Joonas Karppinen – Eino Ruuth
Niko Meriluoto – Toni Hirvikallio

Software engineering project

I-SPY-U sensor management system

Metropolia University of Applied Sciences
Information and Communication Technologies
Software Engineering
Software Engineering Project
02.05.2025

1. Project development process

1.1. Applied Methodology

Project was developed using the Agile methodology.
The team was organized using the Scrum principles and Kanban for backlog and task management.
Development used DevOps methodology by employing continuous integration and delivery with automated testing.

1.2. Planning and Risk analysis

- **Risk Description**:
  - Inadequate timeframe
  - Device malfunctions for sensors, microcontrollers or computers
  - Server malfunctions or breakdowns
  - Software limitations
  - Supply chain issues for sensors and chips
- **Likelihood**:
  - Timeframe medium
  - Hardware low
  - Software medium
- **Impact**:
  - Timeframe low
  - Hardware high
  - Software medium
- **Mitigation Strategies**:
  - Timeframe: careful planning
  - Hardware: backup server, microcontroller and sensors
  - Software: thorough research

1.3. Agile implementation Approach

Backlog and tasks were managed using the Kanban system with Trello. Scrum practices were upheld with daily scrums.
The project was implemented in 2 week sprints with sprint reviews at the end of each sprint. New sprint was planned, backlog updated and postponed tasks transferred to the next sprint.

The development, integration, testing and deployment were done in a continuous cycle aiming to always produce a minimum viable product with a functional and modular approach.

As we had a small team of four there were really no strict roles, e.g. dedicated testing engineer or operations engineer, but everyone had to fill every role. Scrum master was rotated every sprint.

GitHub was used for version control and our own servers were running Jenkins for CI/CD and SonarQube for code analysis. This allowed us to work seamlessly even remotely and get the latest artifact for testing through build automation and GitHub releases or DockerHub for every developer. This improved consistency for our development-testing cycle.

1.4. Scrum team roles and Responsibilities

Product owner with the responsibility of looking out for the clients point of view.
Scrum master for managing the sprint and development coordinating development and client needs.
Developer with the responsibility of designing and implementing needed architecture and code.
Planning was the whole team's cooperative effort.

2. Product documentation

2.1. Introduction

I-SPY-U is a sensor management suite with DB for storing sensor data, REST API for client connectivity and expandability for client software, messaging broker and solution for sensor connectivity and JavaFX UI for multiplatform (Windows and Linux) compatibility.

Server can be run containerized in VM's or Kubernetes cloud environments and client GUI in Windows or Linux desktops. The API allows easy implementation of additional UI's for example web or mobile. Client has also a containerized version so it can be remotely used already with VNC or with minor changes in a browser via NoVNC or for systems not compatible, e.g. Mac, mobile etc.

The link to the project github repository: https://github.com/0x6A4B/OTP1

2.2. Design

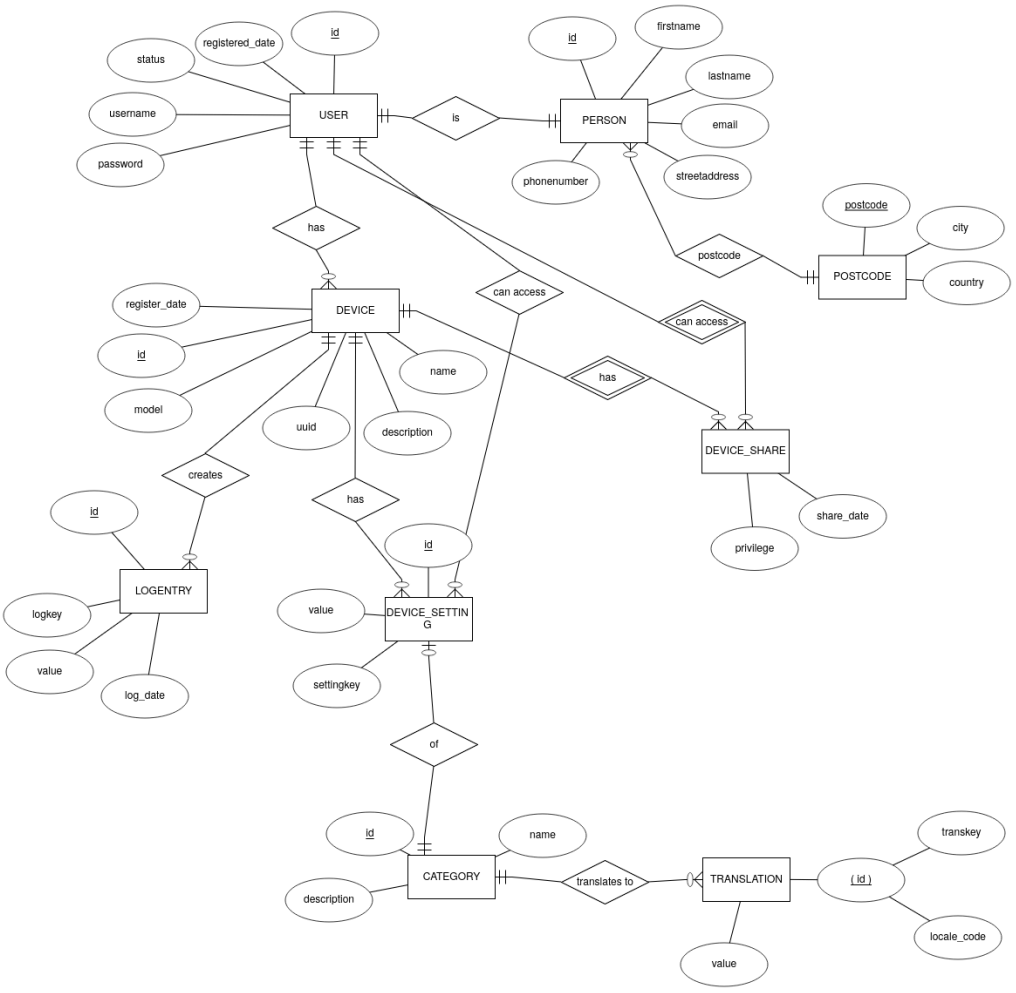
Client was designed using MVC-model separating model from view and allowing easy additional GUI development for example for Android or another GUI library other than JavaFX.
I18N and L10N were also kept in mind while designing and were implemented in the client application with dynamic translations for easy addition of languages without the need to alter source code. Localization implemented text direction, converted units and local numbering formats. DB has a localization table available for future localization and translation development and possibility to use translations from DB instead of local configuration files if so desired.
GUI uses JavaFX library and FXML.
Object oriented programming and Java was chosen for the paradigm and strong emphasis was put on good programming practices with single responsibility, minimising side effects and design patterns chosen with thought.
This has left the software easy to maintain and develop further for future improvements.
It has been implemented as a multiplatform client with a single JAR available for download for both Linux and Windows.

