OULU UNIVERSITY OF
APPLIED SCIENCES

Lam Pham

DEVELOPING SENSOR MANAGEMENT APP

DEVELOPING SENSOR MANAGEMENT APP

Lam Pham
Bachelor's Thesis
Spring 2022
Degree in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Bachelor's degree in Information Technology

Author(s): Lam Pham
Title of the bachelor's thesis: Developing Sensor management app
Supervisor(s): Lasse Haverinen
Term and year of completion: Spring 2022
Number of pages: 52

Nome Oy needed a management system for their sensors, which refer to both sensors in stock and those belonging to their customer. The company wanted to keep track of the sensors' data and update their status. This thesis aims to create a system to manage the data flow of the sensor and monitor the ownership as well as the logistic process of sensors delivery. The work was commissioned by Nome Oy.

The working process started with research on current data storage, data flow management, and sensor monitoring system. Next, the system was designed and implemented based on the result of the research. A database was created to store all information about the sensors. Then, a system was built to manage data flow, update sensor data, and interact with sensors' owners. MQTT is the main transmission protocol for sensor data.

As a result, at the end of this thesis, the system passed the testing process and was running on the company server for management purposes. We create one private website for monitoring sensors on the client-side and one web application for database interaction graphically. A mobile application has been created for monitoring sensors with BLE advertisements.

Keywords: Sensor, monitor, MQTT, BLE

CONTENTS

VOCABULARY

API	Application Programming Interface
APK	Android application package
BLE	Bluetooth Low Energy
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
IT	Information Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
MAC	Media access control
MVC	Model-View-Controller programming structure
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
URI	Uniform Resource Identifier

1 INTRODUCTION

1.1 General Background

Nome Oy is a Finnish technology company, based in Oulu, which specializes in providing high-quality condition monitoring products and services. One of the main products that they are supplying to their B2B customer are sensors. These sensors measure different figures of the machine that they were attached to; the figures included temperature, vibration, battery status, and other scientific data. Data collected from this measurement decides the condition status of the machine and the sensor then sent an alarm to the machine owner if there is a problem. The system needed the following services:

- A database and data structure for sensors.
- A platform for monitoring sensors and data flow.
- Services for customers to manage their sensor.
- Sensor real-time update.

The condition of these sensors was hard to monitor manually and the company had to manage a vast number of sensors. Therefore, the main objective of the thesis was to develop a technological system that helps both technical supervisors and sensors provider easily handle sensor situations. Furthermore, the major purpose of the project was to provide sensor providers with a method for remotely controlling the condition of sensors belonging to specific customers. There are different users in this system, therefore, the system was also required for security and authentication methods. These methods aim to set a limitation of permission to an individual user and only allow users to interfere with the system data based on their permission.

The system can be described as the following:

- Data is exchanged between sensors and the server using the MQTT protocol; this data is collected and stored in the database.
- Web application was developed for interacting with the database and providing data visualization for the interface.

- A Web application provided a UI interface for managing sensors.
- A mobile application was also created to monitor nearby sensors using BLE advertisement technology.

1.2 Requirements

Requirements for this project were established at the start of this project to provide visions and scopes for the final products. The aim of these requirements was to ensure the performance and quality of the system at the end of the development process.

Firstly, data processing should be handled accurately and well-structured as the main target for the project was managing the dataflow of the system. To archive this goal, the system architecture should be well designed and the functionality of each component needs to work properly. It was important that every component in the system should work closely and in association with each other. In case of error occurred in the component, there should be warnings and immediate action must be taken to maintain the dataflow.

The second requirement is that the system should provide good quality UI to guarantee user experience. In this software, sensor data was demanded to be displayed logically for content presentation, users can search and view the data easily. Indeed, the UI needed to be well-organized and well-structured so that users can follow and understand it. Besides, the interaction between the user and the application was required to be handled on a real-time basis, every function and service needed to be executed step-by-step with quick response and accuracy. The UI software was divided into 2 modules:

1. Warehouse module: Which managed shipment information of sensors to warehouses including arrival time, calibration certificates, date of sales.
2. Monitoring module following sensor data, sensor status.

Another factor that should be considered is security. The data should be secured according to user permission. Every user has their own account which allows them to log in. Each account has permission that allows them to access a specific

field of data within their authorities and these permissions were granted by the system administrator.

For deployment environment, all the web application and API application is developed and deployed to company's Linux server running in Node.js. The database is also hosted in Linux server and store in Websockets. Moreover, the server also has duty to run MQTT client script and keep that process run forever. Finally, the mobile application should be created as APK file for testing and further deployment.

2 THEORETICAL BACKGROUND

2.1 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that runs on a V8 engine and executes codes outside the web browser [1]. Node.js allows the creation of web servers, and other networking tools using JavaScript and numerous supporting modules that help to handle core functionalities. Node.js is widely used by developers and IT companies in their projects. Moreover, it has a big and healthy community that continuously contributes new or updated libraries to this open-source service.

Node.js is outstanding in its ability to build fast in response, scalable network applications, faster and easier development, and compatibility with most OS [1]. Above all, the most important feature that Node.js provides to the current market and why most software project uses this environment is the ability to process real-time data fast in multi-user conditions.

2.1.1 React

React is a JavaScript library for building user interfaces based on UI components. The basic concept behind this library is that it allows developers to build a single web page using reusable components [2].

The application web page created in React is divided into components. Each component can also have a child component which makes it convenient to build and manage a webpage in terms of UI development. These components are created with HTML tag and JavaScript and can be written in JSX syntax which is another form that acts as DOM createElement method. Each component in React can be written as a function or a class. The components are used to tell React what should be seen in a web page screen. The method of displaying components items in one web page is called rendering.

‘State’ in React is where data in React application is declared and stored. The React component uses this state to show data on the screen. Each component can store its data by creating its own state. The React component will be re-rendered when the state is updated. The child component can also get data and function from the parent component using ‘props’ [2].

React is widely known for its efficiency in developing application user interfaces There are some advantages of using React compared to other libraries in web development:

- Easy to build and deploy
- Using component concept: Separate components into different folders and files which helps developers to improve the quality of code structure and file structure, making it easier for them to manage each component and manipulate it when they want to make an adjustment to the component.
- Using JSX syntax which helps coding easier
- Having many supporting library packages
- Quick in updating data and re-rendering
- Components are reusable
- Easy to learn and understand

2.1.1.1 MaterialUI

Material UI is a robust, customizable, and accessible Node.js library that provides ready-made advanced components that help the developer to build and develop React applications faster [3]. Material UI offers a huge of number customized styling components. The developer can also modify the configuration of the ready-made component if he wants by stating the change in the ThemeProvider component.

Material UI makes the styling implementation process quicker, easier, and more satisfying. It is well-documented so that provided features can be easy to apply. One aspect that Material UI is the favourite UI library compared to other CSS libraries is it supports components in React, every component in Material UI has

been developed with React components and can be added directly to the rendering function.

2.1.1.2 ForceGraph

ForceGraph is a UI library that provides abilities to display graph data in a 2D layout on a web platform. The graph is rendered in HTML canvas on the physic engine of d3 [4]. It is commonly used to visualize the connections between objects in a network.

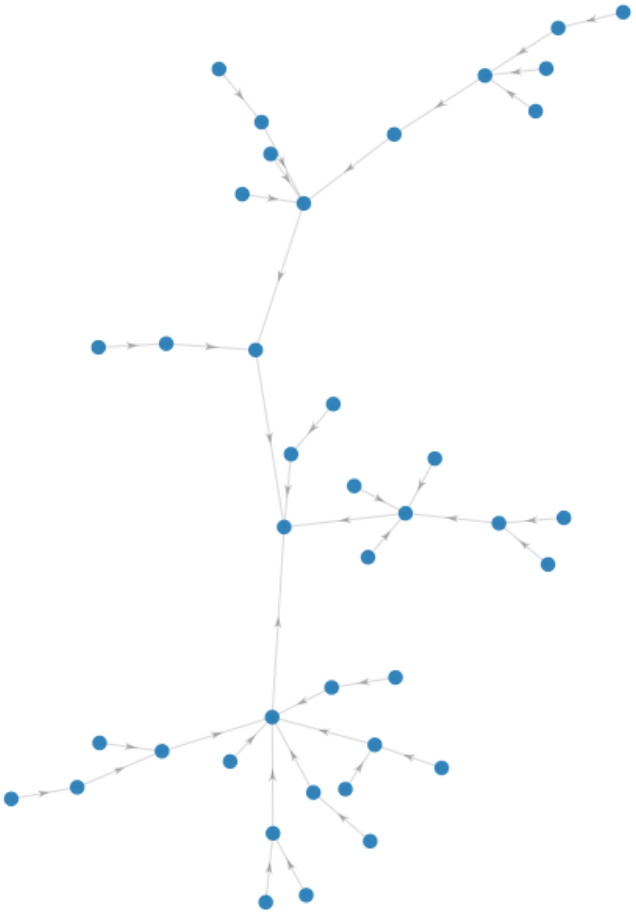

Figure 2.1 How nodes and links are shown in Forcegraph [5]

Figure 2.1 is an example of how objects and the links between them are displayed on. The module gathers a set of information about objects and links and then displays them automatically based on that data. In this layout, the user can see

clearly the relationship between objects, and all the graph data and interact directly with the data. In this project, we use this library to create a web application for visualizing data and manipulating the data on a user-friendly web platform.

