Given the distinctive characteristics of the system discussed in Chapter 3, it is essential to adopt an architecture or combination of multiple architectures to efficiently address these specific requirements while ensuring scalability, reliability, and seamless communication between different components. This chapter aims to deliver an in-depth exploration of the architecture design implemented in the project and provide a detailed understanding of each element in the system.

## 0.1 Architecture design

### 0.1.1 Software architecture selection



**Hình 0.1:** Overall architecture of the system

**Figure ??** illustrates the system's general architecture and provides an overview of its data flow. The system strategically uses the combination of MVCS (Model-View-Controller-Service) and microservice architectures to take advantage of their unique strengths in specific aspects. This hybrid architecture leverages the benefits of both architectural styles, such as scalability, flexibility, and the ability to use different technologies and patterns within each service.

Microservices architecture is a software development method that structures an application as a collection of loosely coupled services. In this architecture, each service can be developed, deployed, and scaled independently, allowing greater scalability, flexibility, and agility than monolithic architectures, as teams can update or fix individual components without impacting the entire system in case of failure. Thanks to its ability to accommodate the dynamic and distributed natures of IoT applications, microservice architecture is frequently used in various IoT projects. It allows organizations to build more agile and adaptive applications to handle the complex demands of modern business environments.

On the other hand, the Model-View-Controller-Service (MVCS) architecture is an extension of the traditional Model-View-Controller (MVC) pattern. It introduces a service layer sitting between the Controller and the Model, responsible for containing business logic and rules. This layer abstracts complex business operations, allowing Controllers to focus on handling incoming requests and delegating the heavy lifting to Services. This design pattern is well-suited to many web applications, providing a structured and organized approach to developing complex applications and making it easier to manage, maintain, and scale the system over time.

By combining these two architectures, the system is better equipped to handle the dynamic and distributed nature of IoT applications. It also enables organizations to build more agile and adaptive applications to address the complex demands of modern business environments while ensuring scalability, reliability, and seamless communication between different components.

### 0.1.2 Overall design

The system is divided into independently deployable services, including the API back-end server and MQTT Server. These services communicate via the MQTT protocol, and the MQTT Server also acts as a broker to relay data to the EPD devices (Figure **??**). API server follows the MVCS pattern with three main components: Controllers handling incoming requests and coordinating responses, Services containing the business logic and rules of the application, and Models interacting with the database, managing data storage and retrieval, while Management UI and EPD devices act as a View component (Figure **??**).
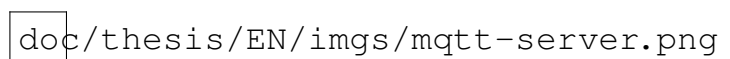
#### a, MVCS architecture

The system has three data types, and the packages of each type handling operation share the same structure: the Controller handles requests about the data type and delegates them to the Service, which processes the requests and uses the Model to interact with the database.

doc/thesis/EN/imgs/api-server.png
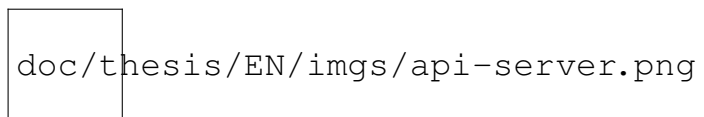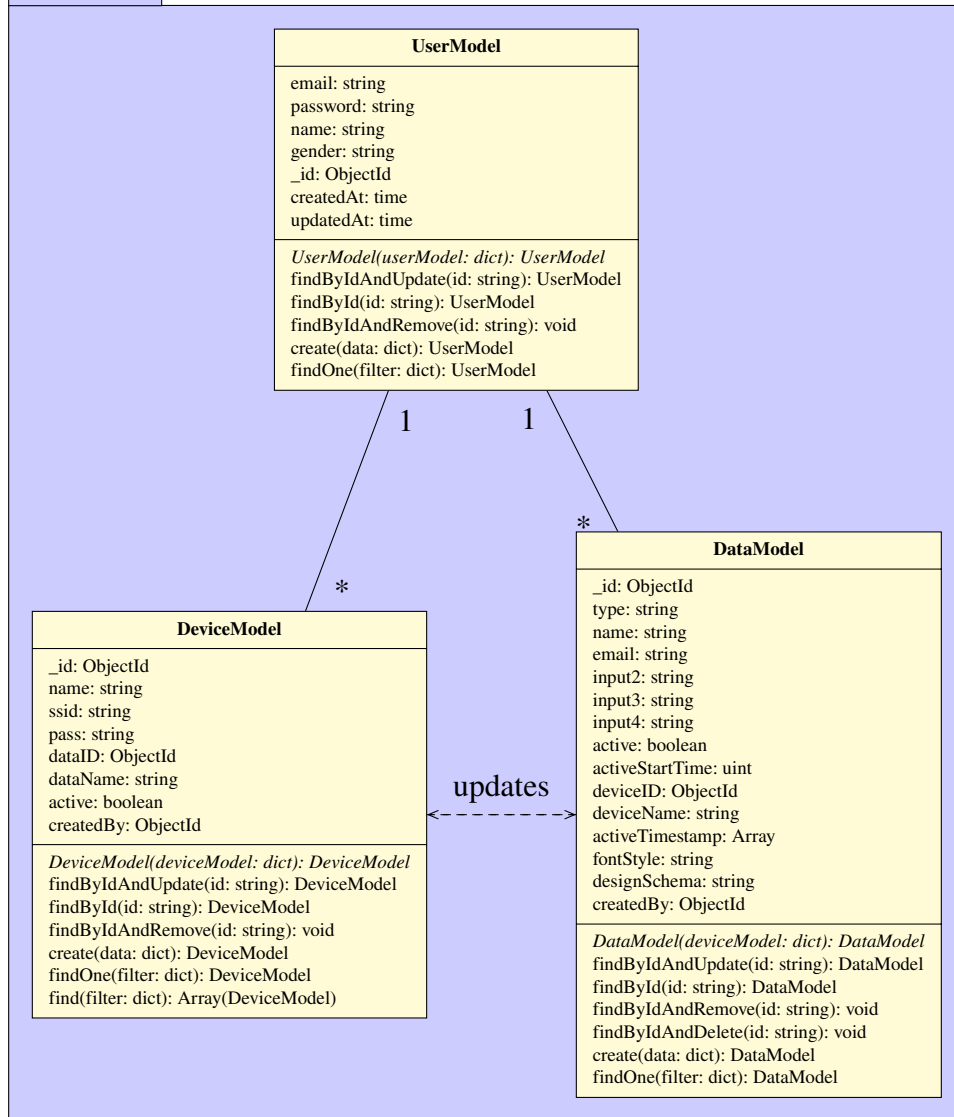
**Hình 0.2:** MVCS architecture

