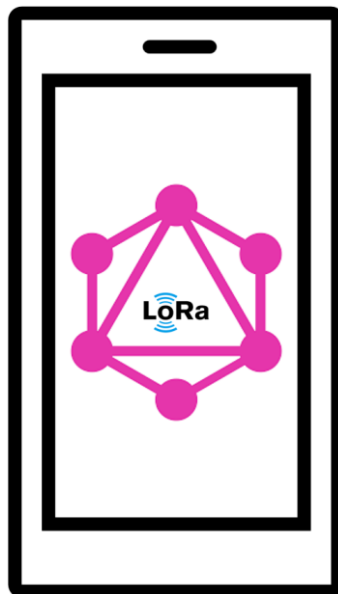


FM4017 Project 2023

Development of Mobile Application for Integration of LoRaWAN Sensors and Infrastructure



MP-14-23

Course: FM4017 Project, 2023

Title: Development of Mobile Application for Integration of LoRaWAN Sensors and Infrastructure

This report forms part of the basis for assessing the student's performance in the course.

Project group: MP-14-23

Group participants: Saba Homayounibaghbidi
Jan-Robin Brustad
Even Tviberg Hope

Supervisor: Hans-Petter Halvorsen

Project partner: Altibox

Summary:

Internet of Things opens the doorway to the future of smart applications. Seeing this as a potential future market segment, Altibox AS has in the recent five years established an IoT network in Norway.

The objective of this project was to utilize this network for connecting LoRaWAN sensors and using this infrastructure for routing of sensor data to a storage solution. This data should then be monitored by a mobile application, developed by the project team.

The methods chapters explore methodology on how to solve this objective. Here GIT, APIs, Cloud databases and mobile operating systems is described, in addition to the Altibox infrastructure, LoRaWAN and the available sensors.

Further on the requirements for the software application are gathered, documented in its own chapter together with a suggestion of GUI.

In conclusion, the completed system delivers a functional mobile application capable of collecting and presenting measurements from LoRaWAN sensors.

Preface

The project originated from the necessity to monitor LoRaWAN sensors remotely. The primary goal was to create a user-friendly mobile application capable of displaying real-time and historical sensor data. A key component to the system is a cloud database to ensure efficient storage of sensor data.

This report adheres to the IMRaD structure and explores topics such as the LoRaWAN protocol, Altibox ThingPark Wireless and ThingPark X, Git version control, GraphQL query language, Dimension Four cloud database, and Android application development. Each of these components have been important in achieving the primary objective of creating an intuitive, responsive, and visually appealing mobile application.

The main outcome of this project is the Android app, utilizing GraphQL API for seamless cloud communication. Emphasizing user experience, we've employed modern design tools to ensure an intuitive interface.

We would like to thank Hans-Petter Halvorsen, our supervisor at the University of South-Eastern Norway for his support and guidance during the project. His comments and advice have been most appreciated. We also extend our gratitude to our external partner, Altibox AS, particularly Daniel Wathne Warholm, for generously providing us with essential tools, materials, and invaluable insights into the Altibox infrastructure, enabling the commencement of this project.

Porsgrunn, 19.11.2023

Saba Homayounibaghbidi

Jan-Robin Brustad

Even Tviberg Hope

Contents

Preface	3
Contents.....	4
Nomenclature/Abbreviations	6
1 Introduction	8
1.1 Background	8
1.2 Objective	8
1.3 Previous works	9
1.4 System overview.....	10
2 LoRaWAN	11
2.1 Specification.....	11
2.2 Available sensors	11
3 Altibox infrastructure.....	12
3.1 ThingPark Wireless – The network server	12
3.2 ThingPark X – Configuration of cloud connections	13
4 Version control system – GIT	14
4.1 GitHub	14
4.2 Repositories	14
4.3 Tools.....	14
5 API research	16
5.1 REST API	16
5.2 GraphQL	16
6 Cloud databases	18
6.1 MS Azure IoT Hub	18
6.2 AWS IoT Core	18
6.3 Dimension Four.....	19
6.3.1 GraphQL schema	19
6.3.2 MQTT	21
7 Mobile operating systems	22
7.1 Android	22
7.2 iOS	22
7.3 Windows	22
7.4 Linux.....	23
8 Requirements and design of the mobile application	24
8.1 Requirements using FURPS+	24
8.2 UML diagrams.	24
8.3 Design	27
8.3.1 Design layout.....	29
8.3.2 System architecture.....	29
9 ThingPark	31

9.1 TP Wireless	31
9.2 TP X	31
9.2.1 <i>Connections</i>	32
9.2.2 <i>Packet inspection</i>	32
9.2.3 <i>Flows</i>	34
10 Dimension Four database	36
10.1 Tenant	36
10.2 Space	37
10.3 Point	37
10.4 Signal	38
11 Android development	40
11.1 Frameworks, language, IDE and build tools	40
11.1.1 <i>Kotlin</i>	40
11.1.2 <i>Android Studio</i>	40
11.1.3 <i>Android SDK</i>	40
11.1.4 <i>Gradle</i>	41
11.1.5 <i>Android emulator</i>	41
11.2 Integrations and libraries	41
11.2.1 <i>Apollo Client</i>	41
11.2.2 <i>Hilt</i>	41
11.2.3 <i>Jetpack</i>	42
11.2.4 <i>Jetpack Compose</i>	42
11.2.5 <i>Material Design 3</i>	42
11.2.6 <i>MPAndroidChart</i>	42
12 LoRaWAN mobile application development	43
12.1 Data layer	43
12.1.1 <i>GraphQL queries</i>	43
12.1.2 <i>Mappers</i>	44
12.2 Application layer (Business layer)	45
12.2.1 <i>Aggregation function</i>	46
12.2.2 <i>Helper methods</i>	47
12.3 User interface	48
12.3.1 <i>Home screen</i>	48
12.3.2 <i>App info</i>	52
12.3.3 <i>PointsScreen</i>	53
12.3.4 <i>Detailed sensor view</i>	54
13 Discussion	63
13.1 Further work	65
14 Conclusion	67
References	68
Appendices	71
Appendix A – Project description	72
Appendix B - Github repositories	74
Appendix C – Dimension Four GraphQL schema	75
Appendix D – How to get the app published at Google Play	77

Nomenclature/Abbreviations

Abbreviation	Definition	Explanation
API	Application Programming Interface	A set of rules and tools to allow different software application to communicate.
AS	Application Server	A software framework or platform providing the environment to run and manage applications, handling tasks like data access, security, and application logic.
CRUD	Create, Read, Update, Delete	Basic operations in a database.
D4	Dimension Four	A company providing an IoT cloud database.
DevEUI	Device Extended Unique Identifier	A unique identification number for each LoRaWAN sensor device
GUI	Graphical User Interface	The interactive component of a computer program, employing visual elements like icons and windows to facilitate human input and interaction.
IDE	Integrated Development Environment	A software suite that provides tools and features for software development, including code editing, compiling, deploying, and debugging software.
IIoT	Industrial Internet of Things	Same as IoT, but with focus on industrial setting.
Industry 4.0	The fourth industrial revolution	Refers to the ongoing transformation driven by technological advancements like automation, artificial intelligence, and connectivity, reshaping industries through the fusion of physical, digital, and biological systems.
IP	Ingress Protection	Rating to classify the degree of protection provided by enclosures against the intrusion of solids or liquids into electrical devices or equipment.
iOS	iPhone Operating System	The Operating system which makes iPhone accessible for the end user.
IoT	Internet of Things	Refers to a network of interconnected devices embedded with sensors and software.
JSON	JavaScript Object Notation	A lightweight data interchange file format that uses human-readable text to transmit and store data in attribute-value pairs.

LoRaWAN	Long Range Wide Area Network	A wireless communication protocol designed for long-range, low-power IoT devices.
MQTT	Message Queuing Telemetry Transport	A lightweight messaging protocol.
OS	Operating System	Software that manages computer hardware, facilitates user interaction, and enables the execution of applications and tasks.
PER	Packet Error Rate	The rate of error for each transmitted package from the LoRaWAN sensor and TPW.
TPW	ThingPark Wireless	ThingPark Wireless (TPW) is a web application that functions as a portal for connecting LoRaWAN sensors to the Altibox network.
TPX	ThingPark X	An AS responsible for managing data flow, handling connections and decoding LoRaWAN payload.
SDK	Software Development Kit	Set of tools, libraries, and documentation provided to developers to build software applications for specific platforms, frameworks, or services
SDL	Schema Definition Language	A syntax or framework used to define the structure, constraints, and relationships within databases or information systems.
SNR	Signal to Noise Ratio	A number that explains the relation between signal and noise.
SSL	Secure Socket Layer	A protocol that ensures secure communication over networks by encrypting data.
SQL	Structured Query Language	SQL is a programming language designed for managing and manipulating relational databases.
URL	Uniform Resource Locator	A reference or address used to access resources on the internet.
USN	University of South-Eastern Norway	University that is facilitating and providing the task description for this project.
QoS	Quality of Service	The level of assurance and reliability in message delivery when using MQTT

1 Introduction

This report presents the project work performed in the development of a mobile application designed to display data from a range of LoRaWAN sensors installed at the University of South-Eastern Norway (USN) in Porsgrunn. The project task was assigned to the team by USN, with Altibox AS serving as the external partner.

The report adheres to the IMRaD structure, starting with the introduction followed by the methods and results, each separated into multiple chapters. Finally, the report concludes with discussion and conclusion chapters.

The introduction chapter provides a concise overview of the project's background, objectives, a review of prior student work, and an introduction to the developed system.

In the methods sections, LoRaWAN protocol is introduced along with the sensors used in the project. The Altibox infrastructure used for connecting to the sensors is presented. It briefly covers different database options and provides an overview of the choices available for developing the mobile application. Additionally, this chapter describes the version control system used during software development.

The results sections summarize the steps taken towards the final solution. It provides more detailed insights into the ThingPark environments, the chosen database solution, and the development of the mobile application.

Finally, in the Discussion and Conclusions chapters, a recap on the achieved results and a brief exploration of the areas that require further attention to improve the application.

1.1 Background

The Internet of Things (IoT) and smart technologies are experiencing rapid growth. The industry has enthusiastically embraced this transformative opportunity, giving rise to the Industrial Internet of Things (IIoT), often referred to as the fourth wave of the industrial revolution or Industry 4.0 [1]. The core concept of IoT involves the interconnection of various “smart” objects within a network, facilitating the collection and exchange of data. This accumulation and storage of data from an array of sensors unlocks a multitude of opportunities. IIoT serves as a powerful enabler of automation, optimization, and heightened efficiency, harnessing data-driven insights that are instrumental in understanding and leveraging concepts like “big data” [2].

Altibox AS have over the last five years established an IoT network in Norway and is currently providing it as a commercial service. This network operates on the LoRaWAN protocol, an open protocol known for efficient signal transmission with low energy consumption and high penetration capabilities. Altibox has provided the team with access to this network and infrastructure, facilitating sensor connectivity and signal routing. The Altibox network is ever growing, now covering more than 100 municipalities and over 1 million Norwegian households [3].

1.2 Objective

The background and motivation for this project stem from the need to remotely monitor LoRaWAN sensors in real-time. The primary objective has been to design and develop a mobile application with an intuitive and user-friendly interface for presentation of both real-

time and historical data. Furthermore, to facilitate the storage of sensor data, a cloud database is required for this purpose.

The main project tasks involve researching the LoRaWAN protocol, understanding the Altibox IoT infrastructure, and mobile app development. Using this research, the aim is to create a mobile application for monitoring data from LoRaWAN sensors available within the Altibox infrastructure. This includes establishing connections with these sensors, setting up a cloud database, and developing a user-friendly mobile app for efficient data monitoring.

The main project tasks involve research related to the LoRaWAN protocol, the Altibox IoT infrastructure, and mobile development. Building on this research a mobile application for monitoring data from available LoRaWAN sensors shall be developed. This includes establishing connections with LoRaWAN sensors within the Altibox infrastructure, designing and creating a cloud database, and ultimately developing a mobile application for the purpose of monitoring the collected data.

The upcoming sub-chapter introduces previous works relevant to this project task. While all four works share similarities with this project, none have involved developing a mobile application.

1.3 Previous works

Previously there has been different LoRaWAN and Dimension Four projects. These projects are listed in Table 1.1.

Table 1.1 Previous works

Title	Date of issue
Development of Weather System with Interface to IoT GraphQL Data Platform	18.11.22
Development and Testing of LoRaWAN Sensor Network for Internet of Things Applications	15.05.23
Development of Open Source Datalogging and Monitoring Resources for IoT Platform	18.05.22
Development of Internet of Things (IoT) Solution using LoRaWAN Infrastructure for Storing and Monitoring Sensor Data	18.11.22

Development of Weather System with Interface to IoT GraphQL Data Platform [4] shows how to use Dimension Four as a cloud storage for sensor data. The project utilizes the MicroSun weather station that is located at USN Porsgrunn. The data from the weather station is exported from the intranet through Dimension Four and visualized on a web application.

Development and Testing of LoRaWAN Sensor Network for Internet of Things Applications [5] shows how to use Altibox network to connect LoRaWAN sensors and how to use a NODE-RED to be able to present the result. Different tools have been used to be able to present and visualize the data. Some of the tools is ThingParkX, ThingParkWireless, niotix, influxdata, Grafana, Dimension Four, HiveMQ Cloud and Azure connection.

Development of Open Source Datalogging and Monitoring Resources for IoT Platform [6] shows how to use Dimension Four as an API and storage of data, as well as making applications for Raspberry Pi, Arduino for datalogging. Also, a ASP.NET web application was made for monitoring of the data.

Development of Internet of Things (IoT) Solution using LoRaWAN Infrastructure for Storing and Monitoring Sensor Data [7] shows how to make a solution for monitoring and analysing

data from LoRaWAN sensors. The solution uses Dimension Four as an IoT platform, and Altibox ThingPark as LoRaWAN gateway. A monitoring application using ASP.NET was also developed.

1.4 System overview

Figure 1.1 shows a high-level system overview, indicating the signal path from sensor through the ThingPark environment to the database and finally presented in the mobile application.

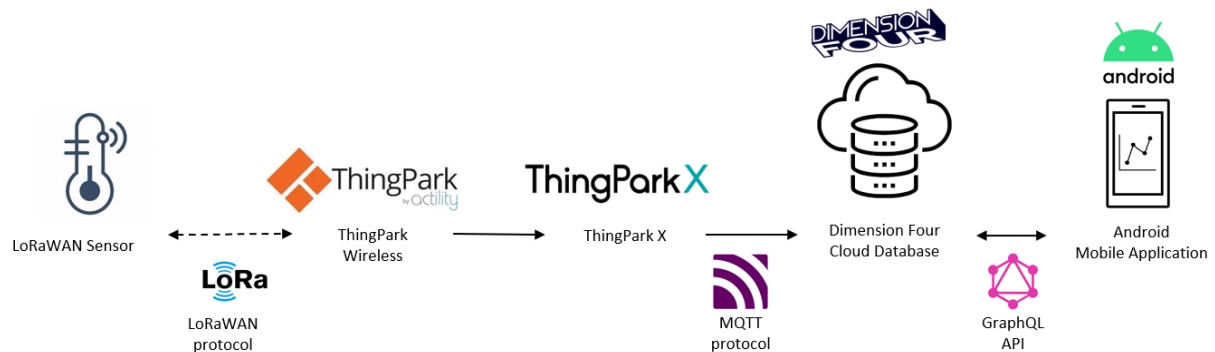


Figure 1.1 Overview of the system

The system gathers data from three sensors: a combined temperature and humidity sensor, a proximity switch, and an outdoor temperature sensor. These sensors payloads traverse the LoRaWAN network, configured through ThingPark Wireless (TPW), then proceed to ThingPark X (TPX) for further processing via the MQTT protocol, routing the data to the appropriate table in the cloud server. TPW is a web application linking LoRaWAN devices to the Altibox IoT network, while TPX serves as an application server managing data flow, connections, and payload decoding.

The chosen cloud server solution, Dimension Four (D4), tailored for IoT applications, supports both MQTT protocol (used for sensor payloads from TPX) and GraphQL API (for data retrieval in the mobile application). The mobile app, developed using Kotlin in Android Studio, retrieves sensor data from the cloud database via GraphQL API, presenting it in two separate views. The first offers an overview of all sensors, including the latest samples, while the second provides detailed information for selected sensor, including the last 10 samples and a 24-hour data graph.

2 LoRaWAN

As mentioned in the introduction, Altibox chose the LoRaWAN protocol for their IoT network. Weighted by arguments such as long transmission range, low energy consumption, and strong penetration capabilities, resulted in LoRaWAN ending up as the preferred choice [8]. Its open nature, hence, no licensing fees, further contributed to its selection. This chapter provides a brief introduction to the protocol, followed by an overview of the sensors available to the project.

2.1 Specification

The LoRaWAN is a low power, wide area networking protocol tailored for the Internet of Things (IoT). It facilitates the wireless connection of battery-operated devices to the internet across global, national, or regional networks, addressing crucial IoT requirements like bi-directional communication, end-to-end security, and mobility. Employing a star-of-stars network architecture, LoRaWAN relies on gateways to relay messages between end-devices and a central network server. It leverages the Long Range characteristics of the LoRa physical layer, allowing for a single-hop link between end-devices and gateways. The specification classifies end-point devices into three classes: Class A for low-power asynchronous communication, Class B for deterministic downlink latency, and Class C for lowest latency, all supporting bi-directional communication.

LoRaWAN's data rates, ranging from 0.3 kbps to 50 kbps, enable dynamic trade-offs between communication range and message duration. Security is a primary focus, with the specification incorporating two layers of cryptography—a 128-bit Network Session Key and a 128-bit Application Session Key, both using AES algorithms. These keys can be activated during production or commissioning (ABP) or over-the-air (OTAA) in the field, providing flexibility for device deployment and re-keying if necessary. Developed and maintained by the LoRa Alliance, the specification ensures interoperability across manufacturers and allows the industry the freedom to innovate and deploy LoRaWAN in diverse models and types, whether public, shared, private, or enterprise, without dictating a specific commercial model [9]–[11].

2.2 Available sensors

The LoRaWAN sensors listed in Table 2.1 were provided by Altibox and installed at various locations at USN in Porsgrunn. All sensors are manufactured by Adeunis, a French company specializing in IoT products and solutions [12]. The Comfort sensor is a sensor for indoor use (IP 20), that measures both temperature and humidity. The TEMP is suitable for outdoor use (IP68) and measures temperature. The DRY CONTACTS is also suitable for outdoor use (IP68) and reports the state of its dry contacts, either 0 or 1. In this case it is used to monitor whether a door is open or closed.

Table 2.1 Sensors available for the project

Device model	Sensor Type	DevEUI	IP
Adeunis COMFORT V2 for LoRaWAN	Temperature and humidity	0018B21000003BBF	20
Adeunis TEMP V4 IP68 LoRaWAN	Temperature	0018B21000004540	68
Adenis DRY CONTACTS LoRaWAN	Proximity switch (0 or 1)	0018B22000001FD2	68

3 Altibox infrastructure

The Altibox partnership has been continuously expanding their LoRaWAN sensor network in Norway, currently spans across 100 municipalities and more than 1 million households [8]. Altibox now offers IoT networking as a commercial service. The product range includes the two products IoT Access and Total:

- IoT Access is the basis service, offering end to end support for a customer's sensors and application server. With this service, a company can utilize the Altibox network for seamless, secure, and efficient data transfer between their sensors and application server.
- IoT Total is a complete service, covering receipt, decoding, structuring, and presentation of sensor data. With this service, a company can integrate its own sensors, while Altibox will supply the sensor network, network server, application server, and present sensor data through a graphical user interface.

When it comes to possibilities within IoT, the only limit is one's imagination. The Altibox network hosts a variety of applications, including measuring water levels in rivers and lakes, smart waste management, monitoring humidity and temperature in municipal housing, smart power boxes, and digital water consumption measurement [13].

3.1 ThingPark Wireless – The network server

ThingPark Wireless (TPW) is a web application that functions as a portal for connecting LoRaWAN sensors to the Altibox network. TPW incorporates the following key integrated applications:

- Network Survey
- Wireless Logger
- Device Manager
- Network Manager

The “Network Survey” application enables a survey of the network to identify signal coverage and quality. “Network Manager” is a toolset for network providers to oversee the radio access network by creating and administering LoRaWAN base stations. In the context of this project, only the “Wireless Logger” and “Device Manager” have been relevant.

