DEVELOPMENT OF A CROSS-PLATFORM MOBILE APPICATION FOR MONITORING PALM OIL MILL (POM) PROCESSES

LIM JUAN HONG

UNIVERSITI TUNKU ABDUL RAHMAN

DEVELOPMENT OF A CROSS-PLATFORM MOBILE APPLICATION FOR MONITORING PALM OIL MILL (POM) PROCESSES

LIM JUAN HONG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Software Engineering (Honours)

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : LIM JUAN HONG

ID No. : 2105435

Date : 17/9/2025

COPYRIGHT STATEMENT

This final year project report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Science (Honours) Software Engineer at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Khor Kok Chin for his invaluable advice, guidance and immense patience throughout the development of this project.

Special thanks to my supportive classmate, Kim Bei Er, a fellow Software Engineering student, who worked closely with me on the same company during my final year project. Her continuous support and collaboration have been truly appreciated.

I would also like to extend my appreciation to the faculty and the departmental members from Lee Kong Chian Faculty of Engineering and Science and Department of Computing (DC), for creating a pleasant learning environment throughout my years in UTAR.

Lastly, my heartfelt gratitude goes to my dearest family and friends for their unlimited help and support, which has been instrumental in the success of this project.

ABSTRACT

This collaborative project with **Novaflow Engineering Sdn. Bhd.** aims to overcome the limitations of the current mobile application used for monitoring palm oil mill (POM) processes. The existing system faces challenges in terms of viewing the real-time data and receiving alarm notifications, which affect operational efficiency and safety. The proposed solution for this project involves designing and developing a cross-platform mobile application that shall provide a visual representation of the POM process layout. The application shall display real-time data updates, alarm notifications for critical conditions, and improvements in the user interface for enhanced usability. Additionally, filtering graph features is added to facilitate easy data analysis and comparison. The system also integrates with Influx DB, the database currently used by the company to store data. The proposed solution ensures more efficient monitoring, enhanced safety, and improved operational performance. In conclusion, the project delivers a practical and efficient solution that meets the operational needs of Novaflow Engineering and provides a robust, cross-platform application to support ongoing improvements in the company's POM operations.

Keywords:
palm oil mill monitoring; cross-platform mobile application; real-time data visualization; alarm notification system; industrial process automation

Subject Area:
TS155–194 Production management. Operations management

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

bar	pressure
psi	pressure
A	current
ch	channel

POM	palm oil mill
SDG	sustainable development goals
BPV	back pressure vessel
BPR	back pressure receiver
JIT	just-in-time
AOT	ahead-of-time
SVG	scalable vector graphics
XML	extensible markup language
PNG	portable network graphics
JPEG	joint photographic experts group
2D	two-dimensional
SUS	System Usability Scale
CSUQ	Computer System Usability Questionnaire
UMUX	Usability Metric for User Experience
SQL	structured query language
BaaS	backend-as-a-service
RDBMS	relational database management system
WBS	work breakdown structure
UML	unified modelling language
IDE	integrated development environment
FR	functional requirements
NFR	non-functional requirements
N/A	not applicable

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Palm oil mills (POM) are essential for processing raw palm fruit into palm oil with large amounts of steam and electricity (Steve et al., 2018). This project focuses on improving the monitoring of POM processes, mainly on the devices such as **sterilizer and digester**, though the implementation of IoT technology. By transforming traditional hardware systems into IoT devices, the project aims to enable real-time monitoring and alarm notifications for critical conditions.

While many POM processes have become mostly automated, some procedures still require human oversight, especially in monitoring the devices. Operators must regularly track the channel data such as **pressure (bar/psi) and current (A)** even though the process is automated, which results in time-consumption and higher labour costs. Therefore, the integration of IoT devices help in reducing on monitoring, lowering labour costs, and eventually improving overall efficiency of POM processes.

The remote monitoring function shall further improve overall management efficiency, as managers and operators able to respond quickly to any alarms or irregularities without being physically present at the mill. This capability also allows for better resource allocation, as managers can identify issues and make decisions remotely. The alarm notifications ensure the critical conditions are solved immediately, reducing the risk of errors and improving the safety throughout the operation.