#### b, MQTT Server

This service in the system acts as an intermediary, managing the state of all MQTT client connections, subscriptions, and message exchanges.
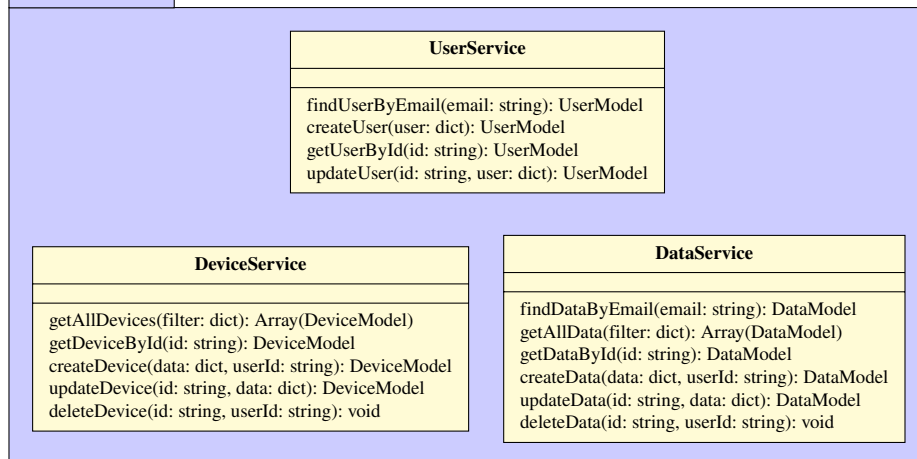
doc/thesis/EN/imgs/mqtt-server.png

**Hình 0.3:** MQTT microservice

## Models

### UserModel

email: string
password: string
name: string
gender: string
_id: ObjectId
createdAt: time
updatedAt: time

---

*UserModel(userModel: dict): UserModel*
findByIdAndUpdate(id: string): UserModel
findById(id: string): UserModel
findByIdAndRemove(id: string): void
create(data: dict): UserModel
findOne(filter: dict): UserModel

1    1

\*

### DeviceModel

_id: ObjectId
name: string
ssid: string
pass: string
dataID: ObjectId
dataName: string
active: boolean
createdBy: ObjectId

---

*DeviceModel(deviceModel: dict): DeviceModel*
findByIdAndUpdate(id: string): DeviceModel
findById(id: string): DeviceModel
findByIdAndRemove(id: string): void
create(data: dict): DeviceModel
findOne(filter: dict): DeviceModel
find(filter: dict): Array(DeviceModel)

updates

\*

### DataModel

_id: ObjectId
type: string
name: string
email: string
input2: string
input3: string
input4: string
active: boolean
activeStartTime: uint
deviceID: ObjectId
deviceName: string
activeTimestamp: Array
fontStyle: string
designSchema: string
createdBy: ObjectId

---

*DataModel(deviceModel: dict): DataModel*
findByIdAndUpdate(id: string): DataModel
findById(id: string): DataModel
findByIdAndRemove(id: string): void
findByIdAndDelete(id: string): void
create(data: dict): DataModel
findOne(filter: dict): DataModel

**Hình 0.4:** General use case diagram of the data management system

## Services

### UserService

---

findUserByEmail(email: string): UserModel
createUser(user: dict): UserModel
getUserById(id: string): UserModel
updateUser(id: string, user: dict): UserModel

### DeviceService

---

getAllDevices(filter: dict): Array(DeviceModel)
getDeviceById(id: string): DeviceModel
createDevice(data: dict, userId: string): DeviceModel
updateDevice(id: string, data: dict): DeviceModel
deleteDevice(id: string, userId: string): void

### DataService

---

findDataByEmail(email: string): DataModel
getAllData(filter: dict): Array(DataModel)
getDataById(id: string): DataModel
createData(data: dict, userId: string): DataModel
updateData(id: string, data: dict): DataModel
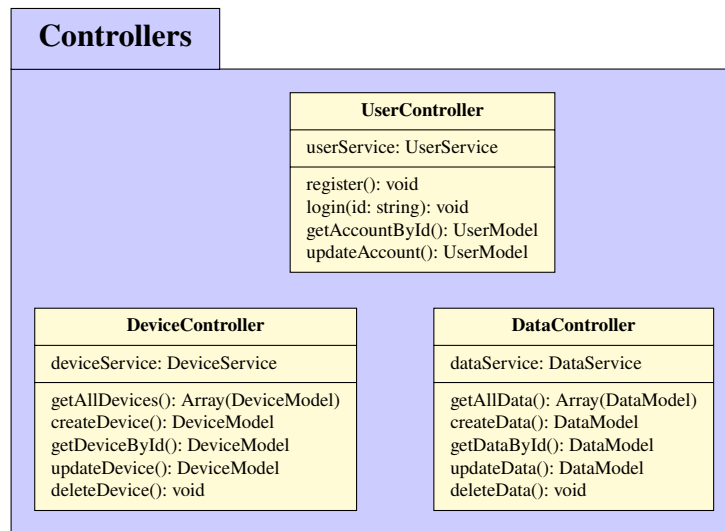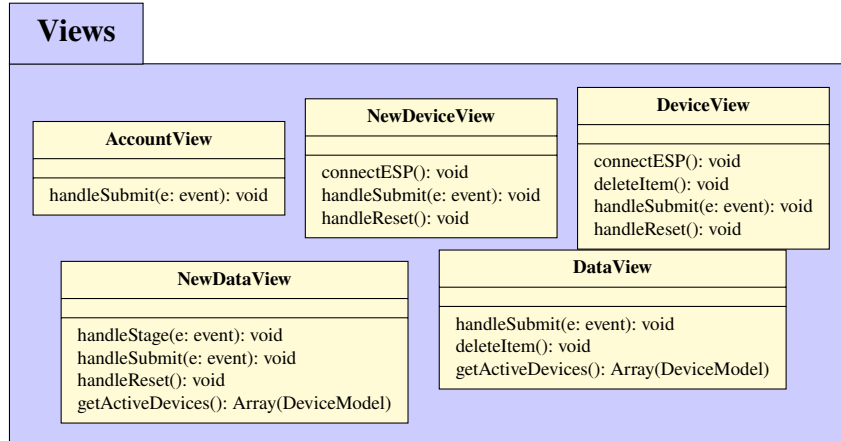deleteData(id: string, userId: string): void