Wireless logger

The Wireless Logger offers the capability to inspect all packets from LoRaWAN devices [14]. It comes with a built-in decoder that allows for the examination of the payload of each packet. This is an especially useful tool for effective fault finding.

Device Manager

The Device Manager serves as a back-end user interface for adding and configuring sensors. In this interface, you can define connectivity plans and establish associations between application servers (AS) and AS routing servers for the devices. These configurations can be explained as follows:

- A connectivity plan defines the access and type of communication to the network.
- An AS is the destination used by an AS routing profile to route device packets to one or more third-party application servers. These servers either receives uplink packets of data from a device or send downlink packages to command a device.
- AS routing profiles define how the device data will be routed to a destination.

3.2 ThingPark X – Configuration of cloud connections

ThingPark X (TPX) simplifies the interface between LoRAWAN-connected sensors and IoT applications, transforming sensors raw data into application-friendly actionable information, that can be fed into the cloud [15].

TPX IoT have the following three key capabilities:

1. Drivers:

The drivers function to decode uplink messages, converting raw hex data strings into readable data. This process transforms the payload into a standardized JSON object. TPX provides support for over 100 sensor models and also offers the option to upload custom drivers [15].

2. Connections:

The connections serve the purpose of adapting transport protocols and forwarding data to external application servers or cloud providers. Connectors ensure the reliable delivery of extracted sensor data (via the driver engine) to the chosen IoT platform [15]. Uplink transformations can be configured to extract payload data and structure it in accordance with the end point specifications.

3. Flows:

The flows are the main function in TPX. It is configured to subscribe to data from one or multiple devices, through the device key which is the DevEUI. Each flow is also connected to one or more connections, and drivers for the device can be specified. Uplink transformations can also be added, if not handled in the connection.

Figure 3.1 illustrates the IoT flow in TPX. TPX only requires two key pieces of information: the data source (DevEUI) and its destination. The device's DevEUI, along with its associated payload, is captured by the flow, triggering the relevant driver's initialization. Once the data is decoded, it is parsed, organized, and subsequently pushed to each connected connection.

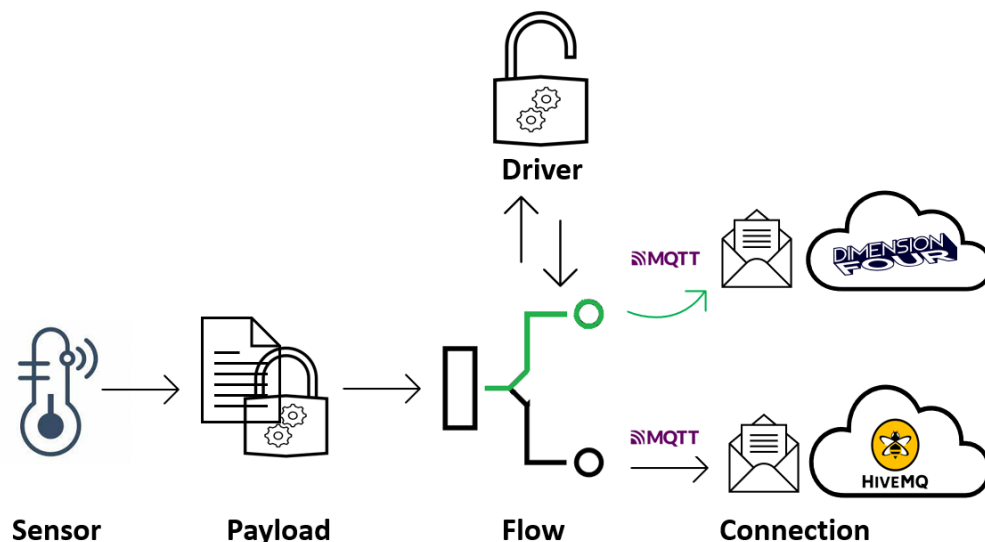


Figure 3.1 TPX IoT flow

4 Version control system – GIT

GIT [16] is a method to have version control for software applications. There are many ways to interact with a GIT system, and GitHub is an online web-based method.

GIT includes methods to make repositories for the code, clone other repositories, track changes through staging and committing a change and branching and merging to make it possible for multiple members to work on different parts.

Pull and push are also included so the user can send their commit to the main repository, and not only their local copy.

The main purpose of using a GIT based system is to track changes in a code, see who made changes, track versions of the software and to collaborate with other team members.

4.1 GitHub

GitHub [17] is an online interface that uses GIT. GitHub supports personal use and organizational use. For this project organizational use is chosen.

The organization in GitHub is called LoRaWANMobileAPP.

The same tools for GIT are included in GitHub, and the main difference is that GitHub is an online version. Meaning it also host code for the different projects and are always online. This means that members working on a project can pull the latest version from GitHub and make their changes in a local clone. When they are done with their changes, they can push the changes back to GitHub. This way every member on the team always has access to the latest commits.

4.2 Repositories

The main repository used in this project is the GraphQLMobileApp and the organization page on GitHub is LoRaWANMobileApp, more on this and links can be found in Appendix B. There exist multiple repositories in the organization pages, but most of them are only for testing. The complete project has been pushed to GraphQLMobileApp.

As GraphQLMobileApp is the main repository for the development of the mobile application, this repository will contain all the commits with comments for the whole development. Here it is possible to roll back to an earlier version of the application in case of bugs in the software or any other problem.

4.3 Tools

This subchapter gives an overview of the tools and methods used to interact with the GitHub Repositories.

The main tool for interaction with GitHub is GitKraken. GitKraken is a free and visual GIT software. GitKraken supports both local and GitHub repositories. When using GitKraken for the mobile application development the repository must first be pulled from GitHub. When the repository is pulled from GitHub it is possible to add changes to the software.

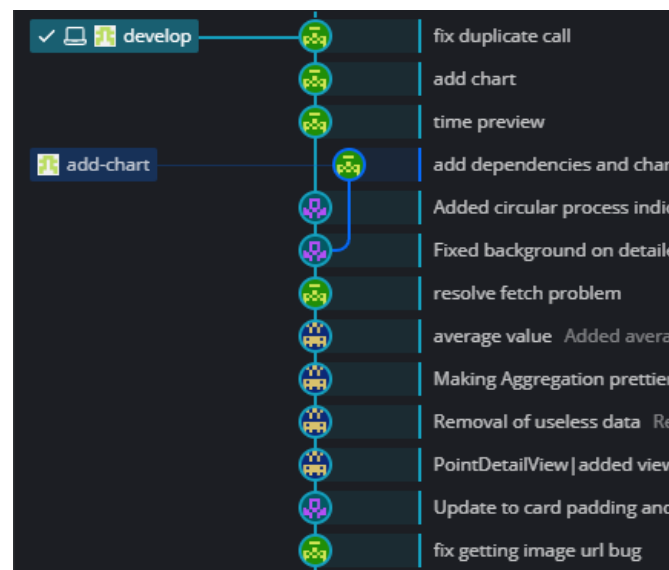


Figure 4.1 GitKraken user interface.

Figure 4.1 shows parts of the GitKraken user interface. Here the visual aspect is shown for the different commits in GraphQLMobileApp. It is also easy to see if the local repository is up to date in accordance with the GitHub version of the repository. This is shown in Figure 4.2.



Figure 4.2 Laptop symbol is local repository, and the icon is the GitHub repository.

5 API research

To make an application for monitoring data on a mobile application it is important to fetch data from a server or database. A way to do it without accessing the database directly is through an Application Programming Interface (API) service. This is a more secure way to access the data instead of directly running SQL queries which would make the system vulnerable for SQL injection and cyber-attacks.

API is a set of rules or protocols that programs, webapps etc has to use in order to communicate with each other. Both the methods and data format are defined by the set of rules. An API service gives a controlled way to get the data and is easy scalable for further development. Choosing the right API service will give the end product an easy and secure way to retrieve data. Here both REST API and GraphQL is considered [18].

5.1 REST API

RESTful is an API which uses HTTP requests to gather data. This means that there must be an endpoint URL for the data it is trying to receive.

REST consist of 4 HTTP methods, GET, POST, PUT and DELETE.

- GET is used to retrieve information. This can be used to retrieve the last measured value from one sensor in the mobile application.
- POST is used to create a new resource. For example, would this be to add a new sensor that can be utilized in the mobile application.
- PUT is used to update an existing resource and add data. This can be the sensor storing new data.
- DELETE is used to delete a resource.

Use of REST would require setting up multiple URL endpoints for the type of data that is retrieved. For example can this be like:

<http://api.webapp.com/sensor/ComfortSensor/measurements>

<http://api.webapp.com/sensor/OutdoorTemperature/measurements>

One big disadvantage is that if “measurements” contain all the measurements, then REST would fetch all measurements. This could potentially be a big load of data that could slow down the mobile application. An approach could involve modifying the HTTP GET request to retrieve only the last 20 measurements. However, this raises concerns: if access to the last 100 measurements is necessary, it necessitates the creation of two separate URL endpoints: one for 20 data points and another for 100 data points. Additionally, retrieving data within specific time intervals becomes challenging when using REST API. Although fetching all data and sorting it within the mobile application is plausible, it may lead to high memory usage due to handling substantial amounts of data.

Another alternative could be to add custom endpoints for data retrieval. This can look like this for example:

<http://api.webapp.com/sensor/ComfortSensor/measurements?start=time1&end=time2>.

5.2 GraphQL

GraphQL [18] is an API that uses a single query to fetch the data that is needed. One of the key features with GraphQL is the elimination of over and under fetching of data, which is a disadvantage with other APIs. For example, if you have a temperature sensor and a lot of data

is stored with the sensor data, like geolocation, device address, data channels etc, it is possible to only fetch the data that is needed. This can be the temperature measured and the timestamp. This is especially relevant for an application focused on displaying real-time temperature updates. By selectively fetching only necessary data, it significantly reduces the amount of data transmitted over the network. By using a single query, the load of the server is reduced, meaning the server can maintain a higher number of users accessing the same data. GraphQL also supports real-time data using their subscription feature, and hereby providing the subscribers with instant update of the latest temperature.

Another advantageous feature of GraphQL is its ability to update only relevant queries whenever the project requirements change. This streamlined approach enables the developed app to easily expand and scale for new features.

GraphQL relies on a schema that serves as a defined protocol dictating how data should be fetched. Adhering to this schema enables clients to retrieve data from the server efficiently.

6 Cloud databases

The LoRaWAN sensors will transmit new payloads at specified intervals. To store historical data and effectively manage it for real-time monitoring, a database is essential. Today, several efficient solutions are available, where this chapter briefly introduce some of the viable alternatives.

6.1 MS Azure IoT Hub

MS Azure IoT Hub, developed by Microsoft Corporation [19], provides a robust platform accessible through the following webpage link: <https://azure.microsoft.com/en-us/products/iot-hub>. This platform serves as a communication channel, enabling the sending and receiving of data from IoT devices. Azure IoT Hub supports two-way communication with devices, facilitating over-the-air deployment for updates to IoT sensors. Furthermore, the solution seamlessly integrates with various Azure services, including Event Grid, Logic Apps, Machine Learning, and IoT Edge. To store sensor data, integration with Azure SQL cloud databases and other services can be established. Azure IoT Hub supports the HTTPS, AMQP, MQTT protocols for connecting devices.

Customized subscription plans are offered with pricing largely dependent on the required number of messages. A free option is available, providing \$200 worth of credits for one month. After this period, users transition to a pay-as-you-go plan, where charges apply only if usage exceeds the free monthly amount. The paid plans are categorized as basic and standard. The primary distinction lies in bidirectional communication support, with basic plans lacking this feature, whereas standard plans offer bidirectional communication. Table 6.1 shows an overview of the available subscription plans.

Table 6.1 MS Azure IoT Hub overview of subscription plans [20]

	Free tier	Basic tier [B1 → B3]	Standard tier [S1 → S3]
Price [NOK]	Free	123 → 6 135	307 → 30 674
Messages / day	8k	400k → 300M	400k → 300M
Units	500	Unlimited	Unlimited

6.2 AWS IoT Core

AWS IoT Core, developed by Amazon Incorporated [21], is a managed cloud platform accessible through the following webpage link: <https://aws.amazon.com/iot-core/>. This platform enables IoT sensors to connect and utilize AWS cloud services. AWS IoT serves as a gateway between IoT devices and AWS services, supporting MQTT, HTTPS, and LoRaWAN protocols. The platform offers two-way communication, opening for over the air configuration and updates. Additionally, storage can be configured in the AWS SiteWise cloud database.

AWS Free Tier is available for 12 months starting the day the user account is created. The connectivity pricing is based calculated in one-minute increments and is based on the total time the devices are connected to AWS IoT Core.

6.3 Dimension Four

Dimension Four AS (D4) is a Norwegian company with their headquarters in Skien. The company is part of the New Normal Group AS. D4 provides of a headless IoT platform [22], which means that they do not provide a user interface to access the database. The platform can be accessed through the following web page link: <https://dimensionfour.io/>

The platform features an intuitive and user-friendly web interface, making the process of setting up the database hierarchy accessible to users with various levels of technical expertise. Within this platform, D4 offers a GraphQL API with real-time subscription capabilities. The API also grants users the power to manage tenants and adjust or add spaces and points as needed. D4 supports various communication protocols, including MQTT, HTTPS, UDP, GraphQL, and more, ensuring compatibility with a wide range of IoT systems and devices. Furthermore, D4 implements Webhooks, automating the sending of messages from D4 to another API when specific events occur. This feature greatly enhances communication and integration possibilities.

D4 provides three subscription plans to cater to diverse user needs: a free plan, ideal for testing and prototyping; the professional plan, well-suited for product launches and system scaling; and the enterprise plan, specifically tailored to meet the robust requirements of organizations with extensive IoT needs. Table 6.2 provides an overview of what is included in each of the different plans.

Table 6.2 D4 overview of subscription plans [23]

	Free	Professional	Enterprise
Price [NOK]	Free	3000 /month	Custom
Sensors	Up to 10	Up to 1000	Inf
Users	Unlimited	Unlimited	Unlimited
API Calls	100 000	1 000 000	Custom
Storage	100 MB	1 GB	Custom

6.3.1 GraphQL schema

The GraphQL schema serves as a contract between the server and client in application development [24]. It defines the structure of available data, including types, fields, and operations that clients can perform [25]. GraphQL schema is defined using a human-readable Schema Definition Language (SDL). This language allows developers to specify types, fields, queries, mutations, and other aspects of the schema in a concise and expressive manner [26]. Dimension Four GraphQL schema provides required information for application development such as type hierarchies, field definitions and their constraints such as nullability, query mutation and subscription type, input types, enum types and response structures [27]. Apollo Client relies on the GraphQL schema for query validation, introspection, and code generation. The schema awareness enhances the developer experience by enabling autocompletion, type validation, and efficient data handling on the client side. Figure 6.1 shows a part of Dimension Four GraphQL Schema. It can be downloaded from Dimension Four website or from

<https://iot.dimensionfour.io/graph>. A part of Dimension Four GraphQL schema is given in Appendix C.

```
directive @cacheControl(  
  maxAge: Int  
  scope: CacheControlScope  
  inheritMaxAge: Boolean  
) on FIELD_DEFINITION | OBJECT | INTERFACE | UNION  
  
type PageInfo {  
  startCursor: Cursor  
  hasNextPage: Boolean  
  hasPreviousPage: Boolean  
  endCursor: Cursor  
}  
  
# Cursor scalar is an opaque string (base64 format)  
scalar Cursor  
  
type BooleanPayload {  
  success: Boolean!  
}  
  
type Price {  
  # All amounts are in a currency's smallest unit. i.e in cent  
  amount: Float  
  currency: String  
  trialDays: Int  
  name: String
```

Figure 6.1 Dimension Four GraphQL Schema

Dimension Four GraphQL Playground is an in-browser IDE enabling developers to interact with GraphQL APIs. With a graphical interface, autocomplete, synthetic schema, and query history, it facilitates exploring, testing, and debugging GraphQL APIs. While documentation is available on the website, using GraphQL Playground is strongly recommended for a more interactive and efficient experience [28]. Figure 6.2 shows Dimension Four GraphQL Playground environment with a sample query executed in it and its result.

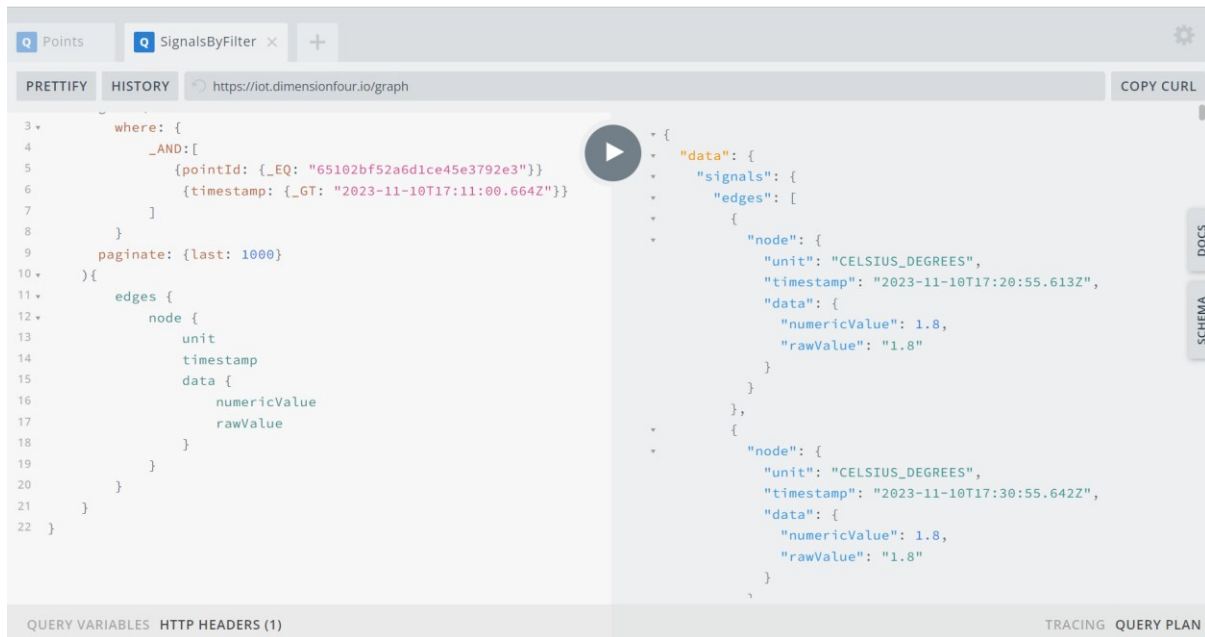


Figure 6.2 Dimension Four GraphQL Playground

6.3.2 MQTT

D4 offers MQTT as a communication protocol. MQTT is well suited for IoT applications, where devices manage limited resources. This lightweight protocol is designed for the exchange of small data packets, making it a good choice for scenarios with constraints like low bandwidth, high latency, or network unreliability. MQTT's minimal resource requirement and emphasis on smaller data exchanges make it well-suited for such applications. MQTT uses a publish-subscribe model, allowing clients to publish messages to specific topics, while others subscribe to these topics, ensuring selective and efficient communication. This approach ensures that clients receive only the information relevant to their interests. Moreover, MQTT simplifies message encryption through TLS and offers modern authentication protocols for client verification. It also supports various Quality of Service (QoS) levels, ensuring reliable message delivery, even under challenging network conditions [29]–[31].

7 Mobile operating systems

There are various methods and integrated development environments (IDEs) to create mobile application, each tailored to specific operating systems. Most mobile phones run either Android or iOS. According to Statcounter [32], as of October 2023 Android stands for 69.64% of the market share with iOS as the runner up on 29.67%. Other alternatives exist, such as e.g., Windows and other Linux based operating systems like Ubuntu Touch.

For mobile development it is crucial to choose the right platform with the right tools.

7.1 Android

Android is a mobile operating system that can run on most modern mobile phones. Android Studio is a good start for programming mobile apps on Android [33]. The reason for this is the intuitive and simple user interface, but still the possibility for advanced software development. The main programming language used for Android is Kotlin, which is used by 60% [34] of the professional Android developers. Other alternatives are Java, C# and Python f. ex.

Android Studio helps you debug the code and offer an intelligent code editor which helps you write code and keep it bug free. There is also a possibility to emulate the application with the built-in Android emulator, and it is possible to develop for a specific Android release.

The community support for Android Studio is relatively big due to the big pool of developers for Android. And there is plenty of online resources both written and through YouTube.com. FreeCodeCamp is an example of a provider for online resources on YouTube [35].