2.1.2 Expressjs

Express.js is a fast assertive web application framework for Node.js. It provides various features to make web application development faster and more comfortable. Moreover, Express.js is convenient and suitable for backend development [6]. The main usage of this framework in this project's backend is creating RESTful API which accepts requests via HTTP and sends back the appropriate response. Express.js provides some core features that support back-end development:

- Providing middleware to support HTTP response
- Providing routes that allow performing different actions with HTTP methods and URL
- Constructed based on the MVC model
- Asynchronous and single thread

These features make Express.js be favourite in build API. It is fast API processing, well-structured, and easy to connect with the database [6]. Express.js is running in Node.js environment so it can be beneficial for its big community in solving problems as well as take advantage of the ready-made supporting NPM package.

2.2 MQTT

MQTT is a lightweight network protocol to transport messages. This protocol runs over TCP/IP and is used widely in the IoT field sending and receiving data between endpoint devices and the central computer [7]. This protocol uses subscribe/publish message transmission model [7].

There are 2 sides of devices in this model: MQTT broker and MQTT client. In order for both parts to interact with each other, the client has to make a connection

request to MQTT broker using username and password as authorization credential. After passing all security step, MQTT client can listen and send information to the broker.

Publishing: MQTT client send a packet information including a message and a topic. When the broker receiving the packet, they will send this message back to all of the clients that subscribed to the topic. This response will be acknowledged depending on the QOS. There is no acknowledgement with a message QOS 0 but QOS 1 and 2 are acknowledged by the server and will send back the message until receiver get the packet successfully [7].

Subscribing: MQTT clients can subscribe to multiple topics from the MQTT broker and receive specific data according to the topic that they are subscribing to.

2.3 PM2

PM2 is a process manager for JavaScript runtime. It allows the system to keep a Node.js application alive forever and automatically reload the application without downtime when the server is rebooted [8].

In this thesis project, PM2 is used to manage the running process of our API application and MQTT client; both applications are written in Node.js. One key aspect of the connection between MQTT client and broker is keeping alive, therefore, it is important to use PM2 to monitor the runtime of this application.

2.4 Janusgraph

JanusGraph is a database designed to process graph data with large storage and computational capacities that beyond a single computer can provide [9]. JanusGraph's outstanding characteristic is scaling data processing for real-time traversals and analytical queries [9].

Graph Database is a kind of database that uses graph structures with nodes, edge, and property data. One of the main concepts of graph database is the relationship (or edge or link). The purpose of graph database is to store data in graph theory. Mathematically, the graph model is made of vertices and edges (10.). Vertices represent the objects in the database while edges represent links between different nodes. In comparison with Graph database, the relational database also stores the relationship between objects, but it appears as the relationship between two sets of data, while the relationship in the graph Database is directly created between two objects. The relationship can also store property to describe more details in the connection between objects.

Different nodes that have the same relationship to a specific node create a network of nodes. As a result, this is easier to create groups of networks and separate a big data set into small groups of data sets which make it easier to manage and traverse for specific data. There are 3 elements in a graph database:

- Node: Each node has one label which defines the type of the node. Nodes that share the same label can be considered to be at the same level.
- Edges: There are 3 aspects that are needed for one edge to be created the start Node, the end Node, and the label of the relationship. Each edge is responsible for connection and shows the relationship between 2 nodes.
- Properties: Property is the data that is stored inside the object. Each node has a set of properties. These properties are the node attributes that can be modified and used in queries.

This kind of database brings some benefits compared to other relational or non-relational databases [10]:

1. Object-oriented thinking: every node acts as an object and it appears in the same platform as each other. It is easier to query and traverse without having much knowledge about the data structure, unlike in relational SQL you have to know all the tables that you are querying for index using FROM [10].
2. Scaling: Graph database can store over big numbers of nodes as it has saved relational information in the edges, so it leaves more storing

capacities for nodes and properties. This is so-called managing big data [10].

3. Updating data in real-time while supporting queries at the same time. There are queries model that help search for vertices and update data at the same time [10].
4. Power full recursive queries: Graph provides queries that are complicated in terms of logic and the structure of data also help user to execute high-level queries for retrieving data whereas in relational SQL you have to include a lot of JOINS command in the query [10].
5. Flexible grouping: It is easy to group object that shares the same common ground: label or connect to the same node, each small group that has common features then create a new group. It is different from a relational database when it is hard to group data as their properties are displayed in different tables [10].

2.5 Node-Red

Node-Red is a flow-based development tool for visual programming [11]. The main purpose of Node-Red is to wire together hardware devices, API, and online services as a part of the Internet of Things. It provides a flow-based editor which makes it easier to manipulate the flows using a wide range of nodes that can be deployed in single click runtime [11].

With the help of Node-Red, the developer only has to configure nodes and functions and wire them to make a sequence for services. There are two types of nodes: inject node and function node. Inject node sends a message to others without requiring input. On the other hand, the function node receives input message from the previous node and runs the algorithm to process the input and then sends the processed output to the next node in the sequence.

Based on that convenience, Node-Red is used to manipulate our MQTT data flow and trigger a telegram bot to send an alarm message to the system manager when the inject node receives an alarm message published from our MQTT client.

2.6 Development tools

2.6.1 Git

Git is an open-source distributed version control system designed to handle everything from small to large projects with speed and efficiency [12]. Git tracks changes in all sets of the files and is used for coordinating work among collaboratively developing source code during software [12].

Github is an internet hosting for software development and version control using Git. Most projects, whether personal or collaborative, use git and Github to manage work processes and version control.

Version control is where every change of the projects is saved, and the developer can have access to the history of modification of the files and other people who want to reuse the code or collaborate in the development process can understand the meaning of the change and code. Not only being used for version control, but Github also provides CI/CD options for team projects such as branch merging, code review.

2.6.2 Visual Studio Code

In this thesis, Visual Studio Code was chosen as the code editor for this project. VS Code is a software that provides developers with tools to write code and manage files in a project repository [13]. There are many advantages in using Visual Studio Code in the project:

- It is a free and lightweight code editor.
- Easy to manage workflow, folder, and files.
- Provide extensions that support the syntax of multiple programming languages.
- There is a terminal in the application to execute code and see logs.

2.6.3 Postman

Postman is an open-source application that provides tools to perform API requests to a specific API's URL [14]. With Postman, developers can perform

different kinds of requests (GET, POST, PUT, DELETE). Postman also provides tools and utilities for making an API request including request header configuration, adding data for request body and authorization then displaying the response after sending the request [14]. Therefore, within this project, the author uses Postman to test and evaluate the performance and accuracy of his RESTful API.

3 SYSTEM ARCHITECTURE

3.1 System Model

System architecture, in information technology, is a conceptual model that defines the structure and behavior, and views of the technical system [15]. An identical system architecture consists of multiple components and the relationship between them and how each component reacts to each other. A system which had a good technical architecture can gain plenty of benefits [15]:

1, Ensure that everything works together.

2, Enable quicker changes in the system during the development process.

3, Provide clear visions and principles for the development and implementation process.

4, Identify the need for the technical requirements.

5, Identify the needs for a financial plan.

In the project, the C4 model was used as the basis for our structure visualization and architecture guidelines. C4 model is a diagramming software architecture, based on the abstractions that reflect how architects and developers think about and build software [16]. In this thesis, our system displays 3 levels of the C4 model: Context, Containers, and Components.

Context:

