

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

**Xây dựng bảng thông tin sử dụng giấy điện tử
ePaperboard**

(Building Information board using ePaperboard)

Vu Le Nhat Minh

minh.vln194333@sis.hust.edu.vn

Major: School of Information Communication and Technology

Supervisor: MSc. Nguyen Duc Tien _____

Signature

Department: Computer Engineering

School: Information and Communication Technology

HANOI, 01/2024

ACKNOWLEDGMENTS

The students' sincere thanks should send to their lover, family, friends, teachers, and themselves for their hard work and determination to achieve the best results. The acknowledgments should be written in a concrete manner and limited in the range of 100-150 words.

ABSTRACT

Sinh viên viết tóm tắt ĐATN của mình trong mục này, với 200 đến 350 từ. Theo trình tự, các nội dung tóm tắt cần có: (i) Giới thiệu vấn đề (tại sao có vấn đề đó, hiện tại được giải quyết chưa, có những hướng tiếp cận nào, các hướng này giải quyết như thế nào, hạn chế là gì), (ii) Hướng tiếp cận sinh viên lựa chọn là gì, vì sao chọn hướng đó, (iii) Tổng quan giải pháp của sinh viên theo hướng tiếp cận đã chọn, và (iv) Đóng góp chính của ĐATN là gì, kết quả đạt được sau cùng là gì. Sinh viên cần viết thành đoạn văn, không được viết ý hoặc gạch đầu dòng.

Student

(Signature and full name)

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION.....	1
1.1 Motivation	1
1.2 Objectives and scope of the graduation thesis	2
1.3 Tentative solution	2
1.4 Thesis organization.....	2
CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS.....	4
2.1 Status survey	4
2.2 Functional Overview.....	6
2.2.1 General use case diagram.....	6
2.2.2 Detailed use case diagram.....	6
2.2.3 Business process	9
2.3 Functional description.....	10
2.3.1 Description of use case "Create new device".....	10
2.3.2 Description of use case "Modify device information"	11
2.3.3 Description of use case "Remove a device".....	13
2.3.4 Description of use case "Add new data"	13
2.3.5 Description of use case "Modify data information"	14
2.3.6 Description of use case "Remove data"	15
2.3.7 Description of use case "Create account"	16
2.3.8 Description of use case "Sign in".....	17
2.4 Non-functional requirement.....	18
2.4.1 Security	18
2.4.2 Performance	19
2.4.3 User Experience.....	19

2.4.4 Flexibility and Scalability	19
CHAPTER 3. METHODOLOGY.....	21
3.1 Management UI (Front-end side)	21
3.1.1 NextJS	21
3.1.2 TailwindCSS	22
3.2 Server (Back-end side)	22
3.2.1 API Server	23
3.2.2 MQTT Broker	23
3.3 EPD device	24
3.3.1 ESP32-C3 Supermini	24
3.3.2 E-paper display	24
3.4 PlatformIO.....	24
3.5 Cloudflare.....	25
CHAPTER 4. DESIGN, IMPLEMENTATION, AND EVALUATION.....	26
4.1 Architecture design.....	26
4.1.1 Software architecture selection.....	26
4.1.2 Overall design.....	27
4.1.3 Detailed package design	28
4.2 Detailed design.....	29
4.2.1 User interface design.....	29
4.2.2 Layer design	30
4.2.3 Database design	47
4.3 Application Building.....	48
4.3.1 Libraries and Tools.....	48
4.3.2 Achievement.....	50
4.3.3 Illustration of main functions	51

4.4 Testing and evaluation.....	57
4.4.1 Evaluation.....	57
4.4.2 Testing.....	59
4.5 Deployment	61
4.5.1 Continuous Integration/Continuous Delivery (CI/CD)	61
4.5.2 Servers.....	62
4.5.3 EPD Devices	63
CHAPTER 5. SOLUTION AND CONTRIBUTION	66
5.1 Display custom text.....	66
5.1.1 UTF-8 vs. UTF-16	66
5.1.2 Two ways of process and display text	67
5.1.3 First Solution	68
5.1.4 Second Solution.....	70
5.2 Solutions for managing devices.....	72
5.2.1 Central Dock	72
5.2.2 Battery-less EPD devices.....	73
5.2.3 Current solution.....	74
5.2.4 Optimization and future work.....	75
5.3 ESP32 Optimization	75
5.3.1 Battery	75
5.3.2 Resources.....	76
5.4 Security and vulnerabilities	79
5.4.1 MongoDB Hack.....	79
5.4.2 TLS/SSL and Certificate Authorities (CA)	81
CHAPTER 6. CONCLUSION AND FUTURE WORK	83
6.1 Conclusion.....	83

6.2 Future work.....	83
REFERENCE	86

LIST OF FIGURES

Figure 2.1	General use case diagram of the EPD display system	7
Figure 2.2	Detailed use case of User's Account Management function .	7
Figure 2.3	Detailed use case of Data Management function	8
Figure 2.4	Detailed use case of Device Management function	9
Figure 2.5	"Creating new device" business flow	9
Figure 2.6	"Creating and displaying new data" business flow	10
Figure 4.1	Overall architecture of the system	26
Figure 4.2	MVCS architecture	27
Figure 4.3	MQTT microservice	27
Figure 4.4	Model class diagram	28
Figure 4.5	Service class diagram	28
Figure 4.6	Controller Class diagram	29
Figure 4.7	View Class diagram	29
Figure 4.8	Mockup display of authentication page	30
Figure 4.9	Mockup display of dashboard page	30
Figure 4.10	Mockup display of modal component	30
Figure 4.11	Mockup display of creation page	30
Figure 4.12	Class diagram of use case "Register a new device"	31
Figure 4.13	Sequence diagram of use case "Register a new device"	31
Figure 4.14	Class diagram of use case "Add a new data"	32
Figure 4.15	Sequence diagram of use case "Add a new data" Cont'd on next page	33
Figure 4.16	cont'd from previous page	35
Figure 4.17	Class diagram of use case "Change a device information" . .	36
Figure 4.18	Sequence diagram of use case "Change a device information"	37
Figure 4.19	Class diagram of use case "Change data information"	38
Figure 4.20	Sequence diagram of use case "Change data information" Cont'd on next page	39
Figure 4.21	cont'd from previous page	40
Figure 4.22	Class diagram of use case "Remove a device"	41
Figure 4.23	Sequence diagram of use case "Remove a device"	42
Figure 4.24	Class diagram of use case "Remove a data"	43
Figure 4.25	Sequence diagram of use case "Remove a data"	44
Figure 4.26	Class diagram of use case "Register new account"	45

Figure 4.27	Sequence diagram of use case "Register new account"	45
Figure 4.28	Class diagram of use case "Sign in"	46
Figure 4.29	Sequence diagram of use case "Register new account"	46
Figure 4.30	Entity-Relationship diagram	48
Figure 4.31	Repository's Project screen	50
Figure 4.32	Login screen	52
Figure 4.33	Dashboard Screen	53
Figure 4.34	"Choose data type" screen in "New Data"	54
Figure 4.35	"Choose device" screen in "New Data"	55
Figure 4.36	"New Device" page	56
Figure 4.37	Data dashboard, with a PiP showing device log	57
Figure 4.38	Performance reports by Lighthouse	58
Figure 4.39	A successful GitHub deploy workflow	62
Figure 4.40	Back and front image of ESP32-C3 Supermini	64
Figure 5.1	UI of TheDotFactory tool	67
Figure 5.2	Enter Caption	77
Figure 5.3	Enter Caption	78
Figure 5.4	Enter Caption	80
Figure 5.5	Enter Caption	81

LIST OF TABLES

Table 2.1	Comparison table of four EPD manufacturers in e-paper display solutions	5
Table 2.2	List of use cases	10
Table 4.1	Database design	47
Table 4.2	List of libraries and tools used in the project	49
Table 4.3	Project service URLs	51
Table 4.4	Lighthouse performance score of four criteria	58
Table 4.5	Input details of "Adding new data" process	59
Table 4.6	Testing results of "Adding new data" process	60
Table 4.7	Input details of "Creating new device" process	60
Table 4.8	Testing results of "Creating new device" process	60
Table 4.9	Testing results of "Removing device" process	61
Table 4.10	Testing results of "Removing device" process	61
Table 4.11	Detail of two Hetzner's dedicated servers	63
Table 4.12	System services	63
Table 4.13	ESP32-C3's specs	65
Table 4.14	SPI connection schematic	65
Table 5.1	System services	76
Table 5.2	System services	76
Table 5.3	System services	79

LIST OF ABBREVIATIONS

Abbreviation	Definition
ADC	Analog to Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
BSS	Block Started by Symbol
CA	Certificate Authority
CI/CD	Continuous Integration/Continuous Deployment
EPD	Electronic Paper Display
FCP	First Contentful Paint
GPIO	General Purpose Input/Output
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
MVCS	Model View Controller Service
MVP	Minimum Viable Product
NFC	Near Field Communication
OTA	Over-The-Air
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
SPA	Single Page Application
SPI	Serial Peripheral Interface
SSR	Server Side Rendering
TLS/SSL	Transport Layer Security/Secure Sockets Layer
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface

CHAPTER 1. INTRODUCTION

1.1 Motivation

In the era of Industry 4.0, businesses are inundated with an unprecedented volume of data, a direct consequence of advancements in digital technology and interconnected systems. This data surge has propelled companies to innovate and adopt sophisticated methods for efficient display and management. The integration of big data analytics, cloud computing, and IoT (Internet of Things) has enabled businesses to process and visualize complex data sets more efficiently than ever before. As a result, data-driven decision-making has become a cornerstone of modern business strategies.

However, the challenge remains in presenting this vast amount of information in a way that is accessible, engaging, and easy to understand for diverse audiences. Traditional display methods, such as printed signs or digital screens, fall short in this fast-paced environment. Printed materials, while visually familiar, incur recurring costs due to the need for constant updates and replacements. On the other hand, conventional digital displays, though easier to update, often involve high energy consumption and maintenance costs. The transition from traditional display methods to more dynamic, cost-effective, and energy-efficient solutions is not just an option but a necessity to keep pace with the rapidly evolving business landscape of the digital age.

By combining the latest advancements in electronic ink technology with energy-efficient design principles, e-paper display technology has emerged as a transformative solution in various industries besides traditional and digital screens. Its unique ability to retain text and images without a constant power supply makes it ideal for a wide range of applications, from dynamic signage in retail and transportation to real-time information displays in healthcare and corporate environments, and even revolutionizes information delivery within the service industry. Also, this technology aligns with the growing emphasis on environmental sustainability and energy efficiency in the industry, offering businesses a cost-effective way to stay ahead of the information curve, ensuring that customers have access to the most current data while reducing their carbon footprint to achieve sustainability goals. This change is not merely an improvement but a necessary evolution to meet the demands of modern service environments where information is both a tool and an asset.

This project aims to focus on building an EPD (Electronic Paper Display) de-

vices system prototype that implements the use of e-paper display technology in various aspects, including warehousing, education, enterprises, etc.

1.2 Objectives and scope of the graduation thesis

There are numerous applications of e-paper being utilized in various large systems, ranging from public transportation and logistics to education. However, these are extensive systems that require the handling of many specialized tasks and call for a complex and costly display system. In other words, EPD systems used in these businesses are highly optimized for the specific needs of each enterprise, making the costs expensive and not suitable for smaller systems.

Building on these limitations, the project aims to develop an EPD devices system that focuses more on smaller businesses and use cases and targets the following main objectives: (i) Provide a cost-effective way of data displaying using e-paper display technology in small businesses; (ii) Provide a management UI and back-end system that suits most aspect's needs, firstly focusing on retails, offices, hospitals, schools, and exhibitions; and (iii) Optimize EPD devices to achieve better performance and higher quality in various harsh environments.

1.3 Tentative solution

To achieve the goals set out in Section 1.2, the project will focus on developing the system using a client-server model combined with microservices to ensure flexibility, ease of management, and scalability when necessary.

On the client side, the project uses the NextJS framework to develop a user-friendly UI and custom library to receive and display data in EPD devices that run on ESP32-C3 supermini development modules. On the back-end side, the project uses ExpressJS and MongoDB to build an API server containing user data and RabbitMQ as an MQTT Broker to communicate with EPD devices via MQTT protocol. Details about the utilized model and the technologies used in the system will be specifically presented in chapters 3 and 4.

About the EPD devices, the project uses an ESP32-C3 Supermini development board to receive and display data on an e-paper display panel. Details of the device are also discussed in chapter 3 and chapter 4.

1.4 Thesis organization

The rest of this thesis is structured as follows:

Chapter 2 presents the survey process and requirement analysis of the project, providing an overview of its functionality and detailing some of the prominent features of the EPD display system.

Based on the requirements outlined in Chapter 2, Chapter 3 will focus on the main technologies for system development. The system will comprise three main parts: (i) a website for managing information and display devices; (ii) a Node.js server and MQTT Broker to handle communication between display devices and the server, as well as process requests from the website; (iii) EPD devices to receive and display information.

Chapter 4 will delve deeper into the architectural design of each component, database design, application development process, testing, and system security.

Chapter 5 discusses the significant contributions and standout solutions that help the system address current issues such as remote device management, retrieving valuable device information, and securing the system. The remaining issues or challenges faced during system development are also discussed in this chapter.

Finally, Chapter 6 concludes with the main contributions of the project and the future development direction of the system.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

Based on the real-world problems discussed in Chapter 1, Chapter 2 aims to delve deeper into the existing products and solutions that can help address these issues. Specifically, it will also analyze some detailed use cases of the EPD display system, as outlined in Section 2.2.

2.1 Status survey

There are several e-paper display manufacturers that take advantage of EPD technology to provide solutions to multiple different businesses. This section will focus on four leading e-paper display providers: **E Ink**, **Seekink**, **Inkcase**, and **Pervasive Displays**.

E Ink is a leader in electronic paper (e-paper) display technology, co-founded in 1997 by MIT undergraduates and a professor from the MIT Media Lab. They offer a variety of e-paper display modules suited for diverse applications. Their products are used in several sectors, including reading and writing, education, business and office, mobile and wearables, retail, logistics and factories, healthcare, transportation, automotive, and innovative design. They also showcase customer-specific applications and provide optimized solutions and system integration for businesses. Although they have quality products and solutions aimed at a broad customer segment spread across various fields, E Ink focuses only on core e-paper technology with a few essential products such as eReaders, electronic shelf labels, and other low-power display needs. While beneficial for brand recognition and expertise in a specific area, this specialization can limit market reach and diversification compared to companies with a broader product range. Also, the manufacturing process for E Ink displays is complex and costly, which can lead to higher product prices, impacting competitiveness, especially in price-sensitive markets. That's why E Ink's specialized technology might not be as widely recognized or preferred by the average consumer in many specific regions, including Vietnam.

Seekink is recognized as a leading electronic paper display (EPD) manufacturer with eight years of experience. Seekink operates its automatic manufacturing base, which is capable of mass production, and emphasizes a high level of customization to meet diverse needs across markets. However, Seekink is not widely recognized in Vietnam, primarily because their marketing and sales strategies are focused on large enterprises. This focus on big businesses typically involves offering high-priced, specialized products and services that might not cater to the broader consumer market or smaller businesses. This approach can limit their market presence

and brand recognition among the general public and smaller enterprises.

InkCase is a product line from the Singapore-based consumer electronics design house Gajah. They specialize in offering protective cases for smartphones with added functionalities and are also involved in various innovative display solutions, including intelligent information display, healthcare, and tracking.

Pervasive Displays is a leading manufacturer in the electronic paper display (EPD) industry, specializing in the design, manufacture, and marketing of e-paper displays. They have a manufacturing capacity of over 2 million pieces per week and are financially stable, being a part of the world's largest display company. Their products and services are tailored to industrial applications, with a focus on the design and manufacturing of ultra-low power displays, and are often ordered in large quantities by other major industrial clients.

With the assessments mentioned above, the features of the four e-paper display manufacturers can be summarized in the table 2.1 below.

Features	E Ink	Seekink	InkCase	Pervasive Displays
Prices	Contact sale	Contact sale	Not available	Contact sale
Client type	[p2.5cm]Business Commercial user	Business	Commercial user	[10pt][-80pt][p2.5cm]Business Commercial user
Use case	[10pt][-10pt][p2cm]Business Education Healthcare Logistics Transportation Automotive	[10pt][-10pt][p2cm]Business Retail Education Transportation Healthcare IoT	[p2cm]Transportation Business	[00pt][-20pt][p2cm]Logistic IoT Healthcare Education Business
Management System	✓	✓	✓	✓
Allow custom display	Customized for each customer	Customized for each customer	Not available	Customized for each customer
Connectivities	[10pt][-10pt][p2cm]Bluetooth Internet Wired	[p5cm]Bluetooth Internet Wired	[p5cm]Bluetooth Internet	Not published
Other Products	Commercial e-paper electronic devices	E-paper display modules and devices	E-paper protective phone cases	E-paper display modules and devices
Available in Vietnam	×	×	×	×

Table 2.1: Comparison table of four EPD manufacturers in e-paper display solutions

Having learned from the survey of the above four systems and other real-life use cases, this project has concluded some objectives of building an EPD display system, including designing and building EPD devices, as well as including some

essential features fitting in multiple use cases.

First, take the advantage of the electrophoretic display technology in displaying information. With the unique characteristic of e-paper display, the EPD device should be capable of displaying data for a long period of time without frequent refreshing. The device also needs to be energy-saving and able to operate in reasonable time without recharging.

Second, manage EPD devices in a system via a central hub, which can be a cloud server, communicating with EPD devices and backend server using MQTT protocol and other wireless connectivities, or a local dock that is responsible for receiving data and displaying on EPD panels. Each solution's advantages and drawbacks are discussed in section 5.3 in chapter 5.

Third, implement the EPD device system in some initial use cases to test the performance and effectiveness of the system in solving real-life problems. The initial use cases include name tags for students, employees, and clients; price tags in mini-markets and logistics services; and room and other signages.

Finally, implement security in the system and enhance user experience.

2.2 Functional Overview

2.2.1 General use case diagram

The general use case diagram of the EPD Display System is illustrated in Figure 2.1. As per the diagram, the system involves three main actors: The Manager, The Administrator, and the EPD devices. The Manager can manage EPD devices, data, and their accounts. The Administrator inherits the functions of the Manager and can also manage and test the devices more advanced. On the other hand, the EPD devices act as end-users, receiving and displaying data on the screen. They also interact with the system via the MQTT protocol.

2.2.2 Detailed use case diagram

a, Detailed use case of User's Account Management function

Figure 2.2 below describes the detailed use case diagram of the User's Account Management function, including the Manager and Administrator. Users can create a new account, log in, and modify personal information.

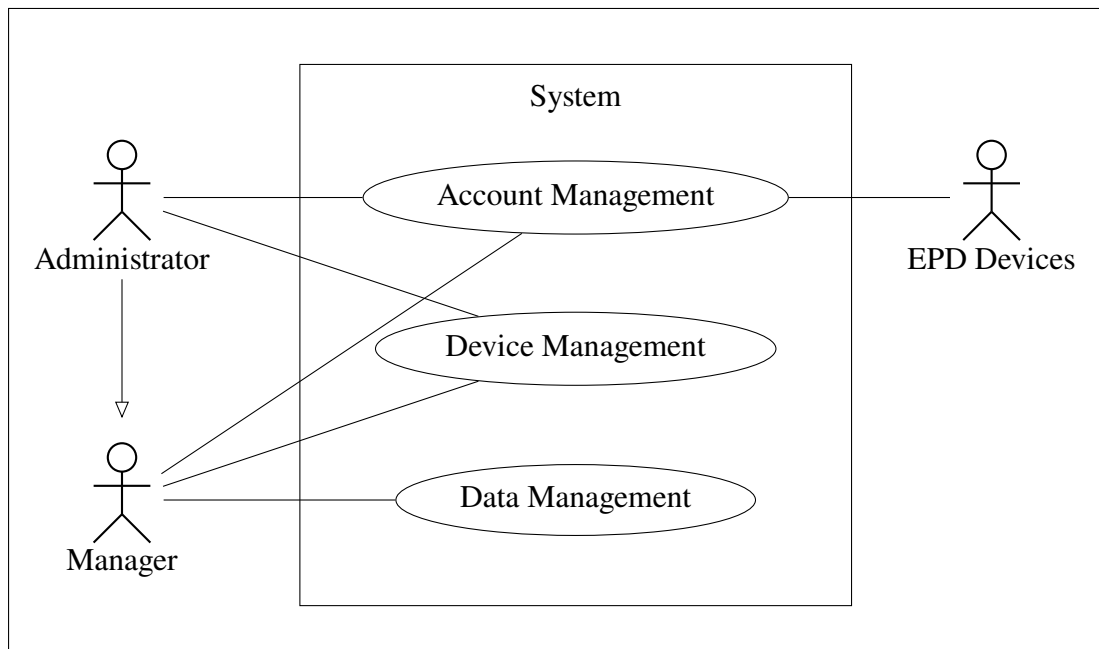


Figure 2.1: General use case diagram of the EPD display system

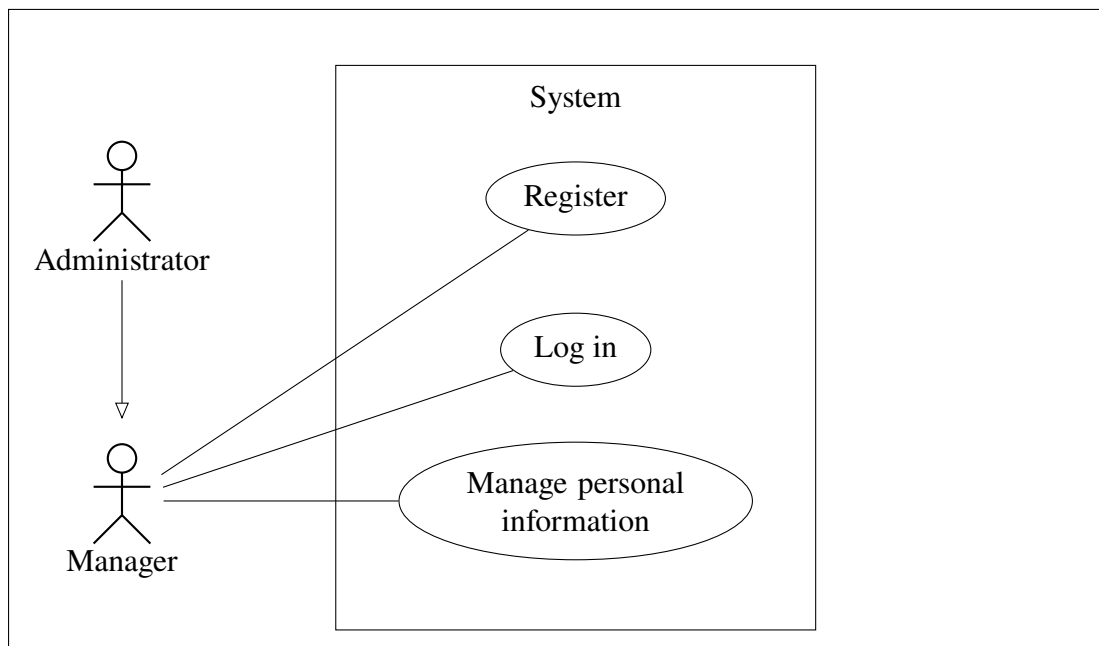


Figure 2.2: Detailed use case of User's Account Management function

b, Detailed use case of Data Management function

A detailed use case diagram of the Data Management function of the Manager is displayed in figure 2.3. This function enables the Manager to view, add, modify, remove data, and choose whether or not to display it on the EPD devices.