Android Studio is also the official IDE for Android application development and is continuously updated by Google to have the latest features and practices for Android.

7.2 iOS

iOS is short for iPhone operating system and uses for example Swift programming language. iPhone operating system is not only limited to iPhone, but also support iPad, iPod etc. Swift is noted as a great and beginner friendly programming language and is designed to be used as a first programming language [36]. Objective C is an alternative to Swift but is not used as much. The reason for this is that Swift is considered the successor and has cleaner syntax and performance benefits.

The official IDE for iOS is Xcode and it supports Swift programming language and objective C. It has the possibility of debugging and is user friendly.

7.3 Windows

Windows Phone is an discontinued mobile operating system for smartphones developed by Microsoft [37]. The latest version is Windows 10 Mobile.

The primary IDE for making applications for Windows Phone is Visual Studio. Windows phone supports these programming languages: C#, Visual Basic, HTML5 and JavaScript and C++.

7.4 Linux

Ubuntu Touch is a mobile operating system based on Linux. There exist many Linux based operating systems, but Ubuntu Touch has good community support and is more focused on privacy and freedom than other mobile operating system.

Clickable is the main tool used to compile or build applications for Ubuntu Touch [38]. The main advantage is that that any IDE can be used to build apps as long as it is built through Clickable.

The user interface can be made using HTML or QML, and the business layer can be made using Python or JavaScript for example.

8 Requirements and design of the mobile application

The specification and customer task description can be found in Appendix A. This document serves as the foundation for utilizing FURPS+ in gathering the requirements for the mobile application, providing a basis for the succeeding development.

8.1 Requirements using FURPS+

FURPS+ stands for functional, usability, reliability, performance and supportability requirements. The + sign stands for additional features.

Functional requirements:

- The mobile application should present measurements from LoRaWAN sensors.
- It should be possible to view information on each sensor, and extra data like minimum, maximum and average data is shown.
- The mobile application should have a front page.
- The mobile application should have an about page where instructions and information is given about the mobile application.

Usability requirements:

- The system should be user-friendly.
- The last measured value for each sensor should be presented in the home page.
- More detailed information about each sensor should be presented in a more detailed view.

Reliability requirements:

- The mobile application needs an external storage of data so it's not relying on local storage. Dimension Four is used as a storage of data.

Performance requirements:

- There shall be an API service to fetch data from Dimension Four. GraphQL is selected for that. The reason for selecting GraphQL is its flexibility and selectivity when it comes to what data is needed.
- There should be a LoRaWAN network that the sensors are connected to. Altibox ThingPark is selected as a sensor network since Altibox is the customer their sensor network covers the needs of this project.
- Dimension Four is using MQTT to gather data from ThingPark X. ThingPark X is used for cloud connections and simplifies the delivery of data to IoT services.

Supportability requirements:

- The mobile application should be easy to expand over time.
- +:
- The system should be open source with documentation on GITHUB.
 - During development and planning Microsoft Teams should be used to cooperate with both the customer and supervisor.
 - A readme file is to be released with the software.

8.2 UML diagrams.

Figure 8.1 shows a use case diagram for the mobile application based on the FURPS+ requirements in chapter 8.1.

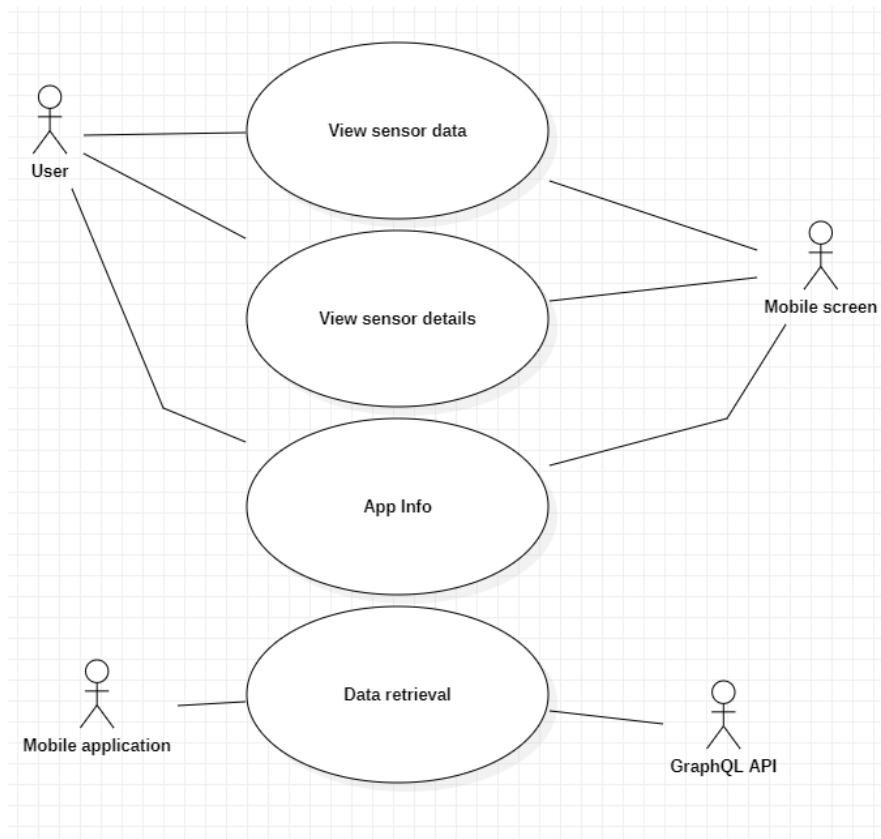


Figure 8.1 Use Case diagram for the mobile application.

In Figure 8.1 the use cases for the mobile application are shown. Here the use case “View sensor data”, “View sensor details”, “App info” and “Data retrieval” are shown. View sensor data is the main use case and is the first point of entry for the end user. Here all sensors are listed together with the latest sample. In the view sensor details, a more detailed perspective of a chosen sensor is shown. Here maximum, minimum, and average sensor data is presented, including a graph with the last 24 hours of data and a table showing the 10 latest samples. App info gives an overview of the mobile application and the system around it. Data retrieval handles the GraphQL queries to retrieve data.

Figure 8.2 shows the system sequence diagram for the mobile application.

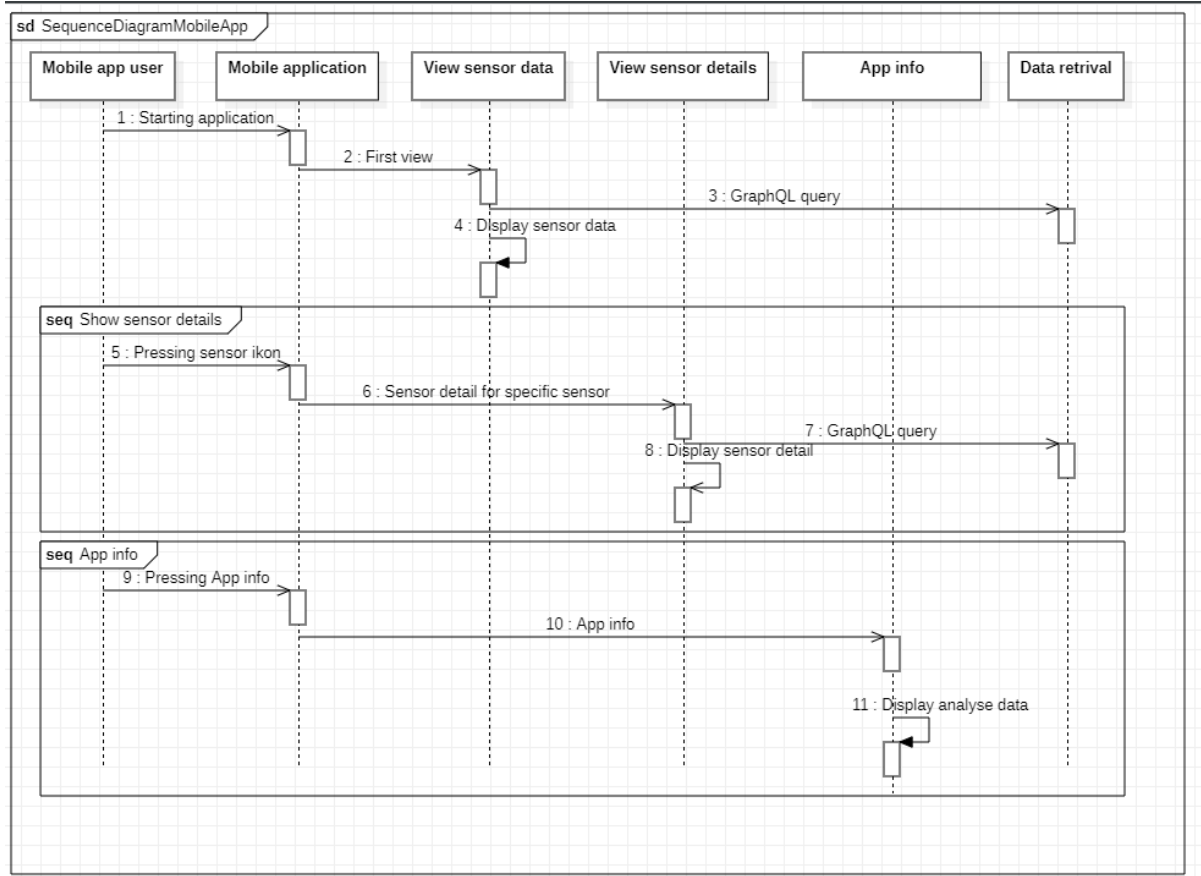


Figure 8.2 System Sequence Diagram for the mobile application.

The system sequence diagram in Figure 8.2 shows a simplified sequence for how the mobile application should function. When the application is started, the main GUI is shown with relevant sensor information. This user interface automatically handles the GraphQL queries to retrieve the measurements.

When a sensor image is pressed, the GUI should update with a new view including a more detailed look at the measurements from the selected sensor. Values like max, min and average and some detailed description of each sensor is shown.

When the user presses the App info some information about the mobile application is shown.

Since the dataflow is crucial a collaboration diagram has been selected as the interaction diagram. This is shown in Figure 8.3.

sd CollaborationDiagram

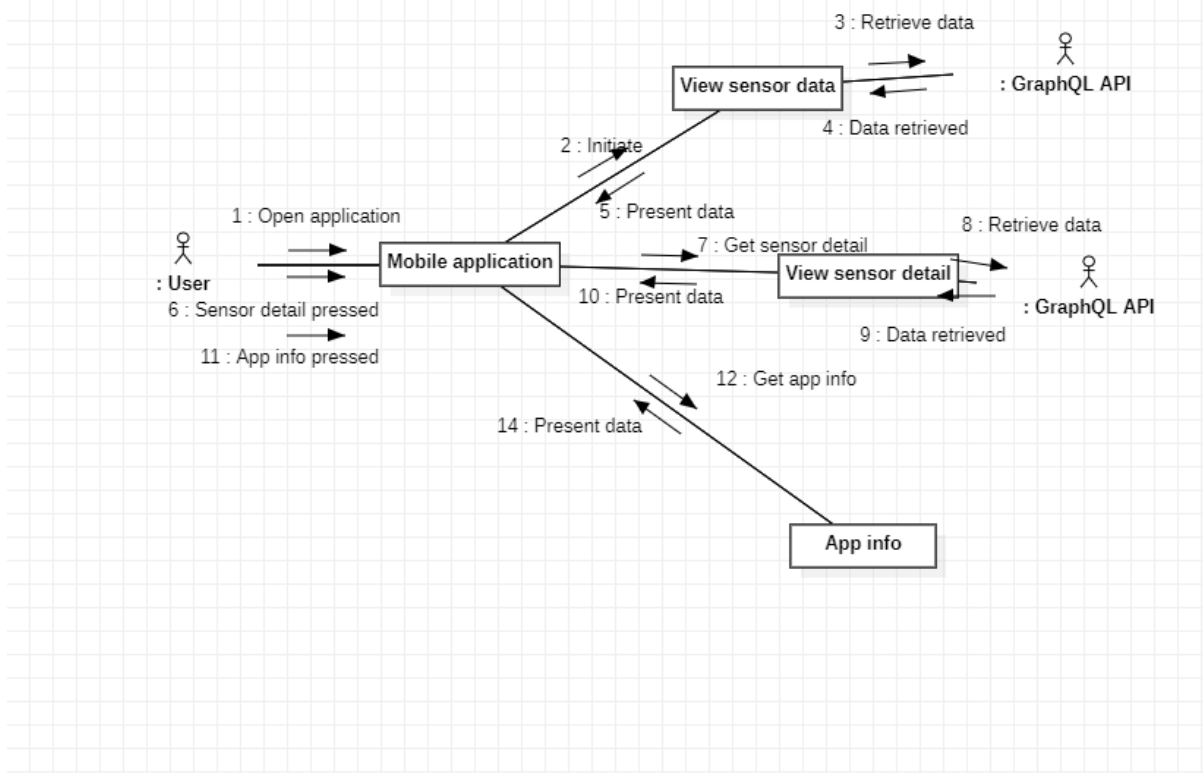


Figure 8.3 Collaboration diagram

The collaboration diagram shows the flow of information in the mobile application. The user interacts with the mobile application, and the first view, the “view sensor data” is shown. Here the user can interact once more and press the icon of a sensor to gather a more detailed view for each sensor. The mobile application should automatically know which sensor the user pressed. And if the user is on the “view sensor data” page it is also possible to get a more detailed description of the mobile application in the “App info” view.

8.3 Design

The mobile application is to be designed as a three-tier architecture, covering a presentation layer, business layer and a data layer.

The presentation layer will consist of view models to present data and give the user the ability to interact with the mobile application. According to the collaboration diagram in Figure 8.3 there will be at least three views for the user. Those are going to be a home screen, app info and a detailed view of the selected sensor.

The home screen is the first view the user will experience and will show the last measured value for each sensor.

App info will give a more detailed description of the mobile application.

The view sensor detail is going to show more detailed data about the measurements to the user.

A simple flow diagram on how the user is going to interact with the application is shown in Figure 8.4. The flow diagram uses the same names as the UML diagrams in chapter 8.2.

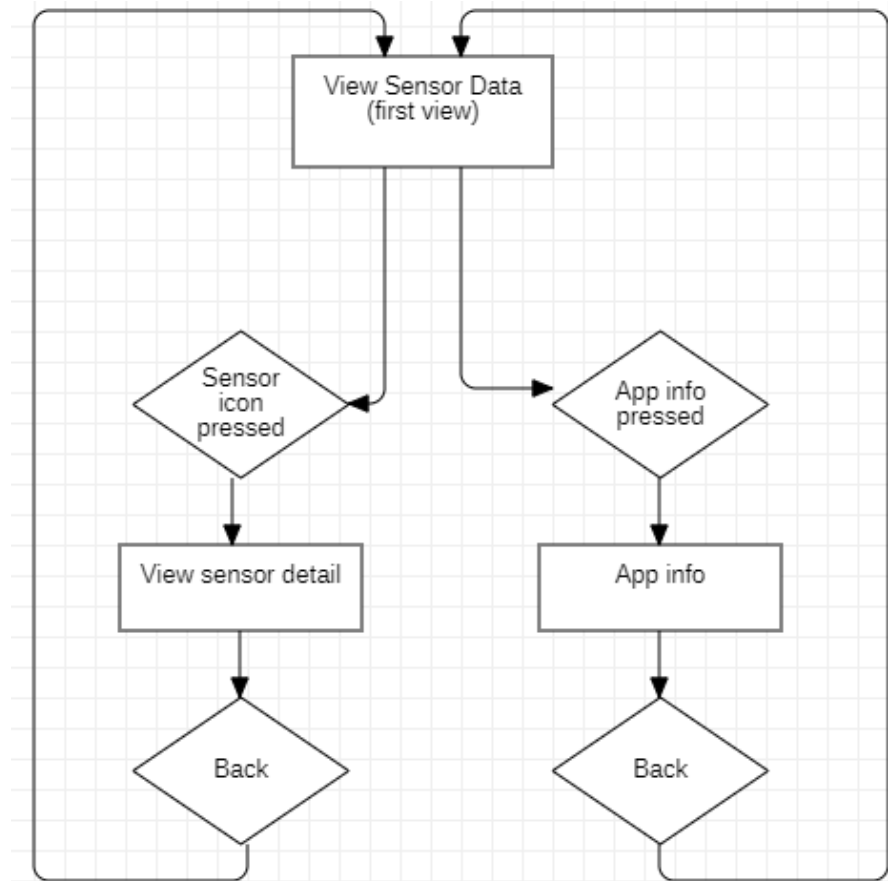


Figure 8.4 Flow diagram

8.3.1 Design layout

To set targets for designing the GUI, simple sketches of the user interface where made. Figure 8.5 and Figure 8.6 depicts the early ideas of how the app views should be presented.



Figure 8.5 Home screen.

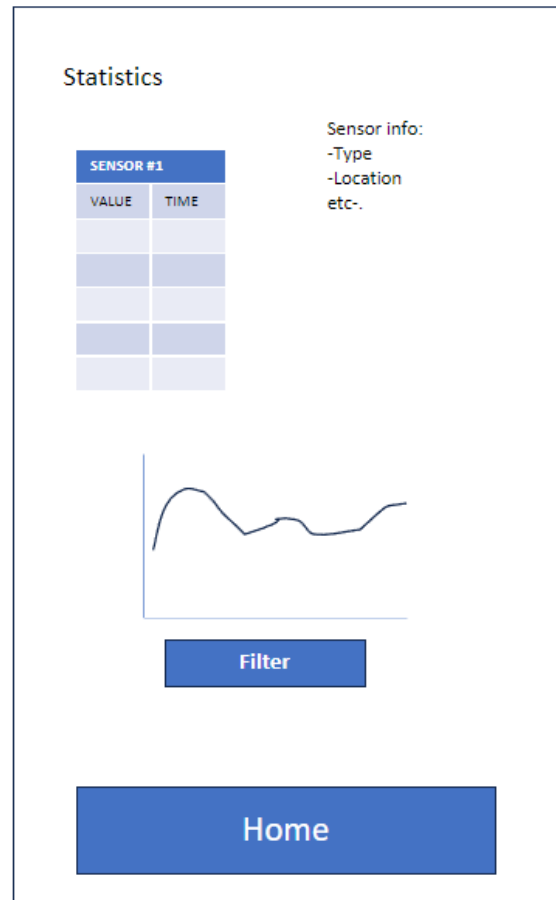


Figure 8.6 Sensor detail view.

The main goal of the user interface is to keep the mobile application simple enough that most people will be able to interact with it. As Figure 8.5 shows the most relevant measurements are shown on the home page. And a more detailed view is shown in Figure 8.6. For a better viewing experience there will be a title of each page and padding towards the edges so no text is on the border of the mobile screen. The sensor detail view will also give a description of a sensor and the location of the sensor.

8.3.2 System architecture

As mentioned in the start of the Design chapter, a three-tier architecture is chosen. The relation between the layers in this type of architecture is shown in Figure 8.7.

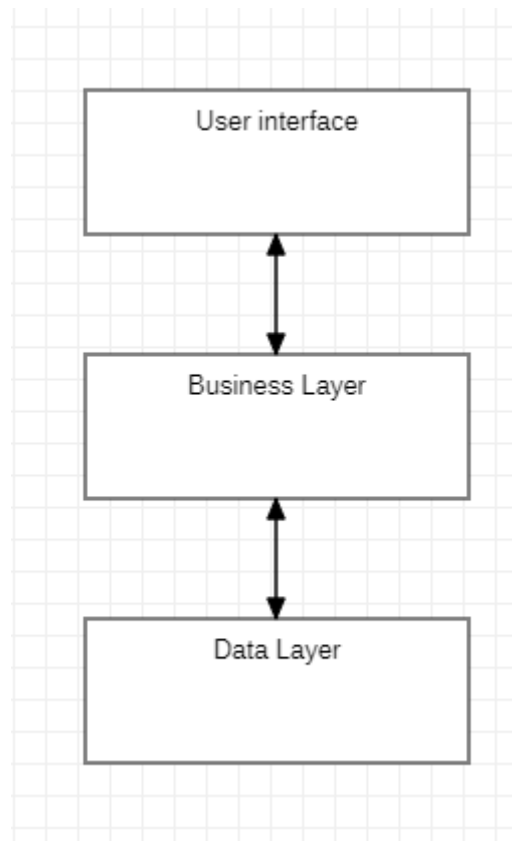


Figure 8.7 System Architecture

This chosen system architecture might seem a bit advanced for a mobile application that only fetches data from LoRaWAN sensors, but for future scalability and improvements this could prove important. The user interface will handle the user's interactions with the mobile application and presentation of data. The data layer is to handle the storage of data. For this project the data layer is handling the communication with the external cloud server through an API service.