Server - client architecture through REST API was chosen also allowing easy expandability to Web or Mobile solutions.
Container architecture was chosen especially for server but also for the client for easy and consistent deployment of server and possibility for remote usage of client via VNC or in a browser with addition of NoVNC.
Database architecture was selected to be MariaDB relational database for easy self or cloud hosted solutions. Using Spring Data JPA for DB access from API server.

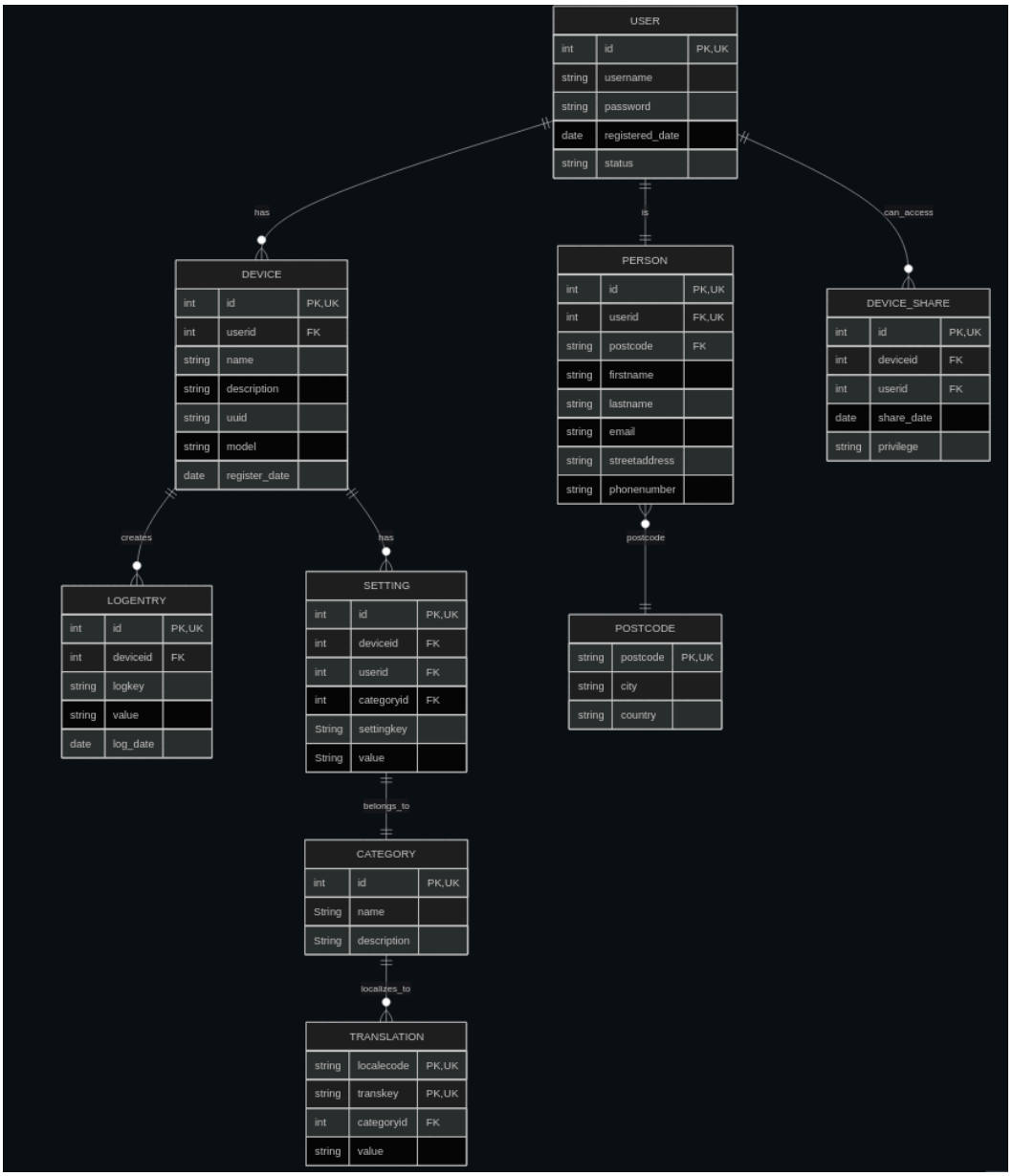
API server deployment to k3s/k3d kubernetes cluster that can also be used with full on-premise or cloud hosted k8s Kubernetes.
Servers are designed to work in a local network isolated from outside connections, leaving only the MQTT broker and API exposed to the internet with the REST API using HTTPS for security. MQTT was chosen for sensor messaging allowing fast, lightweight, resilient and scalable communication between sensors and the server. This has also left the possibility for easy future development for features in communication between sensor and client directly via broker allowing the showing of sensor status for example.