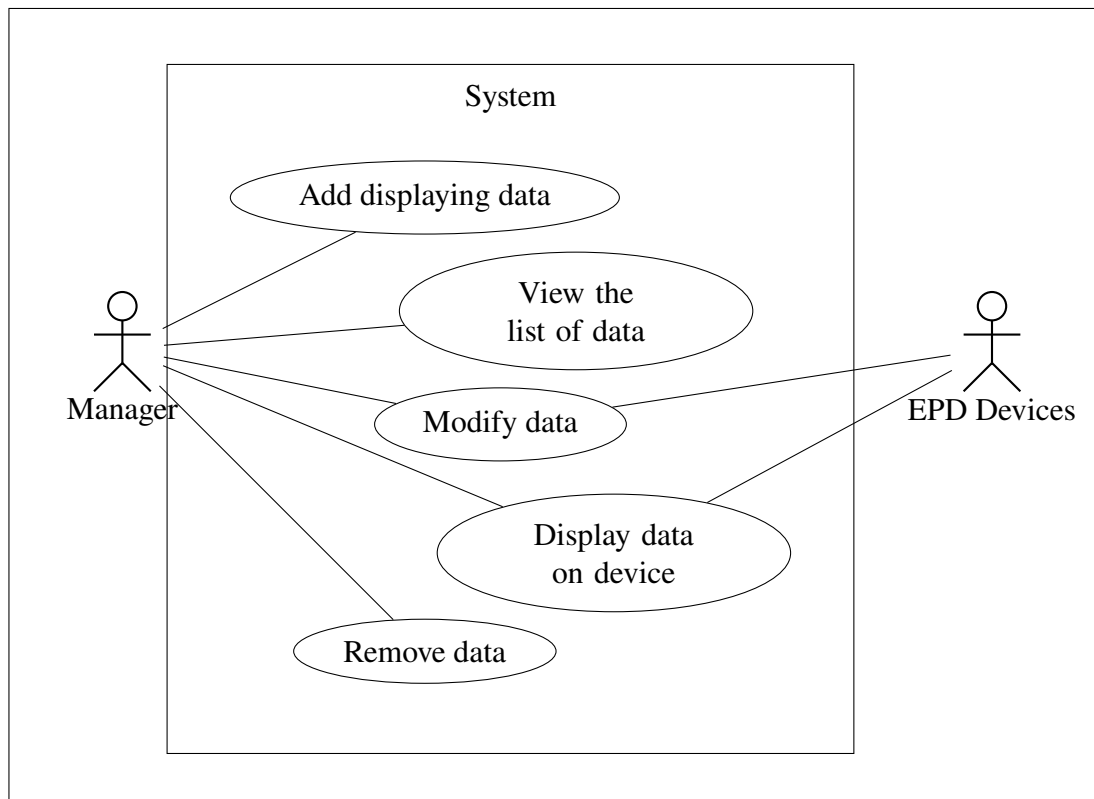


Figure 2.3: Detailed use case of Data Management function

c, Detailed use case of Device Management function

Figure 2.4 illustrates the use case diagram of the Device Management function, showing two actors participating in the system. This function enables the Manager to view, add, modify, or remove device information. The Administrator, inheriting the functions of the Manager, can also debug the device after connecting it to the computer via a USB port.

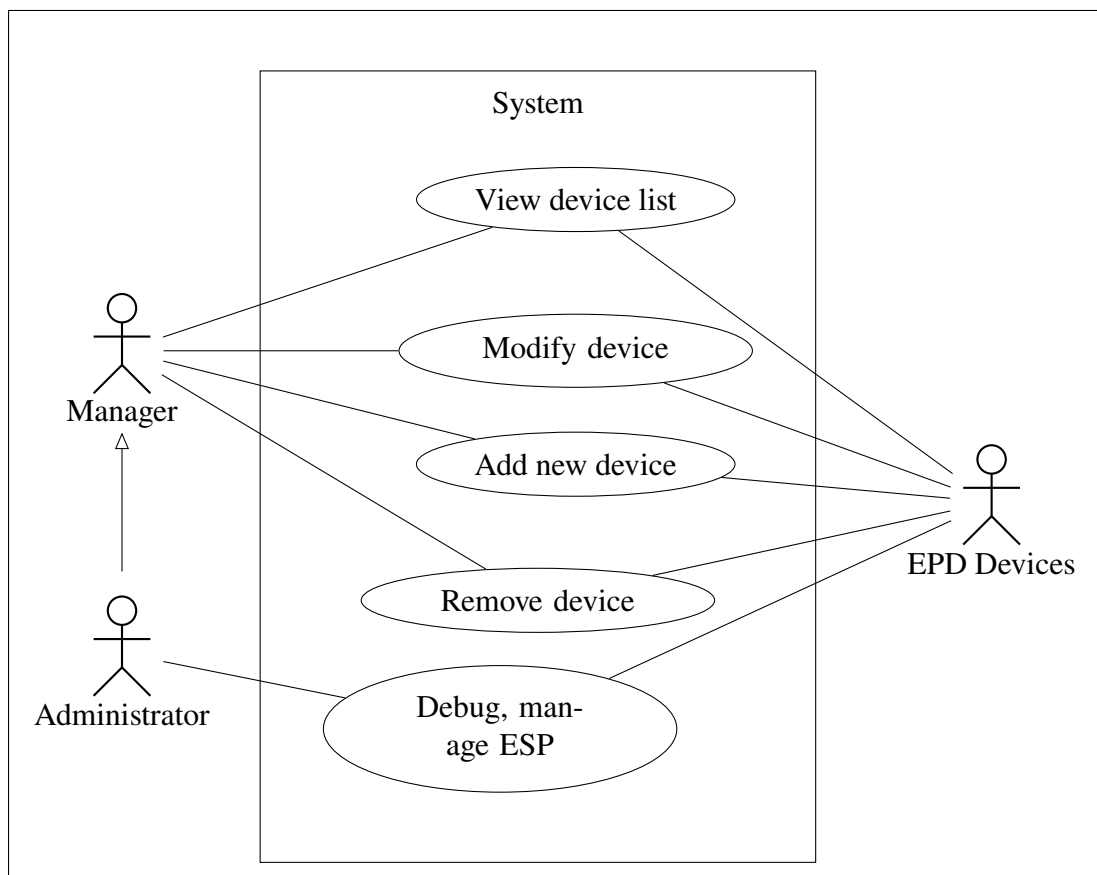


Figure 2.4: Detailed use case of Device Management function

2.2.3 Business process

The system comprises several business processes, with the most prominent ones being the process of adding data and displaying it on EPD devices and the process of adding EPD devices to the system. Each flow showcases how the system communicates with the device via USB and the MQTT protocol and how the device receives and processes data.

a, "Creating new device" flow

Figure 2.5 below shows the business process when the Manager creates and registers a new device to the system. This process requires the users to connect the EPD device to the computer via a USB port, choose from the list of connected devices, and then fill in the required information before submitting it to the system. Upon receipt of the submitted data, the system sends a data write request to the device via Serial Port. The EPD device then connects to the Internet with the received information and connects to the MQTT Broker before publishing its connection status to the system.

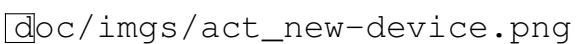
 doc/imgs/act_new-device.png

Figure 2.5: "Creating new device" business flow

b, "Creating and displaying new data" flow

The following describes the business process when users add new data to the system, as depicted in Figure 2.6. Firstly, the user selects the type of data they want to display. Afterward, they add the corresponding detailed information based on the chosen data type. Users can also decide whether or not to allow display on the EPD device. If they choose to do so, the system will provide a list of devices connected to the MQTT Broker for the user to choose from. The user will then provide additional information about how to display the data on the selected device. Once all the necessary information has been provided, the system will send MQTT messages through the MQTT Broker. The selected device will receive and display the data and send the status back to the system via the MQTT protocol.

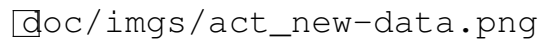
 doc/imgs/act_new-data.png

Figure 2.6: "Creating and displaying new data" business flow

2.3 Functional description

ID	Name
UC01	Create new device
UC02	Modify device information
UC03	Remove device
UC04	Add new data
UC05	Modify data information
UC06	Remove data
UC07	Create account
UC08	Sign in

Table 2.2: List of use cases

2.3.1 Description of use case "Create new device"

ID	UC01	Name	Create new device
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager. To create a new device and edit device information in case the device is not connected to the Internet, users need to connect the device to the computer via a USB port.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Select "New Device" function
	2	System	Retrieve and display the list of devices connected via USB

Main scenario (success)	No.	Executed by	Action
	3	Manager	Choose a device from the list
	4	System	Connect to the device via Serial Port
	5	System	Display the interface to enter device information
	6	Manager	Fill in the device information (<i>described below *</i>)
	7	Manager	Send a request to create a new device
	8	System	Store device information at the server and transmit data to the connected device
	8.1	System	Store device information at the server
	8.2	System	Transmit data to the EPD device
	8.2.1	Connected EPD device	Process the received information
	8.2.2	Connected EPD device	Connect to the Internet and MQTT Broker
	8.2.3	Connected EPD device	Send status information to the server
	8.2.4	System	Receive and edit information of the new device
	9	System	Notify successful device creation
Extensions	No.	Executed by	Action
	4a.	System	Error message: Unable to connect to the device
	8.1a	System	Error message: Need to enter all required fields of the device if the Manager misses any
	8.2.2a	System	If no update response is received from the device, save the device information
Post-condition	The system store new device information in database		

2.3.2 Description of use case "Modify device information"

ID	UC02	Name	Modify device information
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager, and the device is registered in the system. To modify a device not connected to the Internet, users need to connect the device to the computer via a USB port.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Access Dashboard, select Device
	2	System	Retrieve and display the list of registered devices
	3	Manager	Choose a device from the list and select Edit
	4	System	Check the status of the device
	4.1	System	If the device is not connected to the system, display a message asking the user to connect the device via USB
	4.2	Manager	Connect the device via USB and choose from the list of connected devices
	5	System	Display the interface to enter device information (described below *)
	6	Manager	Fill in and send a request to change device information
	7	System	Save the new updated information
	7.1	System	If the device is connected via USB, transmit data to the connected EPD device
	7.2	System	If the device is connected to the MQTT server, send a request to change information on the device via the MQTT protocol
	8	Connected EPD device	Process the received information
	8.1	Connected EPD device	If the device is connected via USB, connect to the Internet and MQTT Broker
	9	Connected EPD device	Send status information to the server
Extensions	10	System	Receive and edit information of the new device
	11	System	Notify successful device information change
	No.	Executed by	Action
	4.2a.	System	Error message: Unable to connect to the device
	7a.	System	Error message: Need to enter all required fields of the device if Manager misses any

Extensions	No.	Executed by	Action
	10a.	System	If no update response is received from the device, save the device information
Post-condition	The system store new device updated information in database		

2.3.3 Description of use case "Remove a device"

ID	UC03	Name	Remove a device
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager, and the device is registered in the system.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Access the Dashboard, select Device
	2	System	Retrieve and display the list of registered devices
	3	Manager	Choose a device from the list and select Delete
	4	System	Display a warning asking the user to confirm deletion
	5	Manager	Confirm deletion of data
	6	System	Check the display status of the device
	6.1	System	If the device is not displaying data, remove the device from the system
	6.2	System	If the device is displaying data, send a request to delete data on the device via MQTT
	6.2.1	EPD Device	Process the received information, delete the displaying data information
	6.2.3	System	Remove device display information from the data
	7	System	Notify successful remove
Extensions	No.	Executed by	Action
	6a.	System	End the use case if the user confirms not to delete the device
	6.2.3a	System	Error message: Unable to delete the device if an error occurs during deletion
Post-condition	No		

2.3.4 Description of use case "Add new data"

ID	UC04	Name	Add new data
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Select "New Data" function
	2	System	Display the interface to choose data type to add
	3	Manager	Choose the type of data from the list
	4	System	Display the interface to enter data information
	5	Manager	Fill in the data information (<i>described below **</i>)
	5.1	System	If the user chooses to display on a device, show the interface to select device and design
	5.2	Manager	Choose device to display, and choose the design
	6	Manager	Send a request to create new data
	7	System	Save new data information on the server
	7.1	System	If the user chooses to display data, transmit data to the connected EPD device
	7.2	EPD Device	Process the received information
	7.3	EPD Device	Display the data
	7.4	EPD Device	Send status information to the server
	7.5	System	Receive and update data information
	8	System	Notify successful creation of the data
Extensions	No.	Executed by	Action
	4a	System	Error: Need to enter all required fields of the device if the Manager misses any
	5.1a	System	If there are no active devices, notify users and finish the use case
	7.1a	System	Error: Can't write data to the device
Post-condition	The system store new data in database		

2.3.5 Description of use case "Modify data information"

ID	UC05	Name	Modify data information
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager, and the data is stored in the system.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Access Dashboard, select Data
	2	System	Retrieve and display the list of data
	3	Manager	Choose a data from the list and select Edit
	4	System	Display the interface to enter data information
	5	Manager	Fill in the data information (<i>described below **</i>)
	5.1	System	If the user chooses to display on a device, show the interface to select device and design
	5.2	Manager	Choose device to display, and choose the design
	6	Manager	Send a request to update data
	7	System	Save the updated data on the server
	7.1	System	If the user chooses to display data, transmit data to the connected EPD device
	7.2	EPD Device	Process the received information
	7.3	EPD Device	Display the data
	7.4	EPD Device	Send status information to the server
	7.5	System	Receive and update data information
	8	System	Notify successful update
Extensions	No.	Executed by	Action
	4a	System	Error: Need to enter all required fields of the device if the Manager misses any
	5.1a	System	If there are no active devices, notify users and finish the use case
	7.1a	System	Error: Can't write data to the device
Post-condition	The system store new updated data in database		

2.3.6 Description of use case "Remove data"

ID	UC06	Name	Remove data
Actor	Manager, System, EPD device		
Pre-condition	The user logs into the system as a Manager, and the data is stored in the system.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Access Dashboard, select Data
	2	System	Retrieve and display the list of data
	3	Manager	Choose a data from the list and select Delete
	4	System	Display a warning asking the user to confirm deletion
	5	Manager	Confirm deletion of data
	6	System	Check the display status of the data
	6.1	System	If the data is being displayed, send a request to delete data on the device via MQTT
	6.2	EPD Device	Process the received information, delete the displaying data information
	6.3	System	Remove data information on the device
	7	System	Remove the data from the system
	8	System	Notify successful remove
Extensions	No.	Executed by	Action
	6a	System	End the use case if the user confirms not to delete the data
	7a	System	Error message: Unable to delete the device if an error occurs during deletion
Post-condition	No		

2.3.7 Description of use case "Create account"

ID	UC07	Name	Create account
Actor	Manager, System		
Pre-condition	No		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Go to the sign-up page
	2	System	Display the interface to fill user information

Main scenario (success)	No.	Executed by	Action
	3	Manager	Fill in the user information (<i>described below ***</i>)
	4	Manager	Send a request to create new account
	5	System	Save new user information on the server
	6	System	Notify successful creation of the user, and direct to log-in page
Extensions	No		
Post-condition	No		

2.3.8 Description of use case "Sign in"

ID	UC08	Name	Sign in
Actor	Manager, System		
Pre-condition	The user have stored account.		
Main scenario (success)	No.	Executed by	Action
	1	Manager	Go to the sign-in page
	2	System	Display the interface to fill user email and password
	3	Manager	Fill in the user credentials
	4	Manager	Send a request to check account
	5	System	Notify successful log-in, and redirect to dashboard page
Extensions	No.	Executed by	Action
	5a	System	Error: User's credentials are incorrect
Post-condition	No		

(*) The input data of the device:

No.	Name	Description	Required
1	Name	Name of the device	✓
2	SSID	Name of the network the device connects to	✓
3	Password	Password of the network the device connects to	✓

(**) The input data of data:

No.	Name	Description	Required
1	Type	Type of the data	✓
2	Name	Name of the data	✓
3	Email	Email data, depend on data type	×
4	Student ID	Student ID data, depend on data type	×
5	Class	Class of the student, depend on data type	×
6	Employee ID	Employee ID data, depend on data type	×
7	Department	Department of employee, depend on data type	×
8	Category	Category of product, depend on data type	×
9	Price	Price of product, depend on data type	×
10	Address	Address data, depend on data type	×
11	Purpose	Room's purpose, depend on data type	×
12	Manager	Room's manager, depend on data type	×
13	Status	Room's status, depend on data type	×
14	Active	Display on device status	✓
15	Font Style	Display main font style	Yes, if <i>Active</i> is true
16	Theme	Display schema	Yes, if <i>Active</i> is true
17	Device	Device to display	Yes, if <i>Active</i> is true

(***) The input data of the account:

No.	Name	Description	Required
1	Name	Name	✓
2	Email	Email	✓
3	Password	Password	✓

2.4 Non-functional requirement

Given the system's unique nature, where a user has to manage a vast array of data across numerous devices in an open environment, system security is a top priority when facilitating communication between devices. Additionally, the system requires transparency and user-friendliness for new users, including those with physical impairments. Moreover, to operate reliably in large enterprise environments with numerous devices, users, and data, the system also demands high fault tolerance, ease of inspection, upgrades, and maintenance.

2.4.1 Security

Containing sensitive user and business data, the system needs to ensure privacy and security among EPD devices and servers. The system also requires authenticated and authorized services so that only users with specific permissions are allowed to access particular services. Moreover, it is also crucial to protect the sys-

tem from outside attacks by implementing secure connections between services using TLS/SSL protocol. The details of this problem and its implemented solution are discussed in chapter 5.

2.4.2 Performance

With the characteristic of having to update in real time, the system also requires multiple performance criteria. First, the EPD devices should process data in a reasonable time and maintain a stable connection with the server in the event of receiving requests. The EPD device also needs long-lasting battery life and sufficient resources to store and handle data. On the other hand, the system also needs to manage many EPD devices and users efficiently, especially in big businesses. Chapter 5 will also provide a clear view of the performance evaluation process and various techniques used to improve both EPD devices and the server's overall quality.

2.4.3 User Experience

UX/UI is also one of the most essential requirements of the system. The web interface needs to be user-friendly, with a consistent layout and color scheme across different pages. Additionally, the interface should support responsive design, making it adaptable to various screen sizes on different devices. Moreover, information in the system needs to be delivered to the users efficiently and clearly, making it easy to understand for all ages.

About the EPD devices, the display needs to have a clear and simple layout, ensuring the data is well-organized and easy to view, especially in complicated use cases. The e-paper screen also needs to refresh clearly and does not show any remaining black pixels after the refresh.

Both the website and the EPD devices should follow a minimalistic design, only showing the users what they want to show. The color, layout, and design patterns should be consistent among devices and pages, overall enhancing user experience and brand recognition.

2.4.4 Flexibility and Scalability

The system is still in the development and expansion phase, so it needs to be designed for easy modification and upgrading to suit new requirements as they arise. Changes or additions to features or devices should also not affect the system's operation. The system also needs to be fault-tolerance provide detailed logs in the event of errors and have backup storage in an emergency. All logic failures must not cause the system to stop running, and the system should have handling strategies

for all of them and a notification system to indicate the error to the users.

CHAPTER 3. METHODOLOGY

Based on the real-life surveyed situation and the detailed system analysis in the chapter 2, this chapter will center on the technologies used to develop the EPD Management System. This system comprises three main components: the Management Front-end, Back-end, and EPD Devices as the separated components. The Management UI is built using NextJS and TailwindCSS to provide an easy-to-use dashboard for managing multiple devices and data efficiently. The UI also incorporates protocols to connect to HTTP, MQTT servers, and EPD devices via Serial Port. All data is processed and stored on the server using MongoDB and ExpressJS. The server communicates with EPD devices through the MQTT protocol, using RabbitMQ as a broker and SSL/TLS to ensure a secure connection. The EPD devices connect to the broker and subscribe to a specific topic to receive and handle any information requested by users from the front-end side and send the status back to the broker.