9 ThingPark

The ThingPark environment offers the services to facilitate the transfer of sensor payloads from sensors to the database. The setup and configuration were achieved through a series of steps within the two primary TP systems, Wireless and X.

9.1 TP Wireless

In ThingPark Wireless, the connection of available sensors to the gateway was managed by a separate project group. Assigning the connectivity plan and application server routing profiles was done as part of the sensor onboarding process within TPW. Altibox provided pre-defined connectivity plans for use in this project. TPX served as the application server and was specified as the destination in the routing profile. Since the sensors were concurrently used in another project, a shared routing profile was established between the projects.

Figure 9.1 illustrates the LoRaWAN sensors connected in TPW, displaying the devices listed in the Device Manager. The list of sensors shown includes relevant details such as the DevEUI, connectivity plan, AS routing profile, average packet rate, mean PER, average SNR, and sensor battery level for each device.









Name / Type	Identifiers	Connectivity	Average packets	Mean PER	Average SNR	Battery
 Comfort Comfort Sensor	0018B21000003BBF FC004EA5	ALTIBOX Standard XL Pipedream	48.0/day	2.0%	5.0 dB	
 Contact Sensor Dry Contact Sensor	0018B22000001FD2 FC004E1F	ALTIBOX Standard XL Pipedream	45.0/day	0.0%	8.5 dB	
 Field Test Device Field Test Device	0018B2000000263A8 FC004F8A	ALTIBOX Standard XL Pipedream	0.0/day	0.0%	6.8 dB	
 Outdoor Temperature Sensor	0018B21000004540 34C194C1	ALTIBOX Standard XL Pipedream	158.0/day	0.0%	6.8 dB	

Figure 9.1 Sensors available to the project shown in the device manager

The Wireless Logger application enables inspection of payloads for each sensor. In Figure 9.2, an example illustrates how to specify a filter to show Comfort sensor payloads. The process involves entering the DevEUI for the device, along with the requested packet type and decoder. The output of the query is a list of the last transmitted packages of that type for the specified device.

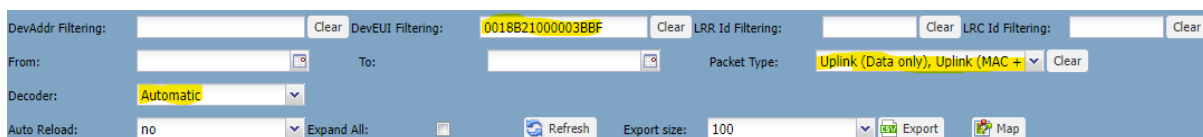


Figure 9.2 Wireless logger filter options

To grasp the configuration of uplink transformations in TPX, it was necessary to comprehend how information is organized within each package. Since the wireless logger provides only a partial view of the packet, revealing the decoded payload and some header information, another alternative method to access the complete package information was explored. Further details on this can be found in the TPX chapter.

9.2 TP X

With the sensors and necessary routings configured in TPW, the next step was to snatch this data in TPX for routing to the database. The initial configuration involved setting up connections, followed by defining the flows.

9.2.1 Connections

The connection determines the protocol used between TPX and the database. As depicted in Figure 9.3, four connections have been established for this project, with all connections utilizing the MQTT protocol as indicated by the MQTT symbol to the left.





	Connection Name	Last Restart	Active Devices (last 1h/24h)	Uplinks (last 1h/24h)	Downlinks (last 1h/24h)	State
	MQTT_D4_MobileApp_DoorSensor	4 days ago	1 / 1	1 / 30	0 / 0	OPENED ▼
	MQTT_D4_MobileApp_TempSensor	4 days ago	1 / 1	6 / 144	0 / 0	OPENED ▼
	MQTT_D4_MobileApp_ComfortSensor	4 days ago	1 / 1	1 / 24	0 / 0	OPENED ▼
	MQTT_Hive_MobileApp	4 days ago	3 / 3	8 / 198	0 / 0	OPENED ▼

Figure 9.3 Connections established in TPX

The specifications enabling TPX to navigate and access the database are outlined in the basic settings. Figure 9.4 illustrates the identical basic settings for the three D4 connections. The necessity for three D4 connections arises from the fact that uplink transformations are executed at this level. These transformations append header information to the signal and specify the corresponding pointId in the database for data entry.

BASIC SETTINGS (Connection id: 4927)













Hostname*  <input type="text" value="mqtt.dimensionfour.io:1883"/> 	Protocol*  <input type="text" value="TCP"/> 
MQTT Username*  <input type="text" value="d4-mqtt-lo-ra-wan-mobile-app"/> 	MQTT Password*  <input type="password" value="....."/> 
Published topic pattern*  <input type="text" value="POINT/PUSH"/> 	Subscribed topic pattern  <input type="text" value="mqtt/things/{DevEUI}/downlink"/> 

Figure 9.4 Connection settings in TPX

9.2.2 Packet inspection

The hive connection was established to facilitate the investigation of a complete package, including all header information. For this purpose, a HiveMQ broker was set up to receive data from all three sensors. To read the data, a local MQTT subscriber, specifically the MQTTX software, was employed. Figure 9.5 illustrates an example of the received message, with the package starting on the left side and concluding on the right side of the paper. This illustration provides insight into the available information and its structure in the JSON format.


```

{
  "DevEUI_uplink" : {
    "Time" : "2023-10-21T20:41:30.707+00:00",
    "DevEUI" : "0018B21000003BBF",
    "FPort" : "1",
    "FCntUp" : "81",
    "LostUplinksAS" : "0",
    "ADRbit" : "1",
    "MType" : "2",
    "FCntDn" : "76",
    "payload_hex" : "4c2000c634",
    "mic_hex" : "f3bc436d",
    "Lrscid" : "00000201",
    "LrrRSSI" : "-109.000000",
    "LrrSNR" : "5.500000",
    "LrrESP" : "-110.078331",
    "SpFact" : "7",
    "SubBand" : "G2",
    "Channel" : "LC6",
    "Lrrid" : "67602C96",
    "Late" : "0",
    "LrrLAT" : "59.190742",
    "LrrLON" : "9.584460",
    "Lrrs" : {
      "Lrr" : [ {
        "Lrrid" : "67602C96",
        "Chain" : "0",
        "LrrRSSI" : "-109.000000",
        "LrrSNR" : "5.500000",
        "LrrESP" : "-110.078331"
      }, {
        "Lrrid" : "67602C7F",
        "Chain" : "0",
        "LrrRSSI" : "-111.000000",
        "LrrSNR" : "4.000000",
        "LrrESP" : "-112.455406"
      }, {
        "Lrrid" : "68951796",
        "Chain" : "0",
        "LrrRSSI" : "-107.000000",
        "LrrSNR" : "-2.250000",
        "LrrESP" : "-111.279411"
      } ]
    },
    "DevLrrCnt" : "3",
    "CustomerID" : "100050938",
    "CustomerData" : {
      "loc" : null,
      "alr" : {
        "pro" : "ADRF/COMFORT",
        "ver" : "1"
      },
      "tags" : [ ],
      "doms" : [ ],
      "name" : "Comfort"
    },
    "ModelCfg" : null,
    "DriverCfg" : {
      "mod" : {
        "pId" : "adeunis",
        "mId" : "comfort",
        "ver" : "1"
      },
      "app" : {
        "pId" : "adeunis",
        "mId" : "comfort",
        "ver" : "1"
      },
      "id" : "actility:adeunis-comfort:1"
    },
    "BatteryLevel" : "254",
    "BatteryTime" : "2023-10-21T20:41:30.707+00:00",
    "Margin" : 6,
    "InstantFER" : "0.000000",
    "MeanFER" : "0.000000",
    "DevAddr" : "FC004EA5",
    "TxPower" : "2.000000",
    "NbTrans" : "1",
    "Frequency" : "867.50",
    "DynamicClass" : "A",
    "payload" : {
      "bytes" : {
        "type" : "0x4c Comfort data",
        "status" : {
          "frameCounter" : 1,
          "hardwareError" : false,
          "lowBattery" : false,
          "configurationDone" : false,
          "configurationInconsistency" : false
        },
        "decodingInfo" : "values: [t=0, t-1, t-2, ...]",
        "temperature" : {
          "unit" : "°C",
          "values" : [ 19.8 ]
        },
        "humidity" : {
          "unit" : "%",
          "values" : [ 52 ]
        }
      },
      "downlinkUrl" : "removed for scaling in report"
    }
  }
}

```

Figure 9.5 Complete message from the Comfort sensor

9.2.2.1.1 Uplink transformation

After inspecting the complete message, the implementation of uplink transformations becomes possible. As mentioned earlier, essential header information must be added to the package for D4 to accept it and determine the data destination. Furthermore, during the transformation process, any data not required for storage in the database can be removed. This transformation ensures that the new JSON includes only the information of interest, significantly reducing complexity at the database end and minimizing both data transmission size and required storage space.

In Figure 9.6, the transformation code for the Comfort sensor is depicted. Header information, including pointId, tenantID, and tenantKey, is required when using D4. The remaining content of the file comprises data intended for storage in the database. For both temperature and humidity recordings, values, units, types, and timestamps are stored. Additionally, metadata for the sensor is included, featuring position coordinates, latitude and longitude, battery low alarm, and a calculation of the battery level.

Figure 9.7 illustrates the resulting JSON text after the transformation is applied to the example in Figure 9.5.

```
{
  "pointId": "6505c5094543cbb034793ef2",
  "tenantId": "lo-ra-wan-mobile-app",
  "tenantKey": "af68048b575d18ed0b7509b9",
  "signals": [
    {
      "value": string(.DevEUI_uplink.payload.bytes.temperature.values[0]),
      "unit": "CELSIUS_DEGREES",
      "type": "Temperature",
      "timestamp": .DevEUI_uplink.Time,
      "metadata": {
        "latitude" : .DevEUI_uplink.LrrLAT,
        "longitude" : .DevEUI_uplink.LrrLON,
        "battery_low" : .DevEUI_uplink.payload.bytes.status.lowBattery,
        "batteryLevel" : (.DevEUI_uplink.BatteryLevel - 1)/253*100
      },
    },
    {
      "value": string(.DevEUI_uplink.payload.bytes.humidity.values[0]),
      "unit": "PERCENTS",
      "type": "Humidity",
      "timestamp": .DevEUI_uplink.Time,
    },
  ]
}
```

Figure 9.6 Transformation code for Comfort Sensor

```
{
  "pointId": "6505c5094543cbb034793ef2",
  "tenantId": "lo-ra-wan-mobile-app",
  "tenantKey": "af68048b575d18ed0b7509b9",
  "signals": [
    {
      "value": "19.8",
      "unit": "CELSIUS_DEGREES",
      "type": "Temperature",
      "timestamp": "2023-10-21T20:41:30.707+00:00",
      "metadata": {
        "latitude": "59.190742",
        "longitude": "9.584460",
        "battery_low": false,
        "batteryLevel": 100
      },
    },
    {
      "value": "52",
      "unit": "PERCENTS",
      "type": "Humidity",
      "timestamp": "2023-10-21T20:41:30.707+00:00"
    },
  ]
}
```

Figure 9.7 Transformation results for Comfort Sensor

9.2.3 Flows

As seen in Figure 9.8, three flows have been configured for this project, each dedicated to a specific sensor. These flows direct the stream of sensor data to their designated connection(s). Each flow scans for data sent from TPW routing, identified by a unique key corresponding to the device's DevEUI. The flow can be set up to automatically detect the required driver to

decode the payload for each device. In this case there was an error with the automatic driver detection for the temperature sensor. Therefore, the correct driver was explicitly specified in the flow for that device.

Each flow is connected to one or more connections. As a MQTT host was used to inspect the data packages, a connection to the D4 server and the MQTT host was required for each flow.













MQTT_D4_MobileApp_TempSensor	  	 MATCHED BY KEYS	<> DRIVER ACTILITY:ADEUNIS-TEMP3:1	ACTIVE
MQTT_D4_MobileApp_DoorSensor	  	 MATCHED BY KEYS	<> DRIVER AUTOMATIC	ACTIVE
MQTT_D4_MobileApp_ComfortSensor	  	 MATCHED BY KEYS	<> DRIVER AUTOMATIC	ACTIVE

Figure 9.8 Flows established in TPX

10 Dimension Four database

After evaluating three database solutions, D4 emerged as the preferred choice. Its user-friendly approach and straightforward terms of use, including a free license for small-scale users, made it stand out. The platform offered systematic guidance in its developer user guide, presenting a detailed step-by-step process for creating the database.

The initial steps involve creating a user account and subsequently configuring a tenant, space, and points. Figure 10.1 illustrates an overview of the data structure adopted within D4 for this project.

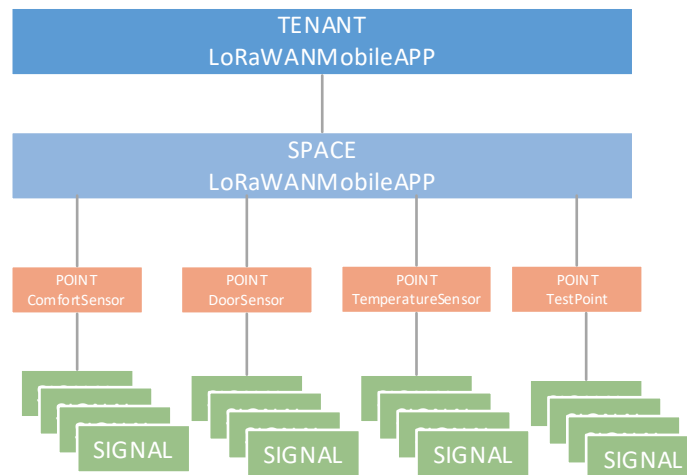



Figure 10.1 D4 General data structure







10.1 Tenant


The tenant serves as the organizational hub, consolidating all sensor-related information owned by a company or entity. Within D4, the tenant serves as a repository for database connection details, containing its ID, associated members, access tokens, and MQTT connection specifics. Figure 10.2 illustrates these tenant details for the project.

Access tokens function as the security layer for the GraphQL API, controlling permissions for spaces, points, and signals. Each access token generates a unique pass-key, allowing individual permissions tailored for various applications.

Additionally, MQTT details specify the necessary configurations for establishing MQTT publisher connections to the database. A password requirement serves as a protective measure to safeguard access.

Tenant ⓘ					
Name	Id	Created	Updated		
LoRaWANMobileAPP	lo-ra-wan-mobile-app	2023/09/16 17:07	2023/11/07 21:31		

Members ⓘ						+ Add member	
First name	Last name	Email	Role	Status			
Jan-Robin	Brustad	238972@usn.no	Owner	Accepted			
Even	Hope	238962@usn.no	Owner	Accepted			
Saba	Homayounibaghbidi	258974@usn.no	Owner	Accepted			

Access Tokens ⓘ						+ Create token	
Name							
TestAT							


MQTT ⓘ					
Username	Address	Port	Password		
d4-mqtt-lo-ra-wan-mobile-app	mqtt.dimensionfour.io	1883		

Figure 10.2 D4 tenant details

10.2 Space

At a level below the tenant, the spaces are defined. These spaces serve as organizational units, offering various structuring options to categorize points—whether by location, type, or other criteria. Spaces act as containers for points, allowing for logical organization. For instance, spaces can have hierarchical structures, where a parent space represents a broader category like a building, and its sub-spaces could denote individual rooms.

For the sake of simplicity, this project has chosen a single virtual space.

10.3 Point

Within each space, multiple points can be contained, each usually representing an individual sensor device. As depicted in Figure 10.3, the project database contains a list of points. For this project, three points have been designated for sensors, and an additional test point has been included. The test point serves as a tool for entering data points using a generator, speeding up the process of API testing between the database and the mobile app.

Name	Space	Id	External id	Created	Last active
ComfortSensor	LoRaWANMobile...	6505c5094543...	0018B21000003B...	2023/09/16 17:08	2023/11/16 14:44
DoorSensor	LoRaWANMobile...	6505c702f490b...	0018B220000001F...	2023/09/16 17:17	2023/11/16 15:24
TemperatureSe...	LoRaWANMobile...	65102bf52a6d1c...	0018B210000004...	2023/09/24 14:30	2023/11/16 15:21
TestPoint	LoRaWANMobile...	654a9e9be0128...		2023/11/07 21:31	2023/11/07 21:49

Figure 10.3 List of points in D4

Displayed in Figure 10.4 is the detailed point view within D4. Each point is characterized by a name and a unique ID. The name corresponds to the device's designation within the database, in this case mirroring the connected sensor model's name but could also be a defined object id or tag. The ID serves as D4's unique identifier for the point, utilized during data imports to specify the respective point.

Additionally, the external ID field, though optional, has been utilized in this context to reference the LoRaWAN DevEUI identifier associated with the connected sensor. Meanwhile, the metadata field, also optional, serves as a container for user-specific information in JSON object format.

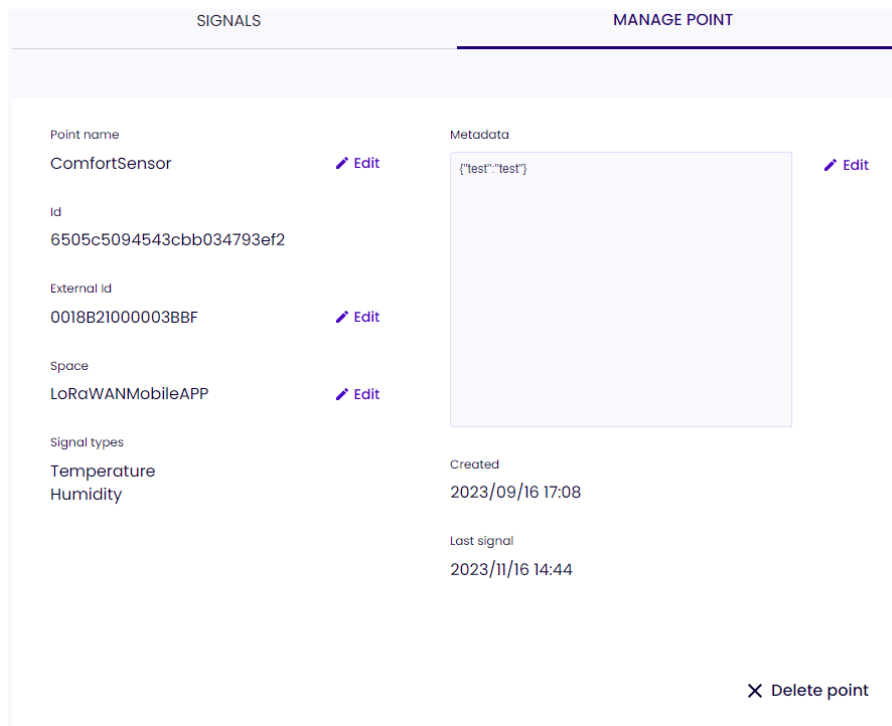


Figure 10.4 Detailed point view in D4

10.4 Signal

Within the database structure, each point accumulates signals, with each signal representing a singular measurement captured at a specific time. These signals hold various data fields detailed in Table 10.1.

Table 10.1 List of data collected for each signal [39]

Data	Description	Example Temperature Sensor
pointId	The origin point of the signal	65102bf52a6d1ce45e3792e3
type	A free text field, where you specify the nature of the signal	Temperature
unit	A set of predefined unit types for convenience and consistency	CELSIUS_DEGREES
value	The actual signal data in string format	2.9
timestamp	The signal's timestamp.	2023-10-21T20:58:10.134+00:00
metadata	A json object containing data related to the signal.	<pre>"metadata": { "latitude": "0.000000", "longitude": "0.000000", "battery_low": false }</pre>

Figure 10.5 is a window in D4 showing the 10 most recent signals from a specific point, in this case the comfort sensor.

