Slovak University of Technology

Faculty of Informatics and Information Technology

Ilkovičova 2, 842 16 Bratislava 4

Team project

LOMON

Documentation for engineering work Academic year: 2021/2022

Team supervisor: Ing. Valach Alexander

Team members (team n. 9):

Bc. Dominik Bucko
Bc. Ján Balucha
Bc. Michal Franczel
Bc. Peter Mačuga
Bc. Patrik Villant
Bc. Soňa Zhan

List of abbreviations

ADC - Analog-to-Digital Converter
AES - Advanced Encryption Standard
ALU - Arithmetic Logic Unit
API - Application Programming Interface
EEPROM - Electrically Erasable Programmable Read-Only Memory
DAC - Digital-to-Analog Converter
DES - Data Encryption Standard
GPIO - General Purpose Input/Output
GPR - General Purpose Register
GPRS - General Packet Radio Service
GPS - Global Positioning System
GSM - Global System for Mobile Communications
I2C - Inter-Integrated Circuit
MCU - Microcontroller Unit
MIPS - Million Instructions per Second
PDI - Program and Debug Interface
PLL - Phase-Locked Loop
PWM - Pulse-Width Modulation
RTC - Real-Time Clock
SPI - Serial Peripheral Interface
SPM - Store Program Memory
SRAM - Static Random Access Memory
TWI - Two-Wire Interface
UART - Universal Asynchronous Receiver-Transmitter
UI - User Interface

Obsah

1. Introduction

This document contains technical documentation for the LOMON project. Aim of this project is to design and implement a system for gathering sensor data from vast areas using LoRa technology, where sufficient coverage by traditional networks is impossible or impractical.
In the document, we describe the problem we are solving, outline specifications for the project and its outputs and propose solutions.
This project is part of the Team Project course that takes place across 2 semesters of the academic year 2021/2022.

1.1. Global goals of the project

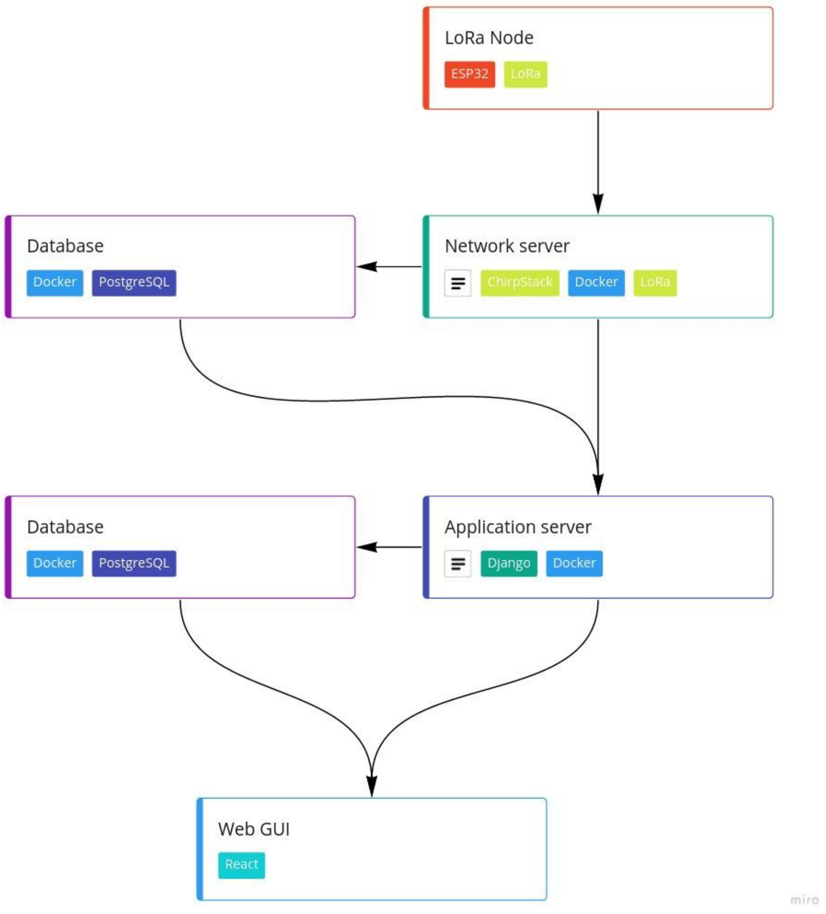
The main goal of the project is to design and implement a system for gathering sensor data from large areas. We will design and prototype an end device, which will be able to measure environmental conditions and transmit the data using LoRa. We will also design a process to update said devices using over-the-air updates. We will also design and implement an application server, which will be able to gather and interpret data sent by devices and a corresponding graphical interface accessible via browser. There users would be able to observe data sent from devices, add and remove devices, categorize them and deploy over-the-air updates. For this application, we need to create frontend and backend parts, along with databases and interfaces which will communicate with the network server.

1.1.1. Goals for 1st semester

There are multiple goals we set out to fulfill in the first semester. First one is to get familiar with LoRaWAN networks and embedded devices, analyze possible options in hardware and create a prototype of the end device including sensors and test them. We will also design a user interface, perform usability testing on wireframes and implement the frontend part of the application. On top of that, we will design a database and backend application, which we will also implement.

2. System overview

2.1. System Architecture



2.2. Database model

A database is a collection of structured data stored in a computer system. Access to this data is in our system provided by Django ORM. Main purpose of using ORM is to eliminate SQL injection attacks. On the images below are physical database models generated from PyCharm.
Main pillar of the database is table Core_devices (devices). Every device has its location to divide devices into logical groups. Sensor and its measurements are mapped to devices to capture measurements like battery level, temperature, humidity, etc.. Table Core_distribution is for storing firmware distributions along with information such as signature, description and firmware version. To handle OTA updates, there is a table named Core_rollout and junction

table Core_rollout_devices to assign devices and store status information about update progress. To simplify listing of devices and its software version, there is a redundant relationship to table Core_distribution to speed up fetching firmware versions. Commands (Core_command) have their command text, description and they will be assigned to the device through tag. Every device would have one unique tag which cannot be deleted or assigned to other devices and multiple non-unique tags. Users and privileges are maintained by Django (Figure 5). For logging there is table Core_log.