3.1 Management UI (Front-end side)

3.1.1 NextJS

Next.js is a leading open-source JavaScript framework built on top of React, designed to simplify the development of single-page applications (SPAs) and server-rendered applications. As a React framework, Next.js enables the development of user interfaces using React components while offering additional features and optimizations. It stands out for its ease of use, performance optimization, and hybrid static and server rendering capabilities, contributing to improved user experiences and faster page loads. Next.js also supports automatic code splitting, ensuring that each page only loads the necessary JavaScript.

With its renowned Server-side Rendering feature (SSR) and other optimizations, Next.js significantly enhances website performance and responsive time while keeping the development task manageable and scalable. The framework is also equipped with tools for automatically measuring, tracking, and visualizing page performance metrics, such as Next.js Speed Insights and the useReportWebVitals hook, enabling developers to continuously maintain and improve websites' performance.

These performance benefits, including enhanced application performance with SSR and effective code management, have made Next.js a suitable choice for the project.

3.1.2 TailwindCSS

Tailwind CSS, a utility-first CSS framework, offers significant benefits for speeding up rendering times in web development. One of its primary benefits is its approach to generating the smallest possible CSS file for a project. The size of CSS files directly impacts website loading speed. Larger CSS files take longer to load, leading to increased page load times. Slow-loading websites can suffer from higher bounce rates, reduced user engagement, and negative impacts on search engine rankings. Tailwind fixes this by only creating CSS for the components actually used in the project. Combined with minification and network compression, this results in remarkably small CSS files, often less than 10kB, even for large projects, which is essential for high performance.

In addition to reducing CSS file size, Tailwind CSS also enhances website accessibility and performance by reducing the amount of HTML code. This is achieved through the use of more efficient selectors, a core aspect of Tailwind's design philosophy. By optimizing the HTML and CSS of a website, Tailwind CSS helps improve both the speed and quality of the user experience.

Tailwind also contributes significantly to the improvement of website vitals. For instance, the First Contentful Paint (FCP) metric, which is crucial for understanding perceived load speed, can see a reduction of up to 36% compared to other styling methods like styled components. Such improvements in Speed Index and Largest Contentful Paint metrics indicate that Tailwind CSS not only accelerates development time but also enhances the performance and responsiveness of the final product.

3.2 Server (Back-end side)

Because of the complexity of the system that needs both processing data from the client side and also communicating with EPD devices, this Server section is divided into two main parts: API server handling user's requests and storing data in MongoDB database, and RabbitMQ broker communicating with devices and sending data back to ExpressJS server to process.

The two servers used in the system are dedicated servers hosted in Hetzner, a prominent web hosting provider and data center operator in Europe. Hetzner is well-known for its robust data center infrastructure, a wide range of hosting services, and commitment to connectivity and performance. Hosted in Hetzner's dedicated servers, the system can benefit from the single allocation of full CPU, RAM, and storage resources that are not shared with other users. It also improves performance, security, and the scalability of the system significantly and enables

developers to fully control and customize for specific needs.

3.2.1 API Server

This part of the back-end server is responsible for receiving user requests, relaying data to the MQTT Server, and storing data in the system. The server is written in the MVCS design pattern for code transparency and management and leverages **ExpressJS**, a minimal and flexible Node.js web application framework, to help enhance the server's capabilities. It enables efficient API creation and middleware integration, streamlining development and improving maintainability.

To store and secure user data, the back-end system uses **MongoDB Community Edition for Linux** (hereafter referred to as '**MongoDB**'), which stands out from other database solutions due to its distinct advantages in storing and processing data. MongoDB is adept at handling vast amounts of user and device data, storing it securely across distributed systems, in this case, the *clusters*. This design enhances data redundancy and security and allows for horizontal scaling, accommodating large-scale data without compromising system performance. In tandem with **ExpressJS**, this also ensures efficient, high-performance query and transaction processing promptly while keeping the development and maintenance processes streamlined and minimal.

Utilized for API design and documentation in this project, Swagger from OpenAPI, known as OpenAPI Specification (OAS), offers a comprehensive, detailed description of the API structure, including endpoints, operations, and parameters. This documentation tool not only aids in API development and testing by allowing for interaction with API resources without implementing logic, but it also streamlines client SDK generation across various programming languages. Leveraging Swagger documentation in the project helps speed up the development cycle and enhance efficiency in identifying bugs and issues during testing, ultimately resulting in a more robust and user-friendly application.

3.2.2 MQTT Broker

To enable real-time communication and data transfer between users and devices, the system also operates as an MQTT broker, facilitating the exchange of information between parties. This is acquired by using RabbitMQ, an open-source message broker software that enables applications to communicate with each other and exchange information efficiently. It acts as an intermediary for messaging by accepting and forwarding messages, making it a critical tool for handling asynchronous communication between EPD devices and the server. Its reliability and scalability make it a preferred choice for enterprises needing to ensure message delivery

without loss, even in high-throughput scenarios. RabbitMQ's ability to decouple processes also leads to more resilient and manageable application architectures.

3.3 EPD device

3.3.1 ESP32-C3 Supermini

The EPD devices in the project run on ESP32-C3 Supermini, a compact, general-purpose Wi-Fi and Bluetooth LE module inheriting the features from ESP32 microcontrollers. The ESP32-C3 Supermini uses ESP32-C3, a 32-bit, RISC-V-based MCU with 400KB of SRAM, capable of running at up to 160 MHz, as its core. It also integrates 2.4 GHz Wi-Fi and Bluetooth 5 (LE) with long-range support. It also features 22 programmable GPIOs (General Purpose Input/Output), supporting a variety of interfaces, including ADC (Analog to Digital Converter), SPI (Serial Peripheral Interface), UART (Universal Asynchronous Receiver/Transmitter), I2C (Inter-Integrated Circuit), RMT (Remote Control), etc. The ESP32-C3 Supermini includes a Type-C USB, 4MB of Flash, and supports 12x IO, which is sufficient for most of the development tasks in the project.

3.3.2 E-paper display

The display panel uses the Waveshare 2.9-inch e-Ink Display Module, which is a versatile and efficient display solution for various applications. It features a 2.9-inch screen with a resolution of 296x128 pixels and supports partial refresh, allowing for low power consumption and a wide viewing angle. The display presents content in black and white with a 4-level grey scale, providing a paper-like effect that is ideal for applications such as price tags, shelf labels, and industrial instruments. It is connected and communicated with the ESP32-C3 Supermini module via the SPI interface, which is particularly notable for its ultra-low power consumption, making it suitable for this system where power efficiency is crucial.

3.4 PlatformIO

PlatformIO is a powerful and versatile open-source ecosystem widely used for IoT and embedded systems development. Its cross-platform compatibility allows integration with Visual Studio Code, offering flexibility for developers. PlatformIO stands out for its extensive support for numerous microcontrollers and development boards, including Arduino, ESP8266, ESP32, and STM32, catering to a diverse range of projects. One of the most renowned features of PlatformIO is its comprehensive suite of development tools, including a robust library manager that simplifies the incorporation and management of external libraries and a unified debugger that supports a wide range of debugging tools. It also supports multiple development environments, which is particularly valuable in complex development scenar-

ios, enabling developers to create versatile applications that can be easily adapted and deployed across different hardware platforms. Last but not least, PlatformIO has also raised a vibrant community around it, enriching the platform and providing extensive documentation, active user forums, and shared knowledge, making it an invaluable tool for both hobbyists and professionals in embedded systems development.

With the aid of PlatformIO, the project has significantly advanced in both development efficiency and technical robustness. It provides a comprehensive toolset for managing, developing, and testing the development boards, which is also integrated with various development environments, making the development and debugging process easier.

3.5 Cloudflare

Cloudflare is a widely recognized internet services company that specializes in providing a range of solutions aimed at enhancing online security and performance. Renowned for its expansive infrastructure designed to handle significant web traffic volumes, Cloudflare serves a diverse clientele, from individual website owners to large enterprises, ensuring their online presence is secure, fast, and reliable. It also provides free SSL/TLS as part of its range of security and performance services for websites, designed to be easily implemented and managed, providing robust encryption for data in transit between the user's browser and the web server.

CHAPTER 4. DESIGN, IMPLEMENTATION, AND EVALUATION

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.

4.1 Architecture design

4.1.1 Software architecture selection

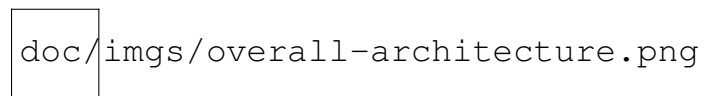


Figure 4.1: Overall architecture of the system

Figure 4.1 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 applica-

tions, 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.

4.1.2 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 4.3). 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 4.2).

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.

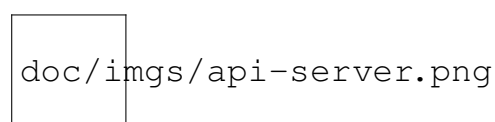


Figure 4.2: MVCS architecture

b, MQTT Service

This service in the system acts as an intermediary, managing the state of all MQTT client connections, subscriptions, and message exchanges.



Figure 4.3: MQTT microservice

4.1.3 Detailed package design

a, Models

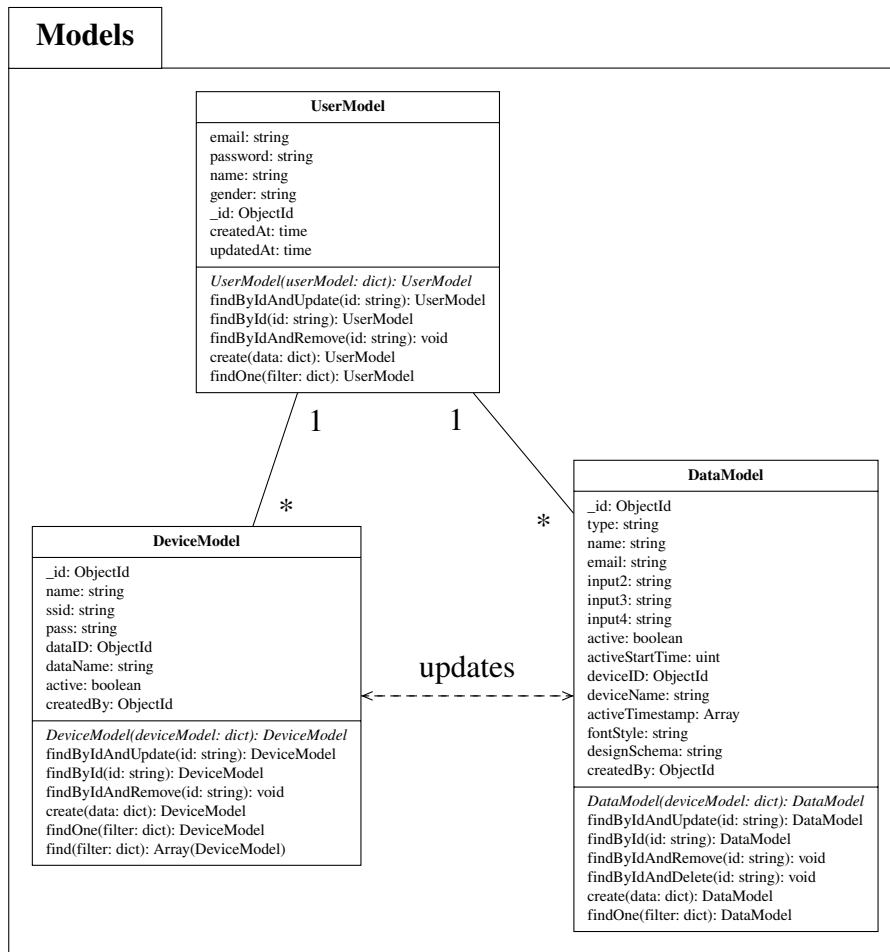


Figure 4.4: Model class diagram

b, Services

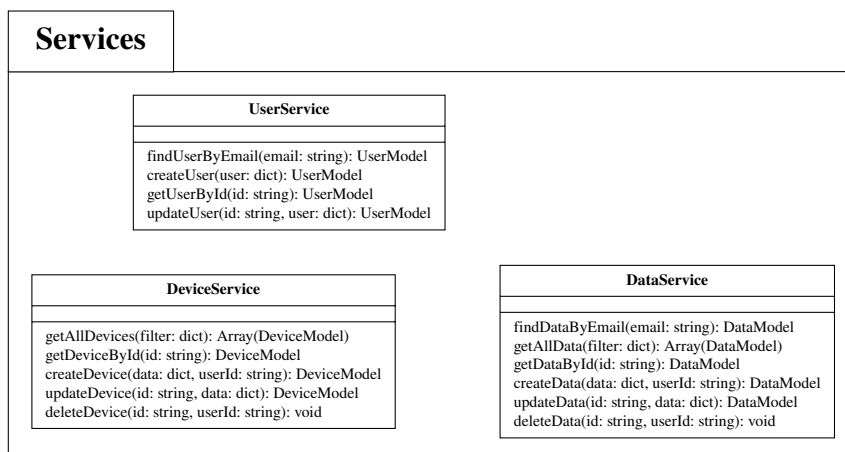
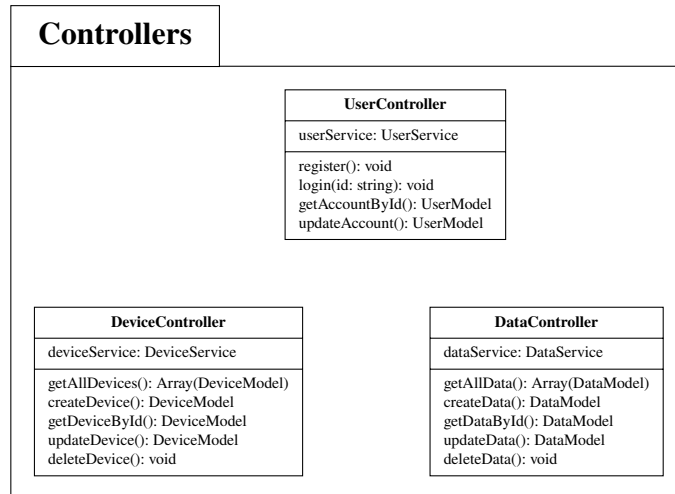
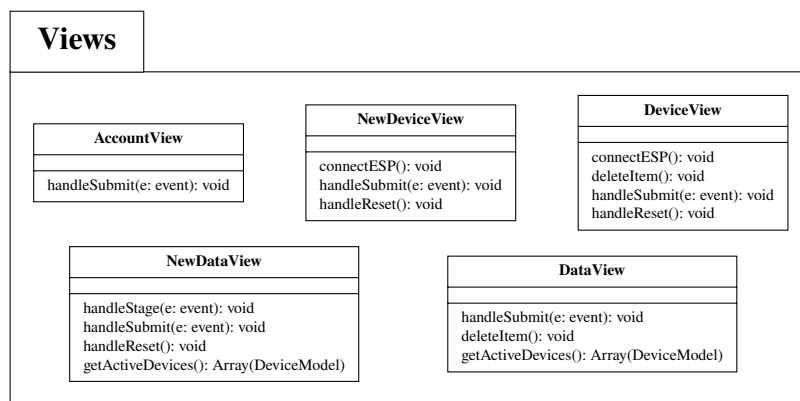


Figure 4.5: Service class diagram

c, Controllers**Figure 4.6:** Controller Class diagram**d, Views****Figure 4.7:** View Class diagram**4.2 Detailed design****4.2.1 User interface design**

This system's architecture and user interface 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 (ranging from 1366×768 and higher), facilitating ease of use and comprehensive information display. This focus on computer-targeted design means that the system is not intended for mobile use. 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. There are four main layouts

used on the website: authentication, dashboard, modal, and creation. The users use the authentication page (figure 4.8) for authenticating purposes, including sign-in and sign-up. The dashboard page layout in picture 4.9 shows a table of EPD devices and data, listing all information about each item and the buttons on the right side to open functional modals to view, edit, and delete, which are illustrated in figure 4.10. In the creation page in figure 4.11, users can create new devices and data by providing details in the inputs and choosing options from the dropdown. The layouts and the buttons follow the minimalistic design pattern and colors, improving the visual experience for users.

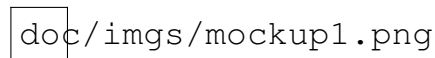
A placeholder box containing the text 'doc/imgs/mockup1.png'.

Figure 4.8: Mockup display of authentication page

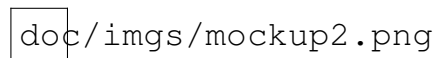
A placeholder box containing the text 'doc/imgs/mockup2.png'.

Figure 4.9: Mockup display of dashboard page

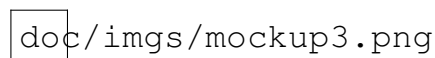
A placeholder box containing the text 'doc/imgs/mockup3.png'.

Figure 4.10: Mockup display of modal component

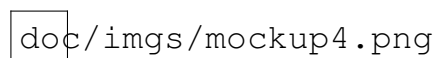
A placeholder box containing the text 'doc/imgs/mockup4.png'.

Figure 4.11: Mockup display of creation page

4.2.2 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"

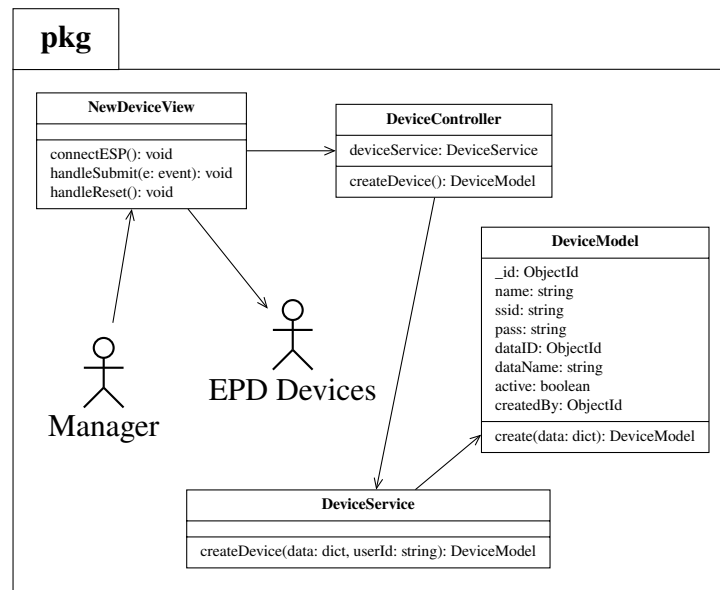


Figure 4.12: Class diagram of use case "Register a new device"

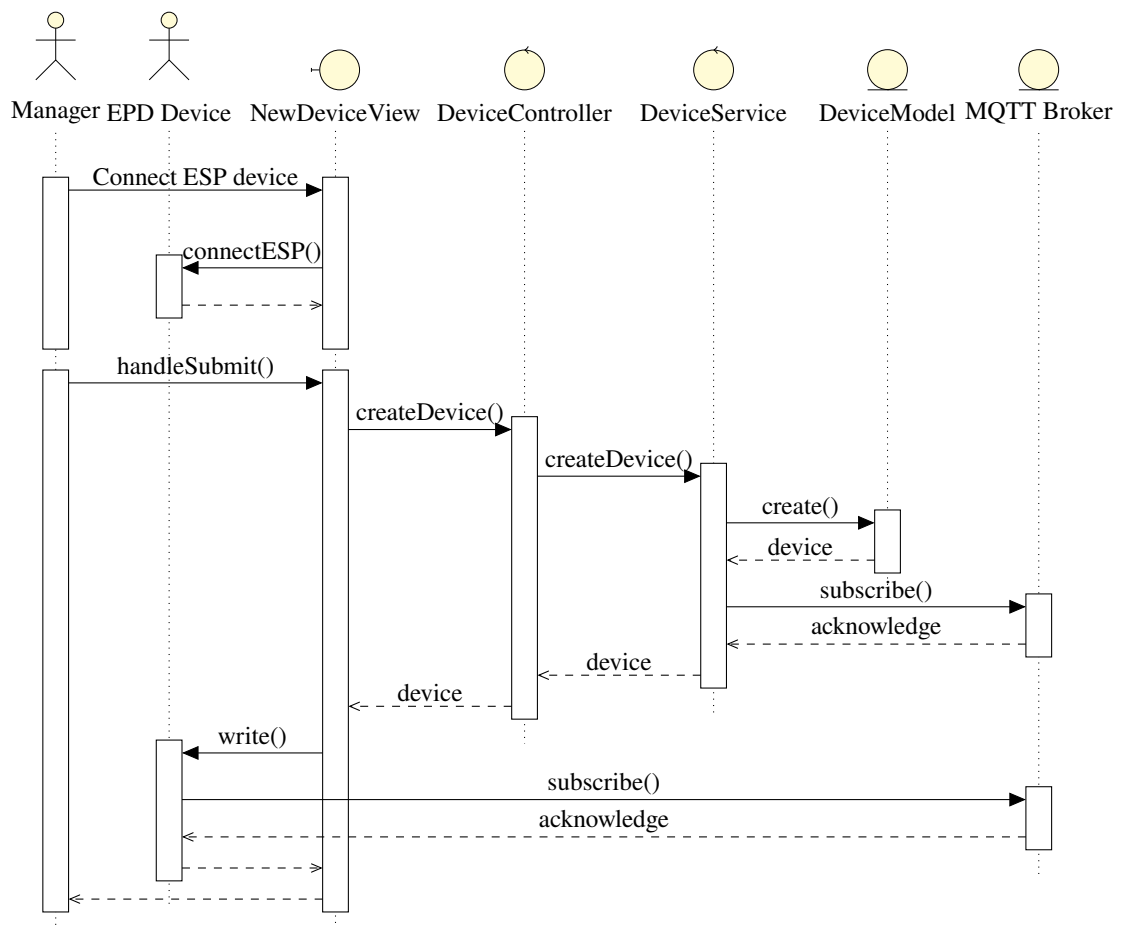


Figure 4.13: Sequence diagram of use case "Register a new device"