SIGNALS			MANAGE POINT
Last 10 signals with real time data			
Type	Unit	Data	Created at
Temperature	CELSIUS_DEGREES	15.1	2023-11-16T13:44:47.526Z
Humidity	PERCENTS	55	2023-11-16T13:44:47.526Z
Humidity	PERCENTS	56	2023-11-16T12:44:47.200Z
Temperature	CELSIUS_DEGREES	15.3	2023-11-16T12:44:47.200Z
Temperature	CELSIUS_DEGREES	15.4	2023-11-16T11:44:46.870Z
Humidity	PERCENTS	56	2023-11-16T11:44:46.870Z
Humidity	PERCENTS	56	2023-11-16T10:44:46.469Z
Temperature	CELSIUS_DEGREES	15.5	2023-11-16T10:44:46.469Z
Humidity	PERCENTS	57	2023-11-16T09:44:46.196Z
Temperature	CELSIUS_DEGREES	15.6	2023-11-16T09:44:46.196Z

Figure 10.5 Signals view in D4, showing last 10 signals

11 Android development

Android development refers to the process of creating applications for devices running the Android operating system. For this project Android was selected due to its widespread global usage and open-source nature. With diverse device support, including smartphones, tablets, and smart TVs, Android offers a broad testing and deployment landscape. It features a broad online database of up-to-date courses, guides, and samples for developers. This chapter gives a summary of topics related to Android development, covering frameworks, languages, tools, libraries, and integrations.

11.1 Frameworks, language, IDE and build tools

Basic requirements for Android development include knowledge of Java or Kotlin, Android Studio (official IDE), Android SDK for development tools, Gradle for building, an Android Emulator or physical device for testing, and a solid understanding of Android components and architecture. This chapter gives an overview of basic requirements for Android development.

11.1.1 Kotlin

Kotlin is an open-source, statically typed programming language developed by JetBrains, designed for multiple platforms, including Java Virtual Machine, Android, JavaScript, Wasm, and Native. Offering a blend of object-oriented and functional programming, Kotlin is both concise and type-safe, providing advantages over Java, such as reduced code length and enhanced type safety. With a current version of 1.9.20, released in November 2023, Kotlin is free under the Apache 2.0 license and is fully interoperable with Java. It finds applications across various domains, including server-side, web, Android, desktop, and native development. Supported by major IDEs like IntelliJ IDEA and Android Studio, Kotlin has a vibrant community, hosts an annual conference (KotlinConf), and offers diverse online resources and courses for learning [40].

11.1.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Android app development. It is designed to provide a comprehensive and user-friendly environment for building Android applications. Android Studio is developed by Google and is the preferred IDE for Android development [41].

11.1.3 Android SDK

The Android SDK, short for Android Software Development Kit, is a toolset developed by Google for the Android platform, facilitating the creation of Android applications. It includes a wide array of components such as libraries, debugger, emulator, and documentation to facilitate the entire app development lifecycle. Google regularly updates the SDK to correspond with new versions or updates of the Android software, introducing additional features with each release. The SDK includes essential tools that contribute to a seamless development process, encompassing tasks from coding to debugging. Notably, Android SDK is compatible with various operating systems, including Windows, Linux, and macOS [42].

11.1.4 Gradle

Gradle is a powerful build automation system used in Android development to manage dependencies, build projects, and streamline the build process. It is the default build system for Android Studio.

11.1.5 Android emulator

The Android emulator is a tool that allows developers to test and run Android applications on their development machine without the need for a physical Android device. It emulates the behaviour of a real Android device, allowing developers to test their apps under different configurations, screen sizes, and Android versions. Android Studio includes Android emulator [43].

11.2 Integrations and libraries

In Android development, integration refers to the process of incorporating external libraries, frameworks, or APIs into your application to leverage pre-built functionalities and features. Libraries enhance the development process by providing reusable code, saving time, and promoting best practices. This chapter gives some examples of libraries and tools in Android Development.

11.2.1 Apollo Client

Apollo Kotlin, formerly known as Apollo Android, is a GraphQL client designed for Java and Kotlin multiplatform development. With its latest version, Apollo Kotlin 3, it provides a strongly-typed and caching approach for handling GraphQL queries. This client generates Kotlin and Java models from GraphQL queries, eliminating the need for manual JSON parsing or writing model types. Apollo Kotlin executes queries and mutations against a GraphQL server, returning results as query-specific Kotlin types. Notably, the generated types are query-specific, ensuring that only the requested data is accessible. The library, primarily tailored for Android, supports Java/Kotlin applications, including multiplatform projects. It incorporates features such as multiplatform code generation, support for queries, mutations, and subscriptions, reflection-free parsing, normalized caching, custom scalar types, and more. In combination with ApolloClient in Android development, Apollo Kotlin facilitates efficient communication with GraphQL servers, simplifying data retrieval and management in a strongly-typed manner [44].

11.2.2 Hilt

Hilt is a framework for dependency injection in Android app development. It is part of the Android Jetpack library and is designed to simplify the process of dependency injection in Android applications. Dependency injection is a design pattern that involves providing the objects that an object needs (dependencies) rather than having it create them. Hilt is built on top of Dagger, another popular dependency injection library for Java and Android. It provides a set of predefined components and annotations that simplify the integration of Dagger into Android projects. Hilt is specifically tailored for Android development and follows best practices recommended by the Android team [45].

11.2.3 Jetpack

Android Jetpack is a suite of tools, and libraries designed by Google to simplify Android app development. Jetpack is not a framework itself, but rather a collection of libraries that address common challenges in app development by providing ready-made solutions and best practices. It comprises components like Activity, Fragment, ViewModel, LiveData, Room, Navigation, and WorkManager, offering benefits such as enhanced productivity, consistency, and simplified lifecycle management [46].

11.2.4 Jetpack Compose

Jetpack Compose stands as Android's recommended toolkit for modern app GUI development [47]. It is designed to simplify and accelerate development through a declarative GUI framework, where one advantage is reducing the required lines of code. This toolkit leverages composable functions as its building blocks for GUI design. Composable functions receive data and generate GUI elements, like a simple text box. They are annotated with `@Composable`, informing the compiler that this function is to convert data into visual elements. Each composable is dedicated to rendering a specific screen segment. They define what should be displayed based on current parameters and state, enhancing modularity and maintainability in GUI creation.

Compose introduces a handy feature called `@Preview`, enabling the generation of a pre-compiled version of GUI elements while coding. This eliminates the need to compile and emulate the entire program to visualize the created elements.

11.2.5 Material Design 3

Material Design encompasses guidelines, components, and tools aimed at facilitating optimal user interface design [48], [49]. The latest iteration, Material 3, builds upon its predecessors and is open-source, providing a framework that organizes GUI components into three primary sections: foundations, styles, and components.

The foundation establishes a unified and adaptable system that maintains consistency across diverse platforms, ensuring a flexible user experience. Styles define the visual aspects of the GUI, including colours, elevations, icons, motion, shape, and typography.

Material 3 features a theme builder allowing exploration and design of various colour palettes, exportable to a theme Kotlin file. The theme builder is available at:

<https://m3.material.io/theme-builder>.

Lastly, components comprise the visual and interactive elements within the GUI, such as buttons, cards, and lists, facilitating information presentation.

11.2.6 MPAndroidChart

MPAndroidChart is a popular open-source chart library for Android applications. It allows developers to create a wide variety of interactive and visually appealing charts in their Android apps. MPAndroidChart supports a range of chart types, including line charts, bar charts, pie charts, radar charts, bubble charts, and more [50].

12 LoRaWAN mobile application development

This chapter gives a summary of the results of developing the LoRaWAN mobile application. The chapter is divided according to the three-layer model structure, first covering the data layer, followed by application layer and the at last the user interface. Figure 12.1 shows the domain model of the application.

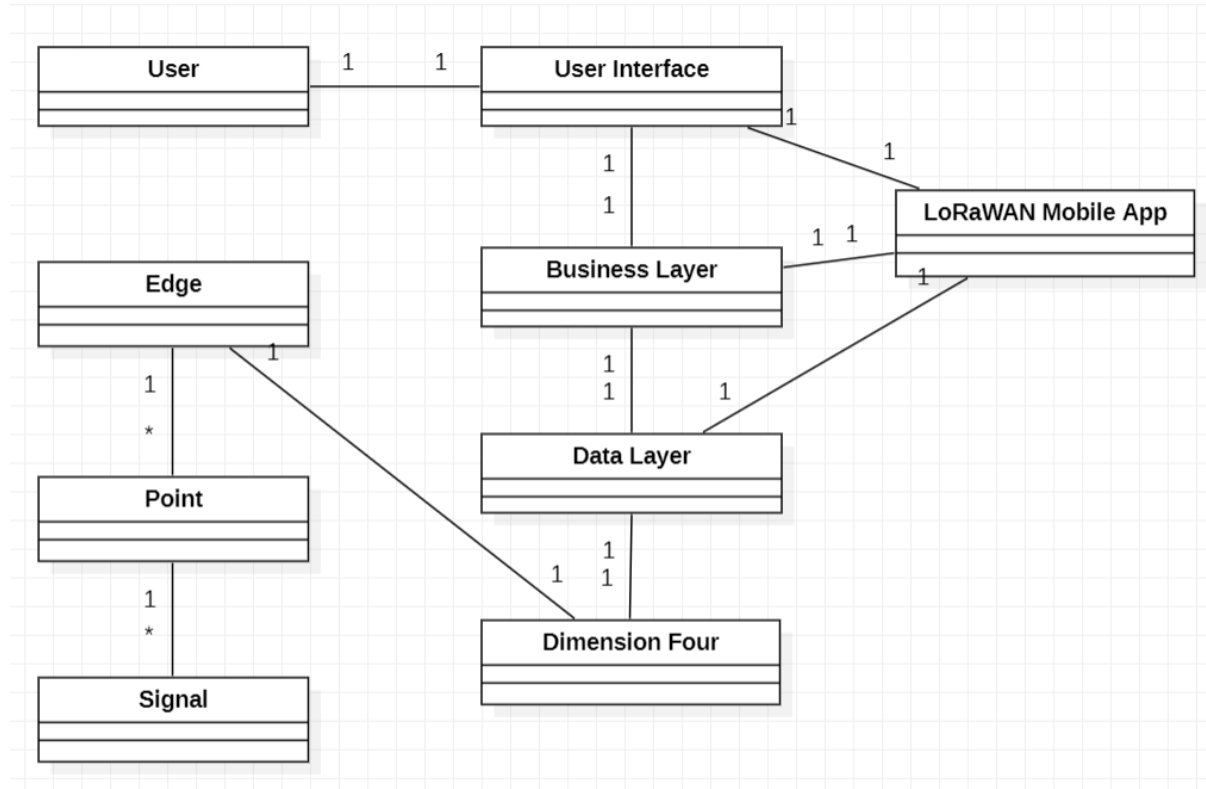


Figure 12.1 Domain model of the application

12.1 Data layer

In LoRaWAN Monitoring Application, Apollo Client acts as the data layer. Generally, Apollo Client provides a way to manage application state, perform GraphQL queries and mutations, and handle caching. In this application Apollo Client is used to perform GraphQL queries and mappers employed to transform raw data received from D4 server into a format that is more suitable for the application. In the data layer there are two interfaces to retrieve required data for presentation layer, `PointClient`, `SignalClient`. `ApolloPointClient` is a Kotlin class that implements these two interfaces. By implementing the application in this way, applying future changes such as retrieving data from a web application instead of D4 server is easier and only needs the changes in the implementation class.

12.1.1 GraphQL queries

DAO or Data Access Object is the most important element of the data layer. The primary purpose of a DAO is to encapsulate the interaction with the underlying data storage, allowing the rest of the application to access data in a more abstract and structured manner [51]. This application uses D4 as database and GraphQL as query language. GraphQL queries are used to read data from the database and the data layer of the application consists of several GraphQL queries. It should be considered that it is possible to handle CRUD (Create, Read, Update and

Delete) operation by GraphQL queries and mutations. Figure 12.2 shows the query to retrieve all points information in addition to last sampled data that have been used in PointsScreen. Figure 12.3 and Figure 12.4 show the queries to retrieve single point information and its signals that have been used in DetailedPointView.

```
query Points($lastParameter: Int!) {
  points {
    edges {
      node {
        id
        name
        description
        signals (
          paginate: { last: $lastParameter }
        ) {
          edges {
            node {
              timestamp
              type
              unit
              data {
                numericValue
                rawValue
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 12.2 Points GraphQL query

```
query Point($id: String!) {
  points( where: {id: {_EQ: $id}} ) {
    edges {
      node {
        signals ( paginate: { last: 1 } ) {
          edges {
            node {
              timestamp
              location {
                lat
                lon
              }
              type
              unit
              point {
                id
                name
                description
              }
              data {
                numericValue
                rawValue
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 12.3 Point GraphQL Query

```
query SignalsByFilter($id: String!, $fromDate: Timestamp!, $toDate: Timestamp!) {
  signals(
    where: {
      _AND: [
        {pointId: {_EQ: $id}}
        {timestamp: {_LT: $toDate}}
        {timestamp: {_GT: $fromDate}}
      ]
    }
    paginate: {last: 1000}
  ){
    edges {
      node {
        unit
        timestamp
        data {
          numericValue
          rawValue
        }
      }
    }
  }
}
```

Figure 12.4 Signals GraphQL query

12.1.2 Mappers

Mapper class in this application consists of different mapper functions that are responsible for mapping or converting data between different representations or formats. In this application

mapper functions convert read data from D4 to the models used in the application. Figure 12.5 shows the mapper class that maps data from D4 server to models shown in presentation layer.

```
package com.plcoding.graphqlmobileapp.data

import ...

@Saba Homayounibaghbidi +1
fun PointsQuery.Points.toSimplePoints(): SimplePoint { ... }

@Saba Homayounibaghbidi +1
fun PointQuery.Points.toDetailedPoint(): DetailedPoint? { ... }

@Saba Homayounibaghbidi
fun ExistingSignalsSpecificPointQuery.Points.toPointSpecification(): List<PointSpecification>? { ... }
```

Figure 12.5 Mapper extension functions

12.2 Application layer (Business layer)

The business layer of this mobile application involves managing and processing the data received from D4, providing relevant features, and potentially implementing additional functionality. There are four classes that act as services in this application. These classes return data from data layer to presentation layer, `GetPointsUseCase`, `GetPointUseCase`, `GetSignalsUseCase` and `GetAggregationInfoUseCase`. Figure 12.6 to Figure 12.9 show these classes respectively. In this application, `GetSignalsUseCase` caches the data read from D4 and uses it in next calls considering the filters and as a result improves the application performance and reduces server load.

```
package com.plcoding.graphqlmobileapp.domain

@Saba Homayounibaghbidi
class GetPointsUseCase(
    private val pointClient: PointClient
) {
    @Saba Homayounibaghbidi
    suspend fun execute(): SimplePoint? {
        return pointClient.getPoints()
    }
}
```

Figure 12.6 `GetPointsUseCase`

```
package com.plcoding.graphqlmobileapp.domain

import Saba Homayounibaghbidi

class GetPointUseCase(
    private val pointClient: PointClient
) {
    import Saba Homayounibaghbidi
    suspend fun execute(id: String): DetailedPoint? {
        return pointClient.getPoint(id)
    }
}
```

Figure 12.7 GetPointUseCase

```
package com.plcoding.graphqlmobileapp.domain

import ...

import Saba Homayounibaghbidi +1 *
class GetSignalUseCase(
    private val signalClient: SignalClient
) {
    val signalsMap = mutableMapOf<String, SignalCache>()
    import Saba Homayounibaghbidi +1
    suspend fun execute(id: String, fromDate: Timestamp?, toDate: Timestamp?): List<DetailedSignalData>? {...}

    import Saba Homayounibaghbidi +1
    private suspend fun fetchAndUpdate(existedSignalCache: SignalCache?, id: String, fromDate: Timestamp, toDate: Timestamp): SignalCache {...}
}

import Saba Homayounibaghbidi
data class SignalCache(...)
```

Figure 12.8 GetSignalsUseCase

```
package com.plcoding.graphqlmobileapp.domain

import Saba Homayounibaghbidi *
class GetAggregatedInfoUseCase {
    import Saba Homayounibaghbidi *
    suspend fun execute(signals: List<DetailedSignalData>?): AggregatedInfo? {
        val signalsWithValue = signals?.filter { it.signalData.numericValue != null }
        val min = signalsWithValue?.minBy { it.signalData.numericValue!! }
        val max = signalsWithValue?.maxBy { it.signalData.numericValue!! }
        val avg = (signalsWithValue?.sumOf { it.signalData.numericValue!! }?: 0.0) / (signalsWithValue?.size?: 1)

        return AggregatedInfo(
            min = min?.signalData?.numericValue,
            timeOfMin = min?.signalData?.time,
            max = max?.signalData?.numericValue,
            timeOfMax = max?.signalData?.time,
            avg = avg
        )
    }
}
```

Figure 12.9 GetAggregationInfoUseCase

12.2.1 Aggregation function

An aggregation function is a mathematical operation performed on a set of values to summarize or combine them into a single value. Aggregation functions are commonly used in databases, data analysis, and programming to extract meaningful information from datasets. In this application and in the business layer there is an aggregation function that extracts minimum,

maximum, and average value for each sensor. In this application, “execute” method in GetAggregationInfoUseCase uses Kotlin aggregation functions to provide aggregated information, min, max and avg for presentation layer.

12.2.2 Helper methods

A helper class in application development is a class that provides utility methods and functionalities to assist other parts of the application. Helper classes encapsulate reusable codes and serve as a collection of related functions that can be used across different parts of the application. In this application there is Helper class that converts the timestamp of read value to more readable format for the user. Figure 12.10 shows Helper object consists of stringToTimestamp method that converts String to Timestamp required for SignalsQuery filter and eliminateMillisecond that converts Timestamp to a more readable format for the users.

```
package com.plcoding.graphqlmobileapp.utils

import java.sql.Timestamp
import java.text.SimpleDateFormat

/* Saba Homayounibaghbidi +1 */
object Helper {
    private const val dimensionFourTimeFormat = "yyyy-MM-dd HH:mm:ss.SSS"

    /* Saba Homayounibaghbidi */
    fun stringToTimestamp(dateString: String): Timestamp {
        val dateFormat = SimpleDateFormat(dimensionFourTimeFormat)
        val parsedDate = dateFormat.parse(dateString.replace( oldValue: "T", newValue: " ").replace( oldValue: "Z", newValue: ""))

        return Timestamp(parsedDate.time)
    }

    /* Saba Homayounibaghbidi */
    fun eliminateMillisecond(timestamp: Timestamp): String {
        val simpleDateFormat = SimpleDateFormat( pattern: "dd-MM-yyyy, HH:mm:ss")
        return simpleDateFormat.format(timestamp)
    }
}
```

Figure 12.10 Helper object

12.3 User interface

The user interface design is based on the early GUI draft in Figure 8.5 and Figure 8.6. For this purpose an example application [52] covering the implementation of Material Design 3 was used as a template.

The user interface is divided into multiple views, where the home screen (My Sensors) is the default view and App Info is available as a tabbed view, and when a specific sensor is pressed a more detailed view will appear of the sensor data. The mobile application has been developed with color themes for both light and dark mode, depending on the mobile device setup.

12.3.1 Home screen

In Figure 12.12 the home screen is shown. Home screen will show My Sensors as default and an option to change tab to see App Info. The mobile application will list every sensor that is registered in D4 automatically and retrieve the last data for them, if the correct header is used. This is tested with the Testpoint device.

To build the home screen, the Scaffold composable from Jetpack Compose is used. This is a structural layout element for building the basic structure of the screen. The scaffold element is shown in Figure 12.11

```
@OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)
@Composable
fun HomeScreen(
    modifier: Modifier = Modifier,
    onPointClick: (SimpleNode) -> Unit = {},
    viewModel: PointViewModel = hiltViewModel(),
) {
    val pagerState = rememberPagerState(pageCount = {
        2
    })
    val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()

    Scaffold(
        modifier = modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
        topBar = {
            HomeTopAppBar(
                scrollBehavior = scrollBehavior
            )
        }
    ) { it: PaddingValues
        HomePagerScreen(
            viewModel = viewModel,
            onPointClick = onPointClick,
            pagerState = pagerState,
            modifier = Modifier.padding(it)
        )
    }
}
```

Figure 12.11 Scaffolding

As Figure 12.11 shows there is both a top bar called HomeTopAppBar and the content for the screen called HomePagerScreen.