Novaflow is an engineering company that mainly focuses on industrial automation, palm oil and sewage & water solutions. The company aims to develop additional monitoring and automation technologies in its palm oil solution to optimize their manufacturing operations. However, Novaflow currently faces limitations on existing iOS mobile application (**SmartMachine365)** for the palm oil industry, which is subscribed from a third-party provider. The current solutions offer only basic functionalities and lack accessibility for Android users. Due to these constraints, Novaflow

decided to develop a customized cross-platform mobile application that provides enhanced monitoring features for managing palm oil mill processes more efficiently.

1.2 Importance of the Study

This project is aligned with the United Nations' Sustainable Development Goals (SDG) and Malaysian's government policies. The importance of the study is outlined as below:

1.2.1 United Nations' Sustainable Development Goals (SDG)

This project is support **SDG9: Industry, Innovation, and Infrastructure** by enhancing the palm oil mills (POM) monitoring processes through technological innovation. The development of real-time monitoring and alarm notification system allow manager or operator to have better decision-making, which can reduce the machine downtime and increase in palm oil mills' operational performance.

Besides that, this project also supports **SDG12: Responsible Consumption and Production** by ensuring the resources in palm oil mills are manage efficiently. The ability to monitor critical process conditions in real time minimizing the error that might lead to production downtime and equipment failures. The alarm notification system can help to prevent unnecessary loss in palm oil mills production, leading to a more sustainable and responsible industrial practices.

1.2.2 Malaysian's Government Policies

This project support with Malaysia's government policies, mainly on **National Policy on Industry 4.0 (Industry 4WRD)**, which aims to make digital transformation on Malaysia's manufacturing sector. By implementing a real-time monitoring system for palm oil mills, this project aligns with Malaysia's four national goals under Industry 4.0. First, the real-time monitoring system can **increase the productivity in manufacturing sector** by applying automation and real-time data analytics to make industry operations more efficient and reduce machine downtime. Secondly, the project **contributes to**

the economy from manufacturing sector by ensuring better resource allocation, where the unnecessary waste is minimized. Palm oil mills can increase their production while reducing unnecessary resource consumption through a smart monitoring system.

Thirdly, this project **strengthens innovation capability**, supporting Malaysia's goal on **improving its global innovation ranking** in manufacturing sector. The integration of smart monitoring and notification system reflects the adoption of Industry 4.0 technologies, leading manufacturing industry to more advanced in technology. Lastly, this project contributes to **increase the number of high-skilled workers** by encouraging the adoption of digital tools in the manufacturing sector. The transformation from a traditional to a smart monitoring system requires workers to develop and apply more technological skills, aligning with Malaysia' goal of increasing more knowledgeable and technology-driven workforce.

1.3 Problem Statement

The **four** main problems that faced by Novaflow in monitoring the POM processes are outline below:

1.3.1 Limited Android access and discontinuation of iOS subscription

Currently, Novaflow's POM monitoring system is only available on iOS devices, which limit the accessibility for Android users. This limitation makes it difficult for Android user to track the devices' channel data in real time, especially when they are not around their desk or outside the office. In cases of happening emergencies, delays in addressing issues can lead to production downtime.

Additionally, Novaflow plans to discontinue the subscription for the current iOS mobile app which is provided by a foreign company. The existing app has limited functionality, offering only basic features. By developing a cross-platform mobile application, both Android and iOS users able to monitor real-time channel data from anywhere using their mobile devices, allowing for faster responses to critical conditions.

1.3.2 Limited visual representation of real-time processes

Existing monitoring interfaces are lack of intuitive graphical layouts, leading to the operators or managers difficult to interpret data efficiently. By integrating SVG graphics, this project aims to provide a clear and dynamic visualization of POM processes for better decision-making.

1.3.3 Delayed response to critical conditions

Many mills rely on manual checks by human operators, increasing the risk of delayed responses to critical conditions. Since operators need to physically track the devices' channel data by following their scheduled rounds, sudden critical conditions for extended periods of time may lead to production downtime or even damage on equipment. The implementation of an alarm notification system ensures that critical conditions are detected and addressed immediately.