**Hình 0.5:** General use case diagram of the data management system

3

**Hình 0.6:** General use case diagram of the data management system

### 0.1.3   Detailed package design

    **a,  Models**

    **b,  Services**

    **c,  Controllers**

    **d,  Views**



**Hình 0.7:** General use case diagram of the data management system

## 0.2   Detailed design

### 0.2.1   User interface design

The architecture and user interface of this system are meticulously tailored for computer displays, ensuring optimal functionality and visual experience on larger screens. The design is customized to leverage the expansive real estate of computer monitors, facilitating ease of use and comprehensive information display. This focus on computer-targeted design means that the system is not intended for mobile use, and as such, it may not provide an ideal user experience or full functionality

on smaller mobile device screens. The decision to specialize in computer displays stems from a commitment to delivering a high-quality, immersive experience that fully utilizes the capabilities and advantages of larger screens typically associated with desktop or laptop computers.

doc/thesis/EN/imgs/mockup1.png

**Hình 0.8:** Mockup display of authentication page

doc/thesis/EN/imgs/mockup2.png

**Hình 0.9:** Mockup display of dashboard page

doc/thesis/EN/imgs/mockup3.png

**Hình 0.10:** Mockup display of modal component

doc/thesis/EN/imgs/mockup4.png

**Hình 0.11:** Mockup display of creation page

### 0.2.2 Layer design

Each sub-section below demonstrates the workflow of each class participating in each use case in the form of a class diagram and a responding sequence diagram. Each class only shows relevant methods used in the use case.

## a, Class and sequence diagram of use case "Register a new device"
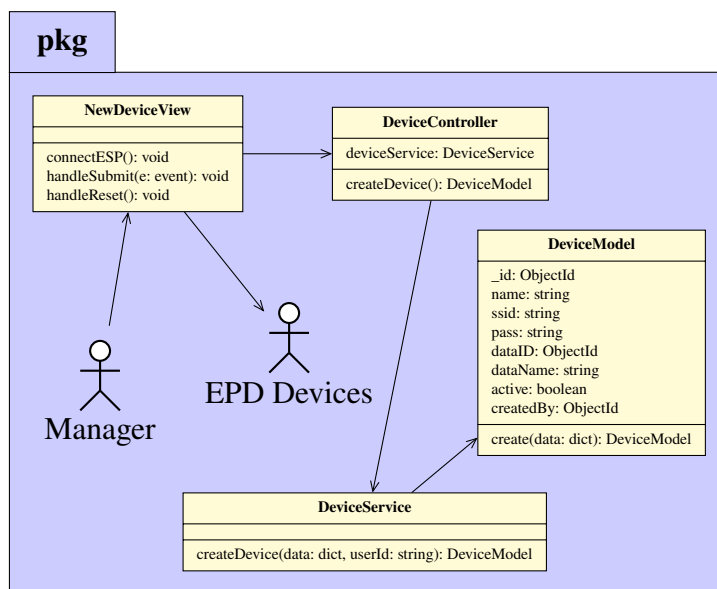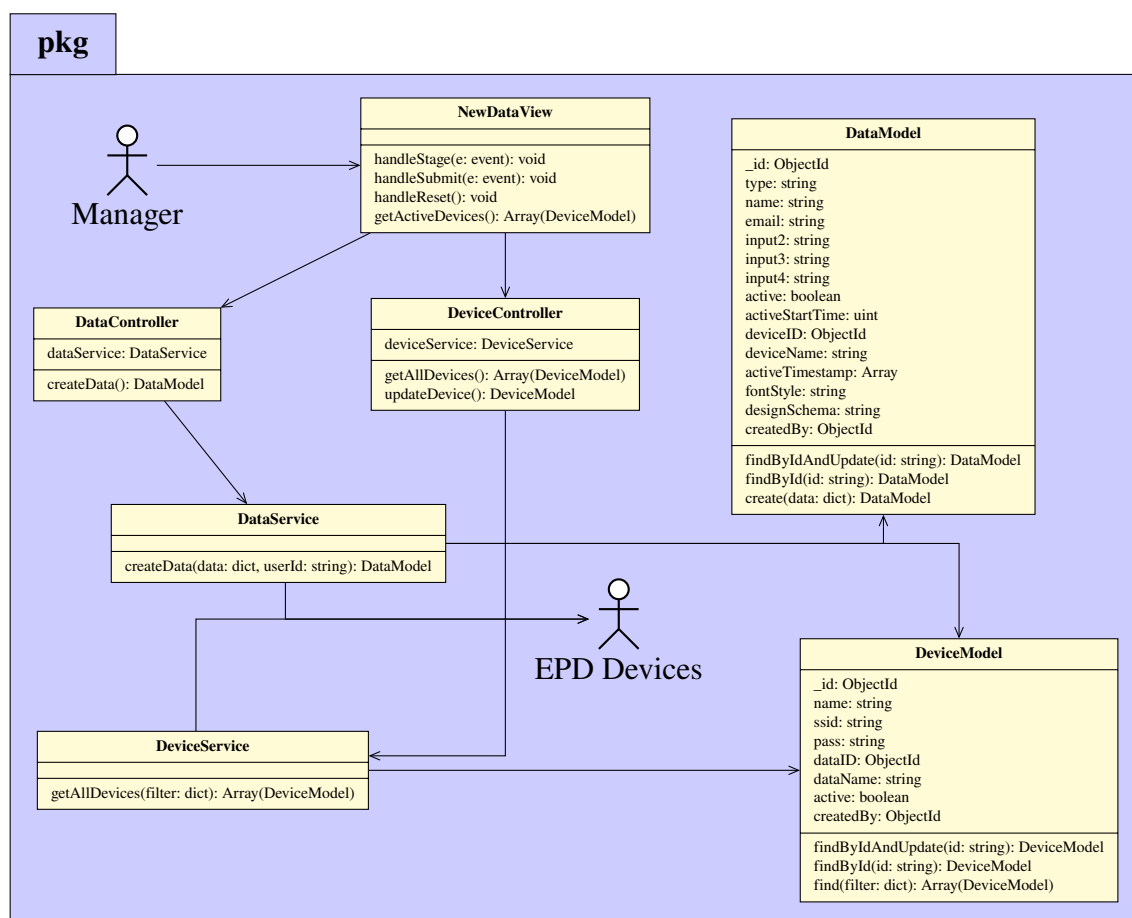