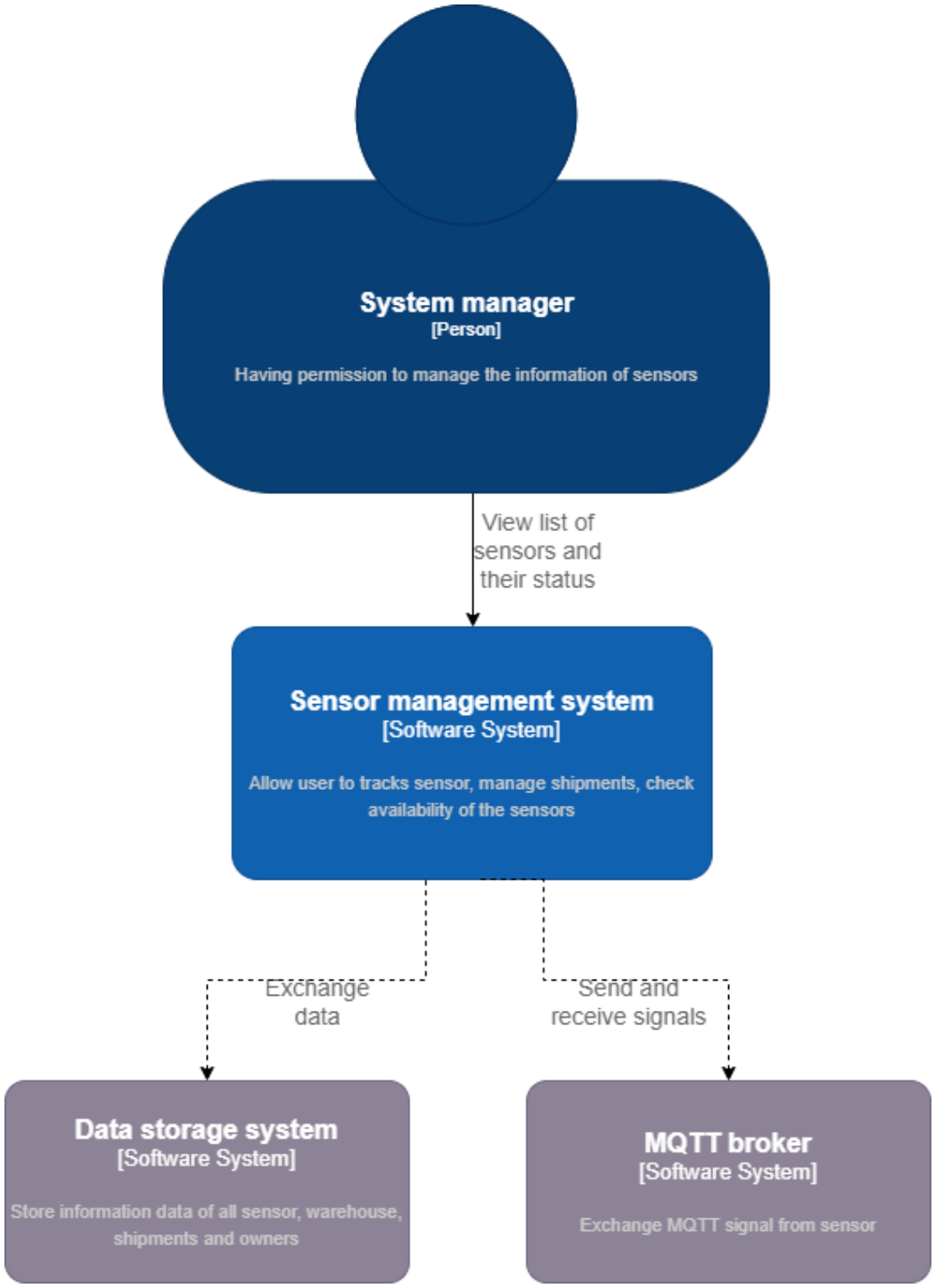

Figure 3.1 System in Context-level C4 model

The context diagram shows how the overall process works and most importantly defined the main output of the internal system [16]. In Figure 3.1, the main user of the system or system manager was the one who had the permission to view and interfered with the data processing; this person can be the sensors providers or the owner of specific sensors.

The system is the center of the data process. The first important functionality that the system had was exchanging data with the database. This activity includes getting data from the database, creating data structure and schema along with making modifications to the database. Another task of this system was using listening and sending MQTT requests to an MQTT broker in the outer server in which the sensor will reply to the request with a response of their data. Users can access the system to view the data and request to trigger the system to use its logic for monitoring data. This action can be done both manually and automatically.

Container:

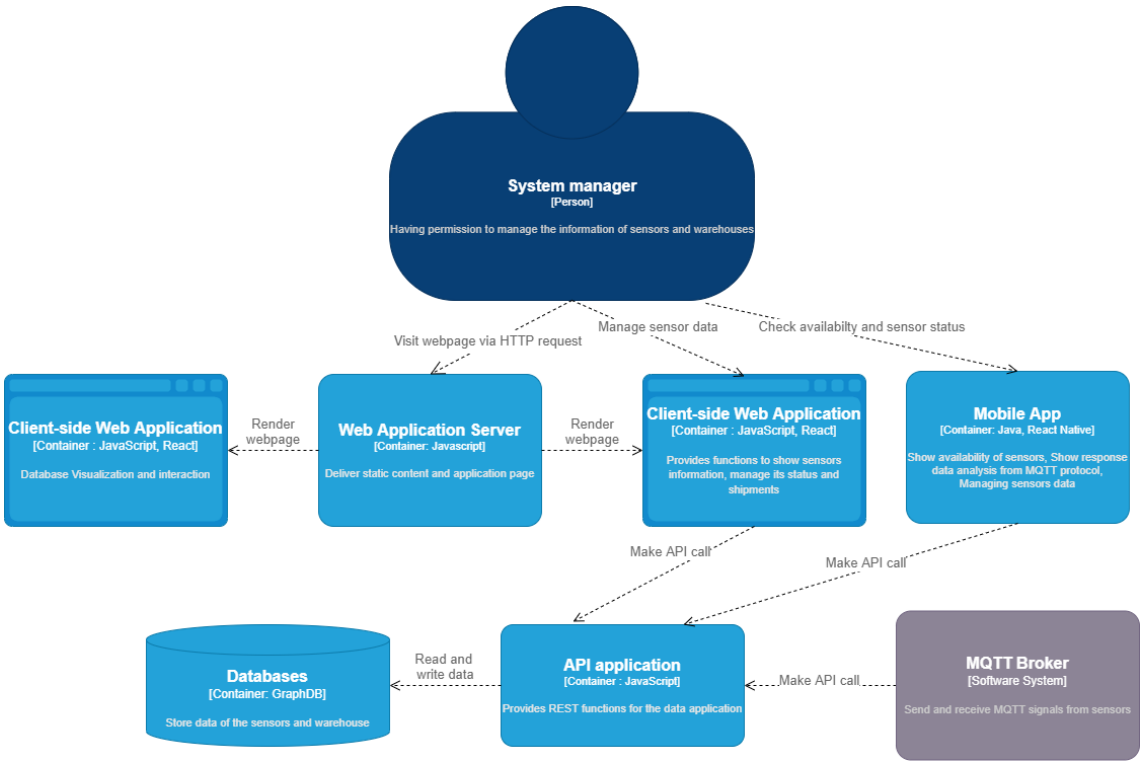

Figure 3.2 System in Container-level C4 model

The container model describes the system in more detail. It emphasizes the applications in the system and how they interact with others. In the figure 3.2, the model shows a list of applications in the system and the life cycle of the whole system. It also illustrates the actions that the system manager can do with front-end applications as well as the types of action that each application needs to take in relation to others for the system to run smoothly and reduce unwanted errors.

There is an API application where it receives API requests from 3 front-end applications and an MQTT broker. This request can be triggered by the user and by the MQTT server. The API, which is connected to the databases, has access to read and modify data in the databases using queries. When API received a request, it starts analyzing the input from the request and processing data to make a response to the client whether the process is a fail or a success. The client-side uses the data received from API and then displays it to the user.

Component

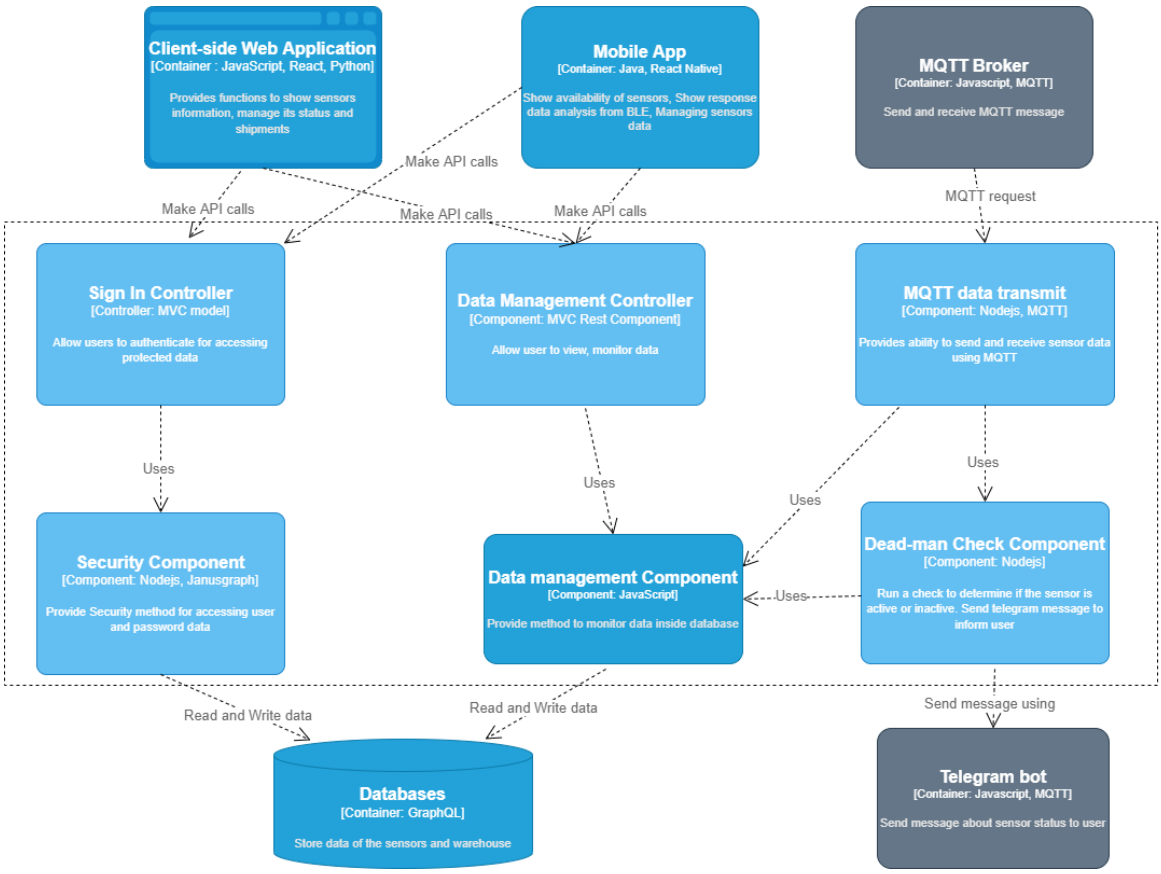

Figure 3.3 System in Component-level C4 model

The final level of architecture in our system is the component model. In this model, the application breaks into multiple components. The main purpose of this model is to show all the functional modules needed in the system and the relationship between the components themselves. Therefore, this concept helps developers to have a clearer view of what modules in the system should be built and how the components should work with each other as well as with other

external systems. Moreover, another important benefit of the component model is to prevent missing functional requirements in the development process to ensure the durability of workflow because all required components that need to be implemented are displayed in this model.