1.3.4 Limited usability and interface constraints

The current Novaflow application has limitation in usability, as it only allows users to view single channel data at a time and lacks alarm notifications for critical conditions. These limitations make real-time monitoring and analysis inefficient, especially in critical situations. Additionally, managers that have different subgroups restricts access to its subgroups' dashboards and their devices' channel data. In this case, managers must manually switch between subgroups by logging in and out, which is time-consuming and inefficient for overseeing multiple groups and devices. Hence, this project aims to enhance the monitoring system by improving accessibility, enabling smoother interaction and allowing simultaneous access to multiple subgroups' dashboards and channel data for better analysis and decision-making.

1.4 Aim and Objectives

The aim of this project is to design and develop a mobile application (**SmartMill365)** that supports both Android and iOS devices for monitoring palm oil mills (POM) processes.

The objectives of this project that refined based on Novaflow's requirements are:

a. To develop a cross-platform mobile application using Flutter that can replace the existing company's iOS app and ensures accessibility for both Android and iOS users in monitoring POM processes.
b. To integrate SVG graphics for displaying process layouts along with real-time data display.
c. To implement an alarm notification system to alert users about critical process conditions.
d. To enhance system usability and interface for more efficient real-time monitoring and analysis.

1.5 Proposed Solution

Developing a cross-platform mobile application for POM processes monitoring system is essential to address the issues described in the problem statement above. Novaflow currently used a web-based monitoring system as the primary platform for monitoring all the devices' channel data. However, the existing mobile solution, which is subscribed from a foreign company, is limited in functionality, restricting accessibility for Android users and offering only basic features.

The new mobile-based monitoring system developed in this project, serve as a companion application to monitor the channel data of all the devices with enhanced features. While the real-time channel data updates for each device shall remain same as the existing system, the new solution have a **multi-graph view** that allow users to view all channel data on a single screen, which improving the monitoring efficiency and analysis.

Additionally, users can **filter and customize displayed graphs** to focus on specific devices, compare among devices for performance analysis,

and adjust the time range for data display to analyse historical performance based on their need. Furthermore, this system enables managers to **access and monitor multiple subgroups' dashboards** without the need to log in and out repeatedly. This enhancement improves operational oversight by providing a consolidated view of all related devices under same company and their channel data.

Another critical enhancement is **the implementation of an alarm notification feature**, which alert users when channel data exceed or below predefined thresholds. This feature ensures immediate awareness of critical conditions, allowing for faster responses times and improved operational performance.

These enhancements shall make the monitoring system become more flexible and feature-rich, allowing users to monitor critical process conditions anytime and anywhere. As a result, the proposed solution ensured better decision-making, enhanced operational efficiency, and improved safety measures.

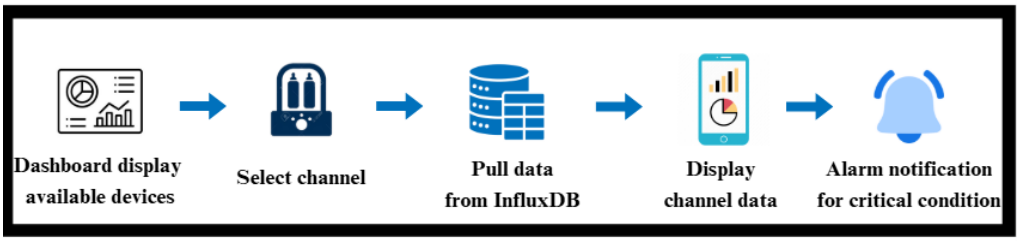


Figure 1.1: Operational Flow of POM Monitoring System.

The operational flow diagram as shown in Figure 1.1 above illustrates how the mobile application monitors channel data of the devices in the palm oil mill (POM) process. Users start by accessing the dashboard, which displays all available devices such as **sterilizer and digester** for selection. After choosing a desired channel, the application retrieves real-time channel data from Influx DB. The channel data such as **pressure (bar/psi) and current (A)** shall then display on the mobile screen, allowing users to monitor and analyse data effectively. Additionally, an alarm notification system is triggered if the channel data exceeds or below a predefined level, ensuring immediate corrective action is taken.