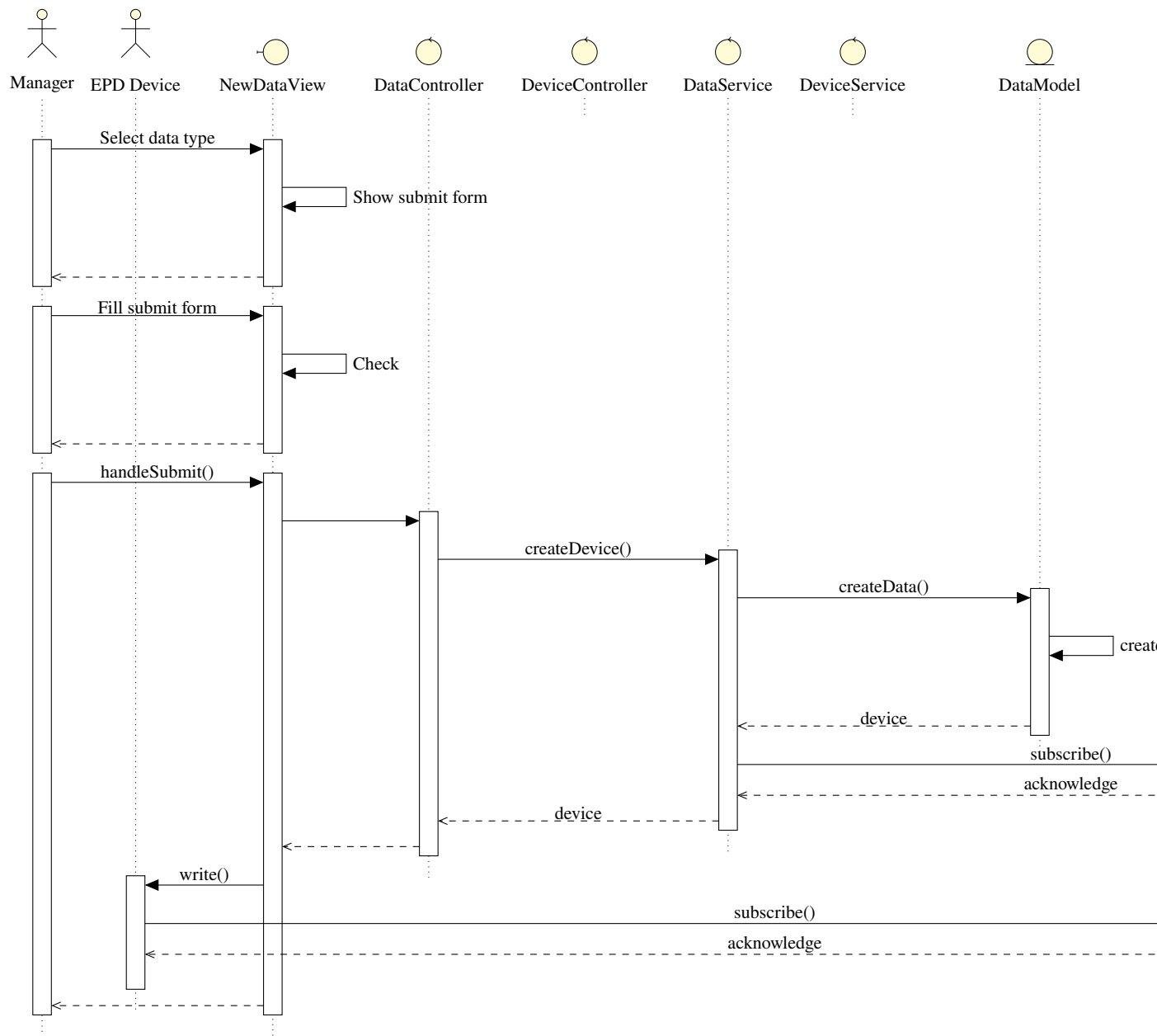**Hình 0.12:** Detailed use case of Data Management function
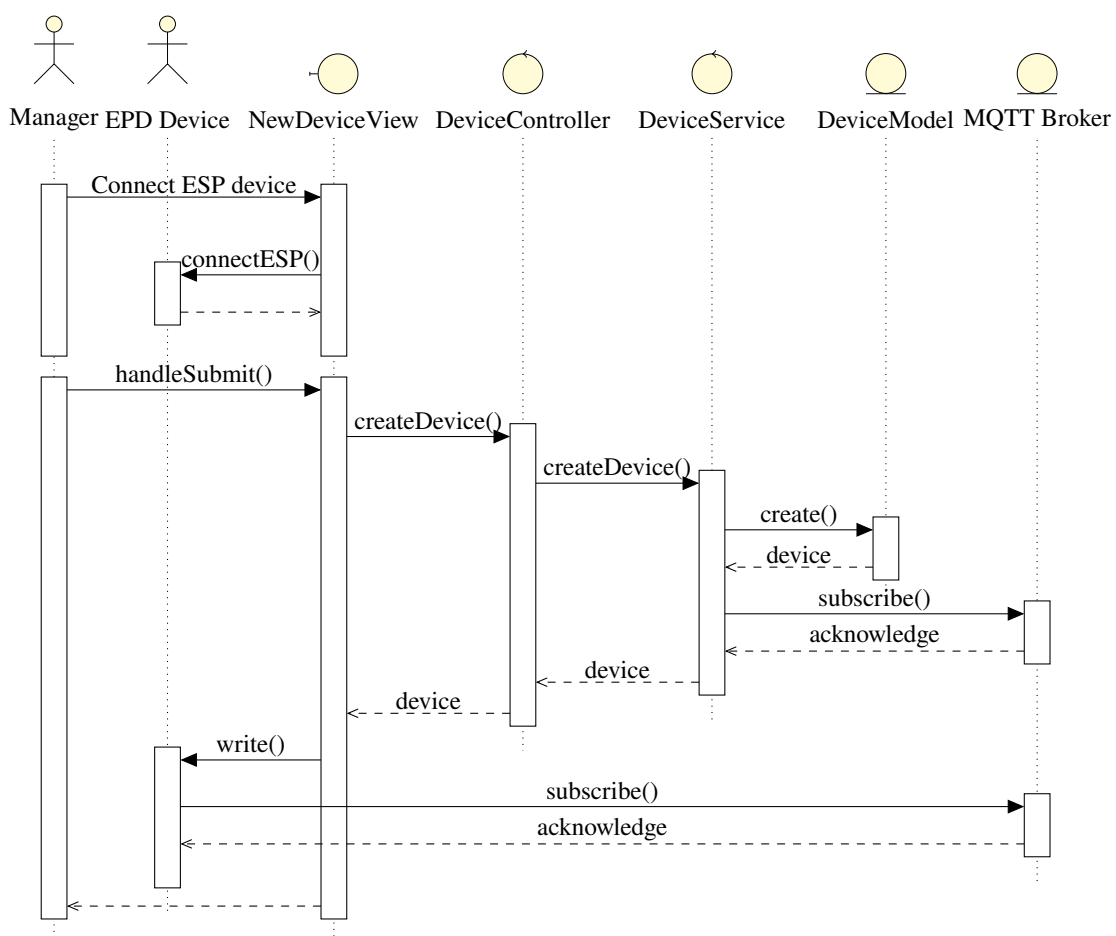


