

There were several patterns applied to the project development to ensure the desired quality attributes were emphasised. One of the quality attributes that were particularly important, due to the open source nature the product, was modifiability. To facilitate modifications for future developers without facing to much difficulty, both Model-View-Controller (MVC) and plug-in system architecture were implemented. MVC, or Model-View-Template in the case of Django applications, separate the business logic and data representation from the presentation logic (ibid.). The plug-in system architecture was a side-effect of using Vue, as it already have extensive libraries that allows developers to implement and abstract complex functionalities. The client-server pattern was also implemented. This pattern is used in the system to enhance modifiability as well as scalability, by delegating tasks between the client and server, separating responsibilities (ibid., p. 126).

9.2 Technical Result

In the previous sections of the report, an overview of the process and software has been described. This section intends to clarify the final result achieved.

The functional and quality requirements have been tested. The results can be found in appendix F.1 and F.2 respectively. 15 of 17 FR tests were successful and 4 of 5 QR tests. In our opinion, this is satisfactory as most requirements were completed. The tests that did not pass were due to time constraints as described in section 8. The FRs and the QR not completed are regarded as future work, some of which are described in section 12.

Figure 26 shows the test-setup the team has been working towards. In this setup the application is running on a computer, with an Android mobile device and nRF-kit connected. The user has created a category, device, flow and added nodes to the flow. The ”LED on” node in the flow is an action node that, when executed, sends instructions to the mobile device. The mobile device, in turn, sends instructions to the connected nRF-kit to turn on its LED light. The following nodes are assert nodes verifying that the LED was in fact turned on.

Figure 26: The application running on a computer, connected to mobile device and nRF kit

10 Innovation

10.1 Sustainability

From the outset of this project we have prioritised sustainability in our software. Given the customer's intent of releasing this product under an open source licence, we placed a particular emphasis on technical sustainability early in the planning phase of this product, key considerations are detailed below. In addition we have made important choices regarding social sustainability, along with concepts from environmental and individual focuses (Moises de Souza. et al. 2024). This includes choosing a darker colour scheme for a lower power usage, as well as ensuring an enjoyable developer and user experience for future users and developers. In the following sections we explore economic, technical, and social sustainability in further detail.

10.1.1 Economic Sustainability

In addition to our four primary sustainability goals, we acknowledge the software's potential contribution to our client's economic sustainability. As an open source solution intended for start-up companies, including our client, our product aims to reduce increased costs in association with testing IoT products compared to existing market solutions. The economic benefit for emerging IoT businesses has therefore been a significant driver for the development and planning of this software (Wright et al. 2024), as well as attributing to the importance of its technical and social sustainability.

10.1.2 Technical Sustainability

Throughout our development phases, a strong emphasis has been placed on creating maintainable software, due to the open source intent of this product. Therefore, we found technical sustainability to be of the essence in our code base. To facilitate the continued development of the product by future developers, we have adhered to the proposed web sustainability guideline (WSG) (Alexander Dawson. et al. 2024). This also builds upon the web content accessibility guideline (WCAG) (W3C World Wide Web Consortium Recommendation 2024), covered below regarding diversity, with a sharpened focus on future developers. The WSG has guided us to create a modular product, focusing on code splitting and reusability, repeating the "Don't Repeat Yourself" clean code practice. It states how you should not be writing repetitive code, but rather create shared methods and functions to handle shared functionality. In conjunction with these guidelines, we have also leveraged our frameworks and code languages to enhance the maintainability of our developed minimum viable product.

As this product is an MVP, we have prioritised future-proofing the development potential. We early recognised the importance of a maintainable code base for our client. To support its maintainability, we have adopted in-code documentation method to facilitate future developers' understanding of our code base. Assuring the possibility of future development to the product. For our server-side code we have also chosen to utilise large open source solutions widely recognised as industry standard within software development, as this will facilitate the maintainability of the code base for future developers.

10.1.3 Social Sustainability

To ensure social sustainability, we have prioritised accessibility for all end-users, particularly those with low or no vision. Although the software is intended for technical developers we have also conducted usability testing with non-technical individuals to guarantee an intuitive interface, this was to assure against bias during its design iterations from the development team and our client. To ensure accessibility for visually impaired users, we have adhered to the WCAG throughout development. This is further detailed in the diversity section below.

To mitigate potential socio-technical biases related to microeconomics, we have prioritised utilising other open source, and free-to-use tools and libraries. This will ensure its long-term viability as a solution for IoT testing needs and secure its potential market position, fronting the social sustainability for start-up businesses and interested individual stakeholders. Building upon this, it has been essential to create a concise documentation of the product to ensure the social sustainability of future developers. As previously mentioned, we have found it of grave importance to ensure an intuitive UI to further enhance the usability of our application for all users, thus we decided to conduct usability testing with non-technical persons.

10.2 Diversity

A cornerstone in software development is recognising the diversities in both development teams and user bases. This socio-technical perspective is crucial to creating applications suitable for all users, regardless of potential biases that can arise between user interfaces and experiences (Ask and Søraa 2024). Additionally, it is essential to consider the needs of users with various abilities, such as visual impairments using the application. For our product, we have ensured its compatibility with screen readers for visually impaired users, as well as carefully selecting colours to accommodate for colour blindness, something we have tested thoroughly throughout the development process.

10.2.1 Team Diversity

Our team is composed of members of similar age and a gender balance, fostering a balanced and collaborative environment. While we share common goals regarding this project we bring

a range of diverse experiences to the team. This includes frontend and backend development experience, algorithms and IoT experience, as well as web development and UI design experience. We believe this diversity has been instrumental in streamlining our development cycles, with diverse experiences allowing tasks to be assigned to the most knowledgable team members. To further optimise our workflow as mentioned in section **??**, we divided our team into two groups, one frontend development team, and one backend development team. This division has allowed each team member to focus on their area of strength, personal interest, and passion, benefitting the end product and our client and our own learning outcome.