2.3. ER Diagrams

Using MariaDB relational database with translation table for localisation needs



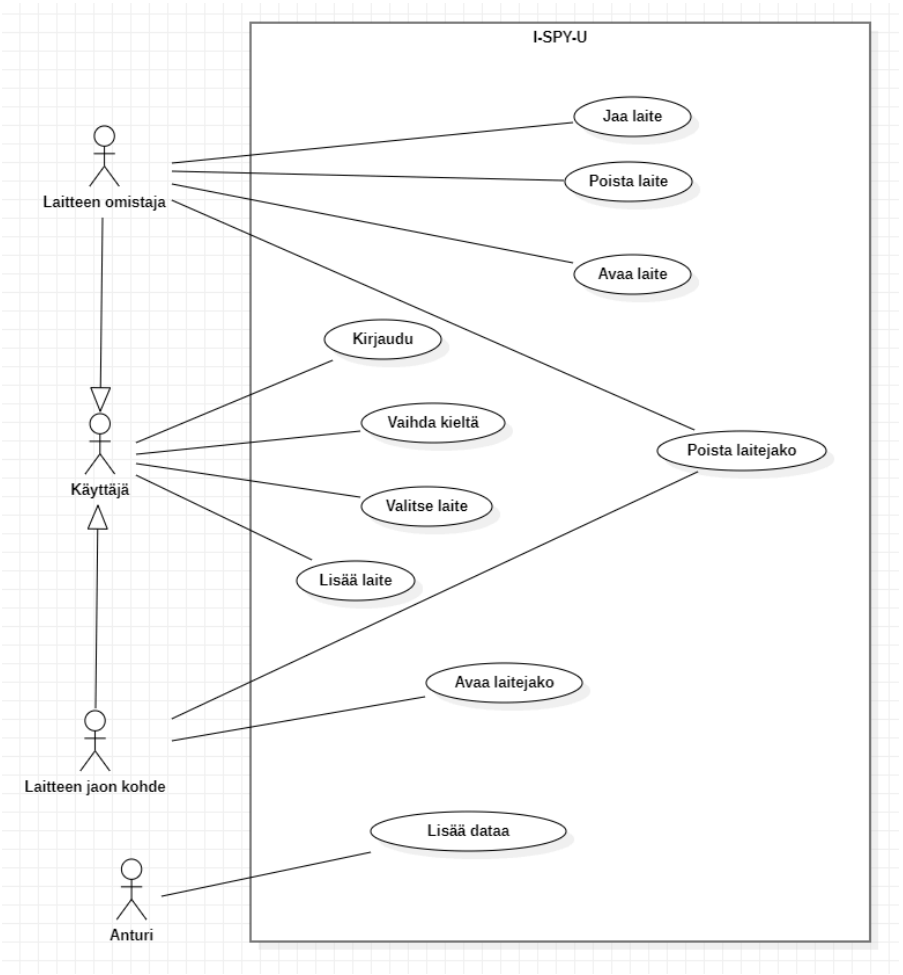
ER diagram of the project DB



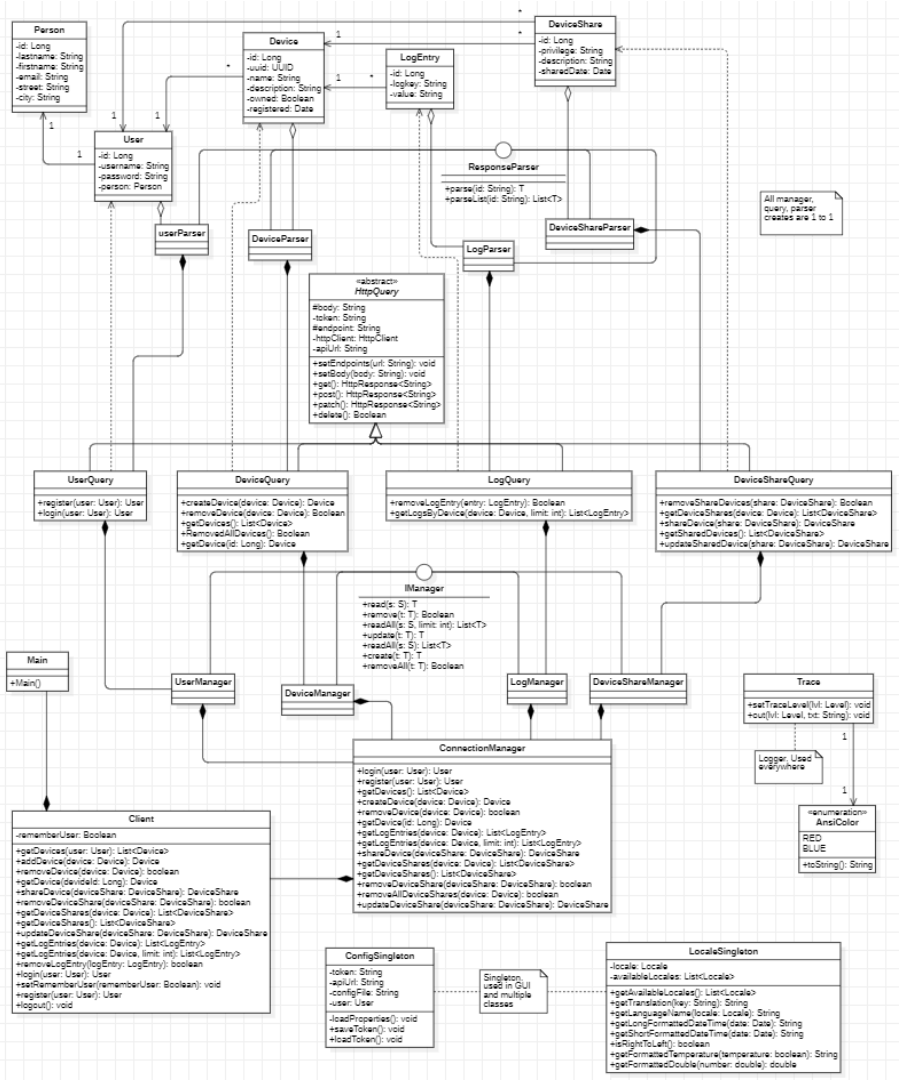
Relational schema of the project DB

2.4. UML Diagrams