**Hình 0.13:** Caption

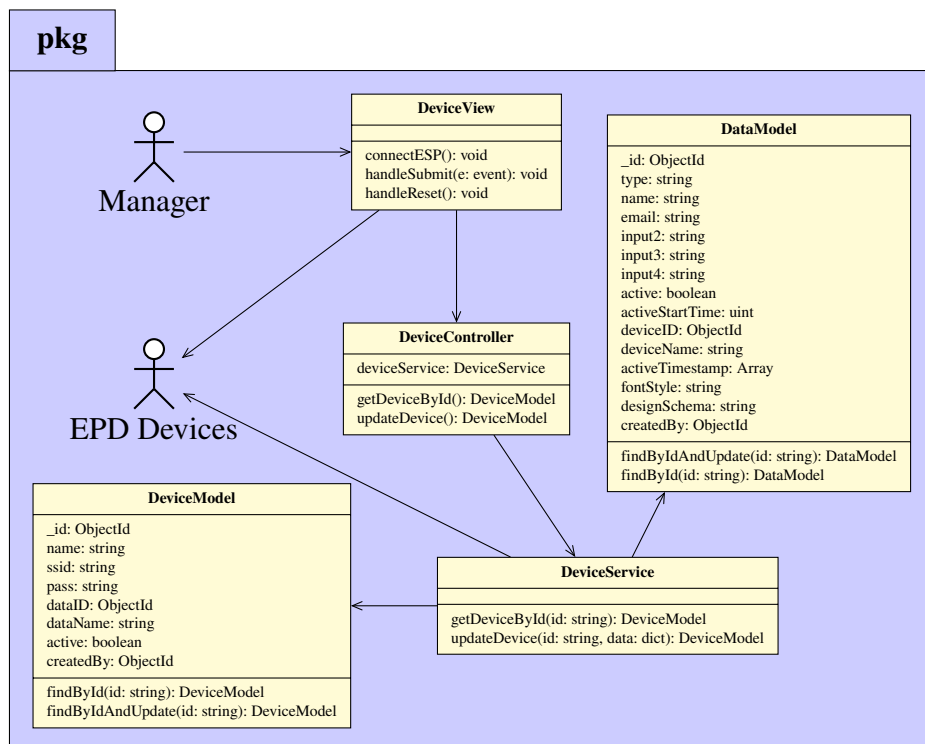**b, Class and sequence diagram of use case "Add a new data"**



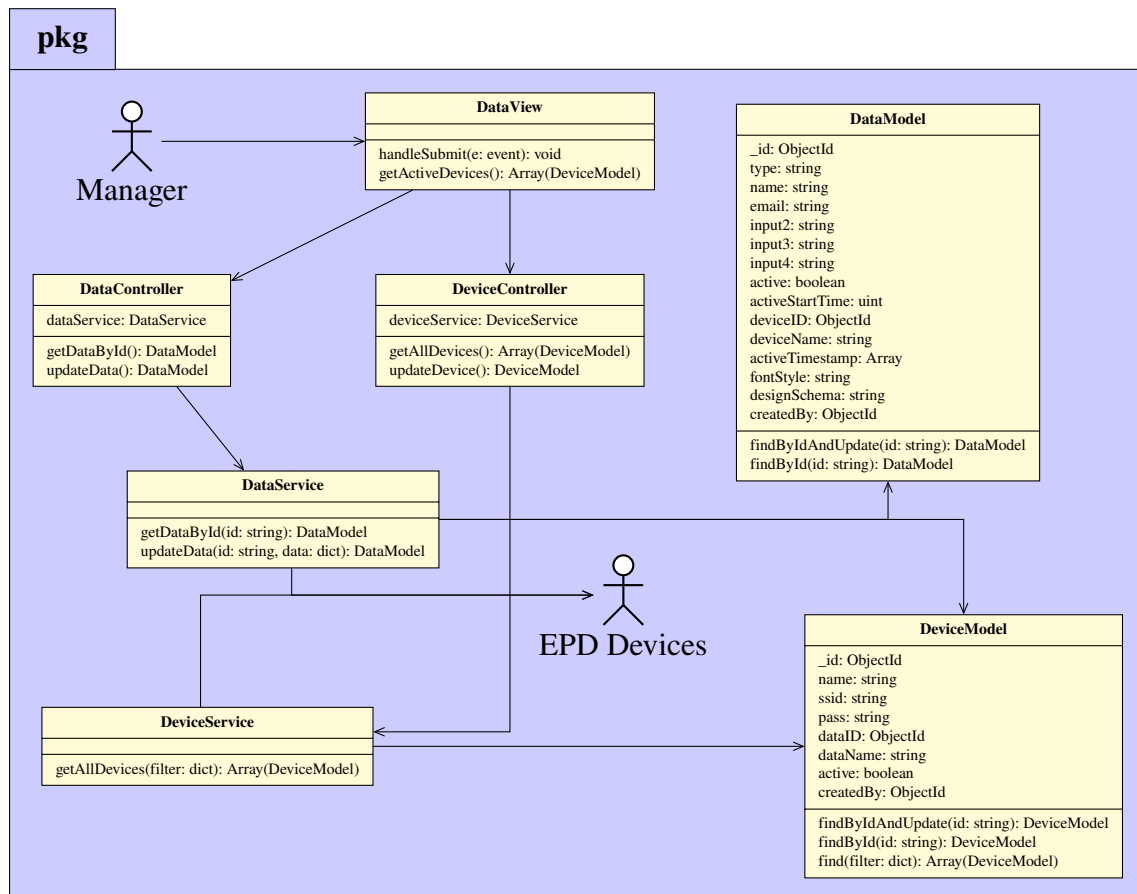**Hình 0.14:** Detailed use case of Data Management function