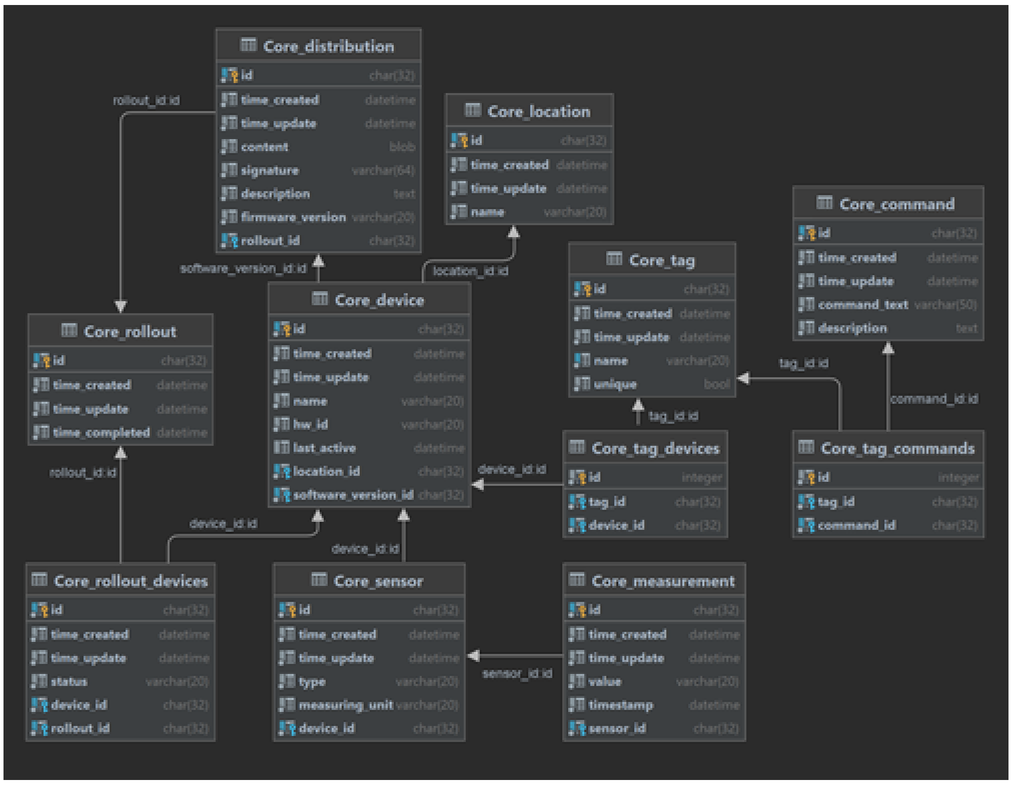


Figure 1 Core database schema

The figures below (Figure 2 – Figure 4) shows the individual blocks of the database for easier understanding.