In Figure 3.3, the area covered by a dotted triangle represents the API system consisting of components and their controller. Modules are created based on the function that the front-end system wants to deliver. Each module consists of a controller and a component. The controller receives requests from the front-end client and transmits the request to other components where data is taken from the database and other logical processes should be done in these components. The result of these processes will then be used by the controller to respond back to the client-side, while some processes can send requests to outer service to continue the workflow. In the sensor management API case, 3 controllers will use the “Data management Component” to get data and services from the database and then use that data to respond back to front-end applications, while the “Deadman Check” use both data from MQTT Broker and “Data management component” to check sensor status to Send an MQTT message to request Telegram bot, which is an external system.

3.2 MQTT broker and Client

MQTT broker is a piece of software running on a server. The broker acts as a post office. MQTT clients do not connect directly to the server for data requests but use a subject called “topic”. Every MQTT clients which subscribe to the significant topic get messages from the broker. These messages are published continuously from other clients, which in this case are sensors. Depending on the types of sensors, after every significant amount of time, a sensor will publish a message to the topic stating its condition; this message data is in Buffer format. MQTT clients can also publish a data message to the broker. MQTT broker server uses TLS encryption with username and password for security protection and it also requires certification from the clients. There are some advantages in using MQTT broker to send and receive data:

- Prevent vulnerable and insecure connections from clients

- Can connect and send data to a huge amount of clients.
- Reducing network strain.

In the sensor management system, the MQTT managing component in API will subscribe to a topic in the MQTT broker to receive a message with data of sensors then these data will be examined and saved into databases. The “Dead-man check components” also use the MQTT signal to publish topics to the broker to trigger the Telegram bot to send an alarm or notification to the user. MQTT client is written in Nodejs and uses MQTTjs package for implementation.

3.3 Front-end

The front-end refers to the layer of software which includes all the applications and software that relates to the user interface. The front-end is a component in system architecture where the user interacts and experiences the service. In the system, the main purpose of front-end components is to provide a platform for managing sensors which requires usability and user experience. This will help users easier to use the system, manipulate all the requests and actions to the back-end side and have a clear view of data as well as the data structure. In sensor management systems, applications that belong to the front end include web applications and a mobile application.

3.3.1 Hybrid mobile application

A native mobile application is a mobile application that is developed using a native programming language and tools specific to that platform [17]. For example, a mobile application used in IOS is written in Swift or Objective-C using Xcode while an android application is written in Kotlin or Java using Android Studio. The advantage of this kind of application is that there are libraries and tools that support specific programming language to help the application performs better and the developer more comfortable with the development process. However, there are some hindrances in developing a native application if a developer wants to build an application that is compatible with multiple OS. For instance, a program written in Swift cannot run in Android-OS while a Java application is unable to work in IOS. So, it means that if developers want to build an application

that is compatible with multiple OS, they have to write codes in different programming language for different platform.

A hybrid mobile application is an application written using web application technologies (HTML, CSS, JavaScript) and then encapsulated in a native application [16]. For better understanding, this app acts as a web application, but instead of being shown on the web browsers, it is run within a native application. This application can be compatible with both IOS and Android.

Developing a hybrid application helps developers with some aspects compare to a native application:

- Developing an application that is cross-platform support
- Code can be reused to develop instead of building new for different OS
- Time-saving for the development process

For these reasons, in our system, the hybrid theory is applied to our mobile application which helps developers to build code once, and then the app can run on different operating systems. The main purpose of the mobile application in front-end development is to scan sensor data using BLE and use that technology to update and monitor sensor data through a Bluetooth connection. The result of the application is APK file hosting on Google play on Android.

3.3.2 Web application

Our web application's main purpose is to manipulate sensors' data and monitor different aspects of a sensor. This application provides a graphical and visual display of sensor data and relationships between owners and their sensors. The application is written in React with Nodejs development environment with Material UI for customization and design system for the user interface. The website is hosted by Linux server.

3.4 Back-end

As for the back-end development, the system uses RESTful API to manage the dataflow from the databases to clients. These clients include front-end applications and the MQTT broker. The main functions of this API are receiving HTTP requests by the client, reading the request, getting the needed data from the database, and processing that data to get a suitable response.

API is a set of definitions and protocols for building and integrating application software. It is sometimes referred to as the contract between information providers and information users – establishing the content required from the consumer through API call and the content required by the producer (the response). In another word, if the user wants the system to retrieve data or perform a function, API help the client communicate with the system so that it can understand and fulfil the request. The system then also uses API to respond to the client according to the request. API acts as a bridge connecting the clients and resources and services, helping both sides to communicate and deliver services.

RESTful API is an API that has patterns of REST architectural styles and allows systems to interact with RESTful web services. REST is a set of architectural constraints. When a client request is made via Restful API, it transfers a package of information via HTTP in different formats; the most popular format currently used is JSON. The information inside this HTTP request can be metadata, URI, authorization, caching, cookies [18]. There are some criteria that API can be considered to be RESTful:

- The architecture of the system includes clients, servers, and resources with requests managed through HTTP.
- Stateless: A request from the client must contain all of the information needed for the server to understand. The application cannot be stored any information on previous requests on the server.
- Cacheable: requires that a response should implicitly or explicitly label itself as non-cacheable or cacheable.

- Layered system: The progress involved in retrieving and processing data is invisible to the client [18].

In our system, we use Nodejs end Expressjs to develop API. There are advantages of using Expressjs to build Rest API.

- Easy to connect to Databases
- Creating routes for API calls
- Data is protected with security components
- Data processing is hidden inside API functions
- Easy to configure and customize
- Error handling middleware [18].

The API has a connection to a database which is also located in the Linux server. It gets services from the database by sending request which includes queries of instructions. The Database that the system used is JanusGraph, a Graph database that stores data of sensors, and system users. After having access to the database, API can create CRUD (create, read, update, delete) functions with queries written in Gremlin language. In the end, the expected output of the API services is a response to the client.

3.5 Database

3.5.1 Requirement

Two requirements are considered to be important and prioritized in this database which are data persistence and scalability. The system needs to store data from a big number of sensors; in the future, this number can increase to hundreds of thousands. Therefore, this demands for a database that has scalable storage capacity.

The database is responsible for storing sensor data, sensor status history and data of owners and the relationship between sensors and owners. Individual sensor gets their new data every 3 hours and with the numbers of sensors equal with the numbers of requests for retrieving and updating data. A request for sensor data update includes quite many properties that need to update. For the

database to perform accurately and securely under these circumstances, the query algorithm that is used in the database needs to be easy to use and can be iterated.

Based on the requirement, JanusGraph database is applied to the project as a type of database for the system. As mentioned in section 3.1, JanusGraph is an open-source and scalable database. Its structure is based on Graph database which is applied to manage relationships and it uses Gremlin query language for traversing and updating data.

3.5.2 Schema and data structure:

In the system, the system has different modules and one of those modules is sensor management. The main purpose of the sensor management module in the database is to store the data of sensors and owners. Using graph theory can simplify the way to control both data and relationships.

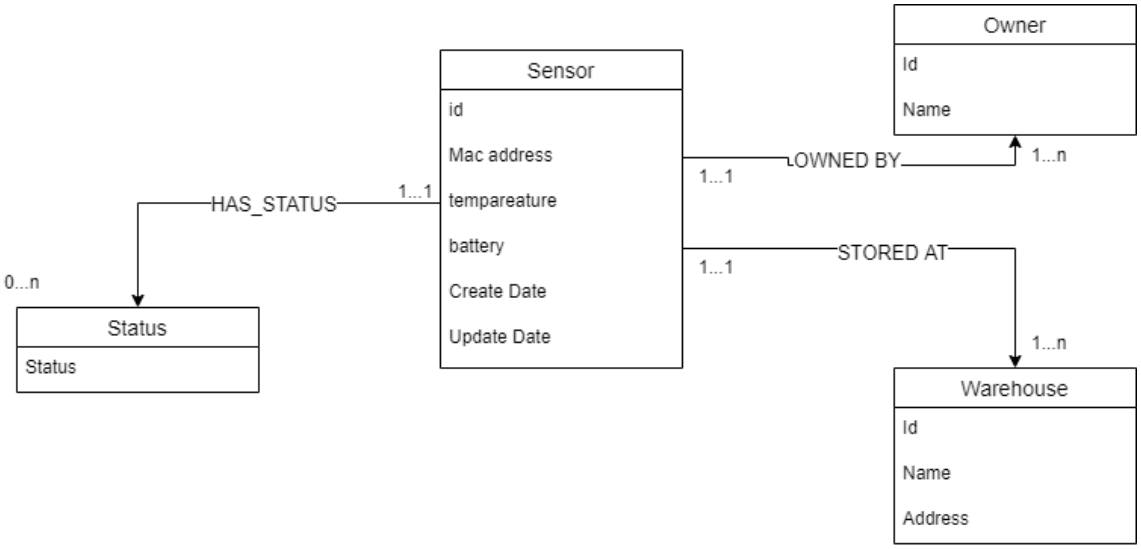

Figure 3.4 Database schema of sensor management

Figure 3.4 illustrates the schema of the database. Graph database provides a schema that is very easy to understand and straightforward. There are 4 types of nodes in this schema.

Sensor: a sensor has some basic properties like name, mac address which are unique properties, the date that a sensor node is created to the database, and all

of the data that has been measured by the actual sensor including the timestamp when the data is updated.