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

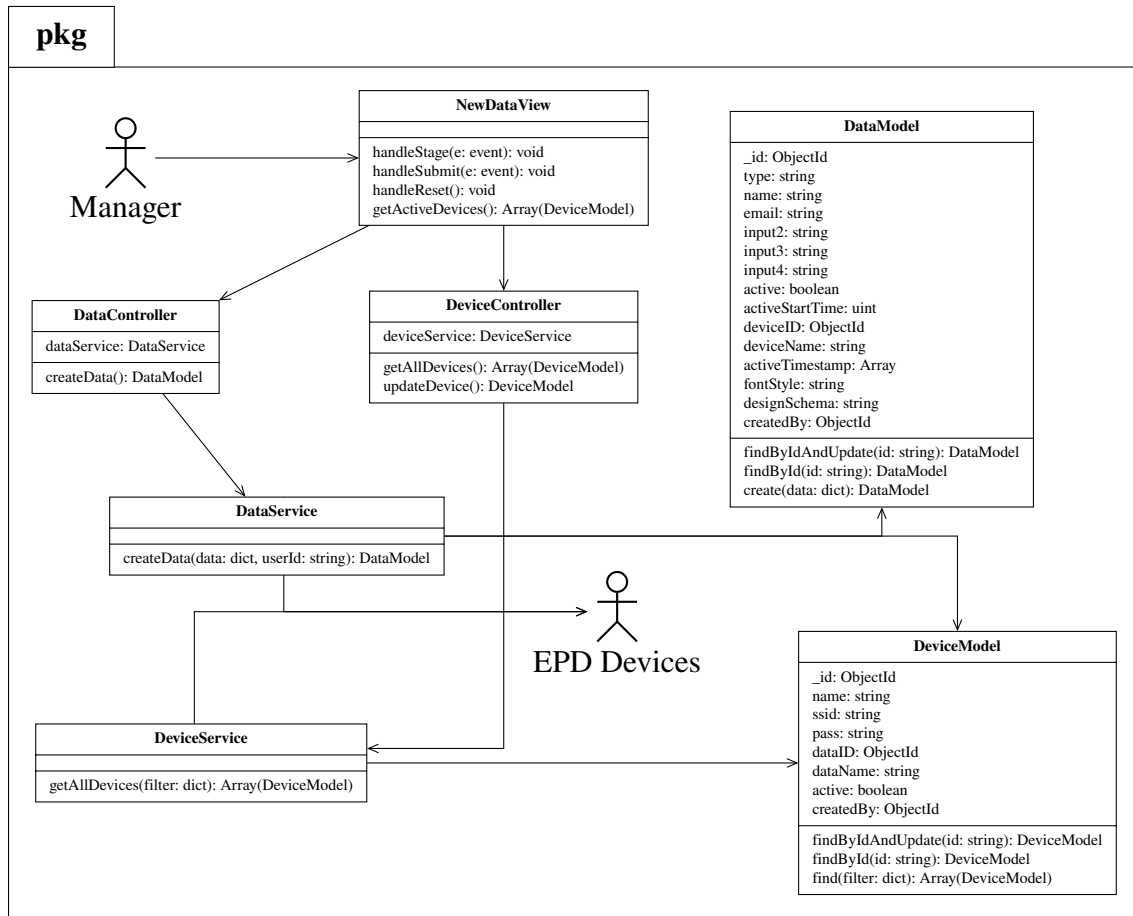


Figure 4.14: Class diagram of use case "Add a new data"

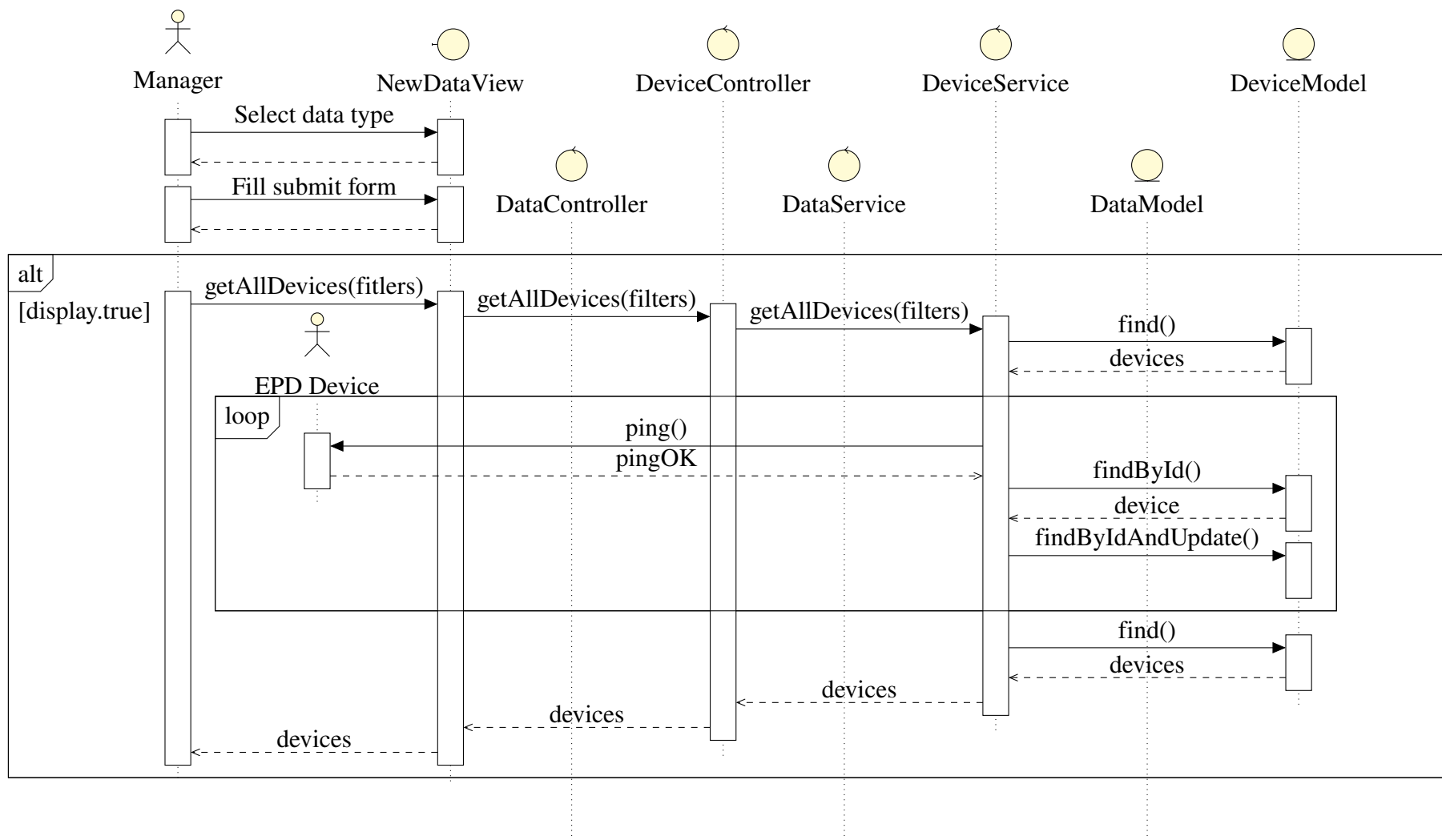


Figure 4.15: Sequence diagram of use case "Add a new data" | Cont'd on next page

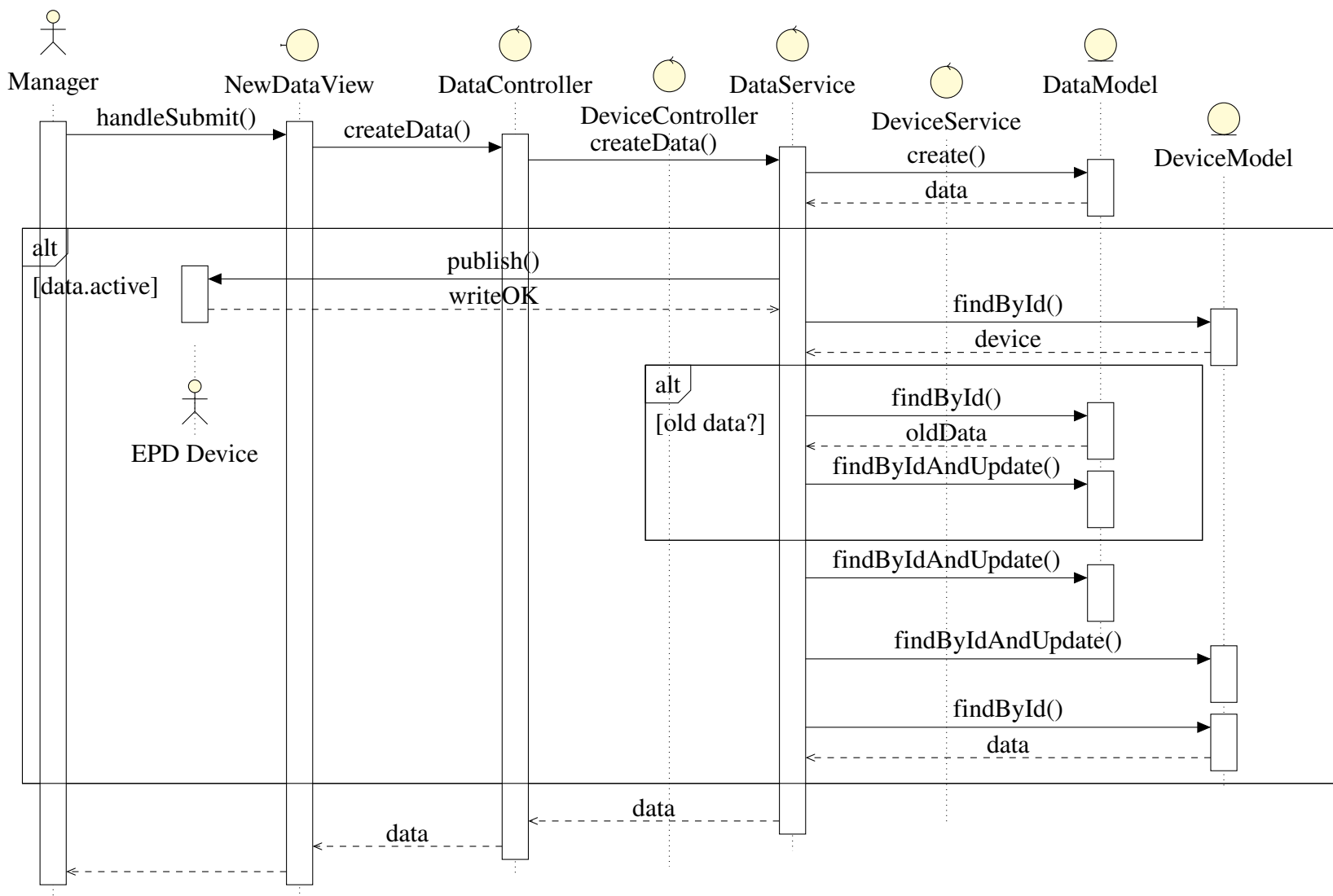


Figure 4.16: cont'd from previous page

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

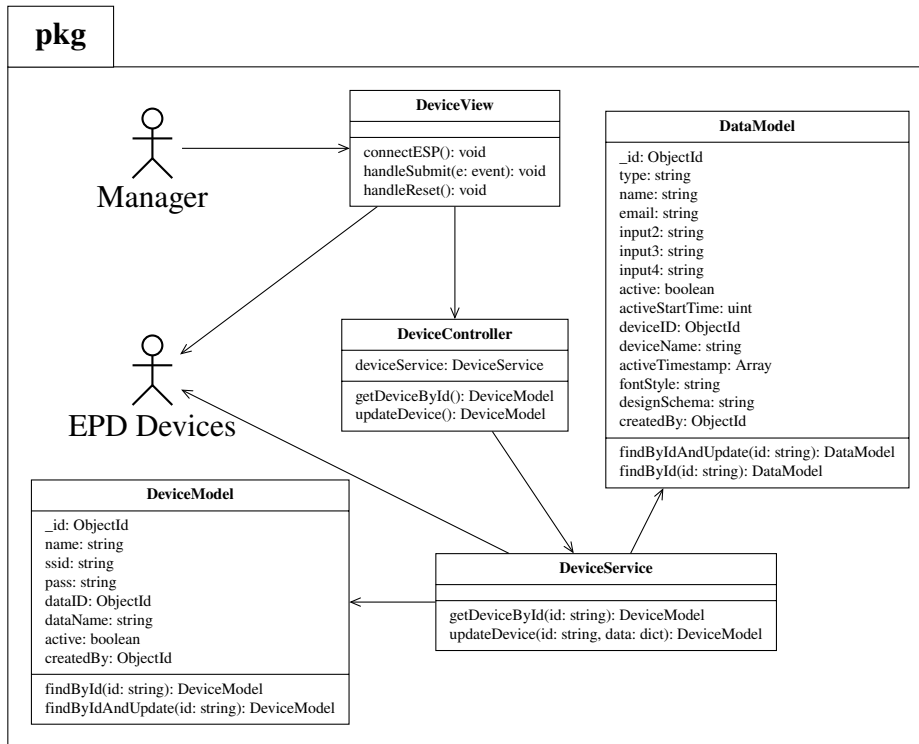
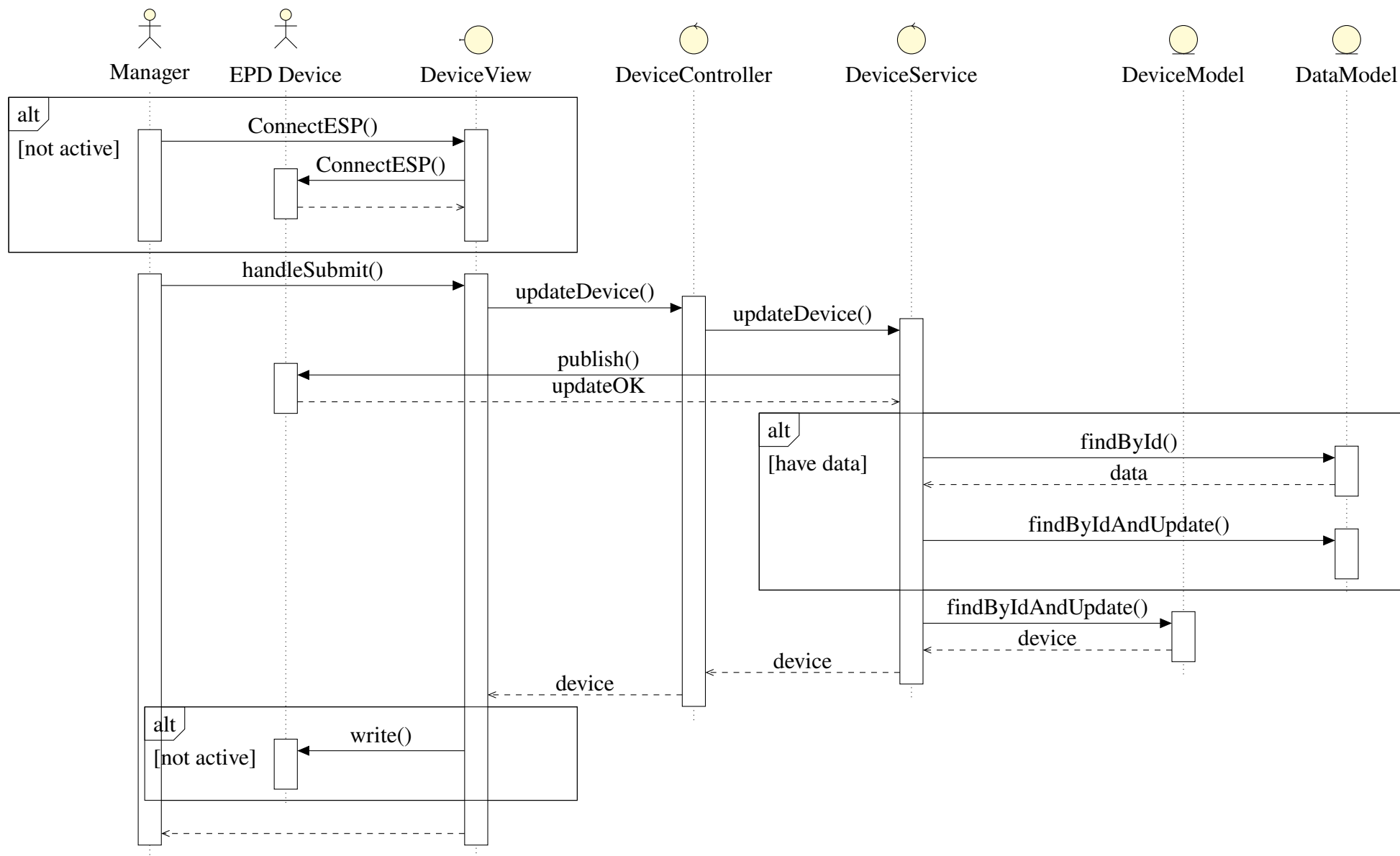


Figure 4.17: Class diagram of use case "Change a device information"

**Figure 4.18:** Sequence diagram of use case "Change a device information"

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

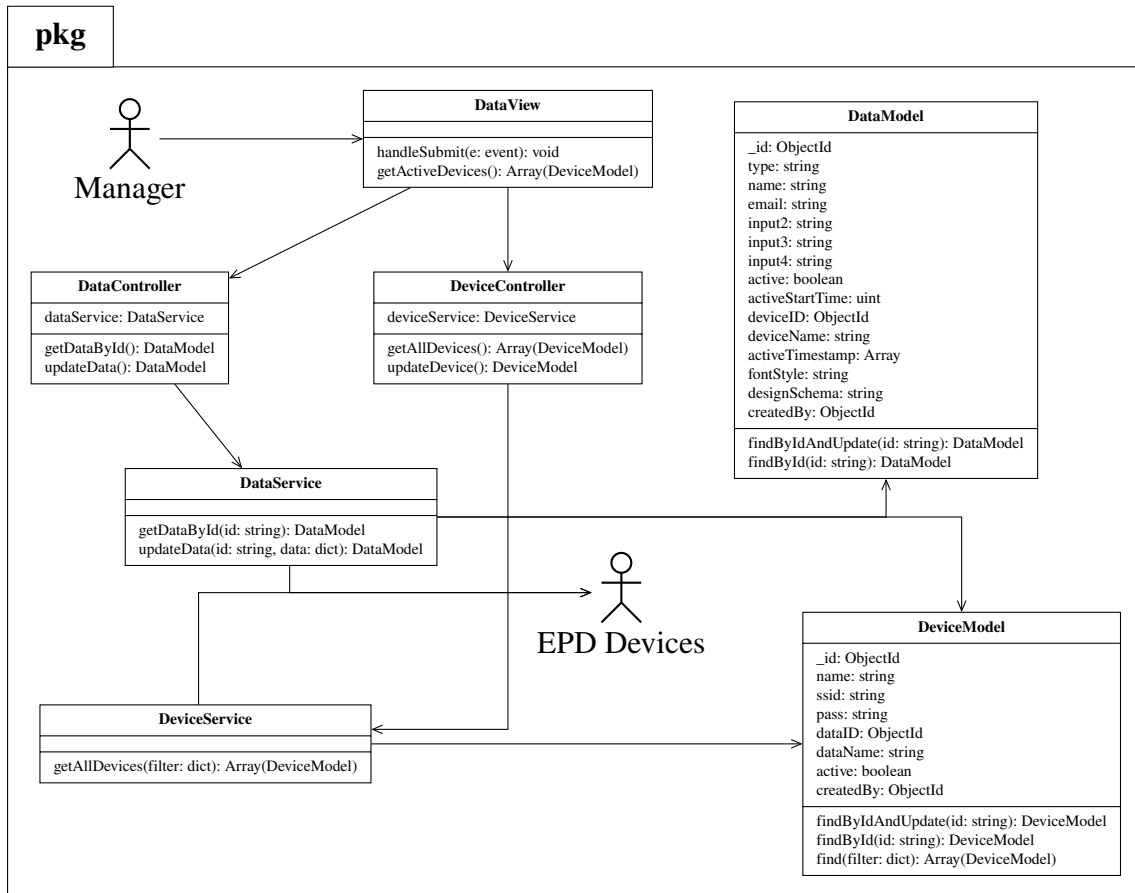


Figure 4.19: Class diagram of use case "Change data information"