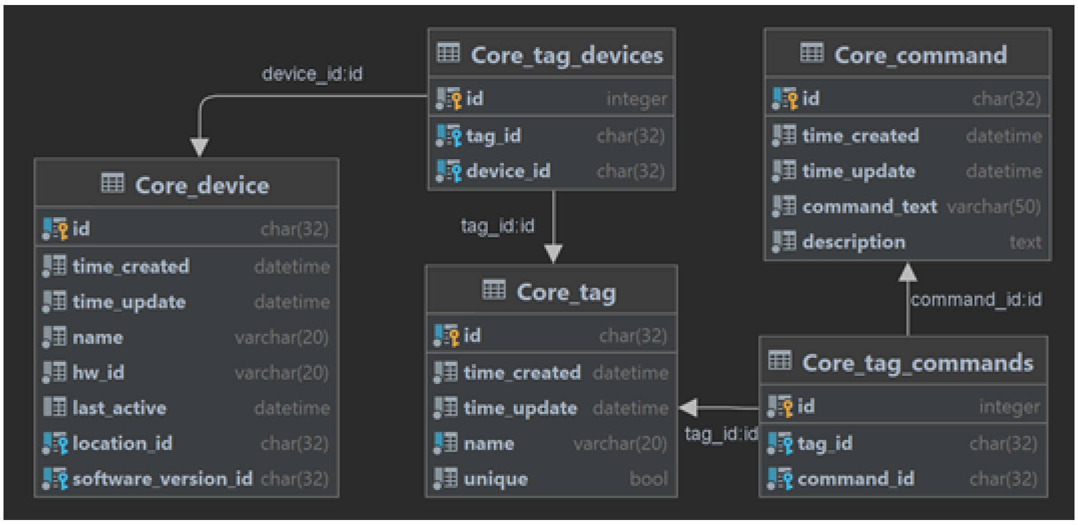


Figure 2 Command schema

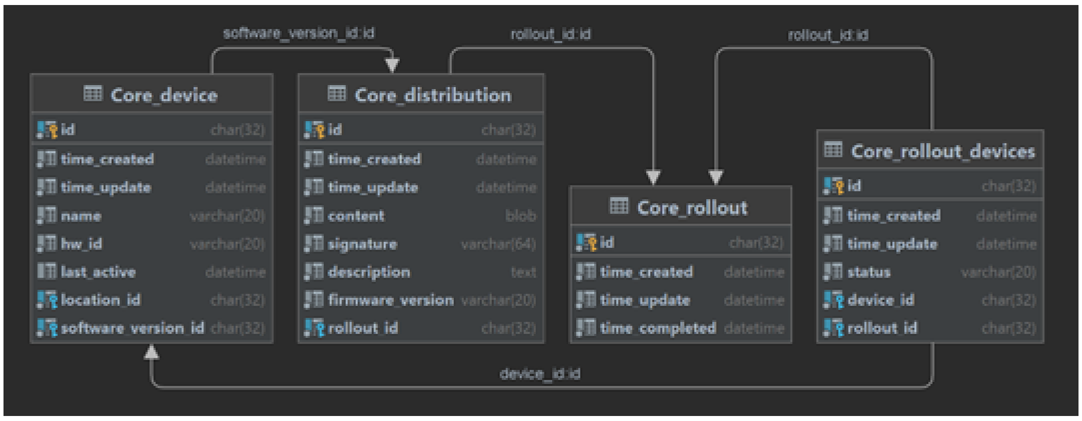


Figure 3 Firmware update schema

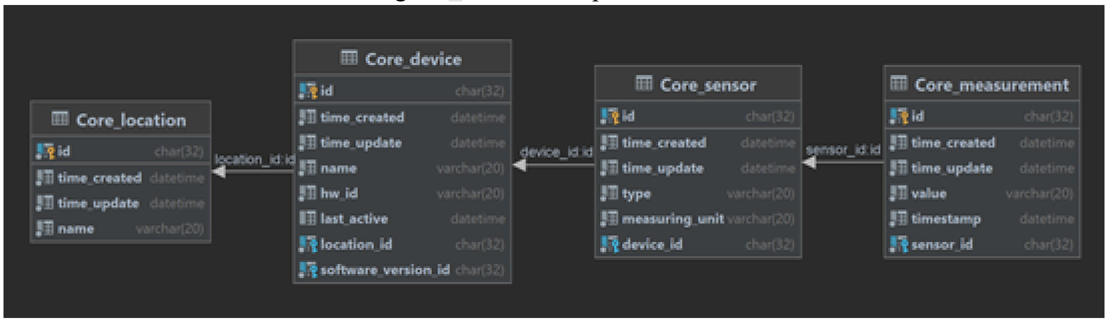


Figure 4 Device and measurement schema

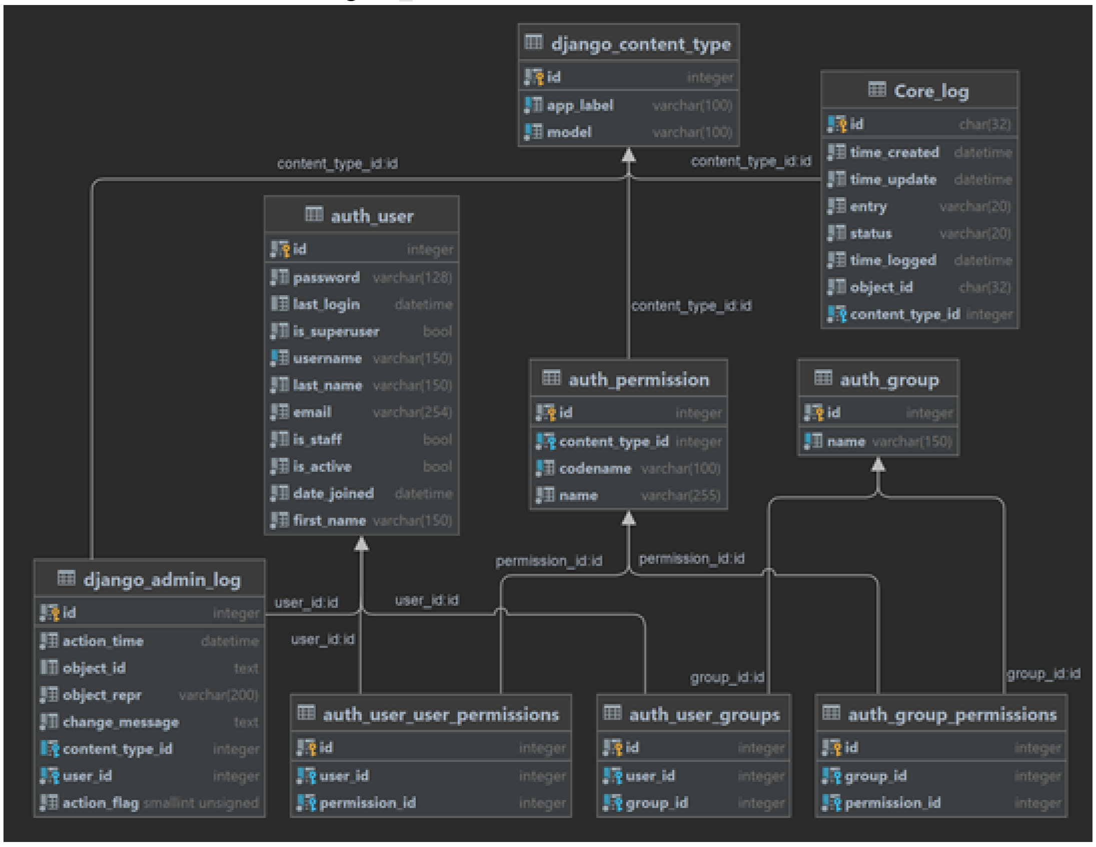


Figure 5 Django authentication schema

2.3. Sensor nodes

2.3.1. Hardware

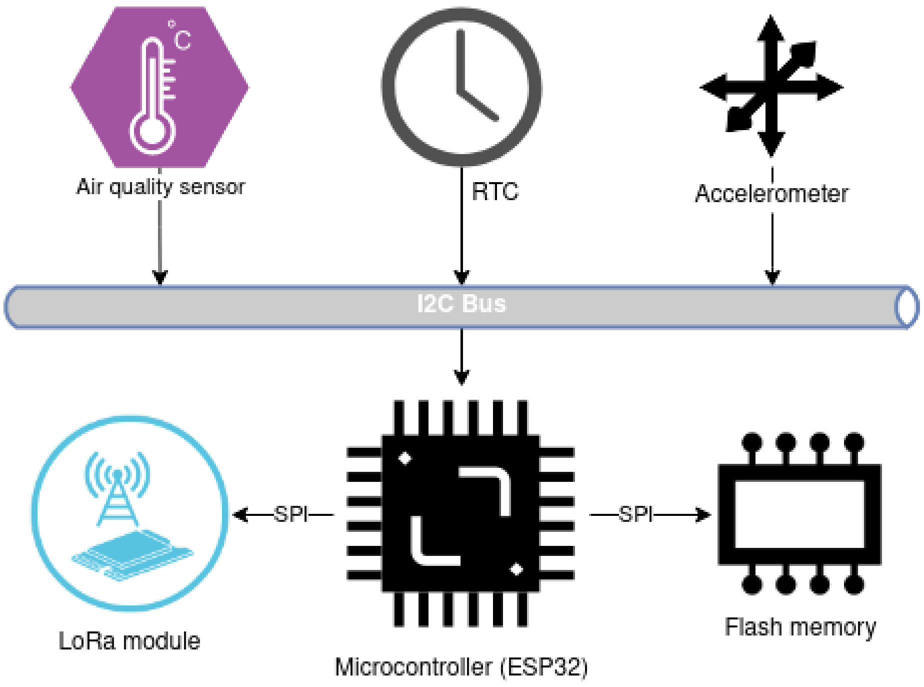


Figure 6 Sensor node – high-level view

2.3.1.1. Microcontroller

Our sensor nodes are based on ESP32, a 32-bit microcontroller developed by Espressif. It is commonly used in embedded devices and low power IoT solutions, due to many features packaged into a single chip. We chose ESP32 over 8-bit AVR microcontrollers due to its higher computational capability and fewer hardware constraints for development. Most importantly, ESP32 includes hardware acceleration for cryptographic use cases, making encryption more energy efficient. This is important, because all network traffic, transmitted and received, needs to be encrypted. We can also use flash encryption and make our devices more secure.

ESP32 contains two Xtensa® 32-bit LX6 cores clocked at 240 MHz. It has 448KB of ROM and 520KB of SRAM, built in Wi-Fi (802.11 b/g/n) and Bluetooth (4.2, low energy support).

There are multiple peripheral interfaces. These include:

- 34 × programmable GPIOs
- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit DAC
- 10 × touch sensors
- 4 × SPI
- 2 × I2S
- 2 × I2C
- 3 × UART
- Ethernet MAC interface with dedicated DMA and IEEE 1588 support
- Motor PWM
- LED PWM up to 16 channels

ESP32 also includes coprocessors for cryptographic acceleration. Supported algorithms are:

- AES
- SHA-2
- RSA
- ECC

There is an option to enable secure boot and flash encryption.

There are multiple power modes that the device can switch between to reduce power draw and prolong battery life. The ones relevant to our use-case (we exclude mode with Wi-Fi and Bluetooth active, because are not using them in this project) include:

- **Active** (Modem-sleep) - CPU is active, power usage is dependent on the clock which we can set in the firmware. 3 clock speeds are supported: 240MHz – 30~68 mA, 160MHz – 27~44mA, 80MHz – 20~31mA.
- **Light sleep** – 0.8mA
- **Deep sleep** – Ultra-Low-Power coprocessor is active, 150 µA
- **Hibernation** – only built-in RTC is active. RTC timer + RTC Memory - 10 µA, RTC timer only – 5 µA.