Status: The sensor object has “HAS_STATUS” one-to-one relationship. One sensor will get only one status node to follow it to decide whether it is active or inactive.

The diagram also shows the relationship between the sensor and owner as well as the warehouse. One sensor can be owned by only one owner and stored at one warehouse. However, one owner can own multiple sensors and one warehouse should store multiple sensors.

4 IMPLEMENTATION

4.1 API Application

API application is running in the environment of Node.js and is written in Express.js framework. The aim of the API application is to provide RESTful API in order to transmit data between the Client application and server via HTTP. It provides CRUD requests for the client-side; this includes GET, POST, PUT, and DELETE.

The API is connected to the JanusGraph database using a remote connection which requires necessary credentials. When the server starts, the application firstly tries to govern the connection to the database; if the database does not exist the server will try to initialize to create the first image of the graph database. The connection code requires the URL of the database in the server and needed credentials; the URL and credentials are stored in the environment configuration file. During connection, API has complete permission to access data in the database. The application uses the gremlin middleware library to perform query execution to the database.

```
async function initializeDriver() {
  try {
    createJSON('./config.json', janusGraphDefaults)
  } catch (err) {
    if (err.code === 'EEXIST') {
      console.log('Config file found.')
    } else {
      console.log(err)
    }
  } finally {
    conf = require('./config.json')
    g = traversal().withRemote(new DriverRemoteConnection(conf.janusgraph.url))
    //tx = g.tx()
  }
}
```

Figure 4.1 Initialize Drive code

Express.js application uses the MVC model, the controller manages routes and receives HTTP requests from clients. In controller files, the request is validated and then it triggers the method provided by the model files to handle the request. The model parts, all logic algorithms to examine this request to retrieve and manipulate data from the database and return data for the controller to respond to the client. If there is an error with the query or input validation, the controller responds with an error.

4.1.1 Routing

The majority of request used by the application is GET and POST request. In this project, we use Express.js routes for each resource. Each type of object in the database will get one route URI for its CRUD operation.

Route Sensors: The central object in the management system is sensors. For CRUD operations, there are some considerations regarding sensor data manipulation. Each sensor has some unique data that is a mac address and id, when creating a new sensor object, the application has to check if there is any sensor that has a duplicate mac address in the database before creating a new one. Apart from these CRUD operations, there are other functions like Deadman check which manipulate the status of all sensors.

Route node: provide CRUD operations for node in general, also provide some Post method URL for searching special node and sensor with the help of traversing query. One API route that is used in the system is getting node upstream when the API will find all nodes and the relationship is a child of the target node.

Route relationship: provide CRUD operations for the relationship between two nodes.

User Route: allow the system to create new users and middleware for authentication. These routes use passport middleware for the authentication and authorization phase. Route ‘login’ requires the user to give basic authentication with username and password, after successfully passing this phase, the user

receives a JSON web token which then is used to bypass the authentication phase for other URI requests.

4.1.2 Development and Deployment

The Application is continuously developed and tested with Postman. This Postman API testing process is in parallel with the coding process and debugging. After an API route is created, the API request is tested with Postman with different scenarios to ensure the algorithm works as expected.

API is running in the Linux server on the basis of pm2. Pm2 helps the application to run forever in the background of the server. There is a public URL for this application which is used by front-end applications. For reusable and code study purposes, all API documentation is written in swagger to instruct the usage of each API call.

4.2 Web Application

After databases and API application is set up, web browser is one platform that uses those resources and brings them into the user interface. Two web application has been developed. One web application focus on managing database data in general. The other application focus on managing the sensor and its ownership.

The core technology that is used by this application is React. React is a very straight-forward JavaScript framework that has a purpose of building web user interfaces.

4.2.1 Database Visualization module

The aim of this module is to provide UI platform for admin and sensor owner to manipulate database’s data and relationship between nodes. It helps system manager to manage all data in the database in an interactive and easy way.

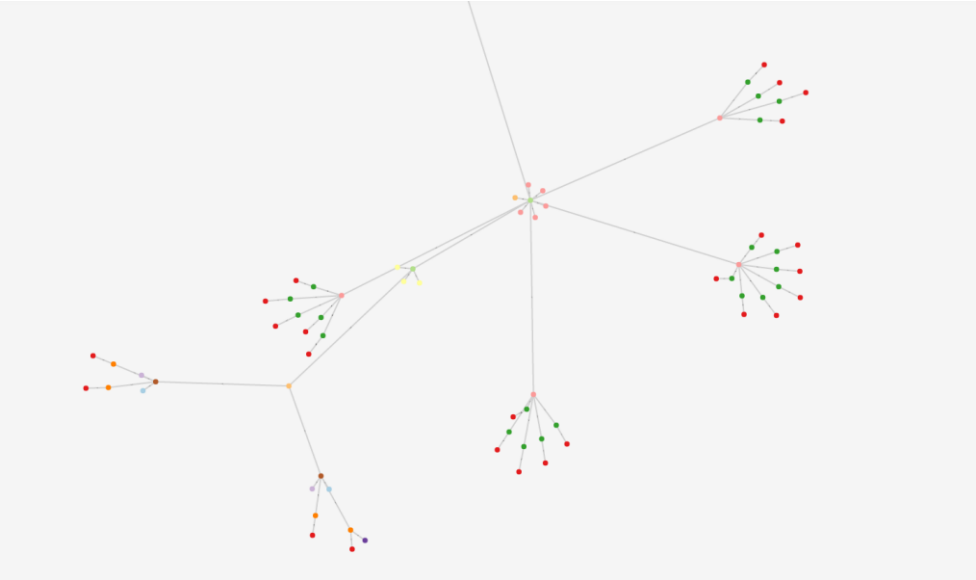

Figure 4.2 Sensors data display in ForceGraph

The first function that related to this application data visualization in graph form. This interface is written with the support of 'force-graph' library which render this graph data on 2d dimension on HTML canvas. Data is retrieve from the database by an API call and is store in React state. The data consists of all nodes and relationships; each node represents one object called vertex in graph. Force graph takes this data set and uses its ready-constructed algorithm to separate nodes and links and render it in a graph-oriented interface.

```
<ForceGraph2D
  ref={graphRef}
  graphData={props.data}
  height={windowHeight}
  width={windowWidth}
  /* dagMode='radialout'
  dagLevelDistance={500} */
  nodeRelSize={NODE_SIZE}
  nodeVal={/* (node) => node.properties.val */ 1}
  nodeLabel={(node) => node.name}
  nodeAutoColorBy= 'label'
  linkDirectionalArrowLength={13}
  linkWidth={(link) => {
    if (link === highlightLink) {
      return 8
    } else {
      return 2
    }
  }}
  cooldownTicks={300}
  maxZoom={10}
  minZoom={0.1}
```

Figure 4.3 ForceGraph 2D configuration implementation

In this ForceGraph2D component, displaying property of nodes and links should be configured and set as props in this component. This property can be node size, displayed name of node, link length, and how nodes should be color, nodes sharing the same label have the same color to distinguish themselves. Under one node, node's name will be displayed and for sensor mac address is displayed. User can interact with the graph, for example, zoom in and out, dragged the nodes and relocate. Through this graph, user can see the whole picture of what data they have on database, which sensor belong to which owner and what is the status of every sensor in a 2D graphic. A list of actions that can be done with a single node can be shown when the user right-clicks to specific node.

Another feature of this application is CRUD operations of nodes and link. When a node in the graph is clicked there should be a popup showing node's properties; these properties are displayed in table format. New node can also be created; a form for creating new node is provided. There are also functions for deleting and modify nodes and its property. The relationship between two nodes can also be create on this platform. Application can also search for individual nodes or a set of nodes.

All of these functions is done by using API call to the API application via axios request. After send request to API, the app receive response which requires callback function to handle.

One special function of this application is "get nodes upstream". In the future there will be hundreds of thousand sensors available in this database so showing that the number of nodes in one graph will be very difficult to follow and understand. That is the reason behind the exist of get node upstream function. In details, the goal of this function is to get all nodes and relationships between those nodes to create a small group of nodes. The function sorts out all of its child nodes and relationship create a group of nodes in which the chosen node is the center one. With this function, the whole database can be split into small part which is easier to manage.

For security, the application use JWT login model, in which when user login for the first time. After a successful authentication, a token is sent to the application

via API. Then the token is stored in local storage and state, this token has encoded user information and authentication credentials which can be used for retrieving protected data.

4.2.2 Sensor Management

Another application built for specializing in sensor manipulation for both sensor owner and system manager. The purpose of this application is to manage data focusing on sensors. The application is built in React and Material UI for building an attractive user interface.

The UI is divided into multiple components on one web page. Each component of UI is separated into different React functions stored in different files in order to improve the code structure and code reusability.