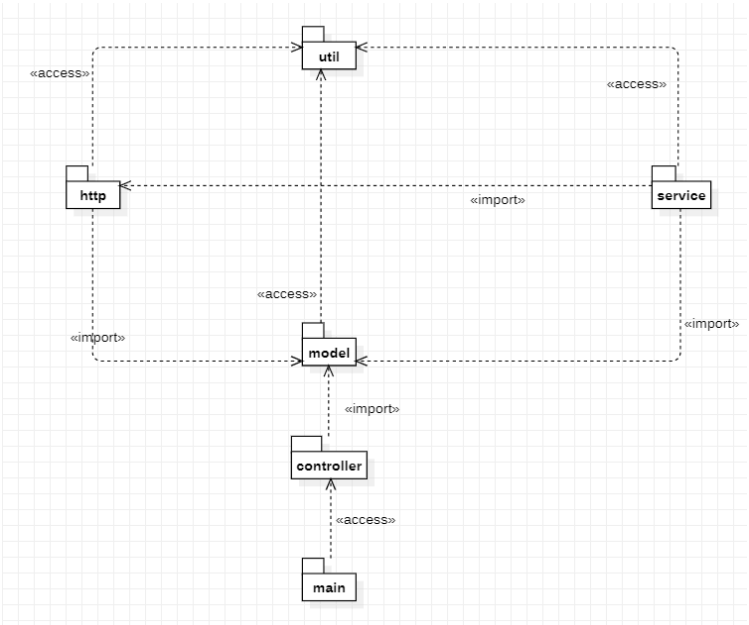
A starUML project file can be found in the project repo in the docs/diagrams folder as “i-spy-u.mdj”, there you can take a closer look at the UML diagrams provided, as some are quite large and are hard to view in this pdf.