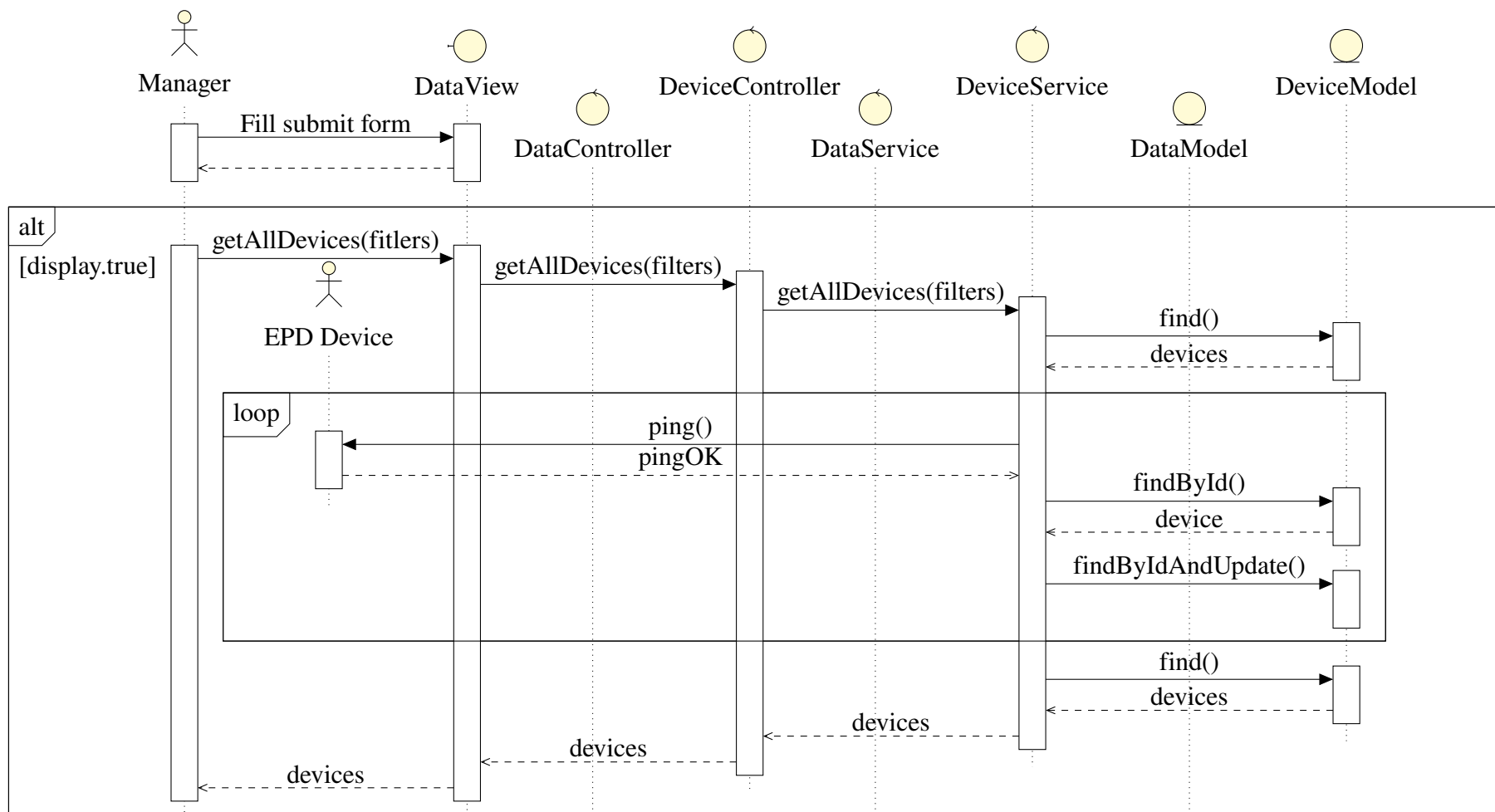


Figure 4.20: Sequence diagram of use case "Change data information" | Cont'd on next page

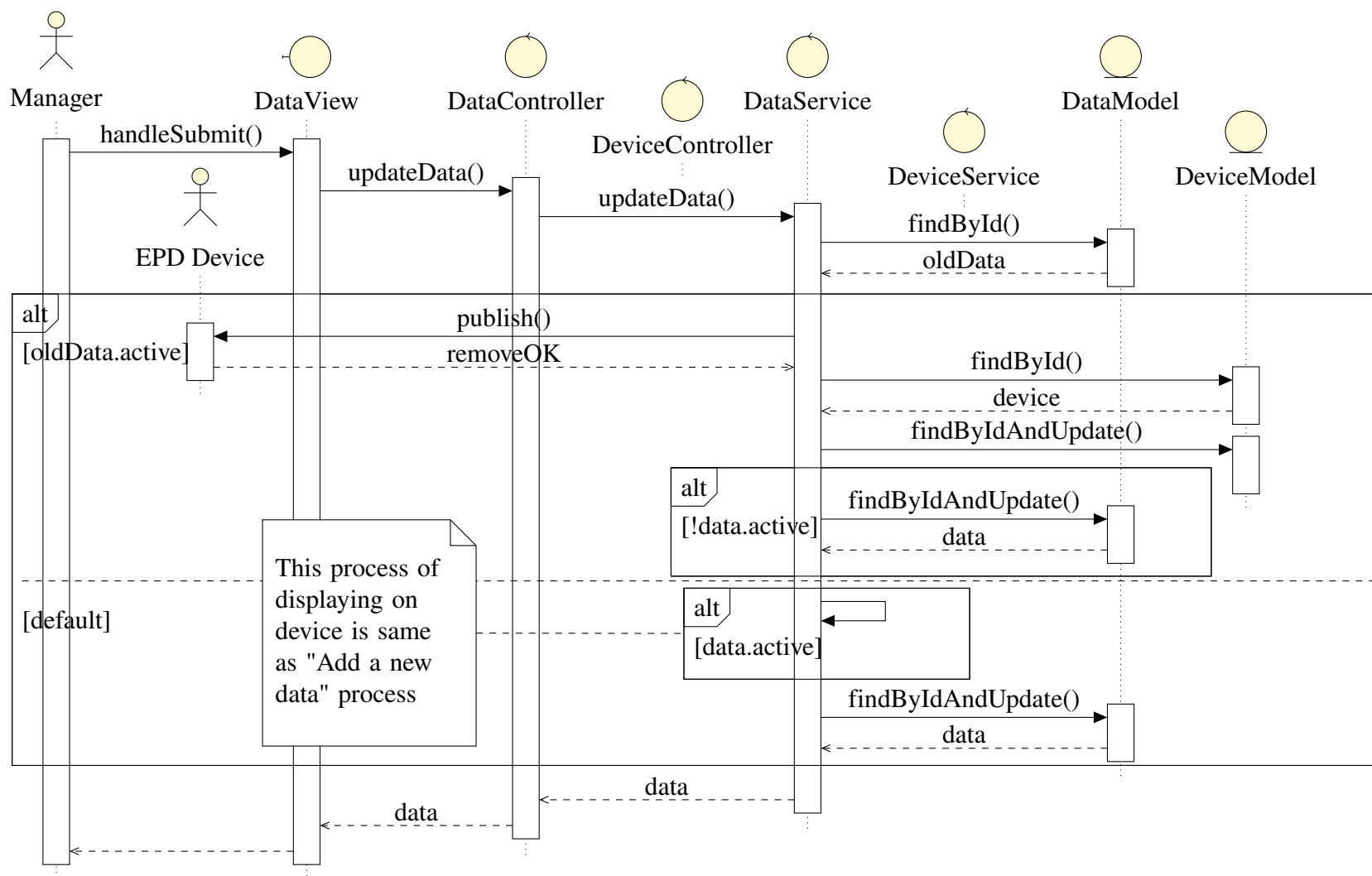


Figure 4.21: cont'd from previous page

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

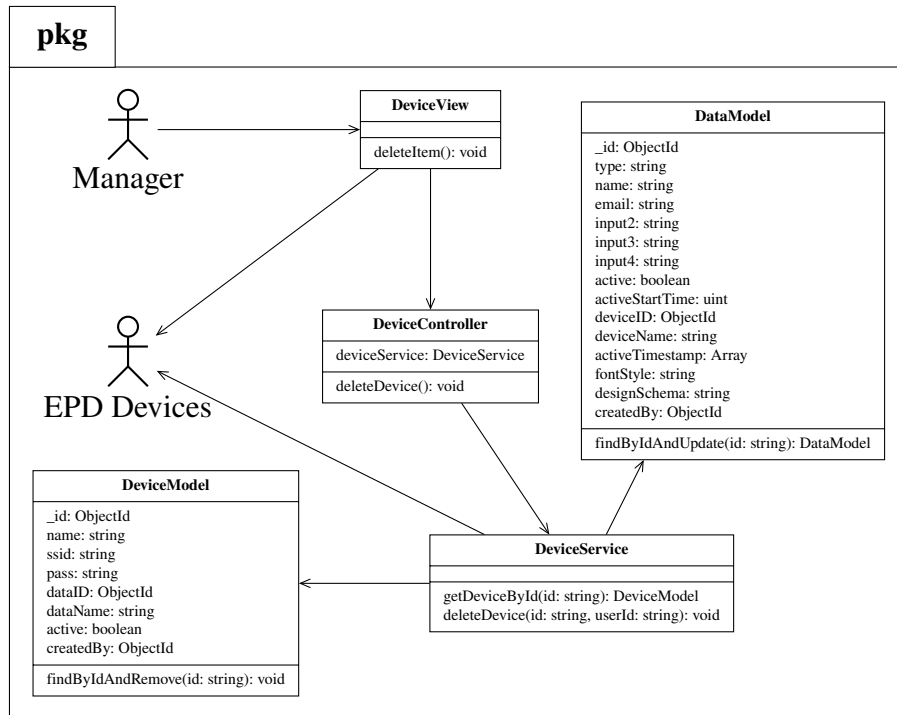


Figure 4.22: Class diagram of use case "Remove a device"

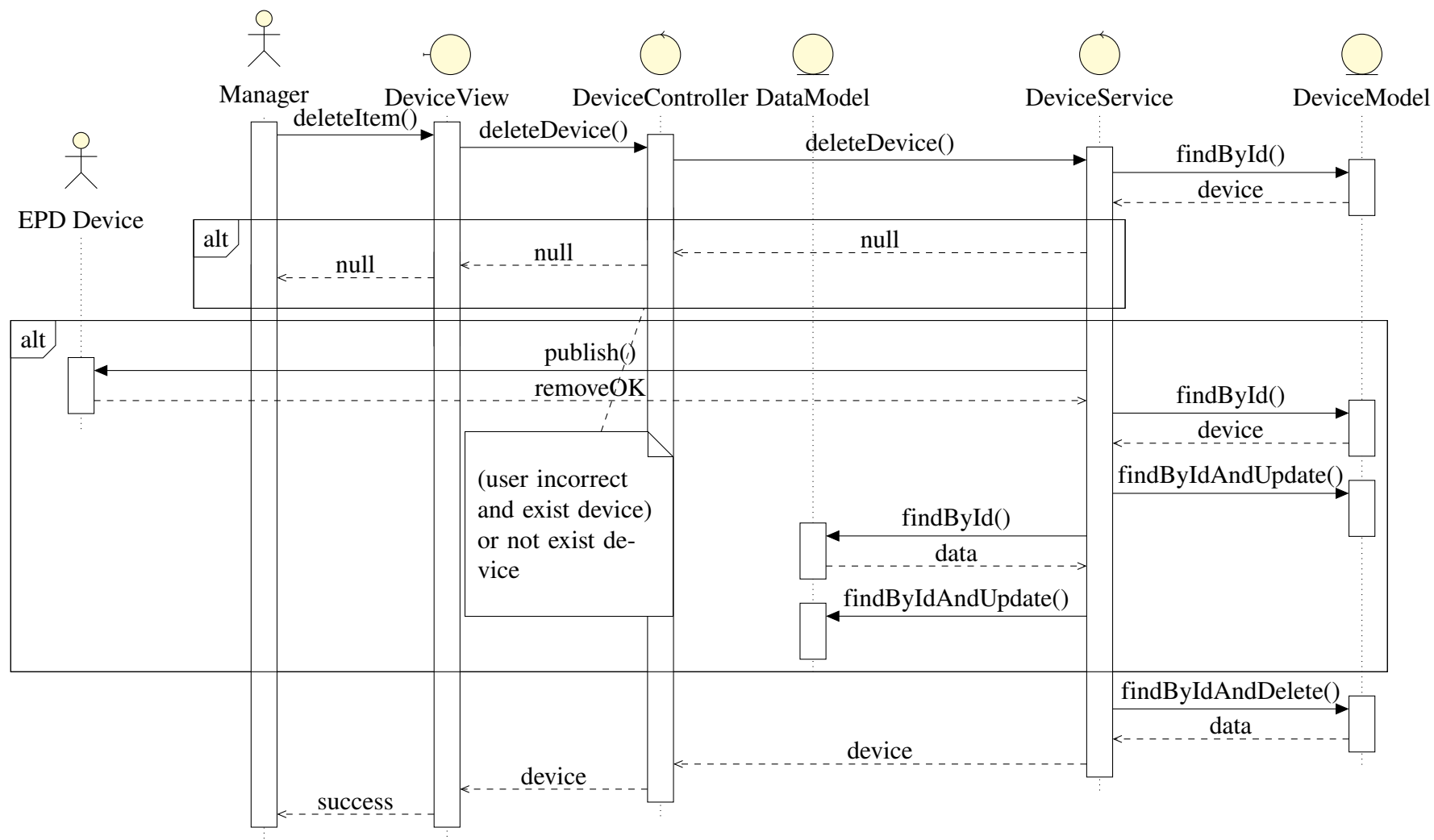


Figure 4.23: Sequence diagram of use case "Remove a device"

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

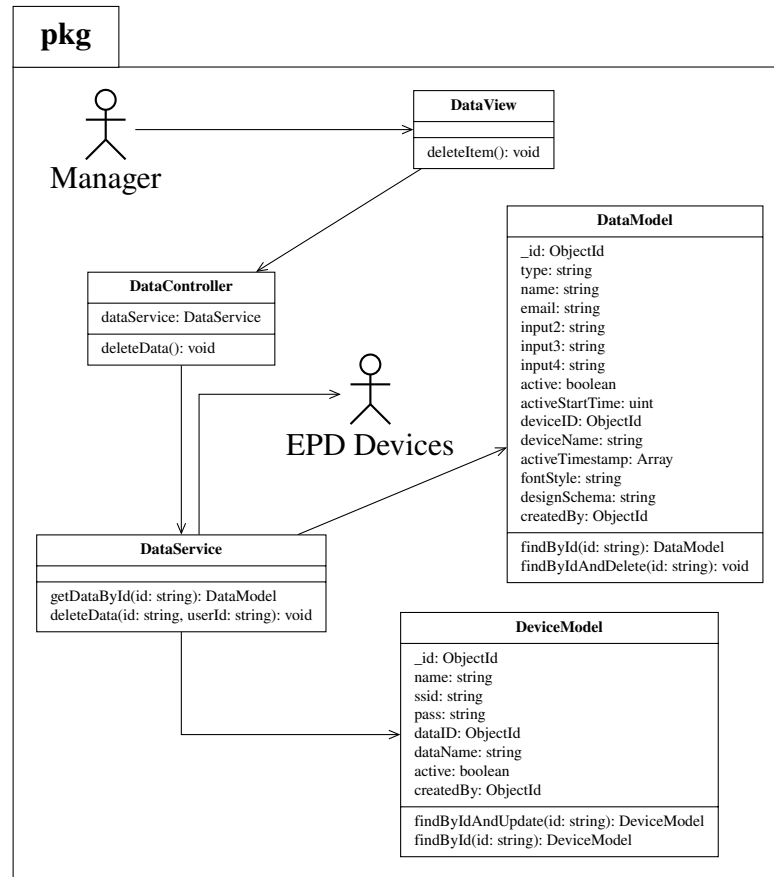


Figure 4.24: Class diagram of use case "Remove a data"

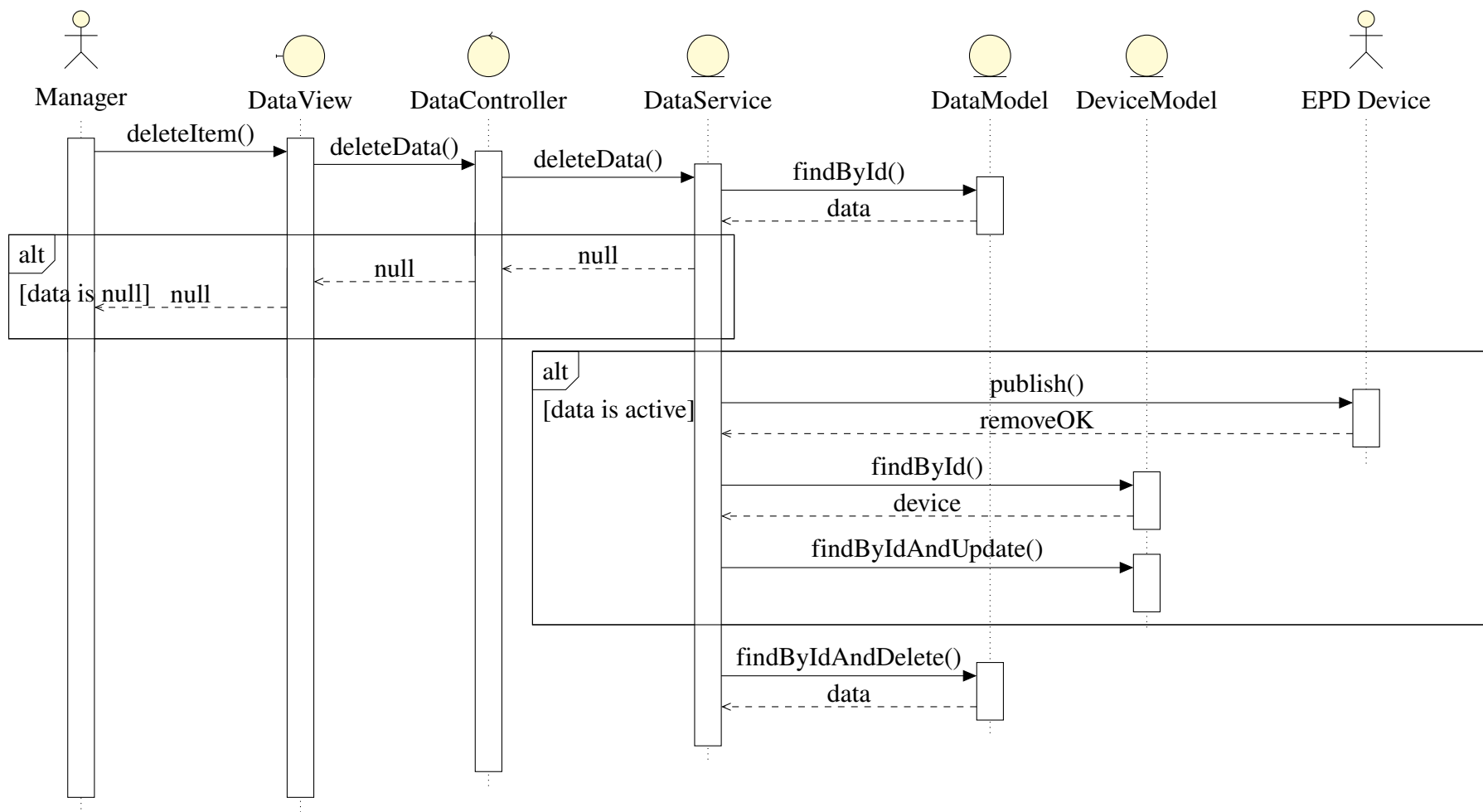


Figure 4.25: Sequence diagram of use case "Remove a data"

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

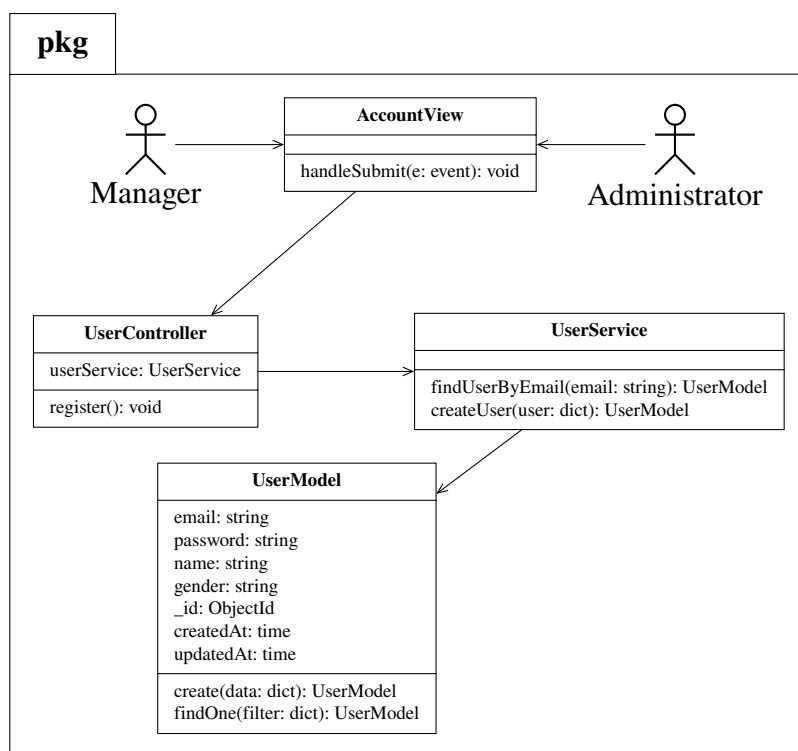


Figure 4.26: Class diagram of use case "Register new account"