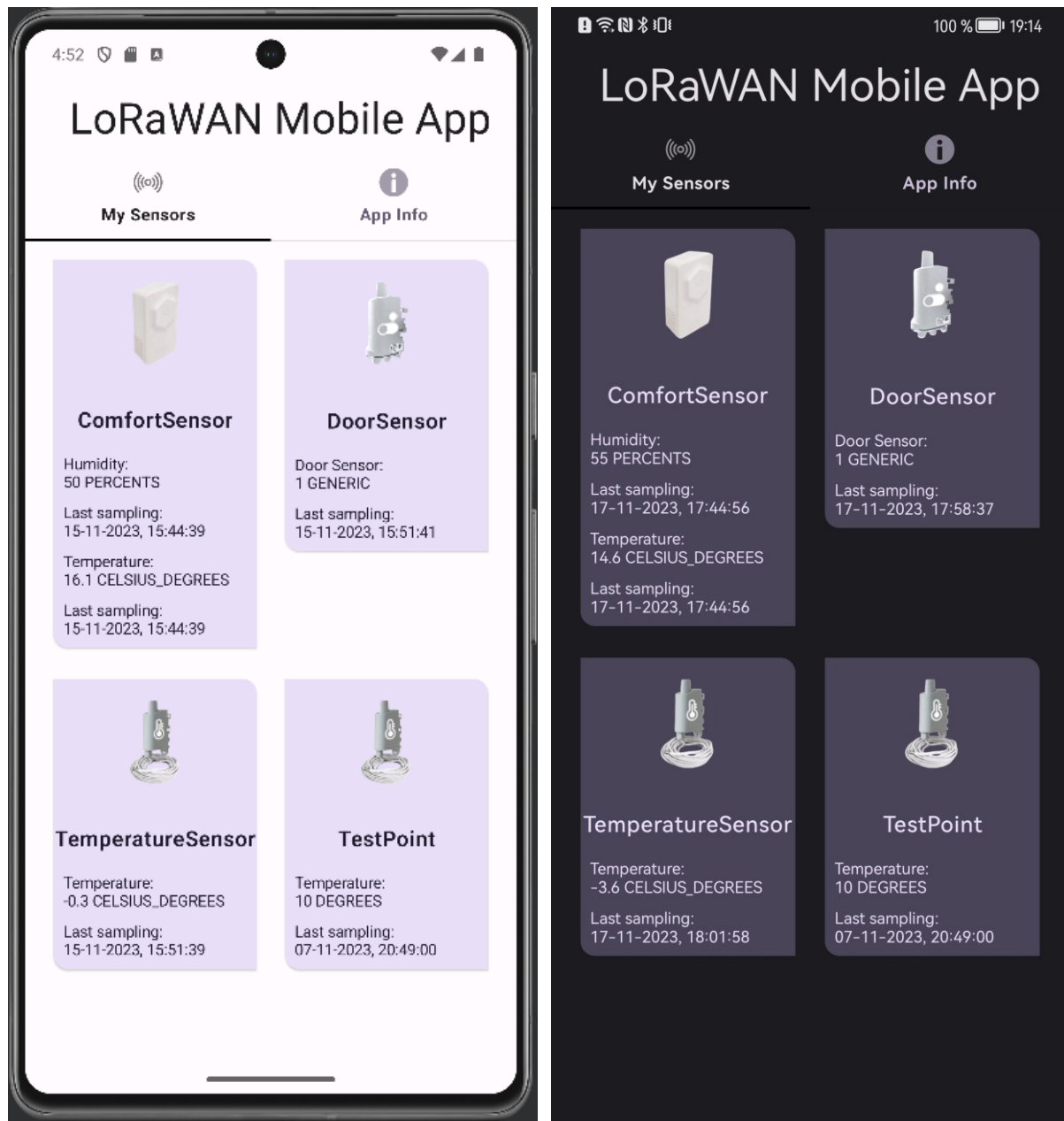


Figure 12.12 Home screen with normal theme and dark theme.

The code snippet in Figure 12.13 displays the top bar, where the text field retrieves the application name from a file containing all string constants. This structural approach to storing string resources enables the app to support multiple languages by utilizing separate string resource files for each required language. Moreover, this method prevents string duplications and ensures that modifications to a string reflect universally across the application, a strategy strived to implemented throughout the entire application design.

```
@OptIn(ExperimentalFoundationApi::class, ExperimentalMaterial3Api::class)
@Composable
private fun HomeTopAppBar(
    scrollBehavior: TopAppBarScrollBehavior,
) {
    TopAppBar(
        title = {
            Row(
                Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.Center,
            ) { this: RowScope
                Text(
                    text = stringResource(id = "LoRaWAN Mobile App"),
                    style = MaterialTheme.typography.displaySmall
                )
            }
        },
        scrollBehavior = scrollBehavior
    )
}
```

Figure 12.13 Top bar

The code behind HomePagerScreen from Figure 12.11 is shown in Figure 12.14 and Figure 12.15.

To get the tabbed view functionality, the TabRow composable shown in Figure 12.14 was implemented. This function makes the number of pages based on a class containing title and images that is needed for each page.

To load the content in My Sensor and App Info the code shown in Figure 12.15. Here PointScreenTest is used to load the sensors with relevant data and InfoScreen is used to load the content for App Info.

```

TabRow(
    selectedTabIndex = pagerState.currentPage,
    backgroundColor = MaterialTheme.colorScheme.background
){
    pages.forEachIndexed { index, page ->
        val title = stringResource(id = page.titleResId)
        Tab(
            selected = pagerState.currentPage == index,
            onClick = { coroutineScope.launch { pagerState.animateScrollToPage(index) } },
            text = { Text(text = title) },
            icon = {
                Icon(
                    painter = painterResource(id = page.drawableResId),
                    contentDescription = title,
                )
            },
            unselectedContentColor = MaterialTheme.colorScheme.secondary
        )
    }
}

```

Figure 12.14 TabRow in HomePagerScreen

```

// Pages
HorizontalPager(
    modifier = Modifier.background(MaterialTheme.colorScheme.background),
    state = pagerState,
    verticalAlignment = Alignment.Top
) { this: PagerScope index ->
    when (pages[index]) {
        LoRaWanPage.MY_SENSORS -> {
            PointScreenTest(
                Modifier.fillMaxSize(),
                onPointClick = onPointClick,
                viewModel = viewModel,
            )
        }

        LoRaWanPage.INFO_VIEW -> {
            InfoScreen(
                modifier = Modifier.fillMaxSize(),
            )
        }
    }
}
}

```

Figure 12.15 HorizontalPager in HomePagerScreen

12.3.2 App info

In Figure 12.17 the App Info page is shown. This page contains a system description with a description of the goal for the application.

The code for App Info page is shown in Figure 12.16. Here it is shown that the page is built up with a column consisting of multiple items. For the first item the system sketch is shown. The second item is a system description, which is just a title. The title is a string constant stored in the string file. The third item is a system description in text form, which is also stored in the string file.

```
@Composable
fun InfoScreen(modifier: Modifier = Modifier){
    val marginNormal = dimensionResource(id = 16dp)

    LazyColumn(modifier.fillMaxWidth()) { this: LazyListScope
        item { this: LazyItemScope
            // System image
            SystemImage(modifier = modifier, marginNormal = marginNormal)
        }
        item { this: LazyItemScope
            Text(
                text = stringResource("System Description"),
                modifier
                    .padding(vertical = marginNormal),
                style = MaterialTheme.typography.titleLarge,
                textAlign = TextAlign.Center
            )
        }
        item { this: LazyItemScope
            Text(
                text = stringResource("Welcome to the Realtime LoRaWAN Sensor Data Viewer"),
                modifier
                    .padding(vertical = marginNormal, horizontal = marginNormal),
                style = MaterialTheme.typography.bodyMedium,
                textAlign = TextAlign.Justify
            )
        }
    }
}
```

Figure 12.16 App Info code

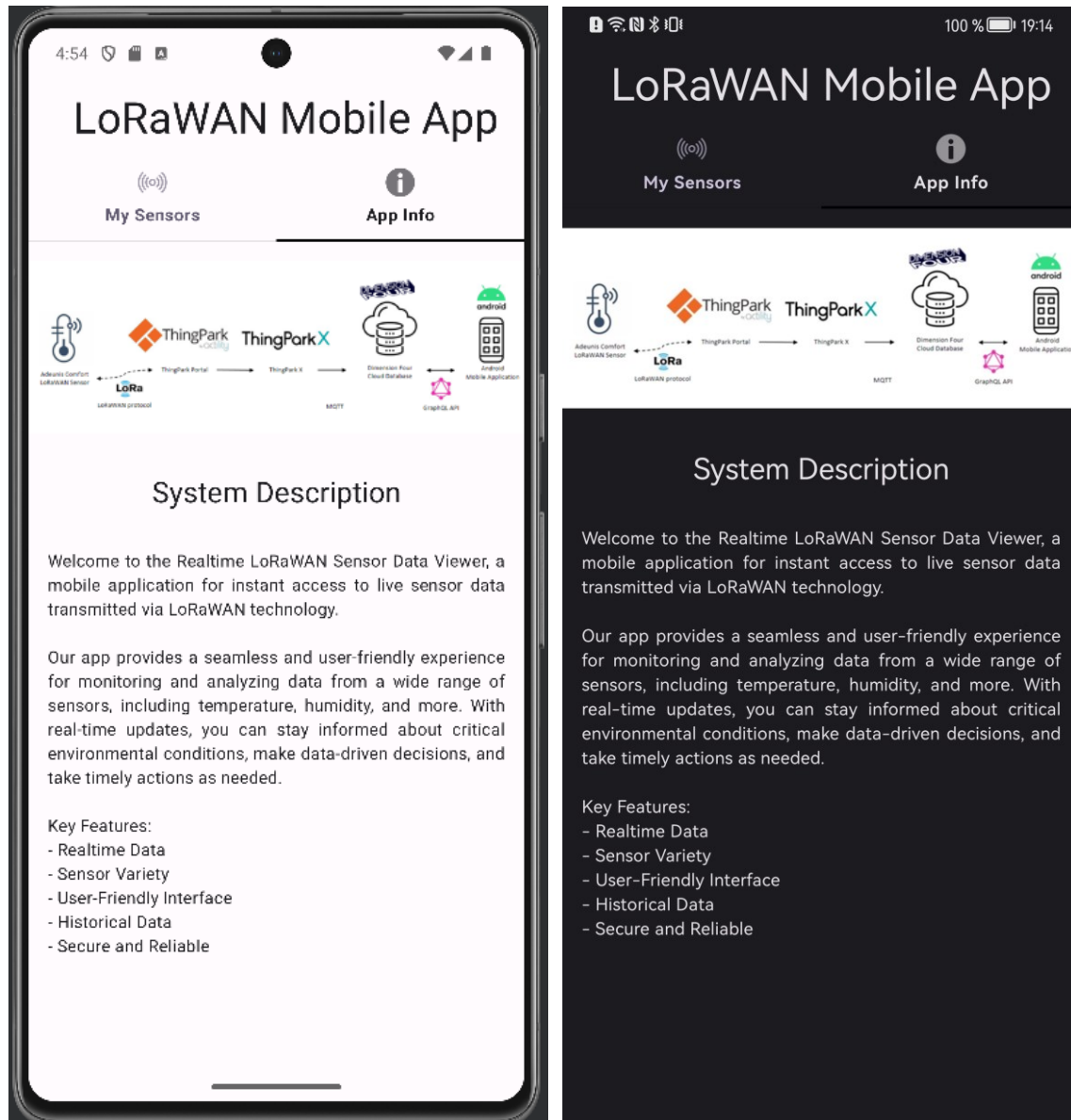


Figure 12.17 App Info with normal theme and dark theme.

12.3.3 PointsScreen

This chapter is describing the results in a more detailed view for the helper file PointsScreen. As mentioned in chapter 12.3.1 there is a function PointScreenTest which is loading the sensors. The function is shown in Figure 12.18. When the sensors are not loaded, a circular loading progress bar is shown. And if no sensors are loaded the EmptyScreen function is run which will show the text “No sensors to load”. And if a successful GraphQL query is executed the complete list of sensors will be shown as in Figure 12.12.

PointsList is a function that makes a list of available sensors that the view model retrieves. The view model has different use cases which the result for is presented in chapter 12.2, note that use cases mentioned here are not the same use cases that the Requirement and design chapter talks about.

```

@Composable
fun PointScreenTest(
    state: PointViewModel.PointState,
    modifier: Modifier = Modifier,
    onPointClick: (SimpleNode) -> Unit = {},
){
    if (state.isLoading){
        Box(modifier.fillMaxSize()){ this: BoxScope
            CircularProgressIndicator(
                modifier = Modifier
                    .align(Alignment.Center)
            )
        }
    }
    else if (state.point?.edges.isNullOrEmpty()) {
        EmptyScreen(modifier = modifier)
    } else {
        PointsList(state = state, onPointClick = onPointClick, modifier = modifier)
    }
}

```

Figure 12.18 Loading the sensors.

12.3.4 Detailed sensor view

Figure 12.29, Figure 12.30 and Figure 12.31 shows the detailed view for each sensor. The detailed view for the different sensors works similarly. The sensor Id gets passed from the home screen through the navigator as shown Figure 12.19. When the sensor Id is received in PointDetailScreen as shown in Figure 12.20 the sensor Id gets sent to PointDetailViewModel where GraphQL queries gets executed through the selectPoint function.

```

composable(
    route: "sensorDetail/{Id}",
    arguments = listOf(navArgument( name: "Id" ) { this: NavArgumentBuilder
        type = NavType.StringType
    })
){ this: AnimatedContentScope
    BackStackEntry ->
    //accessing sensorID from backstackentry
    val sensorId = BackStackEntry.arguments?.getString( key: "Id")
    PointDetailScreen(
        onBackClick = { navController.navigateUp() },
        sensorId = sensorId //passing the sensorID to PointDetailView.
    )
}

```

Figure 12.19 parts of the navigator code.

```
@Composable
fun PointDetailScreen(
    onBackClick: () -> Unit,
    sensorId: String?,
    viewModel: PointDetailViewModel = hiltViewModel(),
```

Figure 12.20 Receiving the sensor ID

```
viewModel.sensorId = sensorId
viewModel.selectPoint()
```

Figure 12.21 sensor ID gets sent to view model

```
fun selectPoint()
{
    println("sensorID is: $sensorId")
    viewModelScope.launch { this: CoroutineScope
        val sensorId = sensorId
        val nonNullableSensorId: String = sensorId ?: "2"
        _state.update { it: PointDetailState
            val signalList = getSignalUseCase.execute(nonNullableSensorId, fromDate: null, toDate: null)

            it.copy(
                selectedPoint = getPointUseCase.execute(nonNullableSensorId),
                signalList = signalList,
                aggregatedInfo = getAggregatedInfoUseCase.execute(signalList),
                isLoading = false
            ) ^update
        }
    }
}
```

Figure 12.22 View model functions.

As Figure 12.22 shows, the view model sends the sensor ID through different use cases, these use cases are responsible to handle the GraphQL queries.

When the selectPoint have retrieved sensor information it is made available in the PointDetailScreen composable, ref Figure 12.20. Now the static sensor data consisting of title, info and a picture shown in Figure 12.23 gets put in a column together with the aggregated information, graph and a table for the last 10 measurements. This is shown in Figure 12.24, Figure 12.25 and Figure 12.26.

```

val sensorData = when (sensorId) {
    "6505c5094543cbb034793ef2" -> {
        Sensor(
            id: "6505c5094543cbb034793ef2",
            stringResource(id = R.string.Comfort_sensor),
            stringResource(
                id = R.string.Comfort_sens_info
            ),
            R.drawable.comfortsensor
        )
    }
}

```

Figure 12.23 static sensor information

```

LazyColumn(
    modifier = Modifier
        .background(MaterialTheme.colorScheme.background)
        .fillMaxSize()
        .padding(top = 40.dp)
) { this: LazyListScope

    item { this: LazyItemScope
        Column(
            modifier = modifier.fillMaxWidth(),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Surface {
                Text(
                    text = sensorData.name,
                    style = MaterialTheme.typography.displaySmall,
                    textAlign = TextAlign.Center,
                )
            }
        }
    }

    item { this: LazyItemScope
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center,
        ) { this: ColumnScope
            Surface {
                Image(
                    painter = painterResource(sensorData.imageUrl), |
                    contentDescription = "My Image",
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(marginNormal)
                )
            }
        }
    }
}

```

Figure 12.24 Static sensor information for the user interface.


```

item { this: LazyItemScope
    Surface {
        Column(modifier = modifier.fillMaxWidth()) { this: ColumnScope
            listData(
                point = state.selectedPoint ?: DetailedPoint(
                    id: "",
                    name: "",
                    description: "",
                    timestamp: null,
                    type: "",
                    UnitType.UNKNOWN,
                    location: null,
                    SignalData( numericValue: 0.0, rawValue: "", time: null),
                ),
                signalList = state.signalList ?: emptyList(),
                aggregatedInfo = state.aggregatedInfo,
                marginNormal = marginNormal
            )
        }
    }
}

```

Figure 12.25 Fetching aggregated info

```

fun AggregatedItem2(
    aggregatedInfo: AggregatedInfo,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier,
        verticalAlignment = Alignment.CenterVertically
    ) { this: RowScope
        Text(
            text = "Max Value: " + aggregatedInfo.max + " ",
            style = MaterialTheme.typography.labelSmall
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = "Time: " + aggregatedInfo.timeOfMax?.let { Helper.eliminateMilisecond(it) },
            style = MaterialTheme.typography.labelSmall
        )
    }
    Row(
        modifier = modifier,
        verticalAlignment = Alignment.CenterVertically
    ) { this: RowScope
        Text(
            text = "Min Value: " + aggregatedInfo.min + " ",
            style = MaterialTheme.typography.labelSmall
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = "Time: " + aggregatedInfo.timeOfMin?.let { Helper.eliminateMilisecond(it) },
            style = MaterialTheme.typography.labelSmall
        )
    }
    Row(
        modifier = modifier,
        verticalAlignment = Alignment.CenterVertically
    ) { this: RowScope
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = "Average value: " + String.format("%.1f", aggregatedInfo.avg),
            style = MaterialTheme.typography.labelSmall
        )
    }
}

```

Figure 12.26 Composable function for aggregated info.

The `AggregatedItem2` function shown in Figure 12.26 gets `aggregatedInfo` and makes it possible to present the minimum value, the maximum value and the average value for each sensor.

```
@Composable
fun LineChartExample(signalList: List<DetailedSignalData>, sensorData: Sensor) {
    val entries = signalList.mapIndexed { index, detailedSignalData ->
        Entry(index.toFloat(), (detailedSignalData.signalData.numericValue ?: 0).toFloat())
    }

    val dataSet = LineDataSet(entries, label: "Sampled data")
    dataSet.colors = ColorTemplate.VORDIPLOM_COLORS.toList()

    val lineData = LineData(dataSet)

    AndroidView(
        factory = { context ->
            LineChart(context).apply { this: LineChart
                data = lineData
                description.text = sensorData.name + "'s data"
            }
        },
        modifier = Modifier
            .height(200.dp)
            .fillMaxWidth()
    )
}
```

Figure 12.27 Line chart

Figure 12.27 shows the implementation for a graph. It uses the `signalList` from Figure 12.25 to get the data. The `LineChartExample` uses `MPAndroidChart` as described in chapter 11.2.6 to make a line chart.

```

@Composable
private fun ScrollBoxesSmooth(signallist: List<DetailedSignalData>) {
    // Smoothly scroll 100px on first composition
    val state = rememberScrollState()
    LaunchedEffect(Unit) { state.animateScrollTo( value: 100) }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .size(200.dp)
            .padding(horizontal = 8.dp)
            .verticalScroll(state)
    ) { this: ColumnScope
        val column1Weight = .2f // 30%
        val column2Weight = .6f // 70%
        val column3Weight = .2f // 70%
        Row(Modifier.background(Color.Transparent)) { this: RowScope
            Text(
                text = "Row", Modifier.weight(column1Weight),
                style = MaterialTheme.typography.titleSmall,)
            Text(
                text = "Time", Modifier.weight(column2Weight),
                style = MaterialTheme.typography.titleSmall,)
            Text(
                text = "Value", Modifier.weight(column3Weight),
                style = MaterialTheme.typography.titleSmall,)
        }
        signallist.subList(0, 10).sortedByDescending { it.signalData.time }
            .mapIndexed { index, dataPair ->
                Row(Modifier.fillMaxWidth()) { this: RowScope
                    Text(
                        text = (index + 1).toString(), Modifier.weight(column1Weight),
                        style = MaterialTheme.typography.labelMedium,)
                    Text(text = dataPair.signalData.time?.let { Helper.eliminateMilisecond(it) }
                        ?: "", Modifier.weight(column2Weight),
                        style = MaterialTheme.typography.labelMedium,)
                    Text(
                        text = dataPair.signalData.numericValue?.toString() ?: "",
                        Modifier.weight(column3Weight),
                        style = MaterialTheme.typography.labelMedium,
                    )
                }
            }
    }
}

```

Figure 12.28 Scroll box for the last 10 samples.