2.3.1.2. LoRa module

LoRa (abbreviation of "long range") is a proprietary radio modulation technique designed and patented by Semtech. It found its use in IoT applications, due to the longer range and lower energy consumption than traditional wireless networks, such as Wi-Fi. The most prominent network protocol built on the LoRa physical layer is LoRaWAN, a wide area network connecting LoRa enabled devices to the internet or other networks via network server using packet concentrators/gateways.

The module we are using in our devices is Semtech SX1276. As LoRa is a proprietary technology developed by Semtech, there are no competing devices on the market, so when using this technology we have to use the chips they provide.

Our chip has following parameters:

Supply voltage	1.8 ~ 3.6V
Frequency band (EU)	433MHz or 863MHz
Communication interface	SPI
Power modes	Sleep, Standby, Receive, Transmit
Power consumption in Sleep mode	0.2~1 µA
Power consumption in Standby mode	1.6~1.8 mA

Power consumption in Receive mode	10.8~12 mA
Power consumption in Transmit mode	20-120 mA, based on transmit power

LoRa allows us to set different transmission parameters, on top of which LoRaWAN creates an abstraction. Supported data rates are shown in following table:

Data Rate	**Modulation**	**Spreading factor**	**Bandwidth**	**bit/s**
0	LoRa	12	125	250
1	LoRa	11	125	440
2	LoRa	10	125	980
3	LoRa	9	125	1760
4	LoRa	8	125	3125
5	LoRa	7	125	5470
6	LoRa	7	250	11000
7	FSK 50 kbps			50000

2.3.1.3. Sensors

On our device, we are using sensors that can measure conditions of the environment it is located in. Specifically, we are measuring temperature, humidity, atmospheric pressure, volatile organic compounds (VOC) and shocks/vibrations. There are multiple sensors to choose from and when choosing which ones to use in our devices, we were looking at their availability, firmware support in the form of available libraries for our platform and interfaces which are available for connection to our device.

For measuring temperature, humidity, atmospheric pressure and VOC we have chosen the sensor **BME680**, manufactured by Bosch. There were no available alternatives to this sensor which would be able to measure all of the abovementioned environmental conditions. Sensor BME260 is similarly capable but lacks capability of VOC measurement. We could add an SGP30 sensor for VOC measurement, but if we want to keep our power consumption as low as possible, using the smallest number of available sensors makes more sense. Cumulative power consumption of multiple sensors would be higher than that of a single sensor and adding more sensors would introduce a need to execute more measurements by microcontroller, requiring more processor cycles and therefore longer active time, also resulting in higher power consumption. BME680 was therefore our natural choice.

To measure shocks and vibrations, there are multiple options. The first one is the MPU6500, which contains a 3-axis accelerometer and a 3-axis gyroscope. It can measure acceleration up to 16G, rotational speed of up to 2000 deg/s. The second sensor is **MMA8452**, which is simpler and less capable, containing only an accelerometer (up to 8G of accel.) and no gyroscope. In terms of power consumption, MMA8452 uses less power (6 µA) at the lowest power mode (12.5 Hz measuring frequency) than MPU6500 (7.3 µA at 0.98 Hz). It is also more efficient at higher frequencies, although we cannot make direct comparison as datasheets specify different values. For example MPU6500 consumes 18.65 µA at 31.25 Hz update rate while MMA8452 uses 14 µA at 50 Hz update rate.

Crucially, MMA8452 can send interrupts via specified interrupt pins when detecting acceleration above user specified threshold, making devices able to detect movement or abnormal shocks/vibrations without need for constant polling of the device for measurements, saving a significant amount of power. This is the primary reason why we chose MMA8452 to use in our devices.

Both selected sensors - BME680 and MMA8452 communicate with microcontrollers via I2C interface, which supports multiple devices on the same bus and addresses them using 16-bit addresses.

2.3.1.4. Development board

For development, we chose Liligo LORA-32 board, which features:

- ESP32-PICO-D4 microcontroller unit
- SX1276 LoRa module
- 4MB of QSPI Flash
- CP2102 USB-to-UART interface for flashing
- Support for 3.7V LiPo batteries (contains charging circuit)
- Built-in antenna for Lo-Ra module
- 0.96" SPI OLED display (not used due to high power consumption.

2.4. Firmware development platform

We will be using Platformio as our development platform. We chose Platformio due to the possibility of integrating it into editors such as VSCode or CLion and because it supports a wide range of development boards with preconfigured profiles for fast development. We are also able to search, download and include proper libraries into our code and we can compile and upload our firmware directly from our editor.

2.5. Containerization

Containerization provides a fast, easy and portable method for creating scalable infrastructure. It helps developers by making their projects transferable and more easily deployable via CI/CD pipelines. Like the other parts of this project, the web application is also dockerized, using node and nginx images.

.

2.6. Backend

We decided to use python as our backend programming language. Python has 2 prominent backend frameworks, Flask and Django. Although very similar, there are some key differences between the two.

2.6.1. Comparison between python backend frameworks:

	Django	Flask
ORM	Django ORM	SQLAlchemy
Setup	Preconfigured admin, easy database access, rigid project structure	Bare structure, available packages, less rigid project structure
Admin	Predefined	No predefined admin site
Authentication	Predefined, can be changed	Have to define yourself

In the end we choose Django, because of its easy setup, and convenience regarding predefined functionalities. Our team also has more experience with Django in larger projects.

2.7. API Specification

For API specification we used swagger and for testing our endpoints we use Postman. Basic definitions of our specification are shown in tables below.

Table 1 Device specification

Devices	
GET	/devices/
POST	/devices/
GET	/devices/{device_id}/
PUT	/devices/{device_id}/
DELETE	/devices/{device_id}/
GET	/devices/{device_id}/graph/

Table 2 Distribution specification

Distributions	
GET	/distributions/
POST	/distributions/
GET	/distributions/{dist_id}/
DELETE	/distributions/{dist_id}/

PUT	/distributions/{dist_id}/

Table 3 Tag specification

Tags	
GET	/tags/
POST	/tags/
DELETE	/tags/{tag_id}/
PUT	/tags/{tag_id}/
POST	/tags/device/
DELETE	/tags/device/

Table 4 Location specification

Location	
GET	/locations

Table 5 Command specification

Commands	
GET	/commands/
POST	/commands /
GET	/commands /{command_id}/
DELETE	/commands /{command_id }/
PUT	/commands /{command_id }/

2.8. Web application

There is a plethora of frontend frameworks that can be used for web development, most popular being ReactJS, Angular and Vue.js. All of them use either Javascript or Typescript, latter being statically typed version of former and all of aforementioned frameworks have advantages and disadvantages and are used based on specific requirements. In this project, we have decided to use the framework React combined with Tailwind CSS.