**Hình 0.15:** Caption

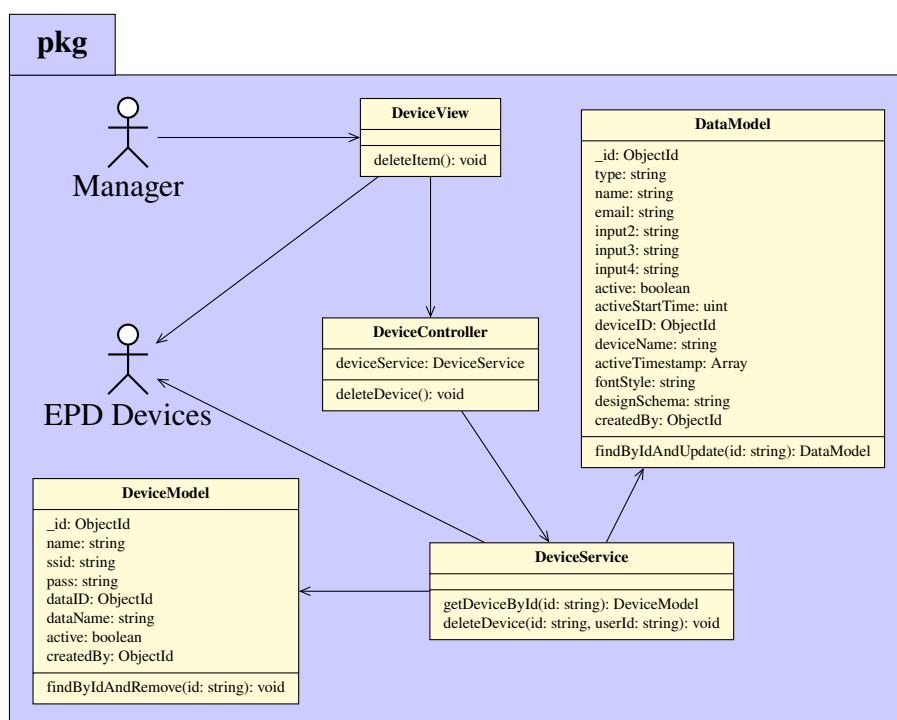## c, Class and sequence diagram of use case "Change a device information"



**Hình 0.16:** Detailed use case of Data Management function

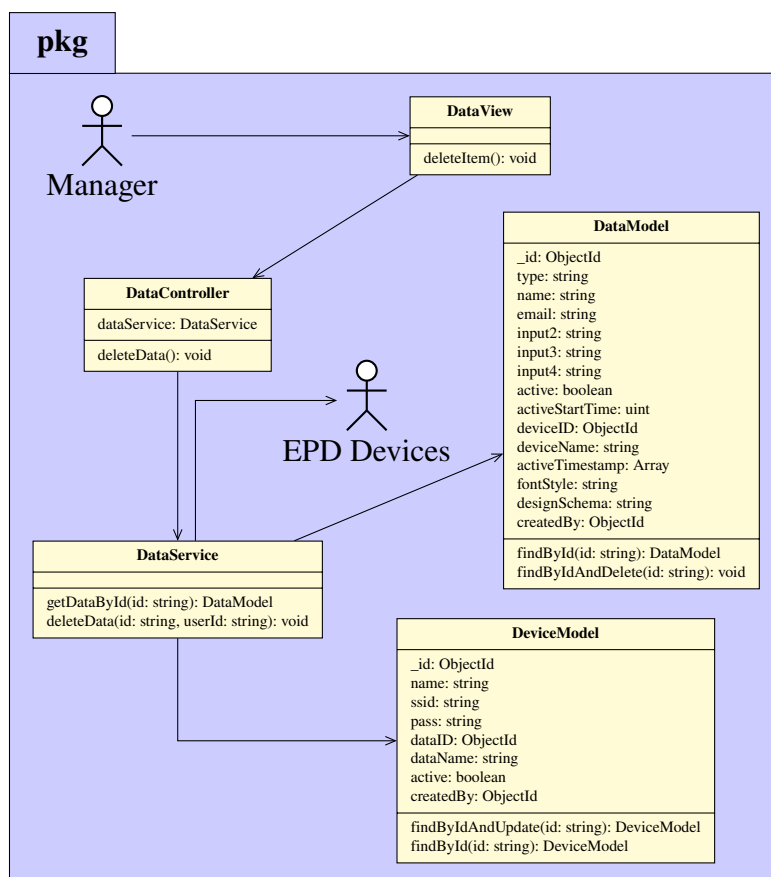**d, Class and sequence diagram of use case "Change data information"**



**Hình 0.17:** Detailed use case of Data Management function

**e, Class and sequence diagram of use case "Remove a device"**



**Hình 0.18:** Detailed use case of Data Management function

## f, Class and sequence diagram of use case "Remove a data"



**Hình 0.19:** Detailed use case of Data Management function

## g, Class and sequence diagram of use case "Register new account"
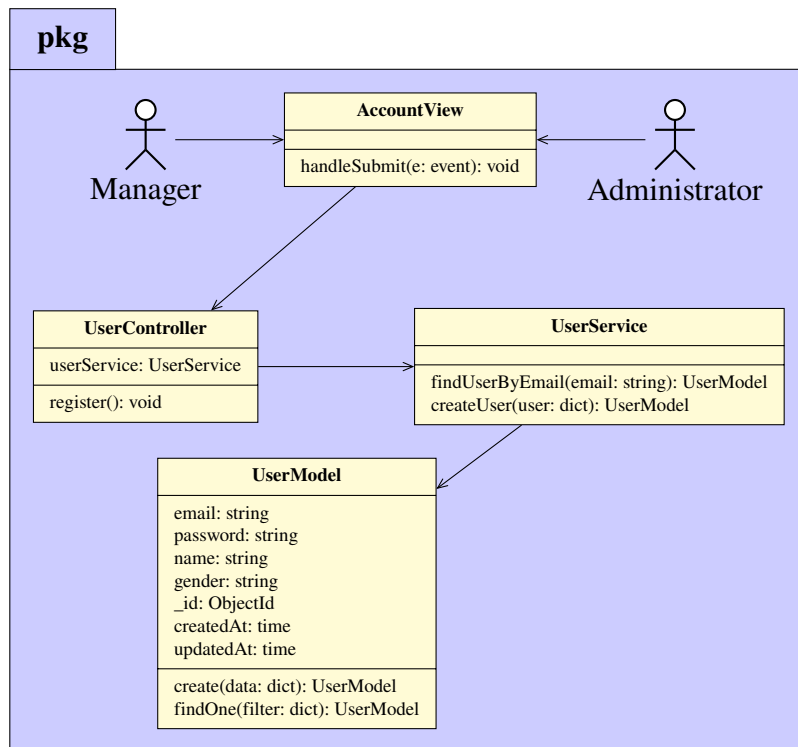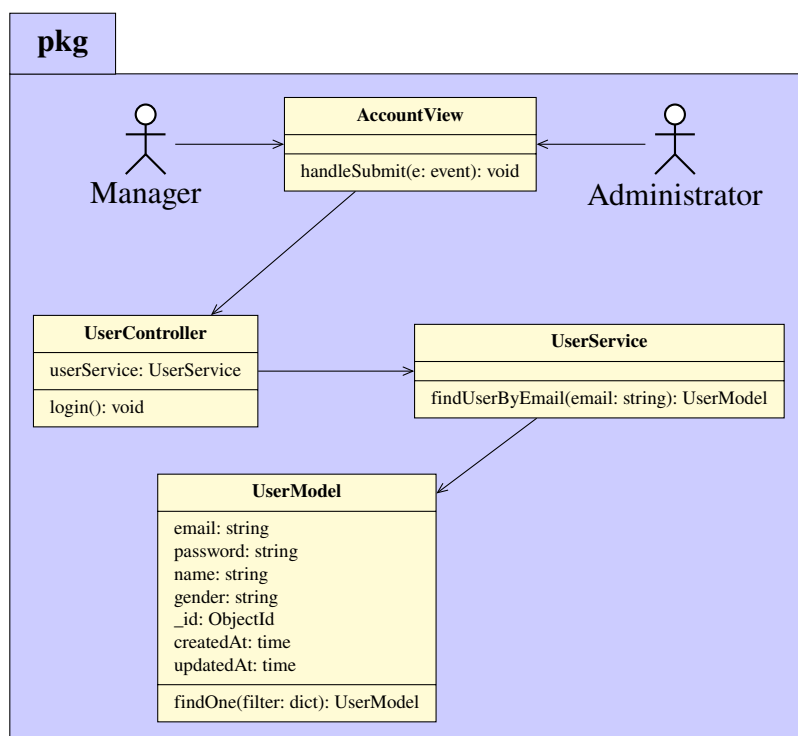