Figure 12.28 uses `signallist` and makes a list of the 10 last samples.

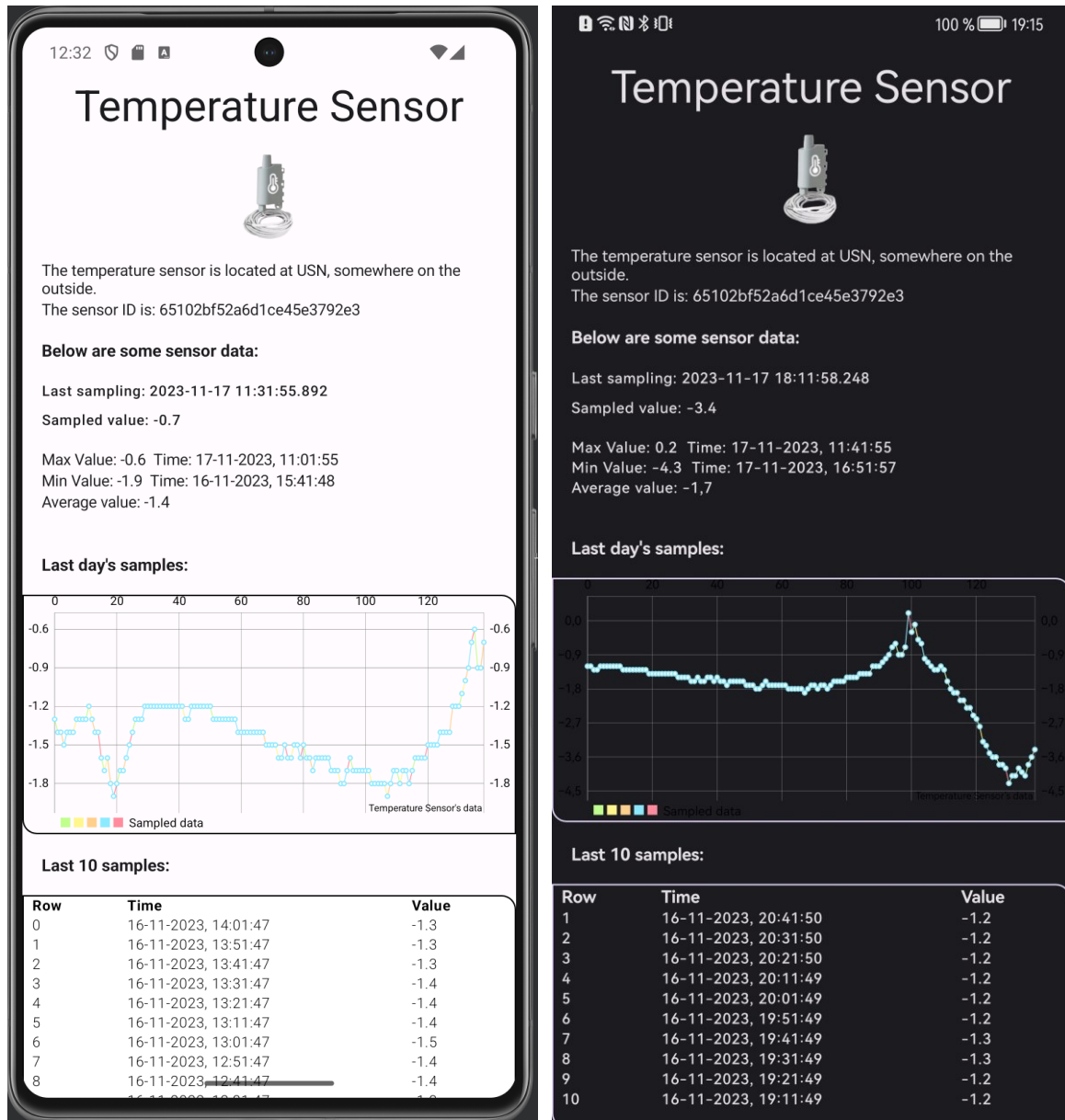


Figure 12.29 Detailed view on the temperature sensor.

Figure 12.29 shows the result seen from the user's perspective on the Temperature Sensor. Here both light and dark theme is available depending on the mobile phone setup. There is a title at the top, with a descriptive picture below and some information about the sensor. The rest is dynamic information that is loaded from D4 through GraphQL queries.

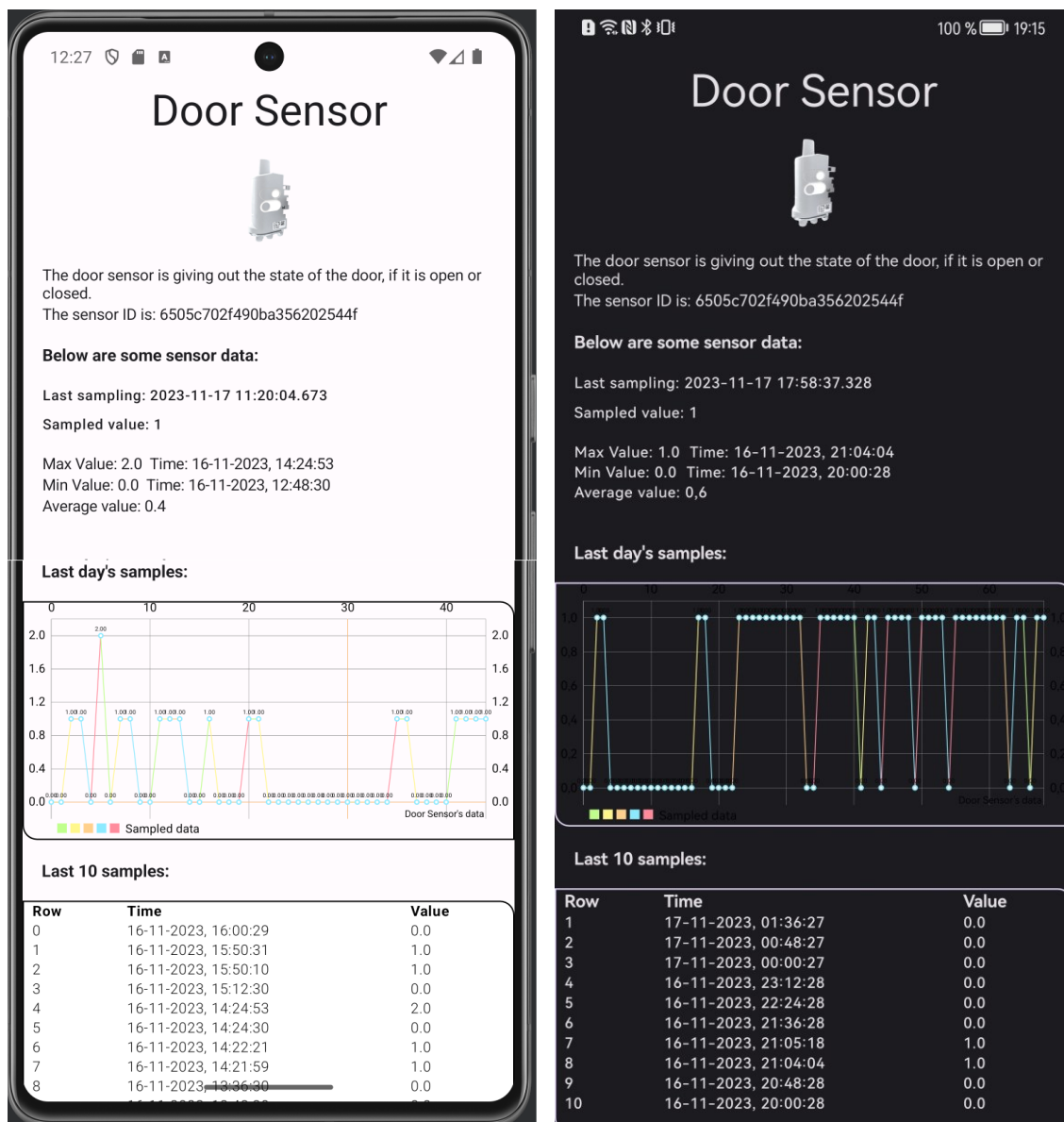


Figure 12.30 Detailed view on the Door Sensor.

Figure 12.30 shows the result regarding the Door Sensor seen from the user's perspective. The provides insights to whether the door is either open or closed.

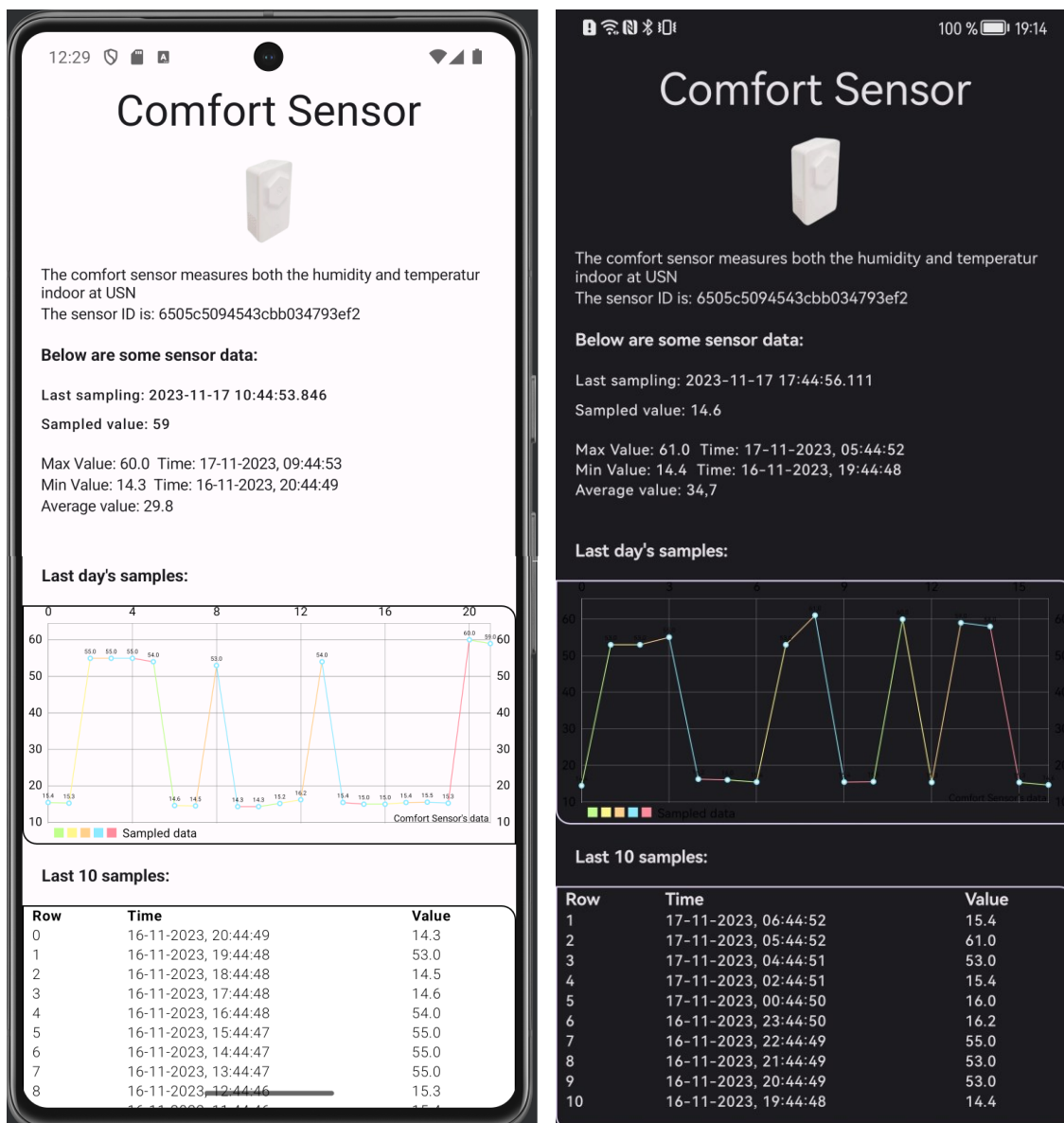


Figure 12.31 Detailed view on the Comfort sensor.

Figure 12.31 shows the result from the Comfort Sensor. The samples from humidity and temperature are mixed in the Last 10 samples and in the graph.

13 Discussion

In the early development stages, it was discovered that having constants(strings) in the application could make the code messy and less scalable for future updates. To solve this, a solution was found by going through the best Android application practices. The recommended way of treating the strings was to store these in a single file. The strings could be the heading of a page, the information text about a sensor. This way it is easy to make future changes and to keep track of the constants through Git.

One function that was originally thought of that was scratched during the development phase was the home button. The reason for this is the need of space for viewing sensor data, the GUI not being very complex with many layers, and that there exist pretty good methods that replaces the need of a home button. When pressing a sensor in the home screen a new view with more detailed sensor data is shown. To get back to the home screen it is possible to swipe from the left edge and to the right. This is more in line with modern practice when it comes to navigating in smart phones.

When looking back at the original GUI in Figure 8.5 and Figure 8.6, it deviates a somewhat from the final product. The final GUI are shown in Figure 12.12, Figure 12.17, Figure 12.29, Figure 12.30 and Figure 12.31. The reason for the changes is to make the user have a better viewing experience, and to include some modern functionality to the application. In the front page there is added a tab feature which allows for having both the home screen and an information screen App Info easily available. The sensor information on the home screen has been kept from then original GUI drawings.

The detailed view for each sensor in Figure 12.29, Figure 12.30 and Figure 12.31 has some bigger deviations than the original GUI drawings. The reasons for this have to do with how the page is structured. The page is structured as a column and then it makes sense to stack the different items in the column. For the Comfort sensor, the maximum, minimum and averages data is not correct. The reason for this is that both humidity and temperature is treated as the same signal, in the aggregation function. To fix this it is possible to filter for type before doing the aggregation. Another option is to divide the sensor in two within D4. The only problem with adding another sensor in D4 is that it is not really an independent sensor, but rather a part of one complex sensor.

To take an average value from the door sensor does provides a value that is hard to make sense of; what is an average door position? From Figure 12.30 it is shown that the average can be 0.4(light theme) and 0.6 (dark theme). The reason for this is that the average value is just the numerical representation of the average value between 1 and 0, open or closed position. It might be better to scale the value to percentage. Then it could be possible to interpret this as e.g., 60% of the time the door is open.

One issue that was detected early was that the ThingPark X automatic detection of driver type created a problem for the temperature sensors. The reason for this was that it chose the wrong driver for the specific sensor. The solution was to force the correct driver for that specific sensor. This issue was not present for the other two sensors.

As for security measures, the D4 token for accessing data through GraphQL is now hardcoded which is not good practice in regard of cyber security. Since the mobile application is not user dependant, meaning every user access the same data it was considered good enough to only make the D4 token read only. This means that the user can only pull data from D4, and not edit it.

There were also detected a problem regarding the heading, title of the mobile application. The problem was that it was not scaling correctly according to different screen sizes. This was not an issue during the development but can be a major issue if the mobile application is to be released to Google Play.

The database structure in D4 is kept simple for this project, but for more complex systems the database structure could be improved. As for now, all the sensors (points) are placed in one Space. It is reasonable to separate the points in different spaces when the number of sensors increase.

D4 incorporates subscription functionality, but an obstacle arose when encountering connection errors during attempts to subscribe to signal. Illustrated in Figure 13.1 is the request executed on the D4 playground along with its error message. To address this issue, a timer was implemented within the application, enabling the retrieval of newly added data every 10 minutes. While this solution may not achieve real-time updates, it provides a practical workaround to mitigate the problem. Figure 13.2 shows this timer.

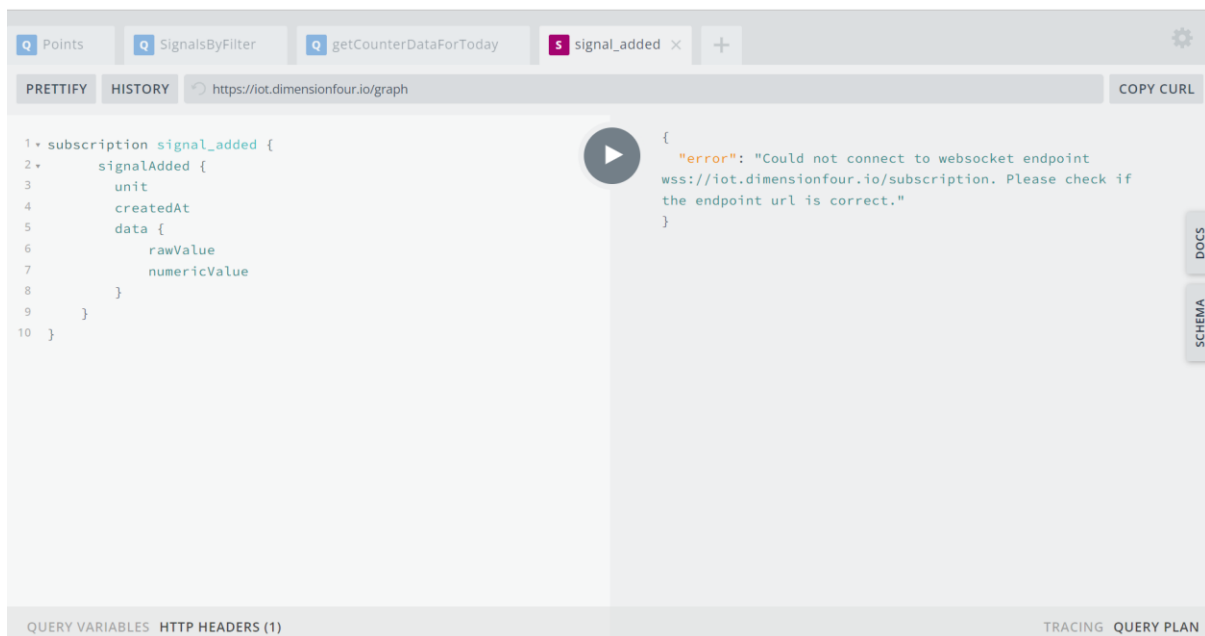


Figure 13.1 Subscription request and the response


```
package com.plcoding.graphqlmobileapp.utils

import ...

new *
class FetchSignalsTimer(
    private val getSignalUseCase: GetSignalUseCase
) {
    new *
    fun start(sensorId: String) {
        while (true) {
            runBlocking { this: CoroutineScope
                delay( timeMillis: 600000)
                getSignalUseCase.execute(sensorId, fromDate: null, toDate: null) ^runBlocking
            }
        }
    }
}
```

Figure 13.2 FetchSignalsTimer

The query to fetch information on the signal type of points proved to be significantly slow. To address this issue and enhance application performance, a helper object was implemented. This object serves to store the general information of points, eliminating the need to repeatedly retrieve this data from Dimension Four.

13.1 Further work

The D4 platform features a functionality for obtaining aggregated information through a GraphQL query. However, an issue was encountered where the query failed to return data. This seems to have been fixed by D4 quite recently, as the function now returns aggregated information as can be seen in Figure 13.3, where the query execution in GraphQL playground is shown. To address this, the function should be enabled in the mobile app. Currently a dedicated method was developed to extract aggregated information from the signal list retrieved from D4.