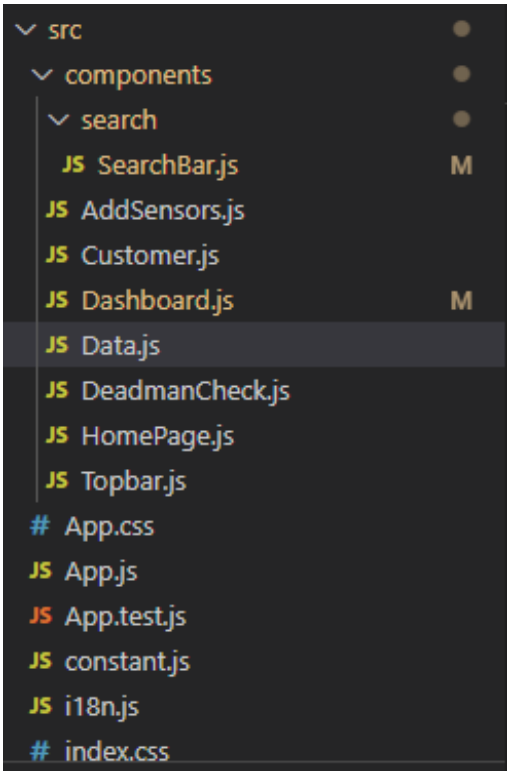

Figure 4.4: File structure

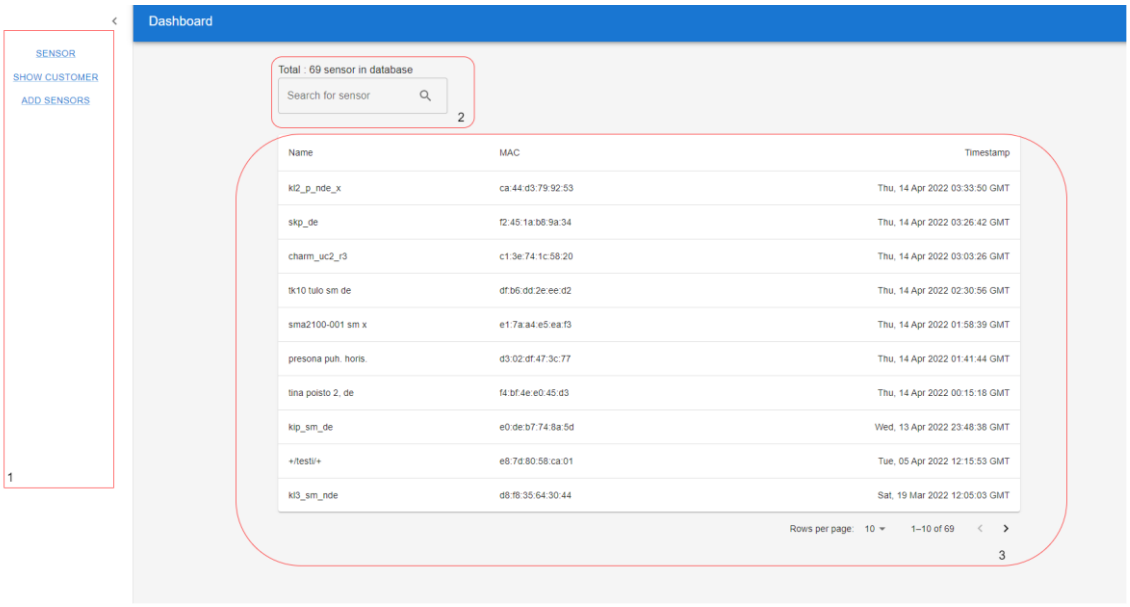

Figure 4.5: Web application components, (1) Route list, (2) Search bar, (3) Table displaying sensors data

Multiple UI components also make it easier to implement and make adjustments for styling purposes. In figure 4.5, the whole web page is divided into many separate components, the developer can change UI properties or add styling directly to those components.

The first feature is to view all of the sensor and owner data. The application sends API requests to the back-end server and gets responses with the data of sensors and owners. After login with API authentication, the owner gets access to all sensor data that belong to him. This data is displayed in a table form and in order by the timestamp of the last sensor update the limitation for the number of sensors displayed on one page is 10 sensors. Another factor that should be applied for easy managing data is searching. When there are a great number of sensors in total but only a few sensors that the user wants to work on within one session, the search engine should be applied to filter a sensor or group of sensors that needs to manipulate. We use factors that are unique for each sensor in consideration in this searching algorithm, the sensor will be filtered according to its MAC address and name.

The second feature is that the sensor owner can manually change the ownership of one sensor from one owner to another. This is very helpful especially when a

new unowned sensor is purchased by a new owner, the owner when they want to discard their sensor can also change ownership of their sensor.

4.3 MQTT Client

The aim of the client is to act as a tool for managing sensor data through MQTT messages. Most of the sensors send data via the MQTT protocol, MQTT client has the mission of receiving sensor data through a TCP connection to the MQTT broker and then using its algorithm to make UPDATE or ADD requests to API

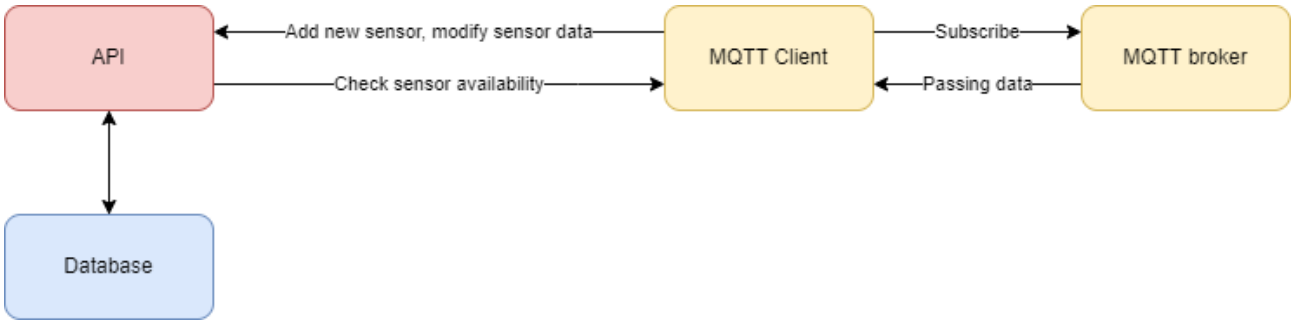

Figure 4.6 Data flow in MQTT connection

Figure 4.6 shows how sensor data is managed with the MQTT client. The basis of this data flow goes along with MQTT transmission.

MQTT client is a script file running on the server. It runs under the control of pm2 to keep the client connection alive. MQTT client is written in JavaScript Nodejs and it uses mqtt.js library to generate the MQTT protocol.

The first thing that an MQTT operates is to connect with an MQTT broker. In order to clarify the connection, there are some arguments needed to be added to the connection's configuration. Some essential arguments that need to be declared are ClientID, username, and password of the broker, broker's URL, and protocol types, and other connection configurations supporting the connection. The connection starts with the function mqtt.connect():

```
let option = {
  clientId: "mqttjs01",
  rejectUnauthorized: false,
  username: "nmas",
  password: "Oulunsalo",
  ca: caFile,
  keepalive: 1,
  clean: false,
  reconnectPeriod: 1000 * 1
};
var client = mqtt.connect("mqtts://nmasiot.nome.fi", option);
```

Figure 4.7 MQTT client configuration

This function will return a Client object which allows users to perform requests and receive messages from the broker. MQTT protocol is based on subscribe and publish model; after being connected to the MQTT broker, the client has the possibility to subscribe and publish a message to the broker.

Subscribe: The client can make a subscription request by executing client.subscribe() function. Each subscription requires a topic to which the MQTT broker sends the message. The topic is a string that helps the MQTT broker to filter messages for each connected client; each topic has different levels and separated in the string by forward slash. Here is one example command:

```
var topic = "nmss/#";
setTimeout(function () {
  client.subscribe(topic, { qos: 1 });
}, 3000)
```

The '#' in the topic string is called wildcard topic where it will tell the broker that the client wants to subscribe to multiple topics without knowing the exact address. When a broker sends a message within a topic that client subscribes, the MQTT client can receive the message with the method on('message'):

```
client.on('message', function (topic, message, packet) {
  console.log(message, topic)
})
```

This function detects the arrival of a message within a topic from the MQTT broker message response.

Publish: The client can also publish a message to the MQTT broker. The broker can then send back this message to other clients that subscribe to the message's topic. With MQTTjs, the MQTT client can publish a message and topic that is related to with client.publish() function.

4.3.1 Data updating

After having built infrastructure code for MQTT data transmission, there comes a need for a method that helps to translate this message and add data from this message to the current database.

The first thing that the client will do when receiving the message is to parse this message into an object of data. The original message, which is published to the broker by a sensor, is a string formed in a Buffer containing some bytes long. This Buffer is created as an output of the sending message function produced by the sensor's hardware, each set of bytes in this Buffer represents a measurement figure in sensor data.

```
function parseAd(mac, message) {
  let data = Buffer.from(message);
  let decimalPoints = 7;
  let payload
  let protocol = data.readUInt8(1);
  if (protocol === 1) {
    console.log(1)
    let stamp = data.toString('utf8', 2, 22);
    let nameLen = data.readUInt16LE(22);
    let name = data.toString('utf8', 24, 24 + nameLen);
    let gpeak = data.readFloatLE(25 + nameLen);
    let vrms = data.readFloatLE(29 + nameLen);
    let env = data.readFloatLE(33 + nameLen);
    let battery = data.readUInt16LE(37 + nameLen)
    let temp = data.readInt16LE(39 + nameLen);
    let humidity = data.readUInt16LE(41 + nameLen);
    let pressure = data.readUInt16LE(43 + nameLen);
    let reserved = data.readUInt16LE(45 + nameLen);
    let reserved2 = data.readUInt8(47 + nameLen);
    let flags = data.subarray(49 + nameLen, 50 + nameLen);
```

Figure 4.8. Data parsing function

Figure 4.8 shows how each set of bytes from Buffer was parsed into different variables. After being parsed, the final result of the message is an object of readable sensor data.