2.8.1. Comparison of ReactJS with other frameworks

- Angular.js

Pros:	**Cons:**
Detailed documentation	Complex syntax

RXJS	Migration issues
Faster compilation	
MVVM 1 that allows developers to work separately	

- Vue.js

Pros:	**Cons:**
Similar characteristics with Angular	Lack of resources
Detailed documentation	Risk of over flexibility
Good integration	
Large scaling	

- React.js

Pros:	**Cons:**
Popularity, huge community	Incompleteness – it is more of a library than a framework, in a sense that it has to be used in combination with other libraries to provide full MVC capabilities
Hot reloading	Poor documentation
Easy to learn and use, with fast development time	
Actively developed by one of the world's largest corporations	
Scalable	
JSX – declarative programming	
Reusable components	

- Tailwind CSS
  - Style component without adding a new style sheet for every new component
  - Better integration
  - Improves maintainability
  - Consistent UI and customizability
- Typescript
  - Strict typing
  - Early spotted bugs

    - Predictability
    - Readability
    - Fast refactoring
    - OOP (classes, interfaces, types, enums, inheritance ...)

2.8.2. Other libraries

- React-router-dom
    - Dynamic routing in a web application
    - Enables the navigation among views of various components in a React application

- Framer-motion
    - Popular animation library
    - Open source
- Nivo
    - Wide variety of data visualization with ease

- React redux
    - Great for managing the state of application
    - Ensure the components behave as expected
    - Good React app architecture

- Fontawesome
    - Scalable vector images
    - Can be customized with CSS
    - Over 1,600 icons in the free set

2.9. Application Design

Our design was created in the Figma environment, where we defined the individual subpages of the web application.

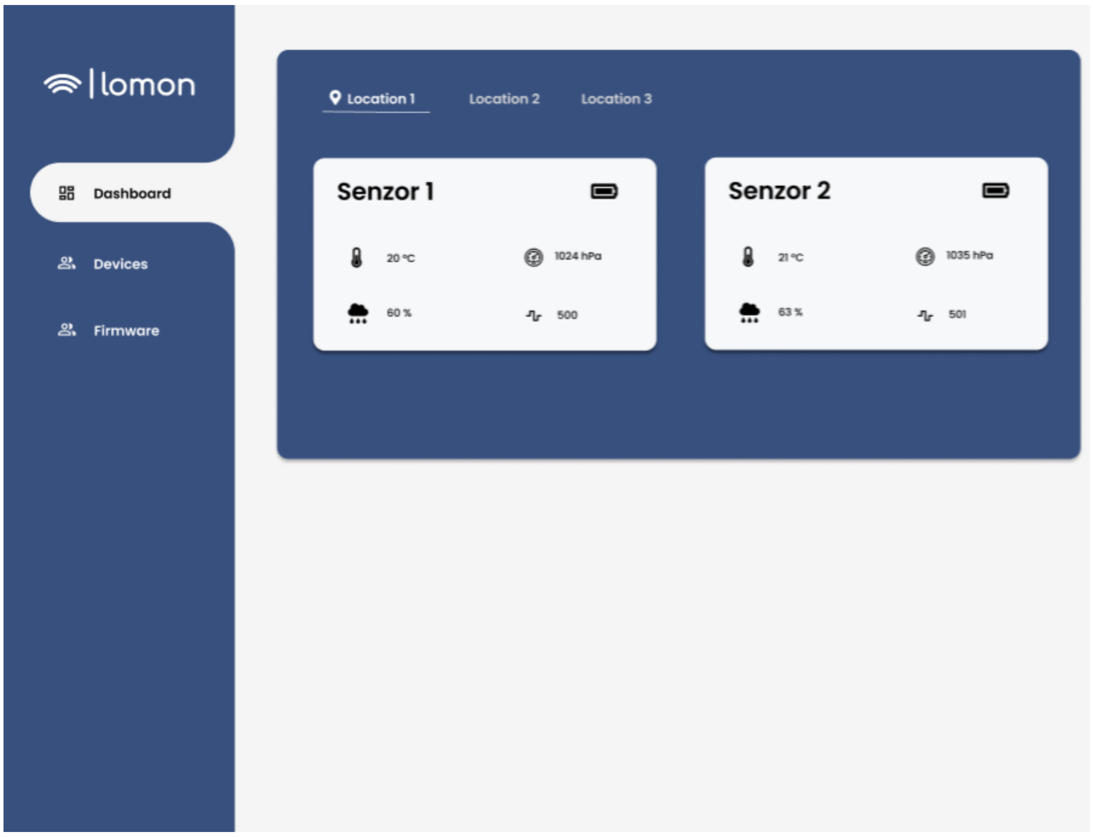
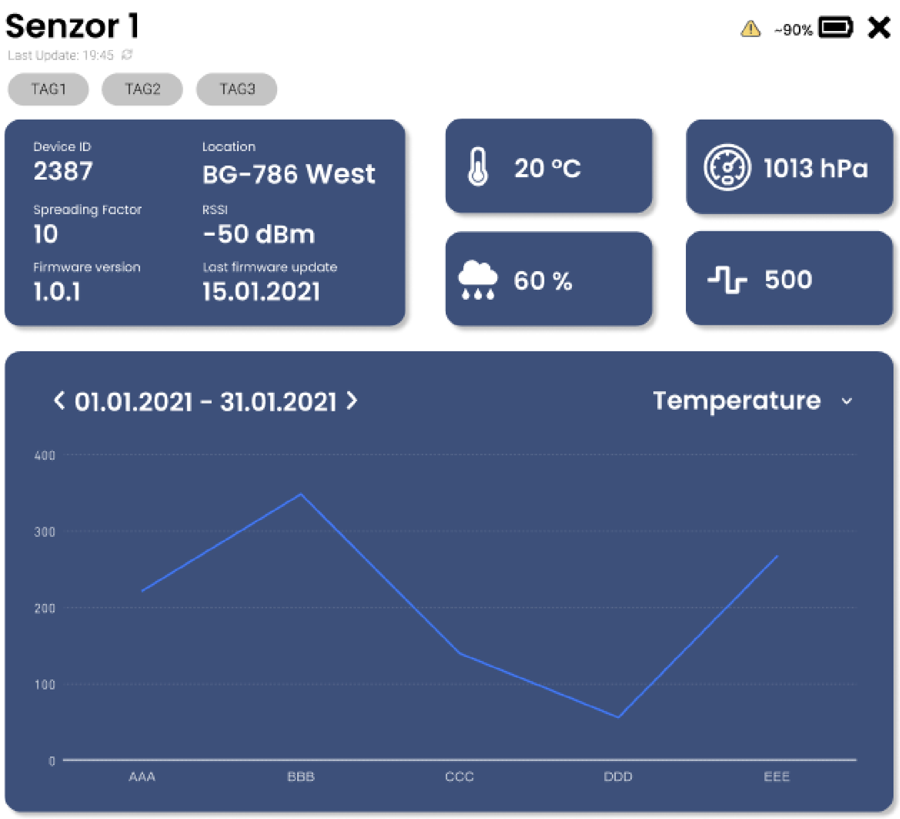