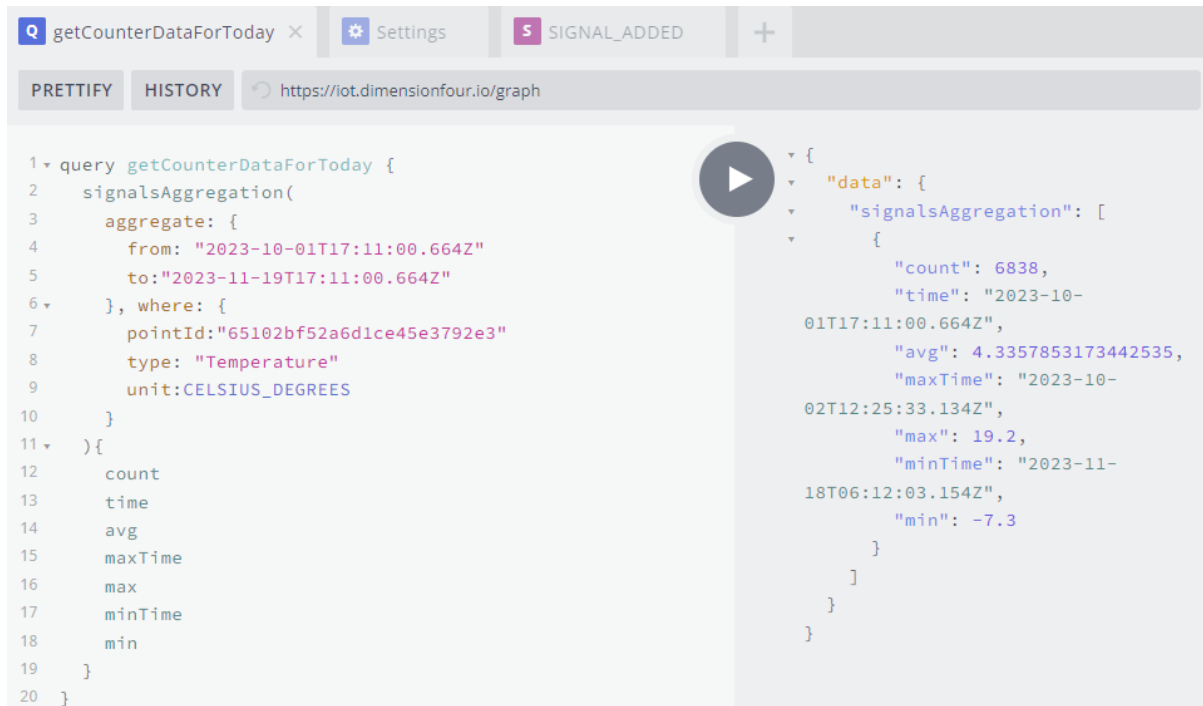


Figure 13.3 Aggregation Information query run in Dimension Four playground

Suggestions for further development of the mobile application have been identified. For example, there might be a need for a configuration page. Here it should be possible to add new spaces and points in D4. This might ease the implementation of new LoRaWAN sensors, yet as things are today it would not be possible to implement the sensors in the ThingPark environments without the supplied web GUI. If an API for the ThingPark environment existed, this could have been utilized for bringing more functionality into the mobile application.

Another function that might be needed is the possibility to filter data between different time intervals. This could be done in the detail sensor view. If implemented both the graph and table should update according to the filter.

There should also be possible to fetch the static data about each sensor, like information, name and picture from a database. This can be done in multiple ways. One way to do it is through the configuration page mentioned. Then the user would enter the information, add the pictures etc. and store it in the database. This would make the app easier to use between the different devices.

To make the mobile application a better experience for the end user there is possible to add a favorite list per user and implement the functionality to have a personal profile for each user. This should be secured with log in so that the user profile is secure against malicious cyber-attacks.

The last defined part for further work is to publish the application to Google play. The process of publishing an Android application is specified in Appendix D. To summarize, there are both general requirements and special requirements for this developed mobile application. The general requirements consist of prioritizing the user experience, conducting thorough testing, optimizing performance and implementing app store optimization. The special requirements can consist of an extra web application to handle multiple users, SSL for secure communication, authentication for multiusers, user personalization and tracking user activities to improve the user experience.

14 Conclusion

The primary objective of this project was to create a user-friendly mobile application, that can present real-time and historical data from LoRaWAN sensors in an intuitive and user-friendly manner. As part of this process, a cloud database was required to facilitate the storing of sensor data.

The project started by diving into the task description, digging into and researching the various fields. As part of this work, the project specification was decomposed, establishing a set of requirements outlining the design criteria of the mobile application.

The Altibox TPW was used to connect the LoRaWAN sensors to the network and specifying TPX as the AS route for the payload. Further on TPX was used to decode the payload followed by routing the data towards its specified location in the cloud database, based on the unique DevEUI. The chosen protocol for communication between the cloud server and TPX was MQTT. D4 was the chosen cloud server, providing an easy-to-use system for use with IoT applications. This system provided a GraphQL API for accessing and retrieving data from the server, which was used towards the developed application.

As presented in the discussion chapter, the final product has room for improvement but aligns well with the primary objective of this project. The choice of OS fell on Android, and the app was developed in Android Studio IDE by use of the Kotlin language. Up-to date libraries and tools were used to achieve the result, such as Apollo Client for handling GraphQL queries, Jetpack Compose for defining the user interface and Material Design 3 for setting the final touch on the visual components.

The app has been made scalable for future updates and was in line with the specification and requirements. The three-tier architecture was utilized to form the software, defining what parts of the system that should be handled in each of the layers. Both the collaboration diagram (Figure 8.3) and the flow diagram (Figure 8.4) from the requirements chapter aligns well with the final application. The developed software is open source, and the code is available for download through the links in appendix B.

References

- [1] R. S. Nakayama, M. de Mesquita Spínola, and J. R. Silva, 'Towards I4.0: A comprehensive analysis of evolution from I3.0', *Computers & Industrial Engineering*, vol. 144, p. 106453, Jun. 2020, doi: 10.1016/j.cie.2020.106453.
- [2] 'What is the Internet of Things (IoT)?' Accessed: Oct. 08, 2023. [Online]. Available: <https://www.oracle.com/internet-of-things/what-is-iot/>
- [3] 'IoT-aksess: Bygg IoT-tjenester på Altibox sin plattform', Altibox. Accessed: Oct. 08, 2023. [Online]. Available: <https://www.altibox.no/bedrift/iot/aksess/>
- [4] H. Jalalian Javadpour, S. Shahrokh, and M. Dizbite Ose, 'Development of Weather System with Interface to IoT GraphQL Data Platform'. USN, Nov. 18, 2022.
- [5] V. Dekhtyarev, 'Development and Testing of LoRaWAN Sensor Network for Internet of Things Applications'. USN, May 15, 2023.
- [6] H. Helgesen, 'Development of Open Source Datalogging and Monitoring Resources for IoT Platform'. USN, May 18, 2022.
- [7] V. Dekhtyarev, N. Syed Kazmi, E. Knudsen, and A. Ruwan Guruge, 'Development of Internet of Things (IoT) Solution using LoRaWAN Infrastructure for Storing and Monitoring Sensor Data'. USN, Nov. 18, 2022.
- [8] 'IoT-teknologi: LoRaWAN: Lang rekkevidde, lavt forbruk, lisensfri, lav pris', Altibox. Accessed: Oct. 09, 2023. [Online]. Available: <https://www.altibox.no/bedrift/iot/lorawan/>
- [9] M. Bloechl, 'What is LoRaWAN [2022 Update] | Link Labs'. Accessed: Nov. 04, 2023. [Online]. Available: <https://www.link-labs.com/blog/what-is-lorawan>
- [10] 'What is LoRaWAN® Specification', LoRa Alliance®. Accessed: Nov. 04, 2023. [Online]. Available: <https://lora-alliance.org/about-lorawan/>
- [11] 'LoRAWAN - Device Classes', The Things Network. Accessed: Nov. 04, 2023. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/classes/>
- [12] 'Optimise your equipment management with IoT | Adeunis'. Accessed: Nov. 18, 2023. [Online]. Available: <https://www.adeunis.com/en/company/>
- [13] 'IoT Total: Full IoT-plattform som ikke krever egne systemer hos kunde', Altibox. Accessed: Oct. 09, 2023. [Online]. Available: <https://www.altibox.no/bedrift/iot/total/>
- [14] 'Getting Started | ThingPark Wireless documentation'. Accessed: Oct. 09, 2023. [Online]. Available: <https://docs.thingpark.com/thingpark-wireless/7.2/docs/user-guide-tpw/getting-started>
- [15] 'ThingPark X IoT Flow Overview'. Accessed: Oct. 09, 2023. [Online]. Available: <https://docs.thingpark.com/thingpark-x/latest/Overview/#thingpark-x-iot-flow-in-the-thingpark-product-stack>
- [16] 'What is GIT'. Accessed: Sep. 28, 2023. [Online]. Available: <https://git-scm.com>
- [17] 'w3 about GIT and GITHUB'. Accessed: Sep. 28, 2023. [Online]. Available: https://www.w3schools.com/git/git_intro.asp?remote=github
- [18] 'GraphQL vs REST - A comparison'. Accessed: Oct. 30, 2023. [Online]. Available: <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>

-
- [19] kgreman, 'IoT concepts and Azure IoT Hub'. Accessed: Nov. 10, 2023. [Online]. Available: <https://learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub>
- [20] 'Priser – IoT Hub | Microsoft Azure'. Accessed: Nov. 10, 2023. [Online]. Available: <https://azure.microsoft.com/nb-no/pricing/details/iot-hub/>
- [21] 'What is AWS IoT? - AWS IoT Core'. Accessed: Nov. 10, 2023. [Online]. Available: <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>
- [22] 'D4 About us - Enabling the power of IoT'. Accessed: Oct. 13, 2023. [Online]. Available: <https://dimensionfour.io/about-us>
- [23] 'D4 - Choose the perfect plan for your company'. Accessed: Oct. 13, 2023. [Online]. Available: <https://dimensionfour.io/pricing>
- [24] 'D4 - What is GraphQL?' Accessed: Sep. 24, 2023. [Online]. Available: <https://dimensionfour.io/resources/concepts/what-is-graphql>
- [25] 'Learn GraphQL: Schema', GraphQL.com. Accessed: Nov. 19, 2023. [Online]. Available: <https://graphql.com>
- [26] 'GraphQL schema basics', Apollo Docs. Accessed: Nov. 16, 2023. [Online]. Available: <https://www.apollographql.com/docs/apollo-server/schema/schema/>
- [27] 'Dimension Four'. Accessed: Nov. 16, 2023. [Online]. Available: <https://dimensionfour.io/resources/developer-docs>
- [28] 'GraphQL Playground'. Accessed: Nov. 16, 2023. [Online]. Available: <https://dimensionfour.io/resources/developer-docs/api-overview/graphql-playground>
- [29] 'D4 Product -Unleashing the power of IoT'. Accessed: Nov. 16, 2023. [Online]. Available: <https://dimensionfour.io/product>
- [30] 'What is MQTT? - MQTT Protocol Explained - AWS', Amazon Web Services, Inc. Accessed: Nov. 16, 2023. [Online]. Available: <https://aws.amazon.com/what-is/mqtt/>
- [31] 'MQTT - The Standard for IoT Messaging'. Accessed: Nov. 16, 2023. [Online]. Available: <https://mqtt.org/>
- [32] 'Mobile Operating System Market Share Worldwide', StatCounter Global Stats. Accessed: Nov. 19, 2023. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide/>
- [33] 'Android Mobile App Developer Tools – Android Developers'. Accessed: Oct. 15, 2023. [Online]. Available: <https://developer.android.com/>
- [34] 'Kotlin and Android', Android Developers. Accessed: Nov. 17, 2023. [Online]. Available: <https://developer.android.com/kotlin>
- [35] 'freeCodeCamp.org'. Accessed: Oct. 15, 2023. [Online]. Available: <https://www.freecodecamp.org/>
- [36] A. Inc, 'Swift - Apple Developer'. Accessed: Oct. 15, 2023. [Online]. Available: <https://developer.apple.com/swift/>
- [37] 'Windows Phone', *Wikipedia*. Oct. 11, 2023. Accessed: Oct. 11, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Windows_Phone&oldid=1179608468

-
- [38] ‘App development - UBports documentation’. Accessed: Oct. 15, 2023. [Online]. Available: <https://docs.ubports.com/en/latest/appdev/index.html>
- [39] ‘D4 Concepts - Signal’. Accessed: Nov. 16, 2023. [Online]. Available: <https://dimensionfour.io/resources/concepts/signal>
- [40] ‘FAQ | Kotlin’, Kotlin Help. Accessed: Nov. 16, 2023. [Online]. Available: <https://kotlinlang.org/docs/faq.html>
- [41] ‘Meet Android Studio’, Android Developers. Accessed: Nov. 18, 2023. [Online]. Available: <https://developer.android.com/studio/intro>
- [42] ‘Android SDK and its Components’, GeeksforGeeks. Accessed: Nov. 16, 2023. [Online]. Available: <https://www.geeksforgeeks.org/android-sdk-and-its-components/>
- [43] ‘What is an Android Emulator?’, GeeksforGeeks. Accessed: Nov. 18, 2023. [Online]. Available: <https://www.geeksforgeeks.org/what-is-an-android-emulator/>
- [44] ‘Introduction to Apollo Android’, Apollo Docs. Accessed: Oct. 08, 2023. [Online]. Available: <https://www.apollographql.com/docs/kotlin/v2/>
- [45] ‘Dependency injection with Hilt’, Android Developers. Accessed: Nov. 16, 2023. [Online]. Available: <https://developer.android.com/training/dependency-injection/hilt-android>
- [46] Mellowacademy, ‘Introduction to Android Jetpack: Enhancing App Development Efficiency’, Medium. Accessed: Nov. 16, 2023. [Online]. Available: <https://medium.com/@mellowacademy0507/introduction-to-android-jetpack-enhancing-app-development-efficiency-6671ffef875a>
- [47] ‘Jetpack Compose UI App Development Toolkit’, Android Developers. Accessed: Nov. 19, 2023. [Online]. Available: <https://developer.android.com/jetpack/compose>
- [48] ‘Jetpack Compose – Material Design 3’, Material Design. Accessed: Nov. 17, 2023. [Online]. Available: <https://m3.material.io/get-started>
- [49] ‘Material Design 3 in Compose | Jetpack Compose | Android Developers’. Accessed: Nov. 17, 2023. [Online]. Available: <https://developer.android.com/jetpack/compose/designsystems/material3>
- [50] ‘MPAndroidChart Documentation’, Weeklycoding. Accessed: Nov. 16, 2023. [Online]. Available: <https://weeklycoding.com/mpandroidchart-documentation/>
- [51] ‘What is the full form of DAO?’ Accessed: Nov. 16, 2023. [Online]. Available: <https://www.tutorialspoint.com/what-is-the-full-form-of-dao>
- [52] ‘Android Sunflower with Compose’. Android, Nov. 10, 2023. Accessed: Nov. 10, 2023. [Online]. Available: <https://github.com/android/sunflower>
- [53] ‘What is App Store Optimization (ASO)? The in-depth guide for 2023’, <https://appradar.com>. Accessed: Nov. 16, 2023. [Online]. Available: <https://appradar.com/academy/what-is-app-store-optimization-aso>

Appendices

Appendix A – Project Description

Appendix B – Github repositories

Appendix C – Dimension Four GraphQL schema

Appendix D – How To get the app published at Google Play

Appendix A – Project description

FM4017 Project

Title: Development of Mobile Application for Integration of LoRaWAN Sensors and Infrastructure

USN supervisor: Hans-Petter Halvorsen

External partner: Altibox

Task background:

Data from different LoRaWAN Sensors located at USN needs to be stored and monitored. In this project the LoRaWAN Infrastructure from Altibox should be used and data should be stored and monitored. Since the autumn of 2018, the Altibox and Altibox partnership has expanded the LoRaWAN Sensor Network in Norway and currently has coverage for more than 1,000,000 households in 100 municipalities. Altibox and Partners now offer IoT Access as a commercial service, so that more people can use the Sensor Network for their own sensors. More information: <https://www.altibox.no/iot/>

Task description:

Examples of activities that should be performed:

- Get an overview of LoRaWAN and other relevant protocols in general and in context of this work.
- Get an overview of the Altibox LoRaWAN Infrastructure.
- Get an overview of Mobile Development in general and in context of this work.
- Development of Mobile Application for Monitoring of Data (temperature, humidity, etc.) from available LoRaWAN sensors.
- The system should be Open-source and should be available at GitHub with proper documentation.
- Microsoft Teams and GitHub should be used during project planning and development.
- The system should be properly documented in form of a technical report, documentation on GitHub and e.g., on YouTube.

The main goal with the project is to get data (temperature, humidity, etc.) from the sensors in an intuitive and user-friendly way for end-users that are available on mobile devices/smartphones.

More project details and activities will be discussed in collaboration with Altibox when the project starts. For online students it is possible to adapt the activities and the content of the project so it can be adjusted to the activities that are relevant for the company that you work in.

Collaboration with another LoRaWAN project is also expected.

Student category: IIA (both for Campus students and Online students)

The task is suitable for students not present at the campus (e.g., online students): Yes, but some time on campus to install and configure the hardware may be expected, then most of the project can be done online. A mix of Campus students and Online students is also possible.

Practical arrangements: None

Signatures:

Supervisor (date and signature):

Students (write clearly in all capitalized letters + date and signature):

Appendix B - Github repositories

Link to the organizational page for the project:

<https://github.com/LoRaWANMobileAPP>

Link to the mobile application repository:

<https://github.com/LoRaWANMobileAPP/GraphQLMobileApp>

Appendix C – Dimension Four GraphQL schema

```
type Signal {
  id: ID!
  createdAt: Timestamp!
  updatedAt: Timestamp!
  timestamp: Timestamp
  location: SignalLocation
  pointId: ID!
  parentPointId: ID
  metadata: JSONObject!
  point: Point!
  parentPoint: Point
  type: String!
  unit: UnitType!
  data: SignalData!
}

type SignalEdge {
  node: Signal!
  cursor: Cursor!
}

type SignalsConnection {
  edges: [SignalEdge!]
  nodes: [Signal!]
  pageInfo: PageInfo
}

type SignalsAggregation {
  # the bucket's start time
  time: Timestamp
  min: Float
  # the timestamp of the min value
  minTime: Timestamp
  max: Float

  # the timestamp of the max value
```

```
maxTime: Timestamp
avg: Float
sum: Float
count: Float
}
type SignalMutations {
  create(input: CreateSignalInput!): [Signal!]
}
input CreateSignalInput {
  # Either pointId or pointExternalId should be provided. if both, only pointId will be
  # considered
  pointId: ID
  pointExternalId: String
  # Either parentPointId or parentPointExternalId should be provided. if both, only
  # parentPointId will be considered
  parentPointId: ID
  parentPointExternalId: String
  signals: [SignalDataInput!]!
}
input SignalDataInput {
  value: String!
  unit: UnitType = UNKNOWN
  type: String!
  timestamp: Timestamp
  location: SignalLocationInput
  # Empty object { } by default
  metadata: JSONObject
}
input SignalLocationInput {
  lat: Float!
  lon: Float!
}
```

Appendix D – How to get the app published at Google Play

Creating and publishing a successful mobile application involves various aspects, from development to marketing. There are general requirements for all applications and some specific requirements for this application.

General requirements

Some of the general requirements to have a successful application published on the app stores are as follows:

1. **Prioritize User Experience and Design:** Design an intuitive and user-friendly interface. Prioritize a positive user experience to keep users engaged and satisfied.
2. **Thorough Testing:** Conduct comprehensive testing to identify and fix bugs. Ensure your app works seamlessly across different devices and Android versions.
3. **App Performance:** Optimize the app's performance. A fast and responsive app contributes to a positive user experience.
4. **App Store Optimization (ASO):** Optimize the app store listing with a compelling title, detailed description, high-quality screenshots, and relevant keywords. This helps improve discoverability.[53]
5. **Gather User Feedback:** Encourage users to provide feedback. Use reviews and ratings to understand user perceptions and make improvements.
6. **Regular Updates:** Stay committed to updating the app. Regular updates can include bug fixes, new features, and improvements based on user feedback.
7. **Build a Support System:** Provide customer support channels. Respond promptly to user inquiries and issues to build trust and loyalty.

Specific requirements

In addition to general aspects considered in the previous section, for LoRaWAN mobile application there are some specific aspects that should be considered.

1. **Web Application and Database:** To be able to handle some key aspects of multiuser applications, it is required to have a web application that returns user information stored in the database. The stored information can be x-tenant-id and x-tenant-key required to retrieve related data for current user from Dimension Four. It also be considered that this information should be stored in encrypted format.
2. **SSL (Secure Socket Layer):** SSL is crucial for securing communication between a mobile application and a web application. SSL ensures that the data transmitted between the mobile application and the web server is encrypted. This encryption prevents unauthorized parties from intercepting and reading sensitive information, such as login credentials, personal details, or any other confidential data.
3. **Authentication:** To change this single user application to a multiuser application one of the most important requirements is applying authentication. Authentication ensures that only authorized users can access the application. It is possible to integrate google authentication with the application to enable users to use their google account to login to the application.

4. **User Personalization:** Authentication enables the creation of personalized user experiences. Authenticated users can have unique profiles, preferences, and settings associated with their accounts. This personalization enhances user engagement and satisfaction.
5. **Track User Activities:** Authenticated sessions allow you to track user activities and behaviour within the app. This information can be valuable for analytics, personalization, and improving the overall user experience.