Conversely, fostering a diverse development team is crucial for producing a high-quality product. This aligns with script theory which states, "[...], developers' world-views shape the technologies they create, that technologies communicate values in addition to executing tasks, [...]" (Ask and Søraa 2024, p.95). This theory highlights how diversities can inadvertently influence the end-product's usability.

Furthermore, diverse teams are more likely to establish inclusive default settings in their product. As noted, "[...], default settings tell us a lot about how developers envision their use. Default settings are also a good example of how design advocates something as "normal," and, consequently, something else as "abnormal"." (ibid., p.102), by incorporating diverse perspectives, developers can mitigate biases during the development and design phases of their product, this allows for the product to cater to a larger and wider range of users.

10.2.2 Product Diversity

To ensure our application's accessibility for a diverse developer user base, we prioritised compliance with the WCAG. In particular, we focused on guidelines relevant to low-vision and non-sighted users. To achieve this we have abided by the WCAG and ARIA guidelines to optimise the application's usability for those operating a screen reader. We also ensured sufficient colour contrast for our colour choices, adhering to section 1.4.3 of WCAG 2.1 to meet the AA standard. Additionally, we have consistently employed the appropriate ARIA roles and labels to enhance the user experience of screen reader users. To validate its usability, we have conducted rigorous testing using VoiceOver to simulate this experience. By doing this we could ensure a seamless page navigation experience for non-sighted users. We have also utilised different rendering options using Google Chrome to assure interface legibility for colour blind and low-vision individuals.

10.3 Artificial Intelligence

Given the final product scope's exclusion of artificial intelligence and machine learning, we have not implemented any of these solutions ourselves. While the planned integration of image recognition for IoT device testing has been deferred to future work due to the shift in product scope, we have leveraged AI tools throughout development. In particular, we have employed large language models and generative pre-trained transformers to ensure swift development cycles. The two utilised AI tools are detailed in the following two sections.

While these AI tools have demonstrated their value in code generation, we find it important to verify their outputs rigorously. Given the inherent inaccuracy and errors in AI-generated content, a thorough review process is essential to ensure the code quality in our code base, as well as ensuring correctness in these generated outputs (Abu-Haifa et al. 2024). This is why we've been particularly strict during our code review process.

10.3.1 GitHub Copilot

Most of the team members run their development environment with GitHub CoPilot installed, an AI-powered code completion tool. Though we acknowledge the potential benefits of this tool for development, we cannot confidently recommend its use for our product code base. GitHub CoPilot has not consistently generated desired outputs, limiting its practical usage for our context.

10.3.2 ChatGPT

The team integrated ChatGPT into their development workflow. This AI-tool has proven particularly valuable in debugging complex IoT solutions and generating comprehensive documentation for the product's code base. Additionally, ChatGPT has demonstrated its strong code production capabilities, aiding in creating the initial database architecture for our product. This accelerated process of establishing a foundational database structure, enabling us to begin testing the product's functionality early.

11 Discussion

11.1 Testing Limitations

Our testing process was highly beneficial in refining the product, particularly through iterative usability testing that enhanced interface intuitiveness and accessibility. Despite some constraints, these efforts resulted in meaningful improvements as seen in section 8.1. Usability testing was conducted throughout the development process, allowing for consistent feedback and refinements. While the testing framework could have been better structured, the insights we gathered were actionable, leading to significant design improvements. For instance, feedback from a participant with low vision confirmed that our high-contrast colour scheme and clear iconography were both accessible and user-friendly, reinforcing our focus on accessibility. However, the participant pool lacked broader diversity, which limited the variety of feedback we received. Expanding the participant base in future projects would enable us to capture a more comprehensive range of user experiences and address potential usability gaps more effectively. Additionally, testing was sometimes conducted on prototypes that were not fully functional, which, while useful for early feedback, limited the depth of insight into real-world usability challenges.

Technical testing, however, faced more significant challenges due to time constraints and shifting priorities as the project evolved. Early delays stemming from an unclear scope led to adjustments in the development timeline, forcing us to extend the code freeze and focus on delivering a functional MVP. As a result, unit and end-to-end testing were deprioritized, leaving some edge cases and integration aspects insufficiently addressed. While developers performed manual checks to ensure the product was functional, the absence of automated testing limited efficiency and consistency. Future projects should prioritize a balanced approach that incorporates automated unit and end-to-end testing earlier in the development process. This would not only improve overall product reliability but also free up resources to address complex technical challenges without compromising testing coverage.

11.2 Cross-Team Communication

Reflecting on our decision on splitting our team into two teams specialising in backend and frontend development, we identified some key areas where our approach could have been more effective. While our intent of splitting our team into specialised groups to increase efficiency of development was sound, a breakdown in cross-team communication hindered our overall efficiency towards the end of the project. Our internal team communication was strong, leading to faster feature development, though the lack of cross-team collaboration led to an array of unforeseen challenges during the final integration phases. Insufficient communication regarding feature approaches, especially regarding our API endpoints, resulted in compatibility issues and unexpected delays. As we neared the project deadline, these discrepancies became apparent, highlighting the importance of maintaining an open communication throughout the entire development process.