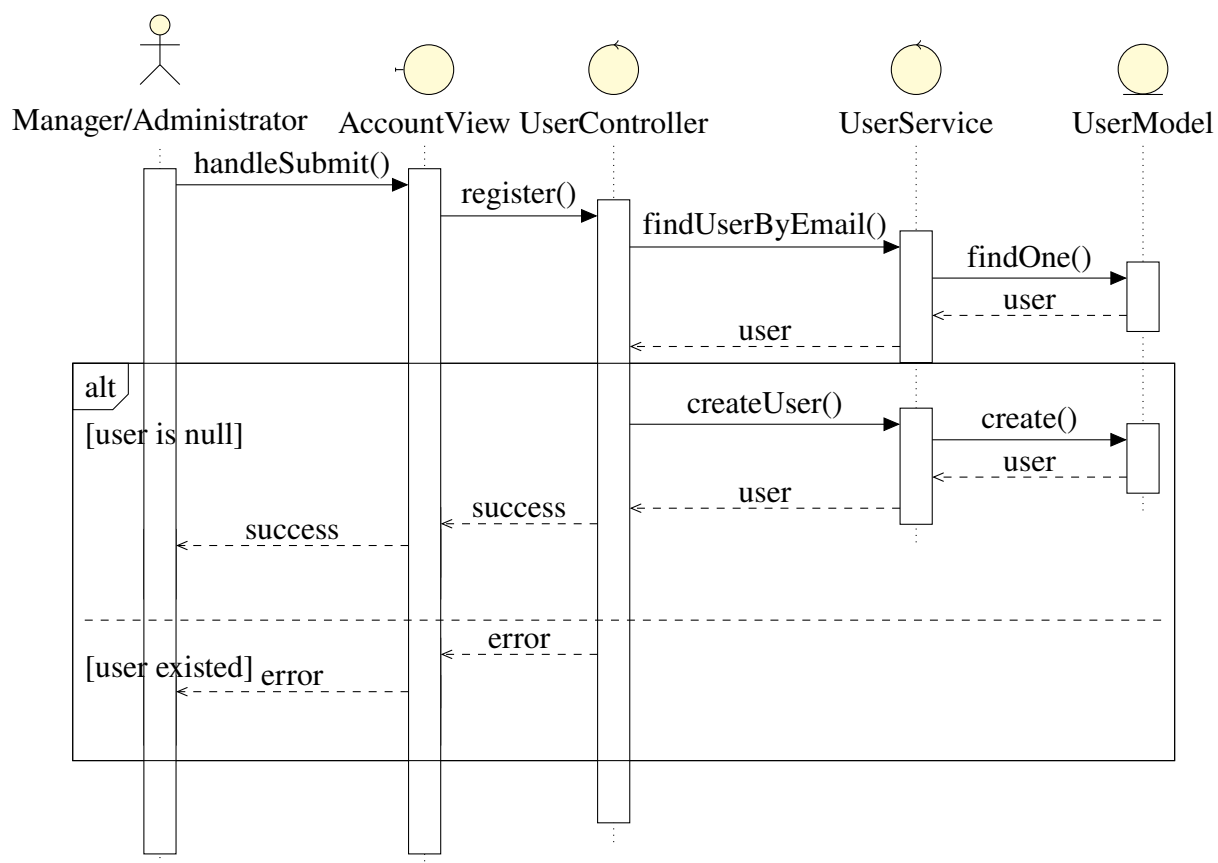
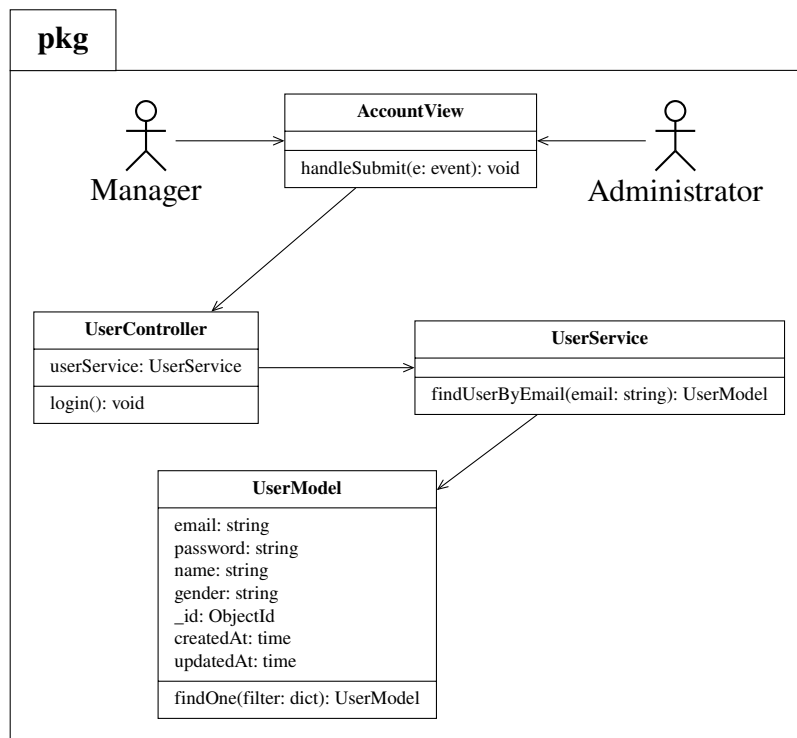
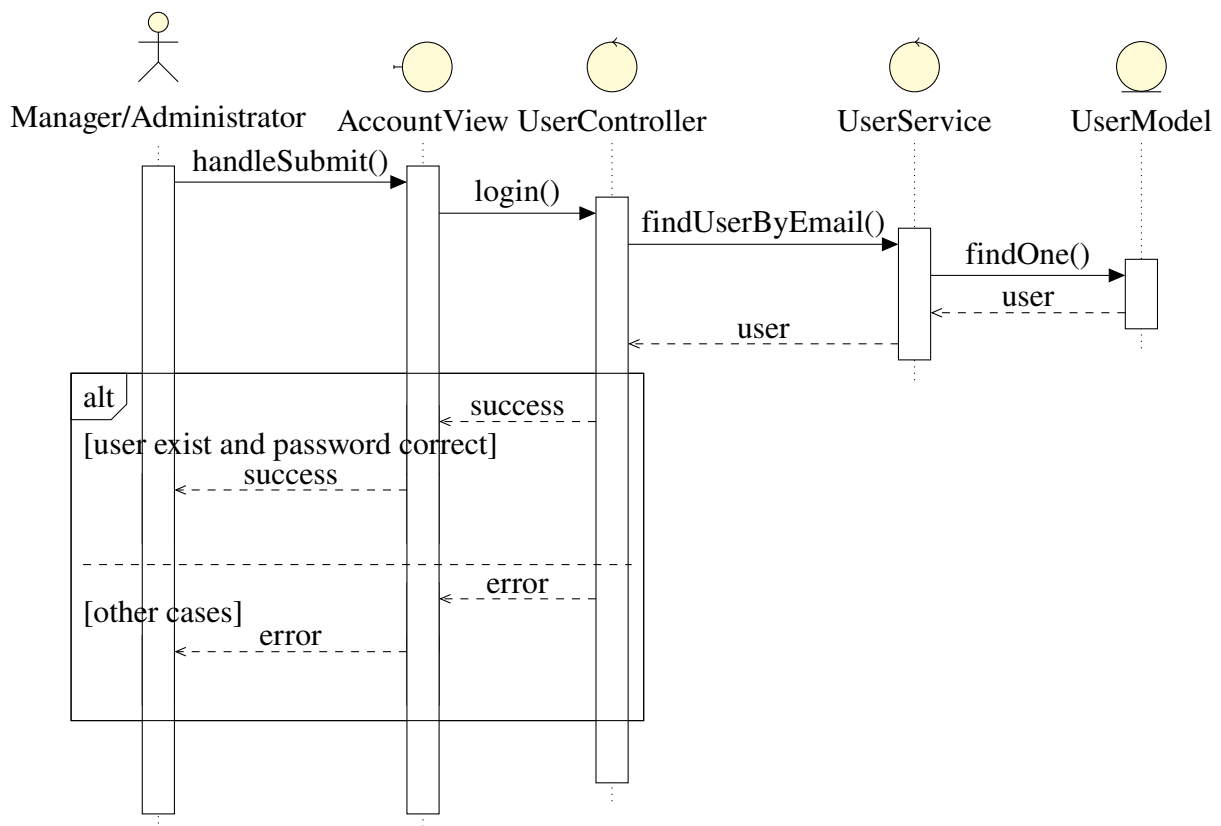


Figure 4.27: Sequence diagram of use case "Register new account"

h, Class and sequence diagram of use case "Sign in"**Figure 4.28:** Class diagram of use case "Sign in"**Figure 4.29:** Sequence diagram of use case "Register new account"

4.2.3 Database design

a, 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
Data	_id	ObjectId	*	Data's ID used in the system
	type	String	*	Data type
	name	String	*	Data name
	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

Table 4.1: Database design

b, Entity-Relationship diagram**Figure 4.30:** Entity-Relationship diagram**4.3 Application Building****4.3.1 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, various 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 4.11 below lists all the tools, systems, and applications that are frequently used during the development process of 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
NextJS	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
ExpressJS	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	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
GitHub	Tools		Source code and project management website	https://github.com

Table 4.2: List of libraries and tools used in the project

A lot of issues and additional tasks arise in the system development process, and a task management tool is required to manage multiple tasks and issues effectively. Included in GitHub, GitHub Project is a perfect tool that matches the requirements while offering users a comprehensive platform for organizing, tracking, and managing issues and tasks in software development projects efficiently.



Figure 4.31: Repository’s Project screen

4.3.2 Achievement

With the help of tools and libraries in the table 4.11 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 runs on the website, so the users don’t need to install any additional applications. Also, the users only 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://mongo.epaper.artsakh.ventures:27017
Back-end server	https://epaper.artsakh.ventures/api
MQTT Broker	mqtt://mqtt.epaper.artsakh.ventures:8883
MQTT Management UI	https://epaper.artsakh.ventures/rabbitmq
OpenAPI Swagger	https://epaper.artsakh.ventures/api/swagger

Table 4.3: Project service URLs

The EPD device consists of an ESP32-C3 Supermini development board with a Waveshare 2.9-inch e-paper display panel and is packaged inside a custom 3D printed case ¹. The battery powering the system is a 250mAh Lithium-ion battery and is rechargeable via the USB-C port of ESP32-C3.

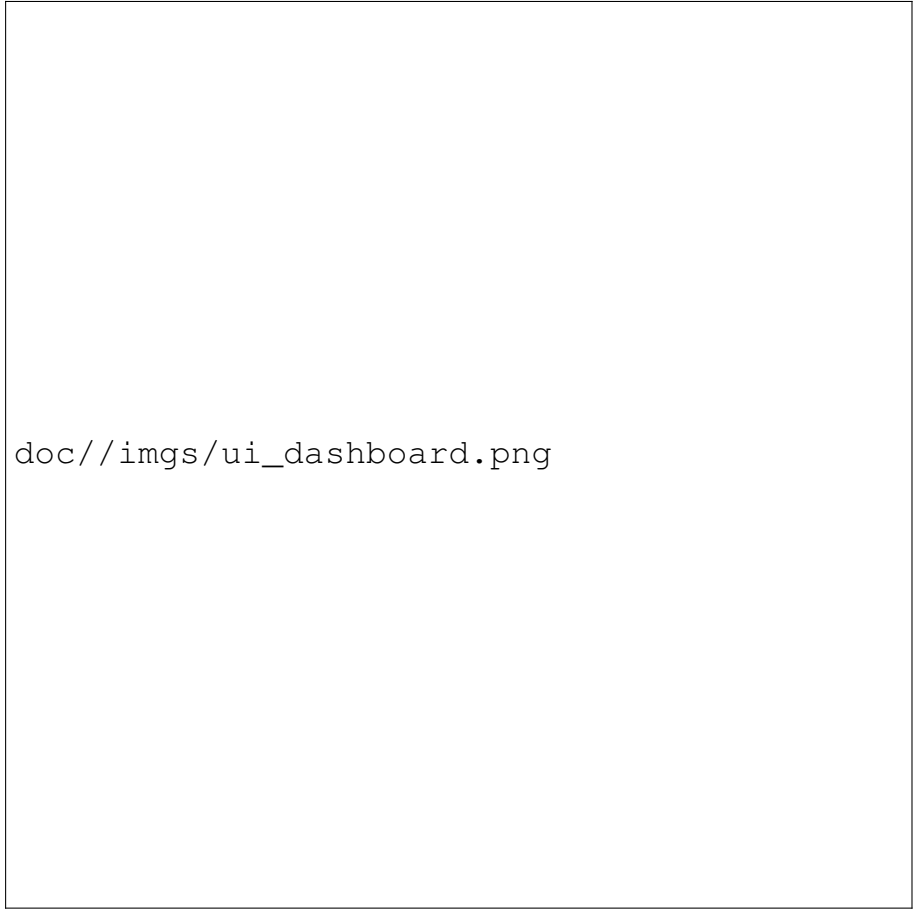
4.3.3 Illustration of main functions

To manage EPD Devices and data, users have to log in with credentials as illustrated in the image 4.32. After logging in, the user can choose to create data/device or go to the dashboard to manage them by choosing the pages from the left sidebar, which is shown in figure 4.33.

¹1



Figure 4.32: Login screen



doc//imgs/ui_dashboard.png

Figure 4.33: Dashboard Screen



doc//imgs/ui_new-data-type.png

Figure 4.34: "Choose data type" screen in "New Data"

When creating data, 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. Figure 4.34 below illustrates the process of creating new data at the data type step, which requires users to choose the type of data before providing more detail and going to the device-choosing step shown in figure 4.35. The user can choose the active device and their preferred display configurations and see the pre-rendered display of the data.



Figure 4.35: "Choose device" screen in "New Data"

When registering a new device, users need to plug the EPD device into the USB port of the PC they are using. Then, users can choose the device from the connected devices list and provide its name and Wi-Fi credentials before submitting to the system.



Figure 4.36: "New Device" page

In the device's dashboard, the users can see information together with the status of each device and also can choose to view more detail, edit, or delete devices via the buttons on the right side. Figure ?? below shows the detailed modal of an active device displaying the data. The users also can check the serial output of the device via Serial Port and even pop it out into the PiP component by choosing the "Debug on-the-go" button, which enables them to test interacting with the device right on the website while still monitoring the device's log (figure 4.37). This feature is very useful in case some problems occur on the device and users don't have the proper instruments to debug further.



doc//imgs/ui_data-dashboard.png

Figure 4.37: Data dashboard, with a PiP showing device log

The EPD Device connects to the MQTT Broker and subscribes to its ID as the topic. The picture ?? on the left side below indicates the device display when no data is stored, and the right side shows the device display when data is stored and displayed in the device.

Users can also put the EPD device in debugging mode to test the devices. Currently, to do so, users need to hold the "Boot" button on the ESP32-C3 for 5 seconds, and it will reboot to the debugging mode and test the functionalities of the screen (figure ??). This access method is not user-friendly and is only temporary in the development of the project. The improvement to this, and also other improvements of the system in the future, are discussed in chapter 6.

4.4 Testing and evaluation

4.4.1 Evaluation

The project utilizes Lighthouse - an open-source tool by Google used for analyzing, measuring, improving quality, and optimizing the Management UI's performance. Lighthouse evaluates a web page's performance based on five criteria:

performance, accessibility, best practices, search engine optimization (SEO), and Progressive Web App (PWA) compatibility. This tool not only provides a comprehensive analysis of web pages but also recommends ways to enhance the quality of the website. In this project, Progressive Web App (PWA) is not accounted for because the website only focuses on mainly desktop users.

The results of the website quality assessment are shown in figure 4.38, and details are listed in table 4.4.



Figure 4.38: Performance reports by Lighthouse

Creticia	Description	Result
Performance	The overall loading and responsive time of the website	77
Accessibility	Evaluating the accessibility of the website for wide-range users	95
Best Practices	Check the security of the website and other safety aspects	100
SEO	Checking if the website is following basic search engine optimization advice	90

Table 4.4: Lighthouse performance score of four criteria

4.4.2 Testing

The project uses a black-box testing technique to conduct the tests on the overall system. This section will discuss the back-box testing techniques used in the main features: adding new data/device and removing a device/data. Test cases for modifying device and data functions are the same with adding new data and device functions, as the functions share mostly the same workflow.

a, "Adding new data" function testing

Required inputs of new data form and device-choosing section in this process are described in table 4.5 below. Table 4.6 lists the conducted test cases and their results.

No.	Name	Input type and format	Required
1	Type	Text, Options	✓
2	Name	String, Input	✓
3	Email	String, Input	×
4	Price	String, Input	×
5	Department	String, Input	×
6	Category	String, Input	×
7	Student ID	String, Input	×
8	Class	String, Input	×
9	Employee ID	String, Input	×
10	Address	String, Input	×
11	Purpose	String, Input	×
12	Manager	String, Input	×
13	Status	String, Input	×
14	Active	Boolean, Check box	✓
15	Font Style	String, Dropdown list	Yes, if active is true
16	Theme	String, Dropdown list	Yes, if active is true
17	Device	String, Dropdown list	Yes, if active is true

Table 4.5: Input details of "Adding new data" process

No.	Test cases	Conditions	Expected Behaviour	Pass
1	User provides information correctly	All inputs are correctly filled	Save new data successfully, and data is correctly displayed on EPD device if Active is true	✓
2	Checking the required information	Some required inputs are empty	Notify users about the error	✓
3	Display on device	Device and display options are properly provided	The data is displayed on EPD screen correctly, and same as the rendered	✓
4	No active devices	All the devices are inactive	Notify users to check the devices	✓
5	Old data handle	Data is displayed on EPD device containing old data	Old data's device information removed, device and data details updated accordingly	✓
6	Handle device error	Device encounters error when processing data	Receive the error and notify users	✓

Table 4.6: Testing results of "Adding new data" process

b, "Register new device" function testing

Required inputs the user needs to provide to register a new device in this process are described in table 4.7.

No.	Name	Input type and format	Required
1	Name	String, Input	✓
2	SSID	String, Input	✓
3	Password	String, Input	✓

Table 4.7: Input details of "Creating new device" process

The testing cases and results are described in the table 4.8 below.

No.	Test cases	Conditions	Expected Behaviour	Pass
1	User provides information correctly	All inputs are correctly filled	Device stored and connects to the broker successfully	✓
2	Checking the required information	Some required inputs are empty	Notify users about the error	✓
3	Device connect error	Device encounters error when connect to the PC	Receive the error and notify users	✓

Table 4.8: Testing results of "Creating new device" process

c, "Removing device" function testing

The testing cases and results are described in the table ?? below.

No.	Test cases	Conditions	Expected Behaviour	Pass
1	Removing device	User confirms to remove device	Device is unregistered from the system, remove all data on device and data's device information is removed	✓
2	Handle device error	Device encounters error when processing request	Unregister device from system and notify users about the error	✓

Table 4.9: Testing results of "Removing device" process

d, "Removing data" function testing

The testing cases and results are described in the table ?? below.

No.	Test cases	Conditions	Expected Behaviour	Pass
1	Removing data	User confirms to remove data	Data is removed from the system, removed from device if displayed and device's data information is removed	✓
2	Handle device error	Device encounters error when processing request	Remove data from system and notify users about the error	✓

Table 4.10: Testing results of "Removing device" process

4.5 Deployment

4.5.1 Continuous Integration/Continuous Delivery (CI/CD)

All of the project's source code is hosted and managed on GitHub and uses Continuous Integration and Continuous Delivery (CI/CD) to automate the deployment. When a new commit on the main branch is pushed, a deploy workflow is triggered to connect to the web server and update the front-end and back-end code. Details and the workflow log can be monitored in the Action section of Github's repository website (Figure 4.39).



Figure 4.39: A successful GitHub deploy workflow

4.5.2 Servers

The system uses two Hetzner dedicated servers to host the Rabbit MQTT broker and web services. Details of the servers and the tools used are shown in the table 4.11 below.

	API Server (web-server)	MQTT Broker (laboratory)
IP address	65.108.79.164	95.217.121.243
Operating System	Ubuntu 22.04.3 LTS x86_64	Ubuntu 22.04.3 LTS x86_64
CPU	AMD Ryzen 5 3600 @ 3.600GHz	AMD Ryzen 7 3700X @ 3.600GHz
RAM	64GB	64GB
Storage	1TB	2TB
Purpose	Management UI, API Server and proxy services	MQTT broker
Services used	NginX, MongoDB, NextJS, NPM	RabbitMQ

Table 4.11: Detail of two Hetzner's dedicated servers

Both servers use RAID 1, creating a backup drive to store an exact copy of data on the main drive simultaneously. This clone mechanism ensures real-time data redundancy and high availability, thereby significantly enhancing data security and reducing the risk of data loss due to hardware failures, although the usable storage of the server is only half. It also offers improved read performance as the data can be read from multiple drives simultaneously.

Services running in each server are configured as a system daemon (`systemctl`), which is placed in `/etc/systemd/system/` and `/lib/systemd/system/`. Details of the services are listed in table 5.1 below.

Services	Name	Location	Server
<code>epaper.service</code>	Epaper Front-end service	<code>/etc/systemd/system/</code>	65.108.79.164
<code>epaper-backend.service</code>	Epaper Back-end service	<code>/etc/systemd/system/</code>	65.108.79.164
<code>rabbitmq-server.service</code>	RabbitMQ broker service	<code>/lib/systemd/system/</code>	95.217.121.243
<code>mongod.service</code>	MongoDB Database service	<code>/lib/systemd/system/</code>	65.108.79.164
<code>nginx.service</code>	NginX daemon service	<code>/lib/systemd/system/</code>	65.108.79.164

Table 4.12: System services

4.5.3 EPD Devices

The code is transferred and stored in ESP32-C3 Supermini flash memory using the PlatformIO plugin integrated into Visual Studio Code. The code takes about 1.5MB in total memory, most of which is stored in the non-volatile flash memory,

meaning the code still remains on the device even after it is powered off. The image of the ESP32-C3 Supermini and its ESP32-C3 microcontroller unit specs are listed in figure 4.40 and table 5.1 below.