Figure 7 Dashboard subpage



Device ID
2387

Location
BG-786 West

Spreading Factor
10

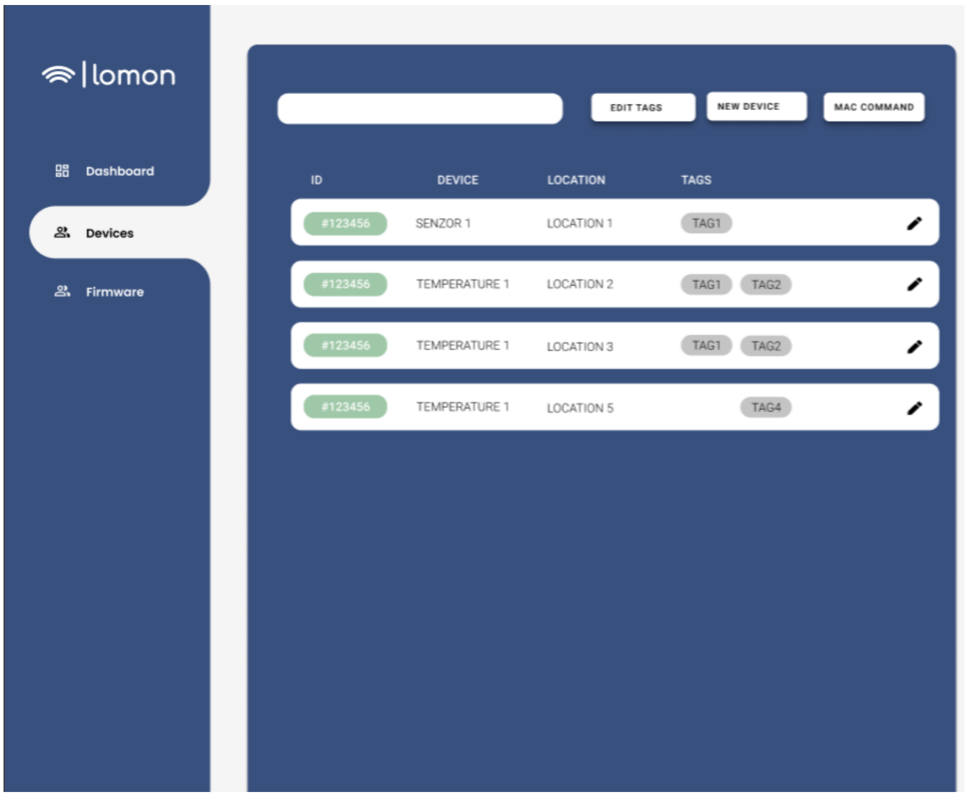
RSSI
-50 dBm

Firmware version
1.0.1

Last firmware update
15.01.2021

Figure 8 Device Details Modal

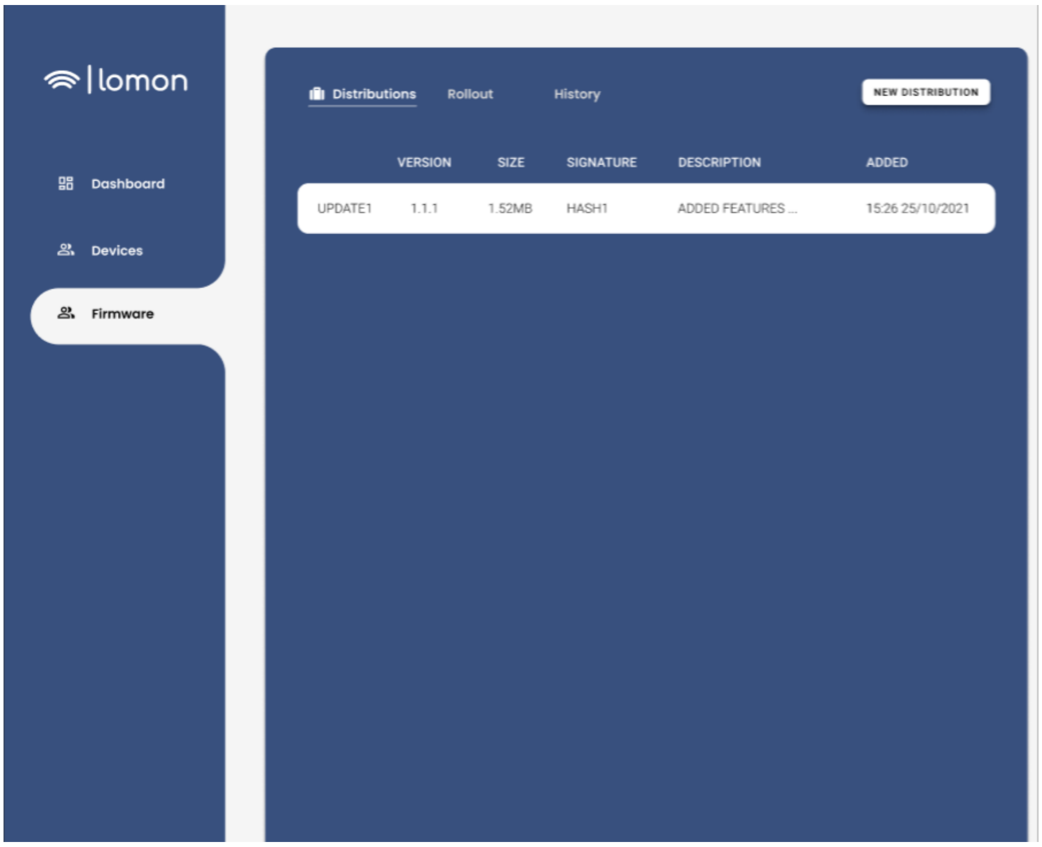
At the beginning of the application, after the user logs in, there is a dashboard with the devices in the individual locations. After clicking on one of the devices, a modal window will appear with a detailed description of the device and its measured values.