While we held our stand-ups, retrospectives and planning meetings in plenum, we subsequently split into our designated development teams thereafter. This hindered important information to be shared as the aforementioned meetings largely focused on the process rather than our product implementation. We have since learned how this could have been mitigated with a more open

communication channel between the teams. In particular, we can see the value of having a product manager creating a more synchronous development process in the future. We see how our communication was lacking, and it is a key issue we would like to highlight for future projects as a critical lesson. By implementing more effective collaboration strategies, we can mitigate similar issues in our future endeavours.

As noted, significant challenges arose during the final integration phase of the back- and frontend code bases. Compatibility issues between APIs and mismatches in verification and behaviour expectations increased the workload in the final sprint, necessitating additional development efforts to complete the project. A primary cause of these issues was the lack of clear and consistent communication regarding the intended functionality and technical functionality of nodes. While both teams had their own interpretations, there were significant misunderstandings across the teams. To mitigate similar problems in the future, we believe a more frequent and in-depth technical communication across teams is essential to create a shared understanding of the product's technical specifications. We see that we would have benefited from discussing the technical implementation of our product in unified meetings, rather than our split approach. This would have created a common overview and understanding of our implementation rather than causing these compatibility issues in the future.

Additionally during our final retrospective meeting we unanimously agreed that end-to-end testing would have been beneficial to mitigate our compatibility issues. This way we could have started with testing the necessary requirements during earlier stages of development, improving our cross-team communication and understanding by discovering the differences made. Possibly preventing the issues that arose during our final sprint, by solving these misunderstandings during initial development.

11.3 Water-Scrum-Fall

As previously noted, the team was inexperienced with development using IoT devices. As the backend team was responsible for the IoT part of the project, they had to figure out how to implement the necessary functionality. During the pre-study and initial development phase, the backend team spent a lot of time trying to learn the necessary knowledge for the project.

This lack of preparedness aligns with findings from a 2023 exploratory study on Water-Scrum-Fall, which highlights that inadequate and unintentional implementation of the methodology often lead to inefficiencies in hybrid methodologies (Krivankova and Remta 2023).

Given the overwhelming amount of possibilities for the development and the misunderstandings regarding the scope mentioned in section 8.2.3, the team ended up doing most tasks in sprint 2 as a learning by doing exercise. By the end of sprint 2, the misunderstandings regarding the scope was cleared up, and the backend team decided on a new approach to the development process. As described earlier, to better understand the complex development tasks at hand, we decided to list all requirements of the IoT communication and then implement them. In retrospect, we see that we inadvertently used Water-Scrum-Fall. Water-Scrum-Fall is a hybrid development approach that combines traditional Waterfall phases, like upfront planning and final deployment, with iterative Scrum practices during the development phase (West 2011).

Planning requirements for the most complex part of the system upfront made it easier to complete the tasks. This reflects one of the advantages highlighted in the study: upfront planning in hybrid methodologies supports the successful handling of complex system requirements. However, our lack of systematic guidelines for Water-Scrum-Fall mirrored the 2023 study's findings that many organizations struggle with poor implementations of this approach (Krivankova and Remta 2023). Looking back, it was the right choice to use the iterative development process for the frontend since the tasks were simpler, and it was easier to change the implementation at a later point. However, the backend should have used the Water-Scrum-Fall intentionally from the start. The required functionality was complex and less modifiable compared to the frontend functionality.

By resolving all required functionality for the IoT part of the software before developing the backend, the responsible team could have been more effective in their development. As the study

indicates, hybrid approaches like Water-Scrum-Fall can deliver high project success rates but require a strong upfront analysis to maximize benefits (Krivankova and Remta 2023). In future projects, we want to be more intentional when choosing the methodology. This includes ensuring proper preparation and improving customer collaboration during iterations, as suggested by the research.

12 Future Work

This product has several intended features planned for its future, and among these are some we did not fully implement during our project. Although log exports were intended from earlier planning phases of our product, we did not manage to implement this feature. We see our aforementioned unexpected challenges as a major cause of this, as we spent significant time in other important areas to enhance our application. Therefore, exporting logs from within the application has been pushed to future work, along with improved aesthetics for the log interface itself. Below we have detailed two of the major future features to be created.

12.1 Artificial Intelligence

The original project proposed by the customer included ML for image recognition and analysis for image-based testing and validation. For example, the customer wanted to be able to provide an image of a button to the software, which would then locate and execute a "press" command and assert that the button was pressed through visual inspection of the expected next state. They had already made progress with the remote command execution, but struggled with locating the buttons due to different resolutions on different phones and layouts. From this we created "FR16: The software could be capable of using ML algorithms to implement image-based testing and validation".

When first investigating Node-RED, we did look into Node-RED libraries and custom nodes that could support the development of these features. Ultimately, when we decided not to use Node-RED, this feature was put off as a won't-have. While the team did have members with ML and AI experience, building the entire application from scratch introduced a large scope, uncertainty about the difficulty of implementing core features, and a steep learning curve. As a result, we chose to focus on the core functionality necessary for the testing framework to function.

With the way the application is currently structured, the described feature could be integrated in the frontend as a new default node, where the customer could input the name of the field they want to press. However, if it is a necessity that the user can provide images of buttons or for the application to target non-text areas, this would require the addition of file uploading to the application. For the backend, both of these options would require an ML model to detect elements and changes, as well as functionality for remote commands such as simulated button presses which the customer could provide.

The ML model itself could be created from scratch using libraries such as Tensorflow or Open Computer Vision Library (OpenCV) or use pre-trained models. If only text-based fields are necessary, one could use Optical Character Recognition (OCR) that converts images of text into machine-readable text, or use pre-trained models for UI field detection (Macchi et al. 1997). For example, the project `web-form-ui-field` from the website "huggingface.co" uses the state-of-the art computer vision model YOLOv8 to do this. Using this model is as easy as importing the model and using it for predictions right away (Kirmer 2024), however, testing and validation of the model beforehand should be conducted to assure that it works as intended and provides adequate performance. In addition, it has many of the same cons mentioned earlier when relying on external programs, such as potential difficulties with maintainability and customization.