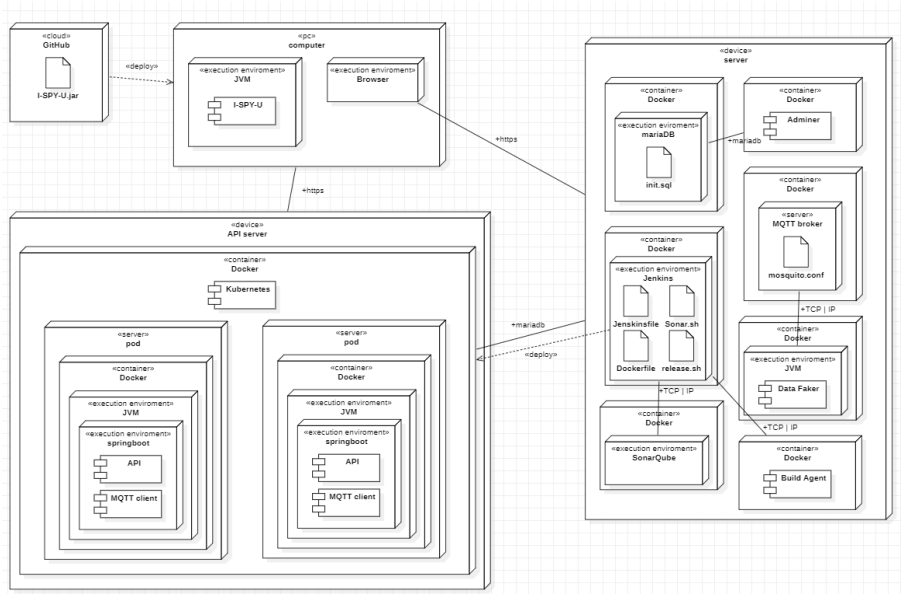
Use case diagram



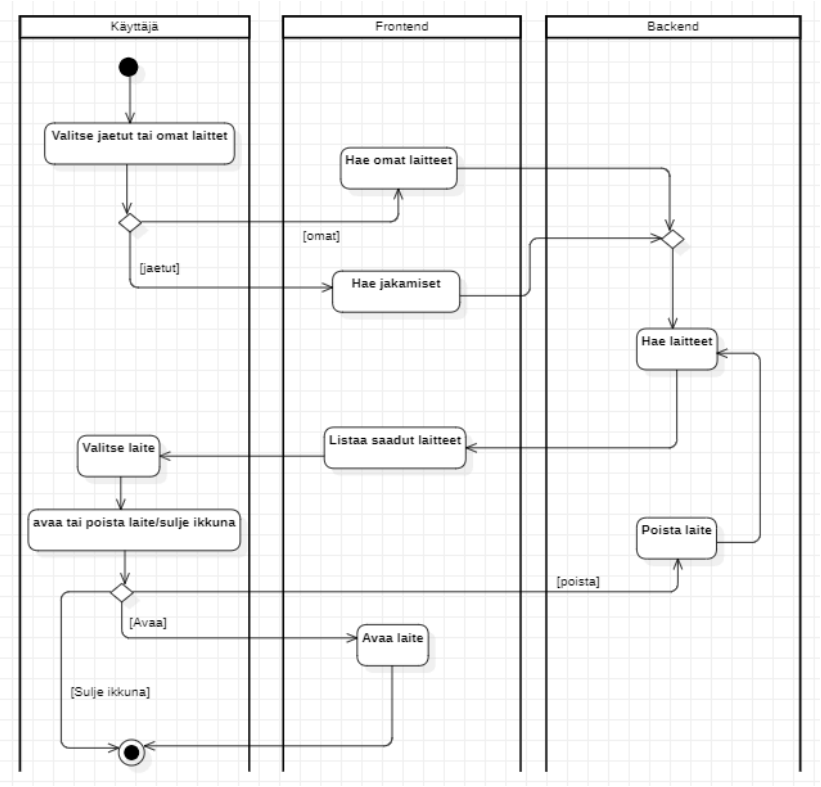
Class diagram



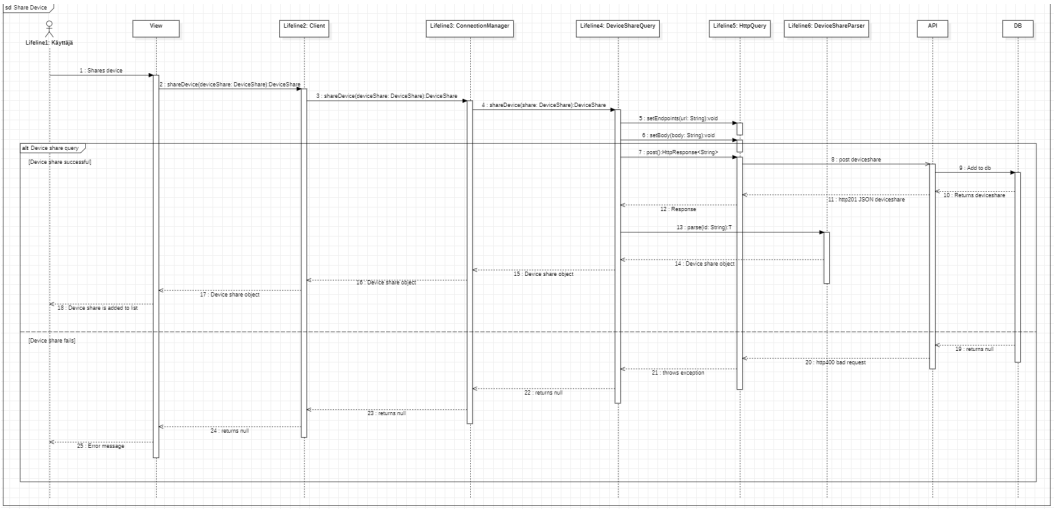
Package diagram



Development environment deployment diagram

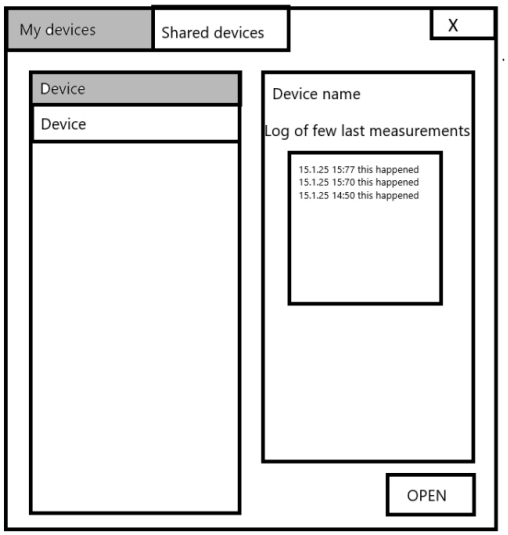


Device list activity diagram

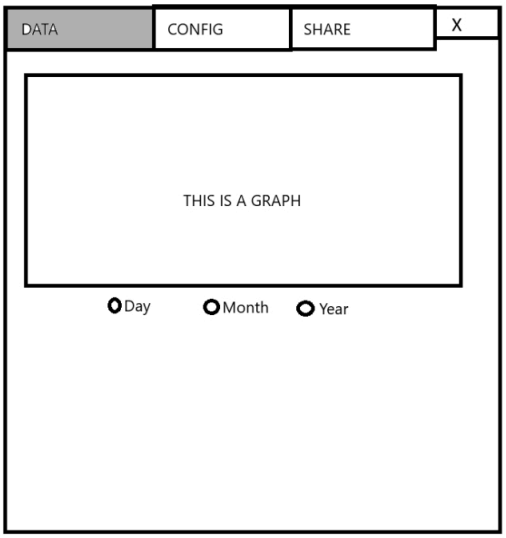


Device sharing sequence diagram

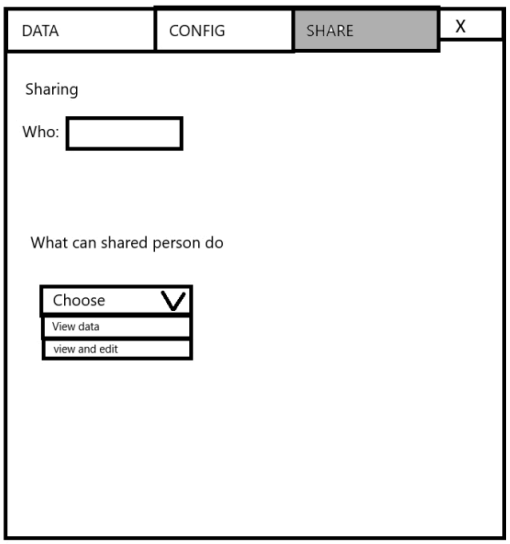
2.5. UI Mockups



This is a UI mockup of the device list



This is a UI mockup of the main device view



This is a UI mockup of the share device view

2.6. Features Overview

App features

- Manage devices
  - Users can add and remove devices, as well as view sensor logs for each device
- Share devices
  - Users can share access to their devices with other users and revoke shared access when needed.
- Language switching
  - Users can change the application's display language, and locale specific formatting, from the available options.

Software infrastructure features

- MQTT
  - MQTT protocol for lightweight and rugged communication between server and sensors. Centralized publish-subscribe messaging queue.
  - Can easily be developed to connect also with client to show whether sensor is online or offline
- Dynamic translations/localizations
  - Translations were implemented to be dynamically instantiated from translations bundle and database for easier future development