ID	DEVICE	LOCATION	TAGS
#123456	SENZOR 1	LOCATION 1	TAG1
#123456	TEMPERATURE 1	LOCATION 2	TAG1 TAG2
#123456	TEMPERATURE 1	LOCATION 3	TAG1 TAG2
#123456	TEMPERATURE 1	LOCATION 5	TAG4

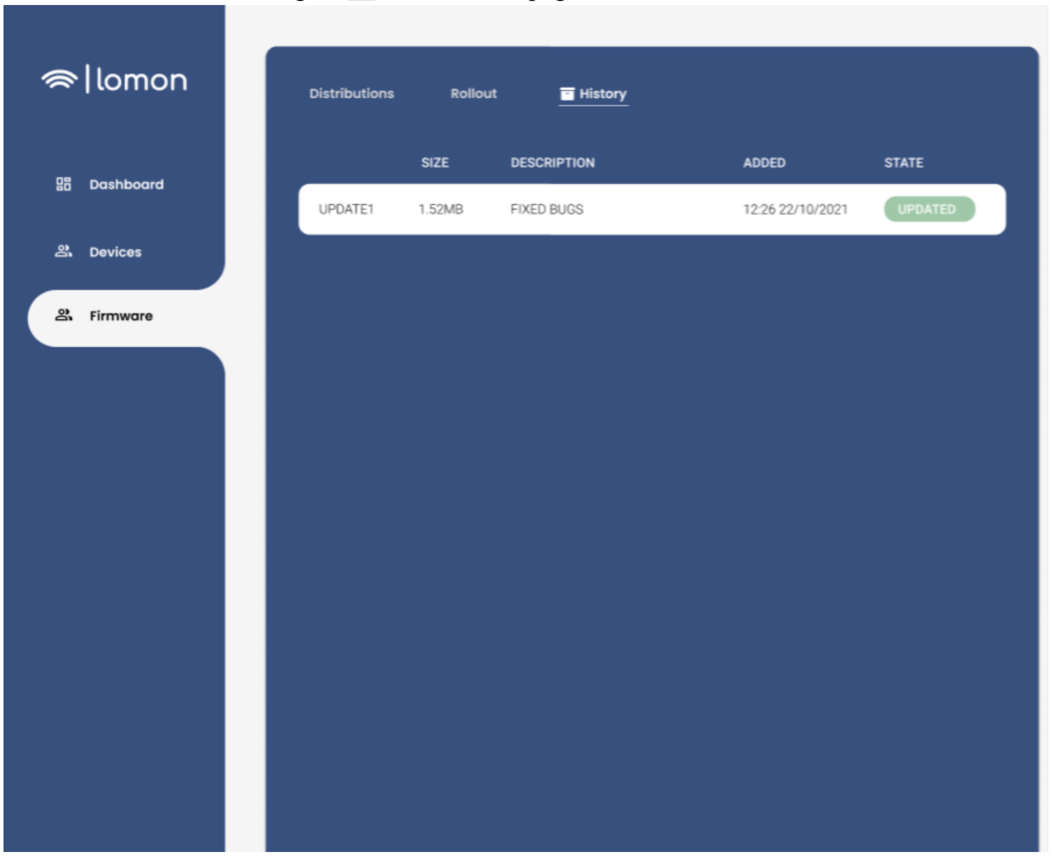
Figure 9 Devices subpage

The Devices subpage allows the user to search for devices, edit tags, add a new device, edit existing devices, and it is also possible to send a MAC command to devices using tags. A corresponding modal window appears for each selected option.



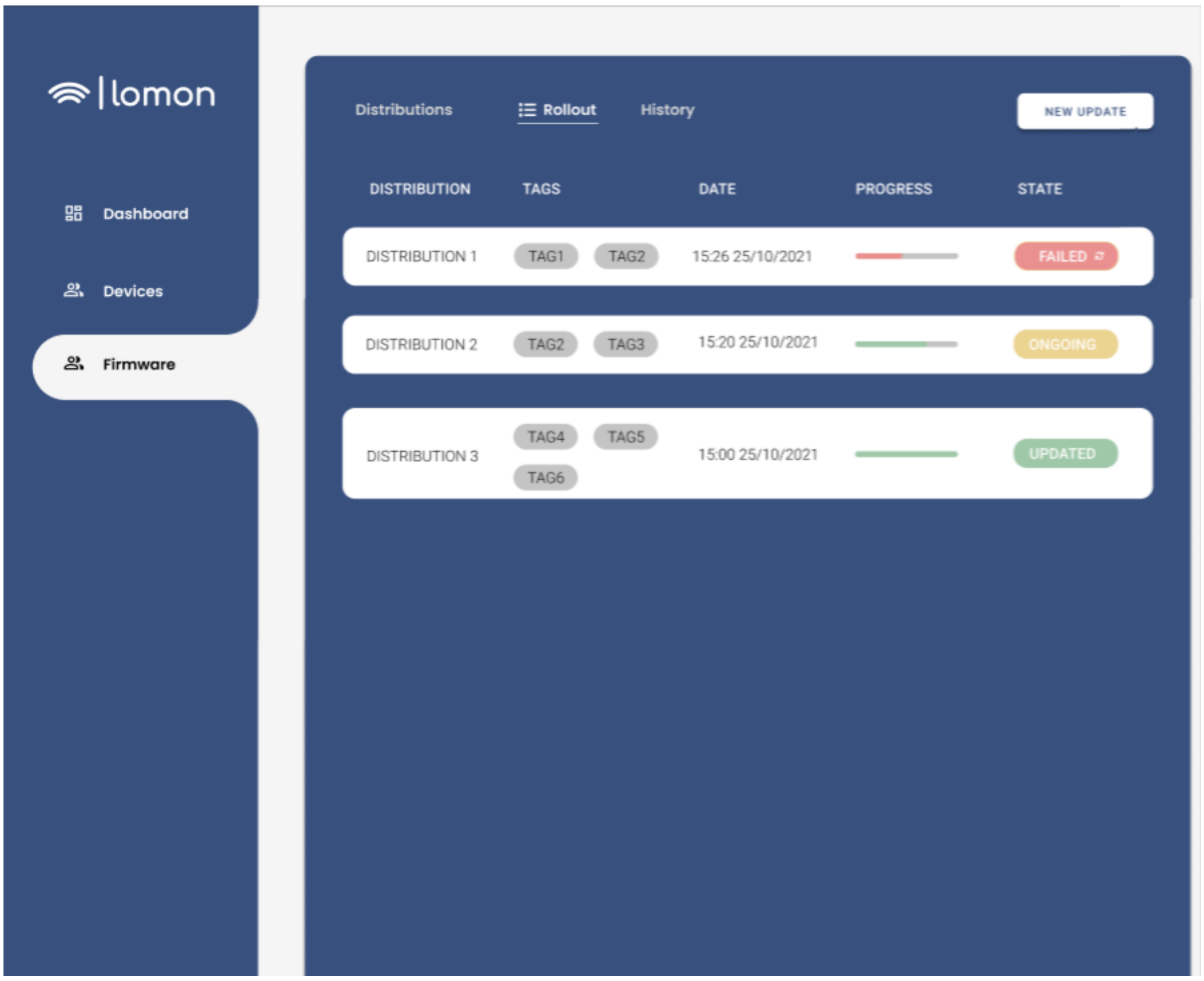
	VERSION	SIZE	SIGNATURE	DESCRIPTION	ADDED
UPDATE1	1.1.1	1.52MB	HASH1	ADDED FEATURES ...	15:26 25/10/2021