If the use case is locating a "button" image within a "phone screen" image, one could use methods such as template matching, i.e. finding the location of a template image in a larger image. That said, this method is sensitive to scale and rotation (without the addition of other techniques), which

could be an issue with different resolutions on different phones. Pattern matching algorithms, such as scale invariant feature transform (SIFT) or other algorithms not affected by these transformations, might handle this task better (Macchi et al. 1997). This depends on what the customer will input, but using an algorithm robust to these transformations off the bat could ultimately save time and effort.

Similar methods could be used for matching the current state with expected if the customer provides an image of the next expected state, e.g. through capturing screenshots with ADB commands. Determining what is an acceptable difference between these states and debugging and logging this could be a potential challenge, and would require more input and examples on how exactly the customer intends to use this feature.

12.2 Communication Protocols

The current software only supports ADB, Wi-Fi, BLE, and UART. Originally, the customer also mentioned LTE, and currently, while Wi-Fi is added as an option, it does not add any functionality in the application aside from marking the device as Wi-Fi. As mentioned in the pre-study, our application functions as a "broker" between the different devices, so these protocols would be most relevant in communication between host and device. Later in the project, the customer communicated that it would be sufficient for the devices to communicate with the host computer solely through cable or an API, thus this is what is currently implemented.

However, with multiple devices, it could prove difficult to connect all of these via cable to the host. If the system is to stay a "broker", and not add sniffing or MitM as discussed earlier, it would therefore be beneficial to add more ways of connecting to the host, such as BLE, Wi-Fi, and LTE.

For BLE, one could use tools such as Bleak, a GATT client software providing an asynchronous, cross-platform Python API to connect and communicate with IoT devices (Blidh n.d.). As long as both the host computer and the connected devices support BLE, implementation is relatively straight forward.

For Wi-Fi and LTE, several approaches could be explored depending on device type and supported protocols:

- TCP sockets for persistent connection, e.g. through the `socket` library in Python (Python Software Foundation n.d.). Would require the IoT device to support TCP
- An MQTT broker, i.e. an intermediary entity that allows MQTT clients to communicate, e.g. `paho-mqtt` (Eclipse n.d.). Would require the IoT device to support MQTT
- ADB over Wi-Fi for Android devices (Google for Developers n.d.[a])

For LTE, the host computer would also require an LTE modem.

Adding this to the current setup could be done as a new view in our modular setup. Collecting connection functionality in a setup node at the beginning could be beneficial for debugging and setting connection-specific or dynamic parameters such as randomized MAC-addresses (Google for Developers n.d.[b]).

Bibliography

Abu-Haifa, Mohammad et al. (2024). 'Comparative Analysis of ChatGPT, GPT-4, and Microsoft Copilot Chatbots for GRE Test'. In: *International Journal of Learning Teaching and Educational Research*.

Aircrack-ng (n.d.). *Aircrack-ng*. URL: https://www.aircrack-ng.org/.

Alexander Dawson., Ines Akrap, Tim Frick. and Mike Gifford. (2024). *Web Sustainability Guidelines (WSG) 1.0*. W3C.

Ask, Kristine and Roger A. Søraa (2024). *Digitalization and Social Change, a Guide to Critical Thinking*. CRC Press.

Bass, Len, Paul Clements and Rick Kazman (2021). *Software architecture in practice*. Fourth edition. Sei series in software engineering. Boston: Addison-Wesley.

Bettiche, Mehdi (2018). *Using Node-Red for testing*. Last Accessed: 2024-11-19. URL: https://medium.com/meetech/using-node-red-for-testing-46df7b8daef9.

Blidh, Henrik (n.d.). *Bleak - Usage*. URL: https://bleak.readthedocs.io/en/latest/usage.html.

Clegg, Dai and Richard Barker (1994). *Case Method Fast-Track: A Rad Approach*. AddisonWesley Longman Publishing Co.

Crisping, Lisa and Janet Gregory (2009). *Agile Testing - A Practical Guide For Testers and Agile Teams*. Addison-Wesley Educational Publishers Inc.

Eclipse (n.d.). *Eclipse paho-mqtt - client module*. URL: https://eclipse.dev/paho/files/paho.mqtt.python/html/client.html.

Econocom Digital Security (n.d.). *BtleJuice Framework*. URL: https://github.com/DigitalSecurity/btlejuice.

EMQX Team (2024). *MQTT Broker: How It Works, Popular Options, and Quickstart*. URL: https://www.emqx.com/en/blog/the-ultimate-guide-to-mqtt-broker-comparison.

Google for Developers (n.d.[a]). *Android Debug Bridge (adb)*. URL: https://developer.android.com/tools/adb.

— (n.d.[b]). *MAC randomization behavior*. URL: https://source.android.com/docs/core/connect/wifi-mac-randomization-behavior.

Jasek, Slawomir (n.d.). *GATTacker - Outsmart the Things*. URL: https://github.com/securing/gattacker.

Kirmer, Stephanie (2024). *Choosing and Implementing Hugging Face Models*. URL: %7Bhttps://towardsdatascience.com/choosing-and-implementing-hugging-face-models-026d71426fbe%7D.

Kniberg, Henrik and Marrias Skarin (2010). *Kanban and Scrum - making the most of both*. C4Media Inc.

— (2015). *Scrum and XP from the trenches*. C4Media Inc.