The next step after getting the parsed data was to update this data of the sensor into the database via API. The sensor of the new data is checked using their mac Address whether the sensor already exists in the database or not. If the database has already stored the sensor the data of that sensor will be added. If the first sketch of the sensor did not exist in the database, the sensor would be added for the first time.

The message from the MQTT broker comes constantly and each sensor publishes their measurement data every 24 hour.

4.3.2 Deadman Check

Another module that is built in MQTT client is Deadman Check. A Deadman check is a test conducted to decide the status of the sensor. The dead-man check is based on the fact that each sensor sends data every 6 hours or 24 hours depending on the type of sensor; if the sensor does not send a signal or message to the MQTT broker once a day, that sensor is considered to be offline.

To perform this module, we have created an algorithm in which the time of the check is compared with the last time the sensor sends data. Due to the difference in the time of each sensor is a maximum of the 1-day time, every 24 hours the check will be done in the client. In one check, the client will get data of every sensor's last timestamp. Then the module performs a loop through all sensors comparing each sensor's last update timestamp with the time of the check. If the module found any sensors that did not respond within 24 hours, those sensors were considered to be inactive. On the other hand, other sensors which passed the test had the status of the active sensor. The client will detect the change in each sensor status, this change first is updated to the database and then reported back to the system manager for notification.

The Dead-man check report module was built with the help of Node-Red. When there is a change in sensor status, the Client will publish a message to the broker with the topic 'status'. This message contains the sensor's mac address which changes its status and its current status. In Node-red, a data flow is built to react to the MQTT message publish.

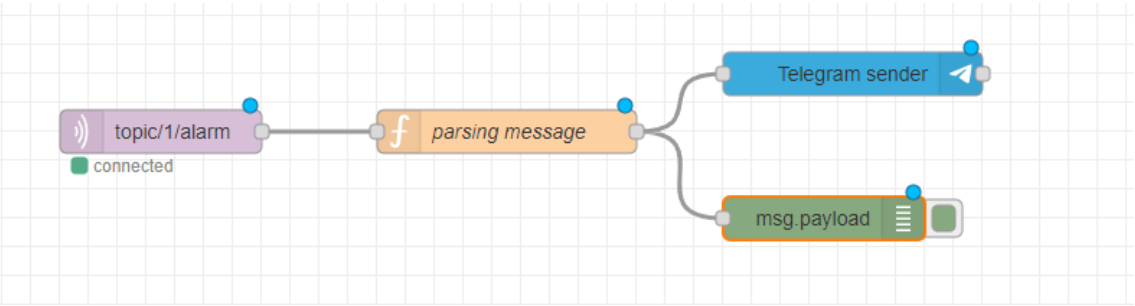

Figure 4.9 Data flow in Node-Red

Figure 4.9 shows how the reporting process works in Node-red. After publishing the notification message to the MQTT broker, Node-red creates an MQTT client only for receiving that status message. The message will then be parsed and sent to the system manager via Telegram by a telegram bot.

4.4 BLE Advertisement Mobile Application

A hybrid mobile application aims to provide a platform for users to manage nearby sensors using Bluetooth Low Energy connections. This application is written in React Native and during the implementation process, the app is running and debugging on an android device.

Every sensor in our company is created with to have Bluetooth Low Energy technology available allowing these sensors to be connected and transfer data to other Bluetooth devices. BLE is a wireless radio signal, adapted to low-power sensors and accessories [19]. By using a BLE connection, the application can use BLE advertisement to find sensors via Bluetooth to read its data and perform other actions through that connection. In BLE, the mobile application act like a central device that has full control over other the surrounding device which is called peripheral devices [19].

The mobile app use ‘ble manager plx” NPM package as a library for BLE integration in React Native. This library provides a class named BleManager which has some ready-made methods that can scan, connect and interact with BLE peripherals. There are 2 stacks in this application, the first stack screen is scanning BLE sensors and the second is show details of the connected sensor and manipulates the sensor data using BLE services.

For sensor scanning, React Native runs a scan when the component is rendered. To perform this scan on Android, the user is asked to give “ACCESS FINE LOCATION” permission. To start the scan, a new object BleManager is defined then the code run this object’s method scanDevices(). The output of this method is data gathered from every device that has BLE found within the signal range. These data are gathered continuously until the application trigger stopScan()

method. These data are saved as React state as an array of sensor objects. The data from this state then will be displayed in React Native Scrollview.

The data from the scanned object contains MAC address, name, connection condition, and manufacturer data and it is originally in the form of an object called Device. Manufacturer data includes measurement data of the sensor. The application also checks if the found sensors exist in the database, if it is an untracked sensor, the user can manually add the sensor to the database. This will give the system many options to add new sensors from the factory to the database when they arrive at the warehouse.

Another function that this library provides is allowing this application to connect to sensors in order to use their services. Our sensor has provided a service to change the name of the sensor. After being connected, each Device object provides a range of services, by method Device.service() the output of this function is a list of Service objects which then be used to interact with sensors. One of the services that the sensor scanner provides is changing its name. This would give sensors meaningful names, so it is easier to track them in the future.

For deployment, the mobile application has produced an APK file and this app is put in Google Play to first use on the android platform.

5 RESULT

5.1 The outcome of the project

Overall, the thesis project provided a system for sensor owners to control and manipulates sensors.

A database for storing the sensors and owner data was built and allowed a connection for managing the data from the API. There were currently 72 sensors and 12 owners in the database, the storage capacity of the database can allow it to store more than a thousand sensors. Data in the database had relationships which each other so there was no stand-alone object. The database allowed sensors to be added and updated with API application using gremlin query which takes about 0.5 milliseconds. This time cost was considered to be acceptable with the requirement of the real-time system.

Sensor data is monitored and manipulated on different platforms and with different algorithms. There are 4 applications is implemented in the system providing many options for users to manage the data. All of these changes are done through the request to the API.

Firstly, sensor data can be continuously updated with MQTT connections. MQTT clients constantly received signals and data from sensors through MQTT protocols, these data then replaced the data in the database creating a continuous development of the data.

Secondly, the system provides services for maintaining the database, especially for sensors, in a user-experienced and interactive platform. There are 2 websites has been created. The first website offers a graphical user interface for database management, this is the platform where users can see all data in the database in graphical formation and interact with this data.