  - Adding/modifying translations requires only adding/modifying the appropriate Translations_xx-XX.properties file, all files named with this convention will be loaded on startup and compared to list of locales using right-to-left writing and those that use fahrenheit instead of celsius
- CI/CD pipeline
  - Jenkins compiles code, runs tests, builds server and client, publishes Docker images, publishes client JAR file release for Windows and Linux, and triggers deployment of server images to Kubernetes.
- Database
  - MariaDB database stores user and sensor data
  - Sensors send their data as MQTT messages to broker which then relays it to subscribed server side application that writes sensor data to the DB
- Sonarqube

        - Used in CI/CD pipeline for Statistical Non-functional Code Analysis
        - Same server also used with SonarScanner or Maven in developer IDE for live/development code analysis
- API server
    - Spring Boot server provides the REST API for client application
    - MQTT client that subscribes to sensor data, this will probably be moved to a separate micro service
- Sensor Faker
    - This is pseudo data generator/pseudo sensor to send data to MQTT broker for development and testing purposes
- Kubernetes
    - For redundancy and automated deployment

2.7. Implementation Details

Java was chosen for the project because of its strong use of object-oriented programming, and its ability to use JavaFX for its UI capabilities
The graphical interface is implemented using JavaFX, its layout and component structure is done using FXML. GUI uses FXMLLoader class to load each FXML file and they each have their own controller to interact with them.