Krivankova, Michaela and Daniel Remta (2023). 'Real-Life Water-Scrum-Fall: Insights from Large Companies in Czech Republic'. In: *Lecture Notes in Business Information Processing*. SPRINGER INTERNATIONAL PUBLISHING AG, pp. 183–192. DOI: 10.1007/978-3-031-33976-9_12.

Krutchen, Philippe (1995). *Architectural Blueprints—The "4+1" View Model of Software Architecture*. IEEE Software 12.

Lildog (2024). *Lildog — Om oss*. URL: https://lildog.com/no/om-oss.

Macchi, Federico et al. (1997). 'Image-based Approaches for Automating GUI Testing of Interactive Web-based Applications'. In: *Pattern Recognition*.

McLaughlin, Stephen (n.d.). *node-red-contrib-image-tools*. URL: https://flows.nodered.org/node/node-red-contrib-image-tools/in/BGIyoBcAAOoc.

Meland, Per Håkon (2024). *Risk Management During Development*. Presentation, Norwegian University of Science and Technology (NTNU), TDT 4237. Photo by Allan Mas from Pexels.

Moises de Souza., Ana Carolina et al. (2024). 'Exploring Social Sustainability Alignment in Software Development Projects'. In: *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*. INSTICC. SciTePress, pp. 250–261. ISBN: 978-989-758-696-5. DOI: 10.5220/0012739600003687.

NASA (2017). *NASA Systems Engineering Handbook*. 12th Media Services.

Node-RED (n.d.). *Node Test Helper*. URL: https://github.com/node-red/node-red-node-test-helper.

Nordic Semiconductor (n.d.[a]). *Android-nRF-Connect - Automated tests*. URL: https://github.com/NordicSemiconductor/Android-nRF-Connect/blob/main/documentation/Automated%20tests/README.md.

Nordic Semiconductor (n.d.[b]). *nRF Sniffer for Bluetooth LE*. URL: https://docs.nordicsemi.com/bundle/nrfutil/page/nrfutil-ble-sniffer/nrfutil-ble-sniffer_0.12.0.html.

OWASP Foundation (2024). *Threat Modeling Process*. Accessed: 2024-11-21. URL: https://owasp.org/www-community/Threat_Modeling_Process#stride.

Pavitt, Joe and Steve McLaughlin (2023). *Use any npm module in Node-RED*. URL: https://flowfuse.com/blog/2023/06/import-modules/.

Pinker, Itiel (2021). *IoT Testing Tutorial Using Locust, Paho, and BlazeMeter*. URL: https://www.blazemeter.com/blog/iot-testing-tutorial.

Python Software Foundation (n.d.). *socket — Low-level networking interface*. URL: https://docs.python.org/3/library/socket.html.

SysSec KAIST (n.d.). *LTESniffer - An Open-source LTE Downlink/Uplink Eavesdropper*. URL: https://github.com/SysSec-KAIST/LTESniffer.

Unknown (n.d.). *node-red-contrib-flowtest*. URL: https://flows.nodered.org/node/node-red-contrib-flowtest.

W3C World Wide Web Consortium Recommendation (2024). *Web Content Accessibility Guidelines 2.0*. W3C.

West, Dave (2011). *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today*. URL: https://www.verheulconsultants.nl/water-scrum-fall_Forrester.pdf.

Wireshark (2020). *WLAN (IEEE 802.11) capture setup*. URL: https://wiki.wireshark.org/CaptureSetup/WLAN.

Wright, Nataliya Langburd, Frank Nagle and Shane Greenstein (2024). 'Contributing to Growth? The Role of Open Source Software for Global Startups'. In: *Harvard Business School*.

Yokoi, Kazuhito (n.d.). *node-red-contrib-tensorflow*. URL: https://flows.nodered.org/node/node-red-contrib-tensorflow.

Appendix

A Planning

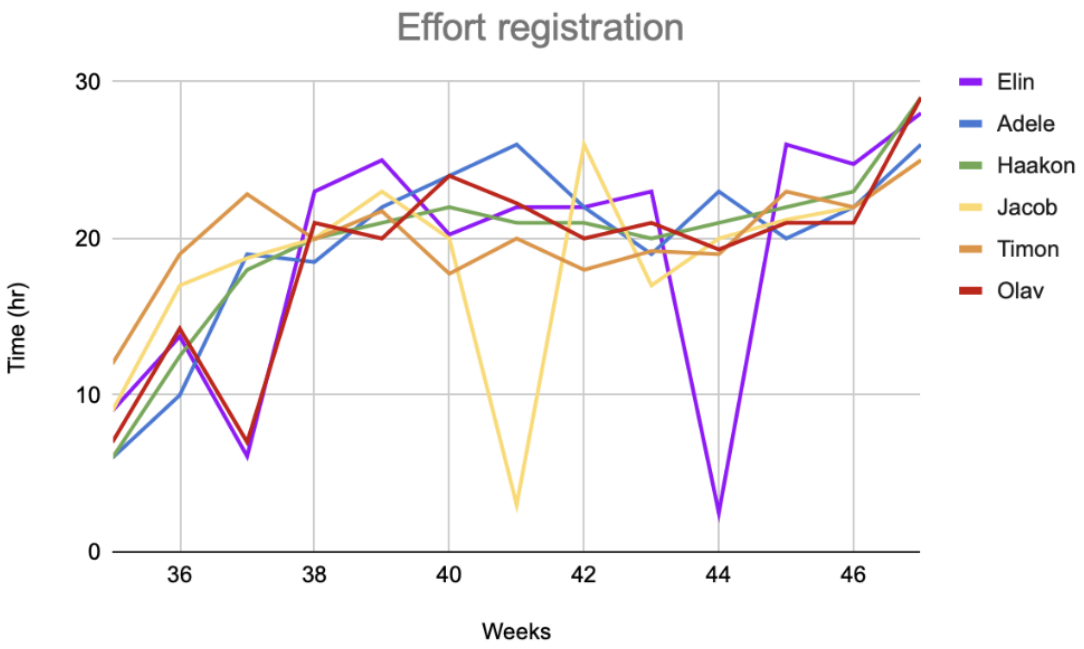

Figure 27: Time efforts by team member

B Usability Test

This is a testing platform to help users set up automated tests for their IoT devices. It is intended as a graphical user interface allowing users to drag test-blocks onto the screen and connect them in the order they wish them to run.