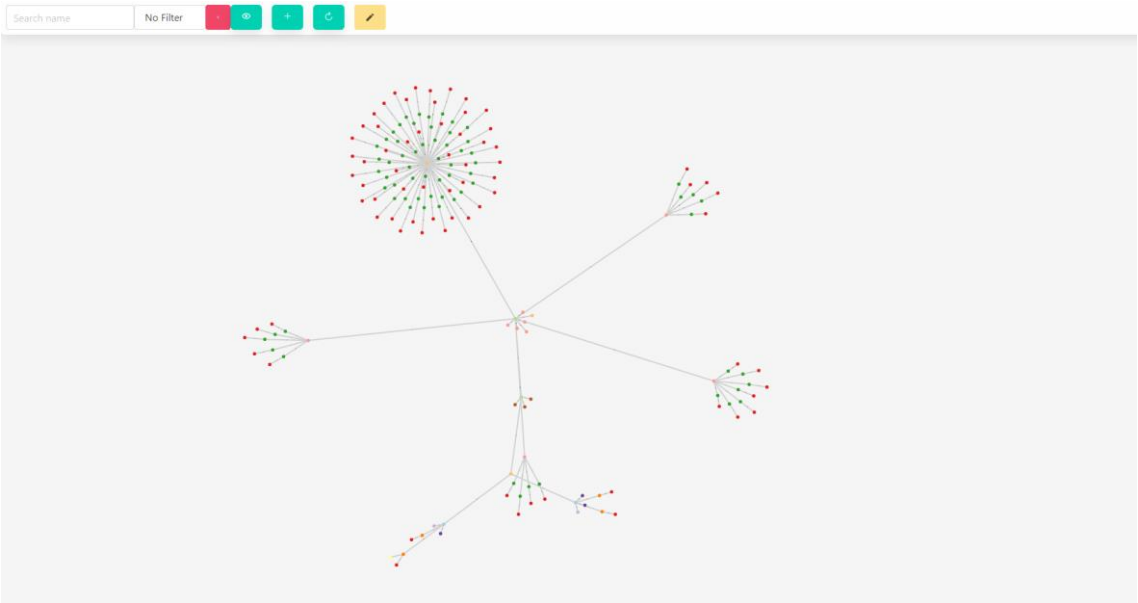

Figure 5.1 The database displayed in web application

Figure 5.2 Information of Object's property

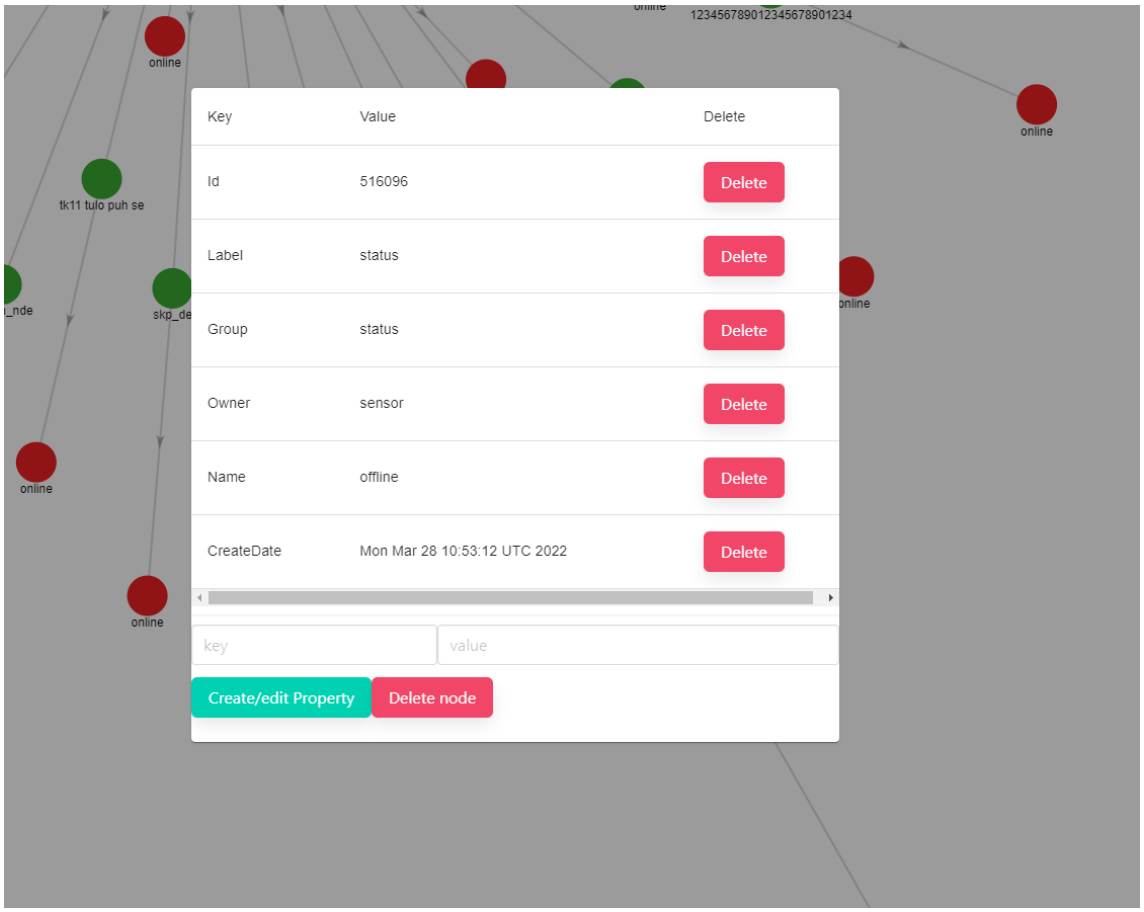

Figure 5.3 Edit/Delete property function of specific object

Another website displays sensors and owner information in a tabular style to display data. Sensor data and its ownership can also be edited manually by this application.

Lastly, a mobile application was created for the purpose of scanning. The mobile application used BLE advertisement to scan and read information from sensors, which have the ability to broadcast their measurement data via BLE signal. With this technology, users can read data in real-time measurement from nearby sensors and change its inner configuration. Users can also add a new sensor to the database using this application.

Data security functions were applied and developed in this project to identify logins, users, and permission. This security system helps the data to be securely retrieved and updated with the user's permission; each user had their own access to certain data and had to provide an authentication identity for active access.

The data security system was developed in API based on the JWT authentication method. The user and permission information was stored in the database.

Sensor status was also controlled continuously with the dead-man check function. The MQTT clients will detect automatically the change in the status of the sensor and update the status in the database. Meantime, it sent a notification to the user's Telegram using a Telegram bot via MQTT message.

Despite some features that were successfully developed, there are some requirements that this thesis did not achieve. The warehouse module, which was planned to be implemented to support monitoring sensors, was not developed at the end of the project. Furthermore, in the mobile application, the user should be able to connect with the sensor via BLE but the tests showed that it failed to deliver this function.

5.2 Usability testing

There was usability testing conducted at the end of the implementation phases. All of the functional requirements were checked and used as test cases. The applications can be accessed in the company's public domain URL. Customers and system managers were asked to perform every function of the application and give feedback on their experiences. Results showed that every available function in the system works finds, however, there would be UI modifications and functions developed in this system in the future. In the end, usability testing helped the author and development team know the drawbacks of the currents system.

5.3 Functional testing

The purpose of functional testing is to ensure every existing function of the applications works according to its original requirement and find to fix any bugs. Based on requirements, there are test cases for each function to perform. Most of the tests are done with API and it is executed with Postman. Each test case has its set of inputs and expected outputs; Postman will run the test with the request URL and input data, the result of the request as compared to the expected output to decide whether the test failed or succeeded. If the test showed

that the function did not meet the requirement, the functions will be fixed in the implementation process.

6 CONCLUSION

The aim of creating a system for Nome's sensor management was achieved. A database for storing sensor data was created with a capacity of storing up to 1000 objects data. Web applications for database management and monitoring sensor were implemented and continuously developed in both the development and production environment. In addition, a mobile application was developed for scanning data from nearby sensors using Bluetooth signals. The sensors' data can also be collected and updated with our MQTT clients. All of the data flow from these applications was connected with the database through API applications. API ensures the UI applications can retrieve and update data to the database securely and precisely.

The development process started with designing the system architecture. When all the requirements were analyzed, rules and principles of component applications were set for the implementation process. The web application is written in React and Material providing UI for interacting with data. Sensor data is gathered based on the concept of MQTT protocols and BLE advertisements for mobile devices. Meanwhile, it is stored and displayed in graph format increasing visual management and manipulation.

The result of the thesis project is applicable for the future development of the whole system of the company. It built a ground infrastructure of sensor data management for the company. However, there are also some functions that this thesis failed to deliver. Therefore, there are more functions that can be applied to this system, based on the foundation that this thesis achieved and its failure. To optimize the benefit of using the graph model, there should be a new algorithm for searching sensors and developing relationships from different modules in the database not only sensor management.

REFERENCES

[1] OpenJS foundation, “About Node.js”, March 18, 2022. [Online]. Available: https://nodejs.org/en/about/. [Accessed Mar. 22, 2022].

[2] Facebook open-source Meta platform, “React documentation”, March 29, 2022. [Online]. Available: https://reactjs.org/docs/getting-started.html. [Accessed Mar. 22, 2022].

[3] Microsoft contributors, “About Material UI”, 2022. [Online]. Available: https://mui.com/. [Accessed Mar. 22, 2022].

[4] V. Asturiano, “react-force-graph package documentation and installation”, February 2022. [Online]. Available: https://www.npmjs.com/package/react-force-graph. [Accessed Apr 26, 2022].

[5] V. Asturiano, “Example of directional links for graph”, February 2022. [Online]. Available: https://vasturiano.github.io/force-graph/example/directional-links-arrows/. [Accessed Mar. 5, 2022].

[6] CM. Syamla, “What Are The Benefits Of Using Express.Js For Backend Development”, October 29, 2019. [Online]. Available: https://www.techomoro.com/what-are-the-benefits-of-using-express-js-for-backend-development/. [Accessed Mar. 22, 2022].

[7] S. Cope, “MQTT Publish and Subscribe Beginners Guide”, December 16, 2021. [Online]. Available: http://www.steves-internet-guide.com/mqtt-publish-subscribe/. [Accessed Feb. 27, 2022].

[8] A. Strzelewicz2022, “Pm2 documentation and installation”, Unitech, February 2022. [Online]. Available: www.npmjs.com/package/pm2. [Accessed Mar. 16, 2022].

[9] JanusGraph's authors, "Introduction of JanusGraph", January 18, 2022. [Online]. Available: https://docs.janusgraph.org/. [Accessed Feb. 19, 2022].

[10] J. O'Conner, "Graph 101: Traversing and Querying JanusGraph using Gremlin", Compose IBM, November 2007. [Online]. Available: https://www.compose.com/articles/graph-101-traversing-and-querying-janusgraph-using-gremlin. [Accessed Feb. 19, 2022].

[11] Openjs Foundation, "Node-Red documentation", 2022. [Online]. Available: https://nodered.org/about/. [Accessed Apr. 2, 2022].

[12] S. Chacon & B. Straub, "Pro Git", 2nd edition. Published: 2014. [E-book]. Available: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git. [Accessed Mar. 2, 2022].

[13] Microsoft, "Why visual studio code", March 30, 2022. [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode. [Accessed Feb. 16, 2022].

[14] O. Gupta, "Basics of API Testing Using Postman", November 18,2021. [Online]. Available: https://www.geeksforgeeks.org/basics-of-api-testing-using-postman/. [Accessed Mar. 12, 2022].

[15] C. Paganini, "Understanding Software and System Architecture", December 5, 2019. [Online]. Available: https://thenewstack.io/primer-understanding-software-and-system-architecture/. [Accessed Apr. 2, 2022].

[16] S. Brown, "C4 model for visualising software structure", 2011. [Online]. Available: https://c4model.com/. [Accessed Feb. 16, 2022].

[17] Stardust Group, "Hybrid Apps: an overview of advantages, limitations, and consequences for your testing phase", April 11, 2022. [Online]. Available: https://www2.stardust-testing.com/en/blog-en/hybrid-apps. [Accessed Mar. 2, 2022].

[18] Redhat, 2020. “What is a REST API?”, May 8, 2020. [Online]. Available: https://www.redhat.com/. [Accessed Feb. 20, 2022].

[19] D. Zasukha and A. Boradenko, 2020. “What to Consider when integrating BLE in your React Native app”, Stormotion, July 12, 2020. [Online] Available: https://stormotion.io/blog/what-to-consider-when-integrating-ble-in-your-react-native-app/. [Accessed Mar. 12, 2022].