Design patterns that the project uses are: *singleton* and *facade*.
ConnectionManager class uses the facade design pattern to simplify the use of many different manager classes into one single simplified class.
The singleton design pattern was used for config and locale. Singleton was used because the same data needs to be used in many different places, for example configSingleton was used to read and write config files and JWT token, allowing the same object to be used everywhere.
LocaleSingleton was used to get locales, get translations and handle date, unit and number formatting, and to keep track of text direction for set locale.

The project uses REST API to handle communication between the server and client components. The API is implemented using Spring Boot and follows standard HTTP conventions.

The application uses MariaDB as the relational database management system for persistent data storage. To interact with the database, the project uses Spring Data JPA as the implementation.

The project uses a MQTT broker to receive data from sensors subscribed to it and send it to the publisher which is the backend that inserts the given data into the database, Eclipse mosquito is used as the MQTT broker.

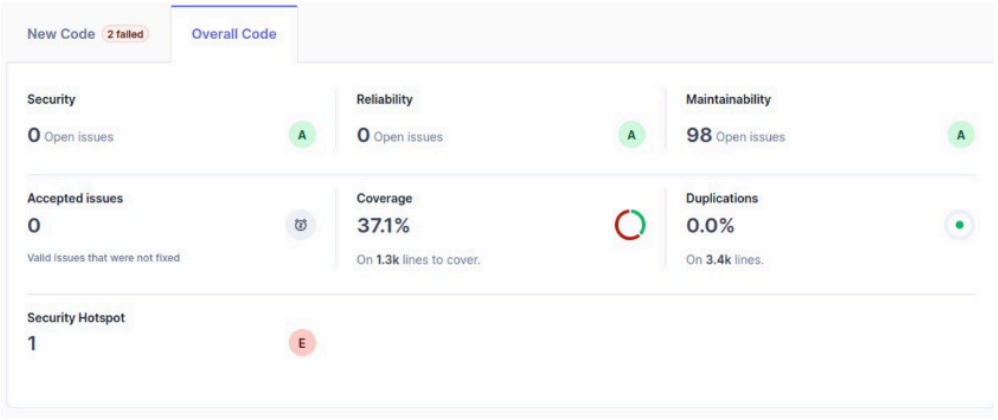
2.8. Testing Strategy and results

Unit, integration and system testing is done in CI/CD pipeline using JUnit with currently around 40% coverage as reported by JaCoCo code coverage. Statistical code analysis is done using SonarQube which also provides an API for SonarScanner to do analysis straight in the developer's IDE.

User acceptance tests were done manually by 4 people with 10 different test cases listed in the results picture below. The issues that were discovered have been noted and resulted in appropriate action.

Heuristic analysis was done by 4 people and the results were combined into 1 table.

Unit, integration and system testing was constantly being done and updated during every sprint of the project. User acceptance testing was done towards the end of the project when the product was near completion.



SonarQube sonar scanner results for whole codebase

	1. User should be able to be automatically logged in	2. User should be able to view devices data	3. User should be able to remove a device	4. User should be able to share a device to another user	5. User is able to see devices shared to them	6. User is able to edit the description of a device share	7. Remove a device share shared to you	8. Remove a share from users own device	9. User is able to create a new device with a valid UUID	10. Edit a users device share permissions
Eino	pass	pass	pass	pass	pass	pass	fail	fail	pass	fail
Severity of fail:				1		1	2	2		4
Comment of fail:				no success message is given to user and form label is wrong		no success message is given to user	no confirmation popup was shown on a destructive action	no confirmation popup was shown on a destructive action		The fields were not filled when the given share was clicked on so the user was unable to complete the test
Joonas	pass	pass	pass	fail	pass	fail	fail	fail	pass	fail
Severity of fail:				1		1	2	2		4
Comment of fail:				No confirmation dialog shown		No confirmation or other feedback given to user	No confirmation dialog shown	No confirmation dialog shown		Share cannot be selected or edited
Niko	pass	pass	fail	fail	pass	pass	fail	fail	pass	fail
Severity of fail:			4	1			2	2		4
Comment of fail:			Can't remove a device that has been shared with another user	No confirmation and wrong name in the label			No confirmation pop up is shown	No confirmation pop up is shown		Existing shares can only be removed and not edited by selecting
Toni	pass	pass	pass	fail	pass	pass	fail	fail	pass	fail
Severity of fail:				1			3	3		4
Comment of fail:				No feedback given on success			User can accidentally remove a shared device	User can accidentally remove a shared device		Device share can only be removed