Task One: "As a user you want to add a new collar device to the platform, add a collar named 'Collar 3' to your devices. Afterwards you want to remove 'Device 1', from the list of available collars. Before navigating back to all your device types."

Observations: Tries clicked device type first, rather than device. Had to choose a connection before adding, took a second. Find the section with the collar, and add the device to the correct place, and remove device 1.

Could be a translation issue, but configuring devices could be a bad word, but the add button does specify type. Maybe just 'add type' could be good?

Time Taken: 2min, 8sec

Task Two As a user you want to modify 'First Flow', navigate to the flow and add a block checking that the light was turned on. Afterwards, check the logs and select three devices to display.

Observations: Goes to flows, and finds the first flow. Tries to click run test multiple times. Then tries edit flow, before just chicken on the card. Tries clicked run to add the block, then tries clicking all the existing blocks. Finds the stored devices, and drags the correct block onto the flow. Finally connects the block to the flow blocks. Finds the log, and adds three devices to the view. Tries saving the log and navigating back and fourth, Redoes adding the light block. And the connection.

Time Taken: 3min

Task Three As a user you wish to change the name 'Halsbånd' to' Collar', edit the name of this device type to match the new requirement.

Observations: Goes to configure devices, comments it being a type and clicks the edit button on the correct card, edits the name

Time Taken: 22sec

Task Four As a user you wish to change the name of the flow 'First Flow' to 'Check that the light turns on', navigate to the appropriate page and make the change. Thereafter create a new flow with a name of your choice!

Observations: Goes to flows and selects the edit button, then changes the name to the new name. Creates a new flow, with the new name

Time Taken: 40sec

Task Five As a user you want a darker colour scheme, enable dark mode for the interface.

Observations: Navigates to the wrong page before finding the toggle.

Time Taken: 9sec

Question	**Rating**
The page was easy to navigate	4
Overall interface rating	4.5
The interface was too complex	0.5
I would need to learn how to use the system	3
There were many inconsistencies in the interface	1
The system did not operate as expected	1
I think people can learn this system quickly	5

Table 16: User feedback on a scale from 1 to 5

Questions and Comments

"Is there anything you feel could be done differently or better?"

Device type vs type, bad to click on the box to open flow without an indicator. Categories could be better than device type?

Might not be straightforward to close the log using the log button. Could be better with a back button as on the other pages, or a breadcrumb. The current back button takes you all the way out, a better label or colour could be better on the log button itself as well to indicate that it is "activated".

"Did you find the interface intuitive?"

Learning to use the system would probably take ca. 30 minutes. Overall aside from the previous

issues I find it understandable with appropriate icons and it was easy to navigate, delete, add etc.

"Did you find any elements too distracting?"

No, I don't think so.

"Do you have any further comments?"

No, I think everything is covered pretty well. Maybe, some feedback on a flow saving when you add a new block, or a save button.

C Dependabot with GitHub Actions

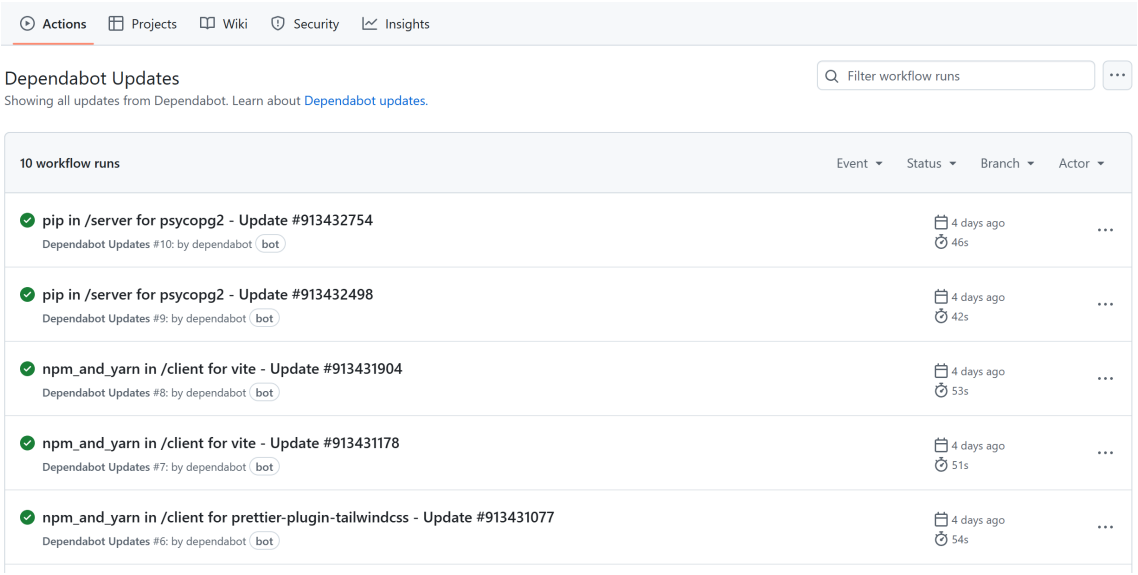

Figure 28: Dependabot workflow runs in GitHub Actions.