Figure 4.40: Back and front image of ESP32-C3 Supermini

Attributes	ESP32-C3
Core	32-bit RISC-V single-core (SoC)
Maximum Clock Frequency	160MHz
RAM size	400KB SRAM
Interfaces	GPIO, I2C, I2S, SPI, UART
Connectivities	BLE 5.0, Wi-Fi (2.4GHz)
Maximum transmit rate	150Mbps with Wi-Fi (2.4GHz), 2Mbps with BLE 5.0
Flash memory size	4MB
ROM	384KB
Supply Voltage	3V - 3.6V

Table 4.13: ESP32-C3's specs

The e-paper display panel is connected and communicates to the board via SPI interfaces. Details of connected pins and their description are listed in table 4.14 below.

Pin	ESP32-C3 Supermini	Description
VCC	3V3	Power input (3.3V)
GND	GND	Ground
DIN	GPIO6	SPI MOSI pin, data input
SCLK	GPIO4	SPI CLK pin, clock signal input
CS	GPIO7	Chip selection, low active
DC	GPIO1	Data/command, low for commands, high for data
RST	GPIO2	Reset, low active
BUSY	GPIO3	Busy status output pin (means busy)

Table 4.14: SPI connection schematic

The device runs on battery, which can last for 3 - 4 hours before needing to recharge. In the scope of this project, this is still reasonable and acceptable as the product is still in the prototype stage and is not optimized for higher endurance in a real-life environment.

CHAPTER 5. SOLUTION AND CONTRIBUTION

The preceding chapters of this document have meticulously delineated the architectural framework and the technological foundations employed in the system's development process. However, without distinct solutions and initiatives, the system might have remained indistinguishable within the extensive landscape of business solutions and unsuitable in other environments like Vietnam. This chapter aims to shed light on how the issues are identified in the development progress of the project and the personalized strategies and innovations that contributed to the system's uniqueness and enhanced its efficacy and relevance in a dynamic business environment.

5.1 Display custom text

In the development process, the system encounters an issue when the user wants to display Vietnamese language texts and other special characters not in the ASCII character map. Vietnamese, with its complex diacritical marks and distinct character set, demands a high degree of precision in rendering, a task that is particularly challenging given the inherent limitations and operational mechanics of e-paper displays. This section will focus on how normal text is stored and processed in constrained environments like ESP32 and the problem of performance over resource usage.

5.1.1 UTF-8 vs. UTF-16

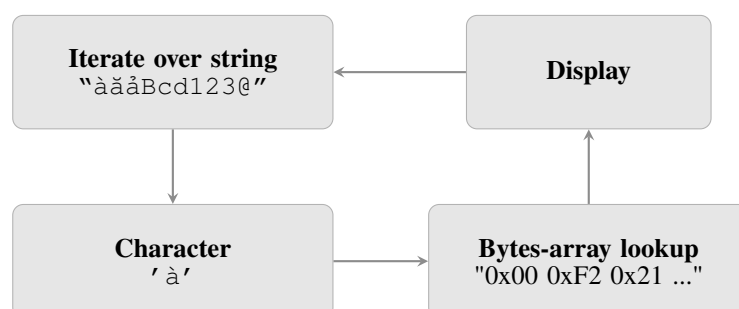
UTF-8 and UTF-16 are both encoding formats used for representing text in computers, each with its unique handling of character sets, including special characters like those in the Vietnamese language. Unlike ASCII, where each character uses one byte to store, UTF-8 uses one to four bytes per character, making it highly capable of 1112064 valid character codes while still being backward-compatible with ASCII's old code. On the other hand, UTF-16 uses fixed-length two or four bytes per character, balancing the space efficiency of UTF-8 and the simplicity of fixed-length encoding. In the scope of this project, both encoding formats are compatible with processing Vietnamese characters to display on screen, each with its unique strengths and weaknesses. UTF-8, while enhancing the efficiency in memory and storage, still has drawbacks when parsing variable-length characters. The ESP32 must decode UTF-8 encoded strings into Unicode code points and then translate these into bitmaps. This process can lead to processing overhead and is more CPU-intensive compared to dealing with a fixed-length encoding like UTF-16.

5.1.2 Two ways of process and display text

To be displayed properly on the screen, the text has to be converted to a bitmap image, stored in the form of an array of bytes. This process can be achieved in two ways: converting from an image of characters (Figure 5.1) or directly from the font file to corresponding byte arrays. The main goal of both two conversion ways is to create a mapping from every character to the corresponding bytes array, which is a step in the display process illustrated in Figure 5.1.2. Due to the specific nature of frequently displaying Vietnamese characters, the project uses the second option, with the help of the TheDotFactory tool, to convert all the Vietnamese and Latin characters mapped to the byte arrays.



Figure 5.1: UI of TheDotFactory tool



In the text-processing task, looking for the bytes array of each character is the most resource-consuming task, especially in an extensive character set. In resource-limited systems like ESP32-C3 Supermini in the project, it is crucial to balance between saving as many resources as possible and processing data in a reasonable time. However, the way data is organized in the exported file from TheDotFactory does not effectively solve the problem of processing performance and resource conservation when having to store and process 228 different characters, including Latin letters, accented Vietnamese, special characters, and numbers. This issue, combined with the encoding formats discussed above, leads to two ways of storing and mapping from the character to its corresponding bytes array, with a big difference in performance and resource usage between the two.

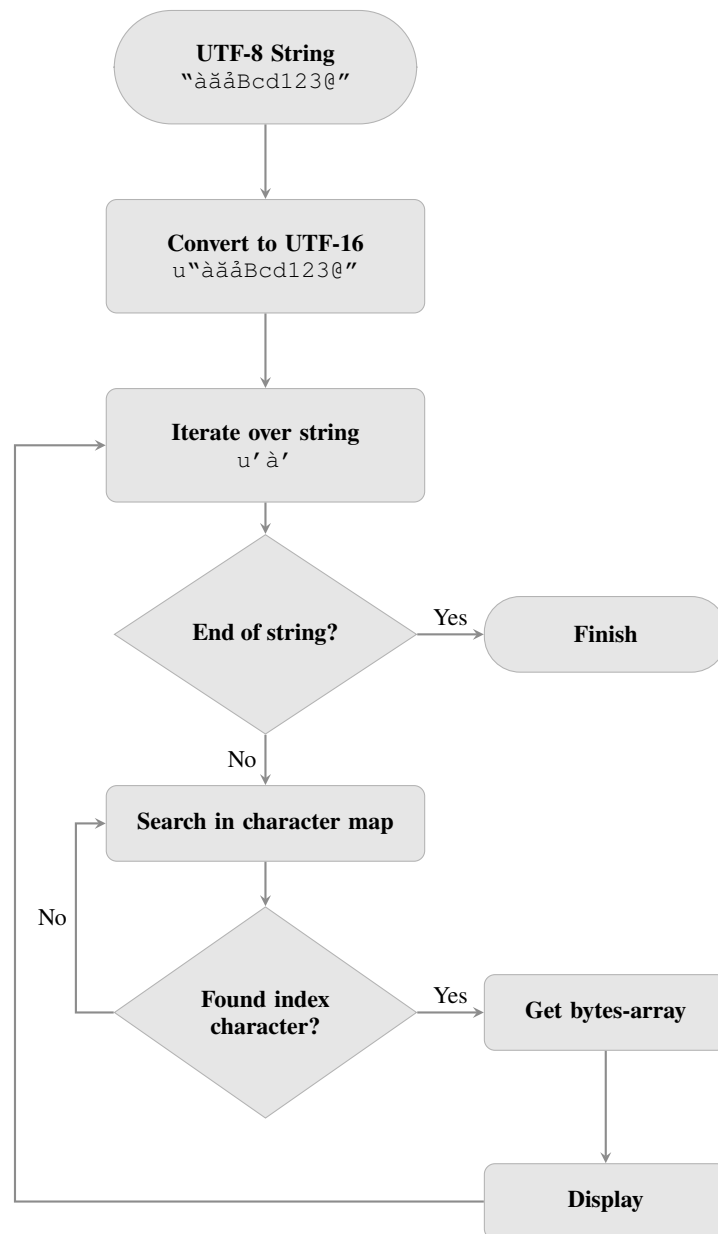
5.1.3 First Solution

The first of the two solutions mentioned above is a practical and straightforward approach focusing on the flexibility and scalability of the character set. The mapped character set is stored in an array of multiple character map definitions, where each character is associated with a structure that contains its display properties (width) and the corresponding byte array for rendering (code structure below), making the retrieval of byte data for each character straightforward and intuitive. This method is particularly suited for systems where a direct and easy way to access the display data for each character is highly recommended, especially for a fixed and known set of characters.

```
typedef struct
{
    char16_t * chr;                                // index character, utf-16
    uint8_t width;                                  // dynamic width
    const char matrix[MAX_HEIGHT_FONT*MAX_WIDTH_FONT/8]; // bytes-array (41*32/8)
} FT_IDX;

typedef struct
{
    const FT_IDX *table;
    uint16_t size;
    uint8_t Height;
} cFONT; // custom Font
```

The index character `chr` defined in the struct above uses UTF-16 as the encoding format. Although it is not as memory-efficient as UTF-8 encoding format as discussed in section 5.1.1, UTF-16 encoding brings ease of character comparisons while UTF-8 requires checking multiple bytes for non-ASCII characters (above 126), which appear the most frequently in the Vietnamese language. The flow chart below shows the details of the process displaying a Vietnamese string on the screen.



In terms of memory, this method stores a character set and occupies around 169 bytes per character map for 228 characters. For all 228 characters in the set, this method occupies a total of $169 \times 228 = 38,532$ bytes or 37 kilobytes. This is also affected by the compiler padding and alignment, so this calculation provides an approximated number based on the struct layout of the character set. However, most bytes-arrays of each character in this project only take up around 40 bytes on average, which, in fact, only costs $228 \times 40 = 9,120$ bytes, or 9 kilobytes of actual data, meaning there are still many redundant space unusable with this method. Also, the data of the processing task, which includes the converted UTF-16 string and other variables in the process, is not accounted for, making an even higher total memory occupation. Currently, this is not a big problem with ESP32-c3supermini, but when the system gets more complicated or is implemented in smaller microcontrollers, another approach is highly recommended to balance performance and resources.

5.1.4 Second Solution

The second solution uses the segment method combined with the lookup table, optimizing resource and time efficiency even more, especially in a constrained environment. In this method, each character is not directly mapped to its bytes-array but to the start index and the length of its array, and the array containing these character maps is a lookup table. In this way, the bytes array will be stored in one extensive single array, which reduces memory overhead significantly compared to the first straightforward method. Also, having a map that links each character to its corresponding byte range in the array allows for fast lookup and retrieval, making the rendering process more efficient, especially when dealing with larger sets of characters or more frequent character lookups, as would be the case when rendering text from strings.

However, finding the index character in the lookup table is also a time- and resource-consuming task as it usually iterates through the array to get the data corresponding to the character. Therefore, the lookup table is divided into three segments based on the use frequency, and each segment contains a group of characters that share mostly the same frequency range. This will boost the lookup time as it does not require iterating over a single array to find needed information. However, because Vietnamese characters in the UTF-8 table are distributed differently from the use frequency and the limited time of the project, the segments are still divided based on the UTF-8 table, with the first segment containing characters of ASCII table, the second segment containing the first half of Vietnamese character set, and the third segment containing the second half. To speed up the lookup process in each segment, the index character is converted to the corresponding Unicode point, and binary search is also used, which is implemented in `binarySearchInSegment()` function in the code structure shown below.

```
typedef struct
{
    int chr; // index character, Unicode point
    uint8_t width; // dynamic width, used to determine the byte array's length
    int index; // start index at byte map table
} FT_MAP;

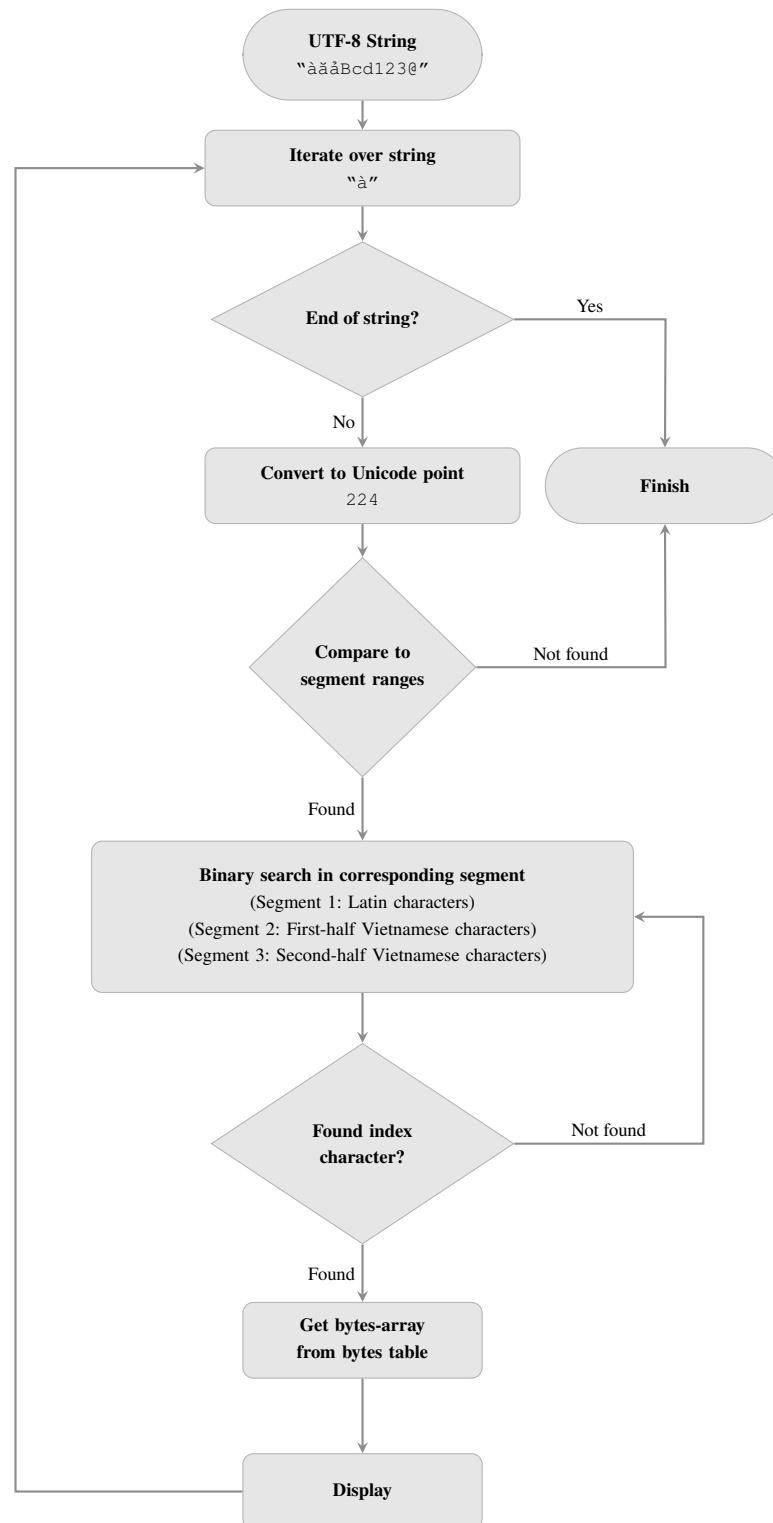
typedef struct
{
    const FT_MAP *ASCII_table; // segmentSize = 95 characters from 32 to 126
    const FT_MAP *vn_table; // segmentSize =
    const FT_MAP *VN_table; // segmentSize =
    uint8_t Height; //character's height, in pixels

    const FT_MAP * binarySearchInSegment (
        int unicodePoint,
        const FT_MAP* segment,
        size_t segmentSize
    );

    const char *table;
```

```
} cFONT_SEGMENT;    // custom Font with Segment Management
```

The flow chart below shows the details of the process displaying a Vietnamese string on the screen in the second method.



In this solution, Unicode Point is used instead of the original UTF-8 encoding format, which performs better in character comparisons than UTF-8 and UTF-16. However, a function to convert from UTF-8 characters to Unicode points is still needed, and the lookup array in each segment is sorted in ascending order based

on the Unicode points of its character elements for the binary search function. In terms of memory, each character map takes up around only 9 bytes, and the single bytes array occupies around 9000 bytes on average, making a total of $9 \times 228 + 9000 = 11,052$ bytes, or 11 kilobytes, of the character set. Also, this calculation is estimated on the code structure, not taking the data during the process into account, and actual memory usage might slightly differ due to factors like alignment and padding specific to the compiler and the architecture.

About performance, ...

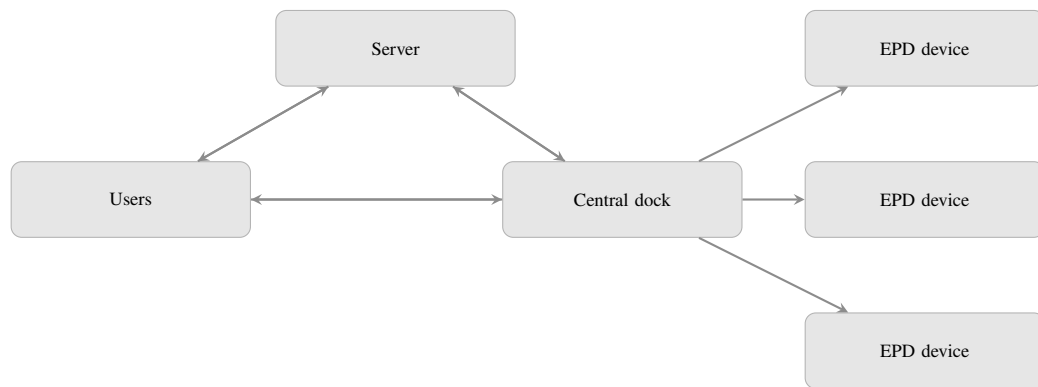
Overall, the logic for accessing and using the data in this second solution becomes slightly more complex, requiring correctly handling the mapping and indexing into the byte array. However, this method leverages the advantages of both segmentation and binary search, effectively reducing the search space and improving lookup times, making it well-suited for handling a large set of characters efficiently on systems with limited resources like the ESP32.

5.2 Solutions for managing devices

Currently, this project handles and communicates to the EPD devices via MQTT Broker, which requires a 2.4GHz Wi-Fi connection from the devices. This method is suitable in most cases when Wi-Fi infrastructures are widely available and easy to set up. There are also other ways of managing EPD devices and each method has its advantages and drawbacks. This section will comprehensively discuss each proposed solution to clearly delineate the differences among them, providing a deeper understanding of their respective strengths and limitations. Additionally, the section will delve into the rationale behind selecting the Wi-Fi solution, offering insights into its suitability and effectiveness for the current project and the future plan to support a wide range of flexible management.

5.2.1 Central Dock

In the general system, a central component is required to act as a bridge between the server and the EPD devices. This component needs to have sufficient resources to handle large numbers of devices and process enormous amounts of data from the server. Also, depending on the EPD devices, this central component will have specific features and functions.



5.2.2 Battery-less EPD devices

With its bi-stable nature that the display can hold an image for a really long time without any power supply, the EPD device may not need a battery to function properly. It only consumes power when the display is being updated, and most of the power used is from the ESP32 board. This advantage of e-paper display leads to a management solution for the use case when minimal power consumption is crucial, and a frequent data update is not important. As such, the central component will have to handle the data update and display task of multiple individual EPD devices, which is now only the display panel.

A Raspberry Pi board can meet these requirements to become a central dock of the system. With 48 GPIO pins equipped, this device can connect and transmit data to the EPD devices via the SPI interface. Also, it is a powerful mini-computer capable of handling multiple tasks simultaneously and supports various programming languages, including Python, which is often used for EPD programming due to its simplicity and vast library support. On top of that, Raspberry Pi is relatively small compared to other equivalent mini-computers, which brings enormous benefits to the system, which has limited space and needs mobilities. This setup also reduces the cost of setting up the EPD display system significantly, which is very crucial in small businesses

To display data, an EPD device connects to the dock via GPIO pins, which also provide power for the display task. After the data is displayed, the EPD device can disconnect from the dock, and the data remains displayed on the screen until a new display is needed. The EPD panel can also connect to the dock via Near Field Communication (NFC), which can also transfer power wirelessly to the EPD devices. This connection method can open great potential for managing EPD devices when users can directly display data on devices right from their phones or any NFC-support devices. The diagram below shows the new device management flow, with the Raspberry Pi acting as the central dock to receive, handle, and transmit data to EPD devices. The central dock can be a Raspberry Pi with the functions discussed

above, or even users' phones and tablets if NFC is used as the connectivity between EPD devices and the dock, and in that case, the users can directly control via the central dock without an intermediate server.

However, there are also drawbacks to this management method, making it only suitable for some minor use cases. First, this method is not capable of frequent updates and handling a lot of EPD devices. Users can only update one EPD device at a time, which will take a lot of time to update multiple EPD devices. Also, EPD devices need to connect to the dock in a short range, which limits the placement and scalability of the displays, especially in larger or more complex setups. In this centralized system, if the central dock encounters an issue or fails, all connected EPD panels will be affected, and this is a significant drawback in critical applications where continuous operation is essential. This drawback also makes the development of the central dock more complex as it has to be stable, fault-tolerant, and effective in dealing with different display content for each panel.

These disadvantages of the central local dock make this solution only effective in small and simple use cases when the disadvantages do not significantly impact the overall functionality or efficiency of the system.