**Hình 0.20:** Detailed use case of Data Management function

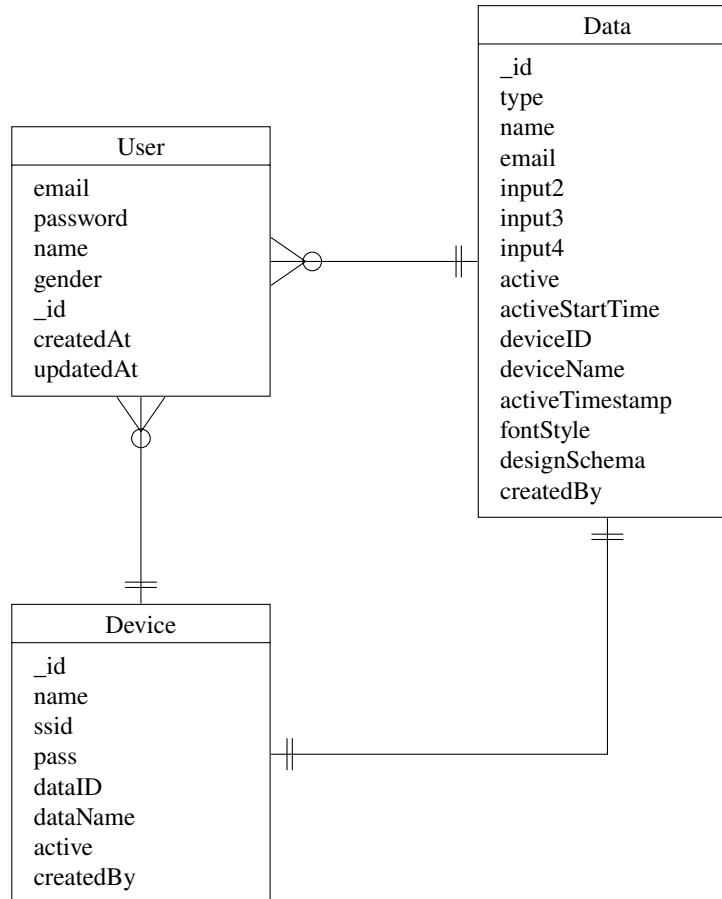## h, Class and sequence diagram of use case "Sign in"



**Hình 0.21:** Detailed use case of Data Management function

### 0.2.3 Database design

Phần này có độ dài từ hai đến bốn trang. Sinh viên thiết kế, vẽ và giải thích biểu đồ thực thể liên kết (E-R diagram). Từ đó, sinh viên thiết kế cơ sở dữ liệu tùy theo hệ quản trị cơ sở dữ liệu mà mình sử dụng (SQL, NoSQL, Firebase, v.v.)

#### a, Entity-Relationship diagram



#### b, Detail database design

| Collection | Field Name | Field Type | Required | Description |
|---|---|---|---|---|
| User | _id | ObjectId | * | User's ID used in the system |
| | email | String | * | User's personal email |
| | password | String | * | Encrypted password of user's account |
| | name | String | * | User's name |
| | gender | String | | User's gender |
| | createdAt | Time | * | Time of account creation |
| | updatedAt | Time | * | Last account update time |
| | _id | ObjectId | * | Data's ID used in the system |
| | type | String | * | Data type |
| | name | String | * | Data name |

14

| Collection | Field Name | Field Type | Required | Description |
|---|---|---|---|---|
| | email | String | | First information: email |
| | input2 | String | | Second information |
| | input3 | String | | Third information |
| | input4 | String | | Fourth information |
| | active | boolean | * | Display status on EPD device |
| | activeStartTime | uint | * | The display start time |
| | deviceID | ObjectId | | ID of displayed device |
| | deviceName | String | | Name of the displayed device |
| | activeTimestamp | Array | * | A list of display time range |
| | fontStyle | String | | Custom display font style |
| | designSchema | String | | Custom display theme |
| | createdBy | ObjectId | * | ID of the user creating the data |
| Device | _id | ObjectId | * | Device's ID used in the system |
| | name | String | | Device name |
| | ssid | String | * | SSID of the network the device is connecting to |
| | pass | String | * | Password of the network the device is connecting to |
| | dataID | ObjectId | | ID of displayed data on the device |
| | dataName | String | | Name of displayed data on the device |
| | active | boolean | * | Connection status of device |
| | createdBy | ObjectId | * | ID of the user creating the data |

**Bảng 1:** Your Table Caption

## 0.3 Application Building

### 0.3.1 Libraries and Tools

In the development of this project, a suite of sophisticated tools was employed to ensure efficiency and quality in the coding process. Central to this toolkit was Visual Studio Code (VS Code), a versatile and powerful code editor known for its user-friendly interface and wide range of extensions. Alongside VS Code, a variety of other tools were integral to the workflow, such as Mongo Compass to manage the MongoDB database and PlatformIO to work with ESP devices. These also included version control systems for tracking changes and collaborating with team members, debugging tools for identifying and resolving issues, and project management software to keep the development process streamlined and on schedule. Table **??** below lists all the tools and applications used during the development process and also the systems used in the project.