D Tech Stack Risk Assessment Matrix

Risk	Likelihood	Impact	Risk level	Mitigation strategy
Vue.js security vulnerabilities	Medium	High	High	Mitigate: Regularly update libraries, use tools like npm audit to detect vulnerabilities
Tailwind inconsistent UI	Low	Medium	Low	Mitigate: Enforce design guidelines and style consistency
Django security vulnerabilities	Low	High	Medium	Mitigate: Focus on security during development and use built-in security features
PostgreSQL injection attacks	Low	Medium	Low	Mitigate: Use parameterized queries and Django ORM to prevent SQL injection.
PostgreSQL backup and recovery failures	Low	Medium	Low	Accept: Implement automated backups and test recovery processes regularly

Figure 29: Risk Assessment Matrix for Tech Stack

E Process risk management

ID	Activity	Risk factor	Consequences	Likelihood	Risk level	Strategy	Deadline	Responsible
R 1	Communication	Miscommunication among team members	M - Delays, overlapping work, unwanted results	M	M	Reduce - Promote open communication, frequent “Stand-ups”	Continuous	Scrum master
R 2	Team commitment	One team member not meeting commitments	H - Reduced team capacity, missed deadlines	M	H	Avoid - Reassign tasks, escalate to advisors if necessary	Immediate	Project leader
R 3	Health issues	Team member gets sick	M - Reduces capacity, delays in deliverables	M	M	Transfer - Delegate tasks to other members, contact course advisors if necessary	Immediate	Project leader
R 4	Technology selection	Members have lack of familiarity with Vue, Node-RED	H - Learning curve, delays, poor quality	M	H	Reduce - Assign research tasks, conduct team workshops to learn technologies	Sprint start	All
R 5	Voluntary work in student organizations	Two team members are active volunteers in student organizations	M - Reduced member capacity, poorer quality of work	M	M	Reduce - Assign clear tasks to mentioned members with clear deadlines	Continuous	Scrum master
R 6	Part-time jobs	Members with part-time jobs prioritize work over the	M - Reduced member capacity, poorer quality of work,	M	M	Reduce - Assign clear tasks to mentioned members with clear deadlines	Continuous	Scrum master

Figure 30: Process risks (1)

		project	missed deadlines					
R 7	Task manage ment	Too many or too complicat ed planned features	H - Infeasible design, missed deadlines	H	H	Reduce - Time estimations for backlogs, prioritize tasks	Weekly review	Scrum master
R 8	Vacatio n time	Members take vacation during critical phases	M - Delays in task completio n, reduced capacity	M	M	Avoid - Plan in advance for member availability, assign tasks before vacations begin	Notify ahead of time, plan at start of each sprint	Project leader

Figure 31: Process risks (2)

F Technical Result

F.1 Functional Requirements Tests

FR1: Add new category	
Executor	Adele Felicia Ellingsen-Grønningsæther
Date:	24/10/2024
Time used:	20 seconds
Evaluation:	Success
Comment:	Category is crated and shown to user

FR2: Add new device	
Executor	Adele Felicia Ellingsen-Grønningsæther
Date:	24/10/2024
Time used:	25 seconds
Evaluation:	Success
Comment:	Device is crated and shown to user

FR3: Create flow	
Executor	Adele Felicia Ellingsen-Grønningsæther
Date:	24/10/2024
Time used:	10 seconds
Evaluation:	Success
Comment:	Flow is created and user has it displayed

FR4: Create node	
Executor	Adele Felicia Ellingsen-Grønningsæther
Date:	24/10/2024
Time used:	45 seconds
Evaluation:	Success
Comment:	Node is created

FR5: Add nodes to flow	
Executor	Adele Felicia Ellingsen-Grønningsæther
Date:	24/10/2024
Time used:	10 seconds
Evaluation:	Success
Comment:	Node is added to flow

FR6: Execute tests	
Executor	Haakon Karstensen
Date:	12/11/2024
Time used:	1 minute
Evaluation:	Success
Comment:	User is a able to execute tests by running the flow

FR7: Trigger interactions	
Executor	Haakon Karstensen
Date:	12/11/2024
Time used:	1 minute
Evaluation:	Success
Comment:	The system sends instructions to mobile device to interact with the connected nRF kit

FR8: Connetion types	
Executor	Haakon Karstensen
Date:	12/11/2024
Time used:	6 minutes
Evaluation:	Success
Comment:	System allows BLE, ADB, UART and API to be utilised in testing

FR9: Mulitiple devices	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	2 minutes
Evaluation:	Success
Comment:	User is able to connect and run device on 2 devices via cable

FR10: External tools	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	2 minutes
Evaluation:	Failed
Comment:	Not able to utilise external tools and system in testing

FR11: Provide feedback	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	2 minutes
Evaluation:	Failed
Comment:	The system does not provide correct result

FR12: Real-time logs	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	1 minutes
Evaluation:	Success
Comment:	The system provides real-time logs for troubleshooting

FR13: Test scenarios	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	3 minutes
Evaluation:	Success
Comment:	The user is able to edit a test scenario

FR14: Test configurations	
Executor	Elin Haugum
Date:	14/11/2024
Time used:	1 minutes
Evaluation:	Success
Comment:	The user is able to reuse test configurations

FR15: Differentiate between devices	
Executor	Jacob Alveberg
Date:	25/10/2024
Time used:	15 seconds
Evaluation:	Success
Comment:	The system uses labels and colours to differentiate between devices.