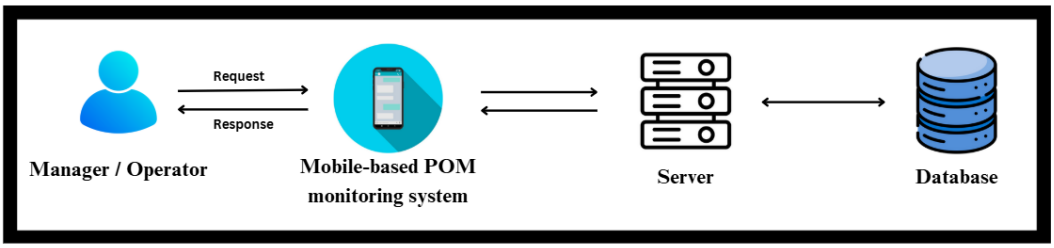


Figure 1.2: Overview of Design Architecture.

1.6 Proposed Approach

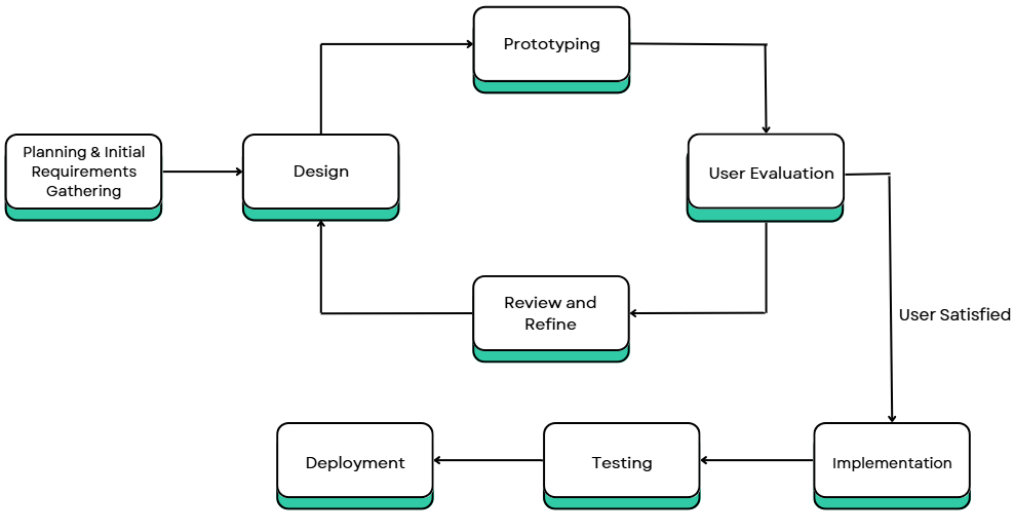


Figure 1.3: Overview of Evolutionary Prototyping Model.

The software development methodology that decided to use is **evolutionary prototyping model** as shown in Figure 1.3. This approach shall develop an initial prototype with essential features, which then **continuously refined through multiple iterations** based on self-evaluation and feedback from company supervisors or industry experts (Camburn et al., 2017). By using this method, the system shall improve progressively, reducing risks and ensuring that the final product meets user expectations.

The project started with requirement gathering to ensure a clear understanding of the system's needs. Next, a prototype is then developed with core functionalities, allowing early testing and identify areas of improvement. This prototype then undergoes self-evaluation and review sessions with the company supervisor to gather feedback. Based on the feedback, refinements

are made, and additional functionalities are incorporated in following iterations.

Each iteration of the prototype is tested before proceeding to the next phase. Once all features are developed and validated through multiple iterations, a final refined prototype is created and reviewed before proceeding with the full implementation. After getting approval, the actual system development begins. Any changes during the development process are based on ongoing evaluations to ensure that system meets the operational needs of the palm oil mill (POM) monitoring system.

1.7 Project Scope

The palm oil mill (POM) processes monitoring system aims to enhance existing monitoring features by **developing a cross-platform mobile application (SmartMill365)** built with **Flutter framework** that provides real-time monitoring of devices' channel data such as pressure, and alarm notifications for critical conditions. The mobile applications are compatible with both Android and iOS platform. **Firebase authentication** is implemented to validate user logins, only allow authorized user to monitor the channel data.

The system shall retrieve real-time channel data from Influx DB and display them on a dashboard, allowing users to monitor all channel data. The system shall offer additional features such as **filtering and comparing channel data across different devices** and **selecting custom time ranges** for historical analysis. An **alarm notification system** is integrated to alert users when channel data exceed certain predefined thresholds, ensuring immediate response are taken through the notifications. In addition, users are able to **monitor and review past alarm notifications** through the application.