| Tools used | Type | Version | Description | URL |
|---|---|---|---|---|
| Visual Studio Code - Insiders (VSCode) | Tools | 1.86.0-insider | Main development environment | https://code.visualstudio.com/insiders |
| PlatformIO | VSCode Plugins | v3.3.2 | A development ecosystem for embedded and IoT applications | https://platformio.org |
| Remote Development | VSCode Plugins | v0.4.1 | A plugin allowing VSCode to develop on remote systems | https://code.visualstudio.com/docs/remote/remote-overview |
| Wokwi Simulator | VSCode Plugins | v2.3.2 | Simulator for Embedded & IoT Systems | https://wokwi.com |
| NodeJS | Language | v20.9.0 | A JavaScript runtime for building scalable network applications | https://nodejs.org |
| Next.JS | Framework | v13.4.5 | A framework for server-side rendered React applications | https://nextjs.org |
| TailwindCSS | Framework | v3.3.2 | A utility-first CSS framework for rapid UI development | https://tailwindcss.com |
| Express.JS | Framework | v4.18.2 | A web application framework for Node.js | https://expressjs.com |
| Mongoose | Library | v8.0.0 | A library for MongoDB and Node.js data modeling | https://mongoosejs.com |
| MQTT NPM Package (mqtt) | Library | v5.3.0 | A library for implementing MQTT protocol in Node.js | https://www.npmjs.com/package/mqtt |
| WaveShare EPD Library | Library | v1.0 | A library for interfacing with e-paper displays | |
| Node Version Manager (nvm) | Tools | v0.39.2 | A tool for managing multiple Node.js versions | https://github.com/nvm-sh/nvm |
| Node Package Manager (npm) | Tools | v10.1.0 | A package manager managing dependencies in Node.js projects | https://www.npmjs.com |
| MongoDB Community Edition for Linux | Tools | v7.0.2 | An open-source document database for Linux systems | https://www.mongodb.com/try/download/community |
| MongoDB Compass | Tools | v1.40.4 | A GUI for MongoDB, simplifying data visualization and management | https://www.mongodb.com/products/compass |
| RabbitMQ | Tools | v3.12.10 | An open-source message broker software | https://www.rabbitmq.com |
| ESP32 C3-Supermini | Device | | A compact microcontroller module for EPD devices | |
| WeAct Studio E-paper 2.9inch display | Device | | A low-power display module for high-contrast, readable content | https://www.weact-tc.cn |
| Ubuntu Server | System | | A dedicated server in Hetzner, for Web hosting and MQTT Broker | https://www.hetzner.com |
| Nginx | Tools | nginx/1.24.0 | A high-performance web server and reverse proxy | https://nginx.org |
| Cloudflare | Tools | | A service for website performance optimization and security | https://www.cloudflare.com |

**Bảng 2:** Danh sách thư viện và công cụ sử dụng

### 0.3.2  Achievement

Sinh viên trước tiên mô tả kết quả đạt được của mình là gì, ví dụ như các sản phẩm được đóng gói là gì, bao gồm những thành phần nào, ý nghĩa, vai trò?

Sinh viên cần thống kê các thông tin về ứng dụng của mình như: số dòng code, số lớp, số gói, dung lượng toàn bộ mã nguồn, dung lượng của từng sản phẩm đóng gói, v.v. Tương tự như phần liệt kê về công cụ sử dụng, sinh viên cũng nên dùng bảng để mô tả phần thông tin thống kê này.

16

With the help of tools and libraries in the table **??** above, the project has reached significant milestones and also released a minimum viable product (MVP) including the Management UI, the back-end server, and a couple of EPD devices that can be implemented in many small business environments. While having some minor performance issues, this MVP still proved its usefulness in various use cases, such as mini-markets, schools, and offices, unveiling the enormous potential of the system in a broader spectrum of the service industry.

The whole system is running on the website so the users don't need to install any additional applications. Also, the users just need to plug the EPD device into the computer via a USB port when needed without additional drivers, and the website will automatically recognize the device. Details of the running system are shown in the table below

| Description | URL |
| --- | --- |
| Management UI | `https://epaper.artsakh.ventures` |
| MongoDB Databases | `mongodb://epaper.artsakh.ventures:27017` |
| Back-end server | `https://epaper.artsakh.ventures/api` |
| MQTT Broker | `mqtts://mqtt.epaper.artsakh.ventures` |
| MQTT Management UI | `https://mqtt.epaper.artsakh.ventures/dashboard` |

**Bảng 3:** Danh sách thư viện và công cụ sử dụng

### 0.3.3 Illustration of main functions

To manage EPD Devices and data, users need to log in with credentials as illustrated in the picture **??**. After logging in, the user can choose to create data/device or go to the dashboard to view from the sidebar on the left.

When creating data/device, the user can easily follow on-screen instructions and fill in the required information. All the steps are annotated in detail, and the rendered display is showing in real-time, which helps improve user experience.

The EPD Device connects to MQTT Broker and subscribe to its ID as topic. The left side of the picture **??** belows indicates the device display when no data is stored, and the right side show the device display when a data is stored in the device. The d evice is running on battery, which can last for 5 - 6 hours before needing to recharge. in the scope of the project, this is still reasonable and acceptable as the product is still in the prototype stage, and not optimized for higher endurance in a real-life environment.

## 0.4 Testing

Phần này có độ dài từ hai đến ba trang. Sinh viên thiết kế các trường hợp kiểm thử cho hai đến ba chức năng quan trọng nhất. Sinh viên cần chỉ rõ các kỹ thuật kiểm thử đã sử dụng. Chi tiết các trường hợp kiểm thử khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục. Sinh viên sau cùng tổng kết về số lượng các trường hợp kiểm thử và kết quả kiểm thử. Sinh viên cần phân tích lý do nếu kết quả kiểm thử không đạt.

## 0.5 Deployment

### 0.5.1 CI/CD

All of the project's source code is hosted and managed on Github, using Continuous Integration and Continuous Delivery (CI/CD) to automate the deployment. When a new commit on main branch is created, a workflow is triggered to update both front-end and back-end in the hosting server. Detail and the log of the workflow can be monitored in Action setion in .

### 0.5.2 Servers

The system uses two dedicated servers both running on Ubuntu 20.04 Jammy, with detail nand purpose are listed in the table below:

| Back-end Server (web-server) | MQTT Broker (laboratory) |
|---|---|
| 65.108.79.164 | 95.217.121.243 |
| Ubuntu 20.04 LTS Jammy | Ubuntu 20.04 LTS Jammy |
| Intel | Intel |
| 64GB | 64GB |
| 4TB | 2TB |
| Nginx, MongoDB, ManagementUI, Back-end, OpenSSL | Nginx, RabbitMQ, OpenSSL |

**Bảng 4:** Danh sách thư viện và công cụ sử dụng

At "web-server" server, Management UI and API server are configured as system daemon, with two service files defined in epaper.service and epaper-backend.service.

### 0.5.3 EPD Devices

Sinh viên trình bày mô hình và/hoặc cách thức triển khai thử nghiệm/thực tế. Ứng dụng của sinh viên được triển khai trên server/thiết bị gì, cấu hình như thế nào. Kết quả triển khai thử nghiệm nếu có (số lượng người dùng, số lượng truy cập, thời gian phản hồi, phản hồi người dùng, khả năng chịu tải, các thống kê, v.v.)