FR16: Connection types	
Executor	Jacob Alveberg
Date:	25/10/2024
Time used:	15 seconds
Evaluation:	Success
Comment:	The system uses labels and colours to highlight connection types

FR17: Multiple logs	
Executor	Jacob Alveberg
Date:	25/10/2024
Time used:	45 seconds
Evaluation:	Success
Comment:	The system shows multiple logs

F.2 Quality Requirements Tests

M1: Add new feature	
Executor	Haakon Karstensen
Date	06/11/2024
Environment	In development
Stimuli	Would like to implement new feature to fit their needs
Expected response measure	Able to implement new feature without needing to alter other parts of the app
Observed response measure	The feature was implemented without issue
Evaluation	Success
Comment	Adding new functionality did not affect the rest of the application.

Table 17: M1 test

M2: Update existing code	
Executor	Haakon Karstensen
Date	06/11/2024
Environment	In development
Stimuli	Would like to update existing code
Expected response measure	Able to update the code without affecting other functionality
Observed response measure	The modification was implemented with no side effects
Evaluation	Success
Comment	Modifying functionality did not affect the rest of the application.

Table 18: M2 test

U1: Add new category and device	
Executor	Timon Alexander Selnes
Date	20/10/2024
Environment	Normal operation
Stimuli	Wants to add new category and device
Expected response measure	The user is able to add a category and device within 1 minute
Observed response measure	The user was able to complete the task in 45 seconds
Evaluation	Success
Comment	-

Table 19: U1 test

U2: Run flow and verify device functionality	
Executor	Elin Haugum
Date	11/11/2024
Environment	In development
Stimuli	Wants to run flow to verify device functionality
Expected response measure	Able to run the flow without needing to alter other parts of the app
Observed response measure	The flow executed without issue
Evaluation	Success
Comment	Running the flow did not affect the rest of the application.

Table 20: U2 test

I1: Connect device with different connection type	
Executor	Elin Haugum
Date	15/11/2024
Environment	Runtime
Stimuli	Wants to connect device with a different connection type
Expected response measure	Able to connect the device without altering other parts of the app
Observed response measure	The user was not able to connect a device with an LTE connection type
Evaluation	Failure
Comment	The application only had BLE as a valid connection type.

Table 21: I1 test

G Monitoring Device Communication

Issue and Context

Since the customer mentioned various communication protocols between devices, we needed a way to capture the data flow between them and their respective logs. This would be necessary both for asserting changes on devices and sending commands. We also potentially needed to assess whether the connection itself was working correctly, rather than just asserting if the results at each endpoint were correct.

We separate here between communication standards that are both protocols and wireless technology standards, i.e. Wi-Fi, LTE, and BLE, and those that are "purely" protocols connected via cable or one of the aforementioned standards, i.e. ADB (Android Debug Bridge) (via cable/Wi-Fi) and UART (via cable). Hereafter, the former is referred to as "communication protocols" and the latter as "connection types". Since attaining data through ADB and UART via cable can be done through Python libraries, we will mainly discuss the ways of capturing data for Wi-Fi, LTE, and BLE.

1. Sniffing (Passive Listener)

The first option was sniffing, which involves passively listening to the communication without

actively participating. This would mean that all devices would connect to each other beforehand, and the host computer of the application would capture packets sent between them.

We found several open source tools that allowed for this, such as Nordic Semiconductor's `nRF sniffer for BLE` (Nordic Semiconductor n.d.[b]), Wireshark for Wi-Fi (Wireshark 2020), and an LTE sniffer by the security company SysSec (KAIST) (SysSec KAIST n.d.).

This would allow us to directly assess and debug the communication between the devices. However, attaining the device logs could be difficult unless the devices were reprogrammed to continuously send beacons or were directly connected to the host computer. Lastly, separating the sources of the sniffed logs could prove difficult. Therefore, this option would most likely not solve our issues on its own.

In addition, this option introduced security issues. Legally, these tools could only decode public unencrypted information, meaning the customer would have to make all their communication public, exposing it to potential malicious use. While this could help the customer uncover security issues, this was not the purpose of the application.

2. Broker (Active participant)

Inpsired by the concept of an MQTT broker (EMQX Team 2024), the second option involved the computer hosting the application acting as a "broker" between the devices, meaning that all the devices would connect to the application beforehand.

There are many tools for connecting devices using various communication protocols available, such as "Bleak" for BLE (Blidh n.d.). This allows explicit control over which devices are connected and what data is accessed. Additionally, automating connection setup and teardown would be especially useful for multi-device flows.

Our main concern was whether it would be possible to assert the connection quality between the devices. However, we found out that this could be done through assessments performed on the devices, which could then be logged. For example, for the nRF kits, we found the `read-rssi` method to measure the RSSI (Received Signal Strength Indicator) between the two devices (Nordic Semiconductor n.d.[a]).

3. Man-in-the-middle

The last option was man-in-the-middle (MitM). This combined the first two options by making the host computer an active participant in the communication, all while the devices remained unaware and believed they were communicating directly.

This would allow for both listening in on the "conversation" between the devices, while also being able to send commands by altering these packages. We found several potential tools for this, such as the BtleJuice Framework (Econocom Digital Security n.d.) and GATTacker (Jasek n.d.) for BLE, or Aircrack-ng (Aircrack-ng n.d.) for Wi-Fi. These tools were however mainly used for security testing, and MitM itself is a cyberattack method. Being unsure about the legal and security concerns, we did not explore this option further.

Decision

We presented these options to the customers, who clarified that assessment of and direct access to the connection between the IoT devices was not necessary. Since the host computer would only need to access the current states at each connection endpoint, we selected option 2, where the host computer acts as a broker between the devices.