User acceptance testing results from the 4 manual tests

Heuristic evaluation

No	Heuristic	Description of the Issue	Severity	Suggested Improvement
1	H1-1: Simple & natural dialog	UI could use a dark/light mode toggle	1	make light/dark mode toggle
2	H1-2: Speak the users' language	no issue		
3	H1-3: Minimize users' memory load	no issue		
4	H1-4: Consistency	no issue		
5	H1-5: Feedback	Feedback could be improved throughout the application	2	Adding feedback to user actions
6	H1-6: Clearly marked exits	no issue		
7	H1-7: Shortcuts	No shortcuts at all	1	The program is very simple, maybe shortcuts wouldn't benefit user
8	H1-8: Precise & constructive error messages	Error messages implemented but more need to be implemented	3	Make sure all errors are handled and present clear messages to users
9	H1-9: Prevent errors	Error preventing lacking, user allowed to input errroneous data	3	Add input validation to all user input fields
10	H1-10: Help and documentation	No help, tooltips or documentation	2	Add tooltips, help functionality and documentation

Combined results from 4 independent heuristic evaluations

2.9. Installation Guide

Requirements:
Client
Windows or Linux OS
Java version 8 and above with Java Runtime Environment version 21 and above

Server
Linux OS, VM/VPS or cloud platform for container deployment
Docker version 28.0 or higher
Kubernetes or cloud service if container orchestration is needed
MariaDB or compatible SQL cloud service
MQTT broker, e.g. Mosquitto
If needed then nginx for reverse proxy or port forwarding

Installation without Kubernetes:

Server

Port forwarding or reverse proxy to direct /api to api container port 8088
Make firewall rules to accept incoming connections:

- 8088 REST API, you may want to forward this to port 80
- 1883 MQTT, TLS uses port 8883 but that is not implemented yet
- 3306 MariaDB
- 8080 if Adminer panel for MariaDB is needed
- 80 if forwarding from http to REST API container

Installation steps:

1. Install MariaDB e.g. with Docker compose from project GitHub. The Docker compose automatically also installs Adminer panel
2. Install MQTT broker e.g. with Docker compose, copy configuration file from GitHub and set password
3. Install REST API image with Docker using image 0x6a4b/otp:server with port 8088 open
4. If needed install nginx with port forwarding or reverse proxy as per needs
5. Make sure all ports are opened in the firewall as needed, if all these services are installed locally only ports that need to be open are 8088(or 80 if forwarded) for REST API and 1883 for MQTT access from sensors.

Client

Installation:

Client can be used as a Java JAR application in Windows and Linux or in a Docker container with VNC or XDisplay. This allows running of the client software even on unsupported platforms remotely or locally with Docker and a VNC client. NoVNC image exists but hasn't yet been added to the build pipeline allowing the use of just a browser to access the client.

1. Download and run JAR file for the latest client build from GitHub releases

or using Docker:

1. Run the Docker image 0x6a4b/otp:client for VNC or 0x6a4b/otp:clientX for XDisplay. Docker container can reside on a remote server or cloud service
2. If using VNC image connect to the VNC container with <server address>:5900

2.10. Usage Instructions

To start using the program you need an account, registering an account needs an email, username and password.

If you already have an account then log in to that account.
Then to view a device you need to add it first, or get someone to share one of their devices with you.

To add a new device to yourself just find the "add new device" button, which will open a form to fill with the sensor's information, and once submitted will be added to the list.

Then navigate to the own devices list if you added a device or shared devices list if you were shared one and find the device you want to open, and click on it.

Then open the device which will bring you to the devices view, which will have the data from the device and the ability to share it if you own the device you opened.

If you want to share one of your devices, navigate to the share tab and fill the form, and share it.

To remove a share just look for the list in the share tab and click the red X on the share you want to remove.

To remove devices or shared devices from your devices list, find the device you want to get rid of, and select it. Then click the red remove device/share.

2.11. Troubleshooting

Client side:

If you are having connection issues make sure there is a working network connection, the port 80, or 8088 if using that for the server, isn't blocked.

The client will print out logs to the console so by running the program from the console with java -jar i-spy-u.jar you can see the program log. More logging info can be accessed when changing logging level to WARN, INFO or DEV in the app.properties located within the JAR for more detailed logging.

Server side:

If there are issues starting the API server:
Make sure the ports aren't blocked and server running the API can access those:
- to access MariaDB you need to be able to access port 3306

Make sure the DB has the correct user and password set also in the API configuration.

If there are issues with getting sensor data:
- to access MQTT you need to be able to access port 1883
Make sure the MQTT configuration is set and credentials are set and correct in the MQTT client
You can troubleshoot with Eclipse Mosquitto cli programs: mosquitto_pub and mosquitto_sub to make sure communication and connection with the broker works.

If there are issues with connections:
- to access the API you need to be able to access port 8088 and if you are forwarding from 80 you need to be able to access that port from client machine
Make sure the API is running. You can view API server's logs with docker logs <containername>