Figure 10 Firmware subpage - Distributions tab



	SIZE	DESCRIPTION	ADDED	STATE
UPDATE1	1.52MB	FIXED BUGS	12:26 22/10/2021	UPDATED

Figure 11 Firmware subpage - History tab



DISTRIBUTION	TAGS	DATE	PROGRESS	STATE
DISTRIBUTION 1	TAG1 TAG2	15:26 25/10/2021		FAILED
DISTRIBUTION 2	TAG2 TAG3	15:20 25/10/2021		ONGOING
DISTRIBUTION 3	TAG4 TAG5 TAG6	15:00 25/10/2021		UPDATED

Figure 12 Firmware subpage - Rollout tab

There are three tabs on the Firmware subpage, namely: Distributions, Rollout and History. On the Distributions tab we can observe all uploaded distributions and it is also possible to upload another one, where the modal window for that purpose will appear. The history tab shows the history of updates that have been made in the past. Finally, on the Rollout tab, we can track updates and their statuses: failed, updated, or ongoing. We can also define a new update here, which will be displayed in the modal window. On this table there is a possibility to click on individual updates for their detailed display in the modal window.

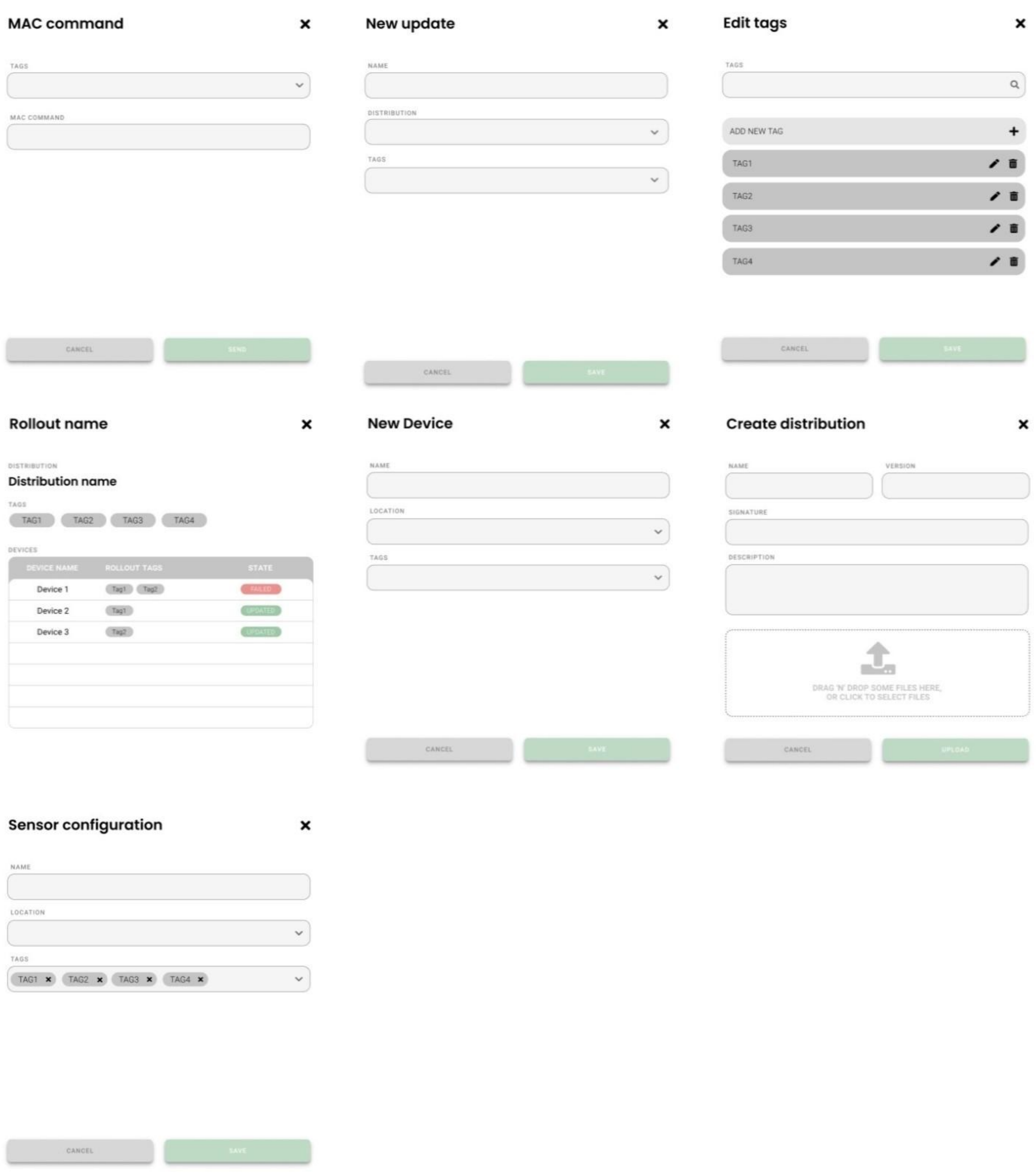


Figure 13 All available modal windows in our web application

The figure above (Figure 13) shows all available modal windows for our application. Modal windows for creating a MAC command, for defining a new update, editing tags, detailed display of the update, creating a new device or distribution, and also for configuring an existing device.

2.10. UX testing

To improve the user experience on our application, we have created UX testing on our design. We used the UXTweak website for testing, where it is possible to upload a prototype from the Figma environment to this web application. The site is used to test web interfaces and prototypes and monitors all aspects of the user who tests the prototype or page. To create

UX testing, we created 5 test scenarios, the aim of which was for the user to go through all the possibilities of the given web application during testing.

1. Scenario

The user's task is to find out information about the end device called "Sensor 1" (spreading factor, RSSI) and its measured values.

2. Scenario

The user's task is to find devices that have a tag named "Tag1" assigned to them and then add a new tag with name "New Tag" to the tag list. Another task will be to modify the settings of the end device named "Sensor 1", where the user must change the name of this end device to "Sensor One", the location to "Location 2" and add a newly created tag named "New Tag" to this device. Finally sends any MAC command to devices that are tagged with a "New Tag".

3. Scenario

The user's task is to add a new end device to the list of end devices. This newly created device will be named "Device New" and the location will be "Location 3". The device will be assigned two tags: "Tag1" and "Tag2".

4. Scenario

The user's task is to examine the existing distribution, especially it’s version number and description. Then create a new distribution called "Update 2", the version name will be "1.1.2", the signature will be "hash1" and the distribution description will contain "New features ...". Also upload the binary file either by using drag and drop functionality or by clicking on the upload location. Another task of the user is to view the history of performed updates and find out their description and time of creation.

5. Scenario

The user's task is to create a new update for a defined tag. In the form created for this, select the distribution "Update 2" and the tags to which the update will apply will be "Tag1" and "Tag2". This update will be called "Thing". Another task is to review the status of other updates, where it must determine the reason for the failure of a particular update.