5.2.3 Current solution

The current solution is designed to handle EPD devices that are ESP32-C3 Supermini boards connected with EPD panels, allowing them to interact with the system individually. In this solution, the central dock does not need to process an extensive amount of data compared to the dock in the above method and only needs to relay data between the server and the EPD devices. The EPD devices can connect to this central dock at long-range distances by Bluetooth Low Energy (BLE) or Internet, maintaining the flexibility and scalability of the system. There can be two types of central dock in this system, depending on the connectivity used by EPD devices. First, if the EPD devices use BLE as connectivity, the central dock has to be a local device, receiving data from the server and transmitting it to the devices via BLE. This connectivity type is beneficial for EPD devices that have poor battery life and limited data size because its data rate is only 2Mbps, not efficient for complex image data. Secondly, if the EPD devices use Wi-Fi, this central device can be an MQTT Broker which receives data from the server and publishes it to the subscribed EPD devices via MQTT protocol. This method is currently used because it can solve the overall problem of managing multiple devices in various business scenarios. However, this approach also comes at the cost of longevity for the EPD devices as Wi-Fi connectivity consumes more power than BLE and does

not support sleeping mode in ESP32.

5.2.4 Optimization and future work

Although having some advantages and handling effectively in various scenarios, this management approach still exposes many downsides in the actual environment. Firstly, EPD devices need to have a good and long-lasting battery, which can increase the thickness and affect the visual look of the devices. Secondly, it increases the setup and maintenance cost of the system as the devices are more complex and expensive to ensure quality and effectiveness. Thirdly, this solution is generally designed for most common cases which limits the flexibility of the system. Section 5.3 will provide more insights into the works conducted to optimize the EPD devices' performance and improve battery life. Another solution is combining the advantages of the first solution into this, making it aligned with small environments while also suiting more complex needs of larger use cases. In this scenario, an additional central dock is implemented between the MQTT broker and the devices for BLE and NFC connectivity in smaller use cases and for some tasks that do not require an Internet connection. In larger use cases when an Internet connection is mandatory, the server will store a wish list of each device, containing the data assigned to but not displayed on the device yet. With this solution, the EPD device can sleep and reconnect to the broker to get and handle missed requests in a short period of time. If the devices need to sleep for a longer period, the intermediate central dock can also wake them up via BLE, which still functions when the device is in sleep mode. However, in the scope of this project, this combined solution has not yet been implemented due to the complexity of the development process. The current solution is tested and can handle well in the ideal environment, achieving the goal of designing and building the prototype of the project and making it the base for further optimizations.

5.3 ESP32 Optimization

As discussed in chapter 4 and the section above, the EPD device's performance is currently just sufficient in the prototype stage and requires further optimization in both resources and battery life. This section will provide more details of the optimizing process, including tracking and addressing the issues.

5.3.1 Battery

In normal working conditions with a continuous Wi-Fi connection, ESP32-C3 takes an average of $324mA \times 3.3V = 1.0692W$ at peak, which is calculated roughly based on resources load and the mode's typical consumption defined in the datasheet. The table below indicates the power consumption of each component in

the device in the ideal environment and working conditions (Room temperature, 3.3V supply, infrequent e-paper refresh, constant Wi-Fi, and MQTT connection).

Components	Active Mode	Note
CPU Consumption	40mA (avg.)	CPU run at max speed 160MHz with other peripherals enabled
RF Consumption	276mA	RF current consumption at peak in 802.11n mode
ePaper Consumption	8mA (avg.)	Typical current consumption with rare refresh frequency (maximum 100mA)
Total	324mA	Total current consumption at peak, actual rate may vary

Table 5.1: System services

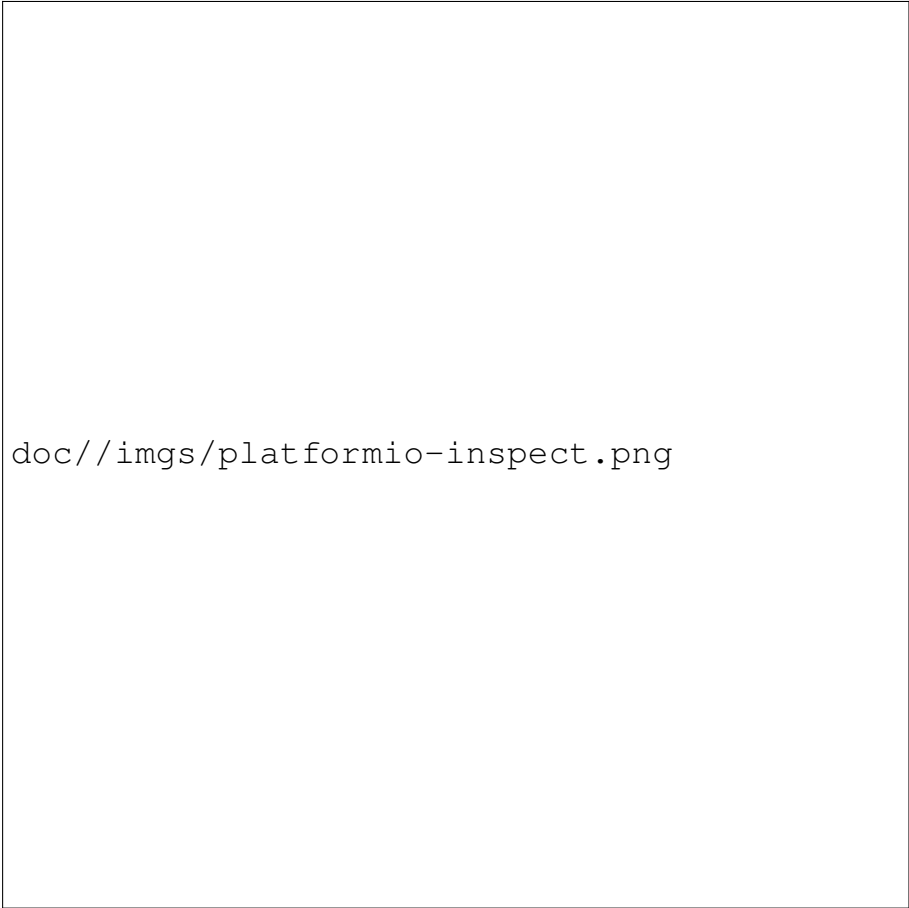
The current only way to improve battery life and reduce the constraints of devices is by optimizing CPU and RF loads, which can be achieved by strategically managing processes in the device. In the future, implementing a bigger battery without affecting the visual look significantly is also a good option to improve the battery life of the devices. The section below will focus on resource optimization, reducing CPU load, overhead, and memory wastes, this also contributes to battery improvement.

5.3.2 Resources

Here are the details of program usage in the ESP32-C3 board reported by PlatformIO before optimization. This VS code plugin also provides more insights about each component's size and allocation, which is helpful for indicating and addressing the issues.

Section	Size (bytes)	Description
Text	677,023	Size of the compiled machine code, stored in the flash memory of the ESP32.
Data	237,964	Size of initialized global and static variables. Initially stored in flash memory and then copied to SRAM
BSS	672,304	Size of uninitialized global and static variables. Stored in SRAM and zero-initialized when the program starts
Total Storage (Text + Data)	914,987	Total storage in flash memory (4MB total)
Total RAM (Data + BSS)	910,268	Total RAM usage (365KB actual total)
Overall (Text + Data + BSS)	1,587,291	Total firmware size, helpful for OTA updates

Table 5.2: System services




doc//imgs/platformio-inspect.png

Figure 5.2: Enter Caption

From the details provided in the table 5.2, it is clear that exceeding 910KB of total RAM usage may lead to potential overhead and memory leaks and also put constraints on the device if not properly managed and allocated. This also poses a risk of malfunctioning and battery draining, which decreases the overall performance of the devices a lot. To reduce Data and BSS usage and address this problem, one of the good practices is reducing the number of global and static variables if redundant and using dynamic allocation. Reducing tasks in the device also helps reduce the burden on CPU load, which can decrease CPU clock speed and thus improve battery life.

With the help of PlatformIO Inspect tools, a detailed analysis of the variables in the program has been conducted, revealing valuable insights into their size and sections. Moreover, ESP's useful libraries in resource management also help track resource usage at specific parts, providing a comprehensive picture of code structure. This detailed breakdown helps in pinpointing specific areas where memory usage can be optimized. Thus, numerous issues have been effectively identified and addressed, making memory management efficient and improving overall application performance.



`doc//imgs/platformio-debug.png`

Figure 5.3: Enter Caption

The most significant memory optimization work is changing the way of storing and processing custom Vietnamese text fonts, which is discussed in section 5.1. The occupied size of an average font decreased three times to just 11KB, which overall reduces memory usage and lookup time, leading to reduced CPU and RAM load. A lot of duplicate, large, and unused variables are also identified, removed, and reallocated dynamically in numerous parts of the code. As a result, the code structure is well-optimized, saving resources while still ensuring the performance of the devices. The table 5.3 below shows the memory usage after optimization.

Section	Old size (bytes)	Size changed (bytes)
Text	692,494	+15,471
Data	198,283	-39,681
BSS	420,482	-251,822
Total Storage (Text + Data)	890,777	-24,210
Total RAM (Data + BSS)	618,765	-291,503
Overall (Text + Data + BSS)	1,320,259	-267,032

Table 5.3: System services

In the actual environment, the frequency of data updates is not high, meaning most of the time the EPD device just keeps the connection to the MQTT Broker and waits for new requests, which is a time-, resource-, and battery-wasting task. A periodical sleep mode can be implemented to address this problem; however, it will also deactivate the RF component in ESP32, which is responsible for the Wi-Fi connection. Although it can reconnect to Wi-Fi and the MQTT broker after a period of sleeping time, the device still has chances of missing requests from users, and if not configured to handle, this may lead to a missing information scenario, which is very crucial in businesses. Another solution to this is also discussed in the previous section 5.3, which is currently not implemented yet. Other than that, the optimized system has been proven to handle tasks effectively in various testing conditions.

5.4 Security and vulnerabilities

A secure and reliable connection is paramount in a system that includes many devices communicating with each other and storing sensitive data. This project, however, had to go through a system hack that resulted in all users' data being deleted and sold on the black market. This experience, together with other security issues in the development process, has brought an urgency to enhance the system's security.

5.4.1 MongoDB Hack

Currently, MongoDB in the system is configured securely with the user's credentials and is being proxied behind a secure connection. However, on the first days of the development, with rudimentary configurations and exposing many vulnerabilities, the MongoDB server is "open" to the public with little protection from the system firewall. Several hacks targeting the MongoDB databases via the open port 27017 of the MongoDB server have been conducted by a hacker who deleted all

collections and overwrote them with a recovery message. This message demanded a payment of 0.025 BTC (equivalent to \$1,000) to recover the data; otherwise, the data will be "published on the darknet forums on the TOR network".

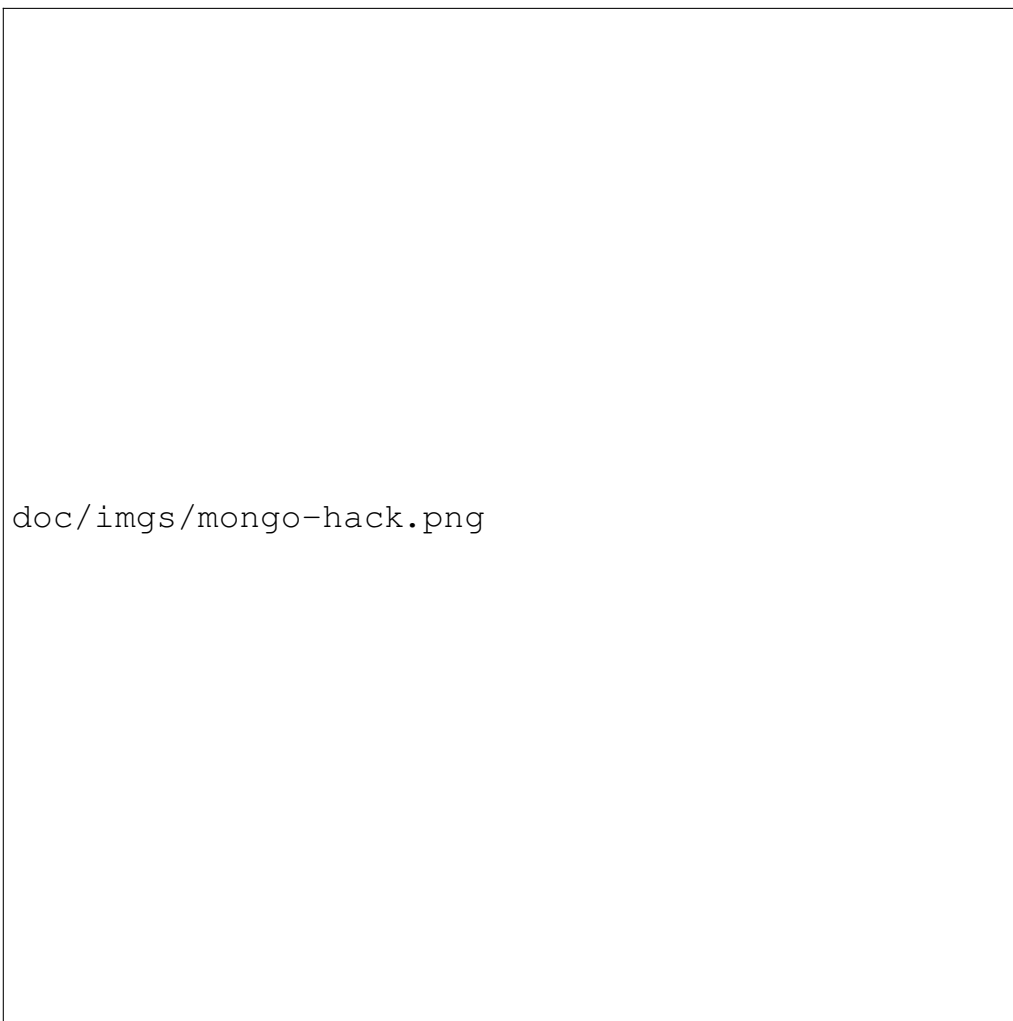


Figure 5.4: Enter Caption

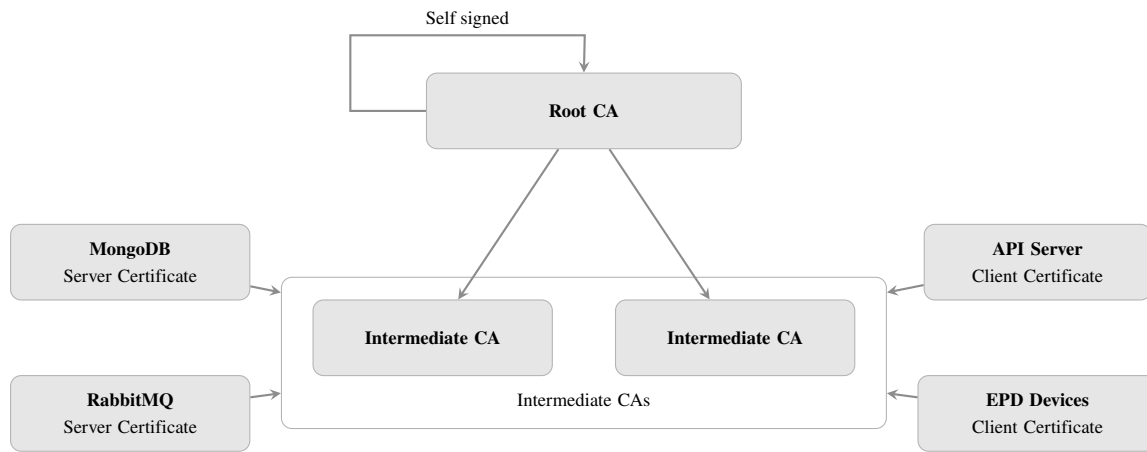
A rectangular box with a thin black border. Inside the box, the text `doc/imgs/mongo-pig.png` is centered in a monospaced font.

Figure 5.5: Enter Caption

5.4.2 TLS/SSL and Certificate Authorities (CA)

MongoDB is one of several tools used in the system that have open connections to other services, together with RabbitMQ and the web servers. Each tool comes with built-in mechanisms to protect data and communications and a logging system to trace down any abnormalities in the system. One of the critical features for enhancing security in both these systems is the implementation of TLS/SSL (Transport Layer Security/Secure Sockets Layer) protocols, using internal Certificate Authorities (CAs) to issue and verify digital server and client certificates.

In the internal CA, Root CA serves as the highest level of trust within the system's certificate hierarchy. It is responsible for self-signing its certificate and issuing Intermediate CA certificates. These Intermediate CAs bridge the gap between the Root CA and lower-level certificates. They inherit trust from the Root CA and are used to issue, verify, and revoke server and client certificates. In the services participating in the system, including MongoDB server, RabbitMQ broker, API server, and EPD devices, server and client certificates are used to secure communication. Server certificates are used to encrypt data transmitted to and from servers, ensuring confidentiality. Client certificates authenticate users or devices, confirming their identity before granting access to specific resources or services. Also, the web server uses Cloudflare SSL certificates to implement secure HTTPS protocol between the user and the system. The below image illustrates the CA system used in the project. Detail of certificate configurations and documentation is placed inside the `/certificates` folder in the project's GitHub repository.



By implementing this Internal CA infrastructure, the system can enable secure TLS/SSL connections between services, ensuring a robust and organized approach to digital security, safeguarding sensitive data, and maintaining the trust and integrity of online interactions.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

Sinh viên so sánh kết quả nghiên cứu hoặc sản phẩm của mình với các nghiên cứu hoặc sản phẩm tương tự.

Sinh viên phân tích trong suốt quá trình thực hiện ĐATN, mình đã làm được gì, chưa làm được gì, các đóng góp nổi bật là gì, và tổng hợp những bài học kinh nghiệm rút ra nếu có.

6.2 Future work

Trong phần này, sinh viên trình bày định hướng công việc trong tương lai để hoàn thiện sản phẩm hoặc nghiên cứu của mình.

Trước tiên, sinh viên trình bày các công việc cần thiết để hoàn thiện các chức năng/nhiệm vụ đã làm. Sau đó sinh viên phân tích các hướng đi mới cho phép cải thiện và nâng cấp các chức năng/nhiệm vụ đã làm.

SHORT NOTICES ON REFERENCE

Lưu ý: Sinh viên không được đưa bài giảng/slide, các trang Wikipedia, hoặc các trang web thông thường làm tài liệu tham khảo.

Một trang web được phép dùng làm tài liệu tham khảo **chỉ khi** nó là công bố chính thống của cá nhân hoặc tổ chức nào đó. Ví dụ, trang web đặc tả ngôn ngữ XML của tổ chức W3C <https://www.w3.org/TR/2008/REC-xml-20081126/> là TLTK hợp lệ.

Có năm loại tài liệu tham khảo mà sinh viên phải tuân thủ đúng quy định về cách thức liệt kê thông tin như sau. Lưu ý: các phần văn bản trong cặp dấu < > dưới đây chỉ là hướng dẫn khai báo cho từng loại tài liệu tham khảo; sinh viên cần xóa các phần văn bản này trong ĐATN của mình.

<**Bài báo đăng trên tạp chí khoa học:** Tên tác giả, tên bài báo, tên tạp chí, volume, từ trang đến trang (nếu có), nhà xuất bản, năm xuất bản >

[1] E. H. Hovy, "Automated discourse generation using discourse structure relations," *Artificial intelligence*, vol. 63, no. 1-2, pp. 341–385, 1993

<**Sách:** Tên tác giả, tên sách, volume (nếu có), lần tái bản (nếu có), nhà xuất bản, năm xuất bản>

[2] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2007.

[3] N. T. Hải, *Mạng máy tính và các hệ thống mở*. Nhà xuất bản giáo dục, 1999.

<**Tập san Báo cáo Hội nghị Khoa học:** Tên tác giả, tên báo cáo, tên hội nghị, ngày (nếu có), địa điểm hội nghị, năm xuất bản>

[4] M. Poesio and B. Di Eugenio, "Discourse structure and anaphoric accessibility," in *ESSLLI workshop on information structure, discourse structure and discourse semantics*, Copenhagen, Denmark, 2001, pp. 129–143.

<**Đồ án tốt nghiệp, Luận văn Thạc sĩ, Tiến sĩ:** Tên tác giả, tên đồ án/luận văn, loại đồ án/luận văn, tên trường, địa điểm, năm xuất bản>

[5] A. Knott, "A data-driven methodology for motivating a set of coherence relations," Ph.D. dissertation, The University of Edinburgh, UK, 1996.

<**Tài liệu tham khảo từ Internet:** Tên tác giả (nếu có), tựa đề, cơ quan (nếu có), địa chỉ trang web, thời gian lần cuối truy cập trang web>

[6] T. Berners-Lee, *Hypertext transfer protocol (HTTP)*. [Online]. Available:

`ftp://info.cern.ch/pub/www/doc/http-spec.txt.Z` (visited on 09/30/2010).

[7] Princeton University, *Wordnet*. [Online]. Available: `http://www.cogsci.princeton.edu/~wn/index.shtml` (visited on 09/30/2010).

REFERENCE

- [1] E. H. Hovy, “Automated discourse generation using discourse structure relations,” *Artificial intelligence*, vol. 63, no. 1-2, pp. 341–385, 1993.
- [2] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2007.
- [3] N. T. Hải, *Mạng máy tính và các hệ thống mở*. Nhà xuất bản giáo dục, 1999.
- [4] M. Poesio and B. Di Eugenio, “Discourse structure and anaphoric accessibility,” in *ESSLLI workshop on information structure, discourse structure and discourse semantics, Copenhagen, Denmark*, 2001, pp. 129–143.
- [5] A. Knott, “A data-driven methodology for motivating a set of coherence relations,” Ph.D. dissertation, The University of Edinburgh, UK, 1996.
- [6] T. Berners-Lee, *Hypertext transfer protocol (HTTP)*. [Online]. Available: <ftp://info.cern.ch/pub/www/doc/http-spec.txt>. Z (visited on 09/30/2010).
- [7] Princeton University, *Wordnet*. [Online]. Available: <http://www.cogsci.princeton.edu/~wn/index.shtml> (visited on 09/30/2010).

APPENDIX