Furthermore, the system shall allow managers to **access and monitor multiple subgroups' dashboards** without requiring repetitive logins. In this case, managers can direct switch between subgroups' dashboard through drop-down options from the home screen. Additionally, the application shall feature a **graphical interface using SVG files** to visually represent the palm oil mill process layout with real-time channel data. This visualization helps users quickly understand the operational status of different devices in an intuitive

way. A clean and user-friendly user interface is prioritized to enhance usability, ensuring that monitoring is simple and accessible for all users.

1.8 Contribution of the Study

This project benefits to various stakeholders such as industrial operators, automation managers, and business owners. By implementing the mobile-based palm oil mills monitoring system, it improves the overall efficiency and accessibility in monitor the palm oil mills operations.

One of the primary advantages of the developed mobile application is allowed to **remote access anytime and anywhere**. Unlike the main existing monitoring system which is web-based, this mobile solution ensures that operators and managers can access the real-time channel data from device remotely. In this situation, they are able to quickly responses to any critical conditions, reducing any unnecessary waste or production downtime.

The implementation of alarm notification system is another significant contribution. By alerting users when channel data exceed predefined thresholds, the system enable users to **detect the issue earlier and make decision immediately**, minimizing the risk of equipment failures and improving operational safety. This feature mainly benefits to operators as they no longer need to physically monitor the devices from time to time.

Additionally, the mobile application able to enhance **data analysis and performance evaluation** by allowing users to compare channel data across multiple devices. Automation managers can utilize graph filtering and time range adjustments to analyse historical data, identify any inefficiencies and improvement area for better productivity in future.

From a broader perspective, this project aligns with Industry4WRD, Malaysia's initiative to drive **digital transformation in manufacturing industry**. By integrating this smart monitoring system, it contributes to the adoption of smart manufacturing technologies in the palm oil industry. This ensures that Nova Flow and its stakeholders remain competitive in this technology-driven era while improving operational efficiency and decision-making processes.

Table 1.1: Challenges Faced by Novaflow and Proposed Solutions.

Challenges	**Proposed Solutions**
Rely on third-party iOS mobile solution	Develop a customized cross-platform mobile application follow company-specific needs.
Lack of comprehensive real-time data visualization	Implement a multi-graph view that displays all channel data on single screen for better monitoring and analysis.
Unable to filter and compare device performance	Implement graph filtering and time range selection, enable users to compare different devices' performance.
Unable to access and monitor subgroups' dashboard simultaneously	Implement multi-subgroup monitoring by allowing managers to access multiple subgroup dashboards without logging in and out.
Delayed response to critical conditions	Implement alarm notification system to alert users immediately when channel data exceed thresholds.
Lack of intuitive graphical layout	Implement SVG-based graphical layouts on dashboard for clear and dynamic visualization of palm oil mill process.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews existing literature and technologies related to the development of the Palm Oil Mill (POM) Monitoring System. It explores previous research, methodologies, and technological advancements in real-time monitoring systems and alarm notification systems, cross-platform mobile development, data visualization techniques, and database management. By analysing past studies and existing applications, this chapter identifies opportunities for improvement, forming the foundation for the proposed solution.

The selection of an appropriate development platform and tools is crucial to formulate a smoother and efficient monitoring system. In this case, **Flutter** with its cross-platform capabilities is chosen for this mobile application development, as single codebase is used for both Android and iOS platforms. Besides that, **Firebase** is used for user authentication and for mapping channel data with Influx DB, ensuring secure login management and accurate data retrieval. For real-time data storage and retrieval, **Influx DB** is selected, as it is optimized for handling time-series data, making it suitable for monitoring machine data in the palm oil mill. Additionally, **SVG graphics** are implemented for dynamic and interactive real-time data visualization, ensuring an intuitive and seamless user interface. These selected tools will then enhance the usability, optimize performance and addressing the limitations of existing Novaflow application.

2.2 Palm Oil Mill Process

In the palm oil milling process, devices such as **boilers, back pressure vessel (BPV), sterilizers and digesters** play an important role in extracting palm oil efficiently. The monitoring system that developed in this project is crucial to ensure that all these devices are operate efficiently, preventing any breakdowns happened and maintaining optimal performance. The channel real-time data such as **sterilizer pressure (bar/psi)** and **digester electric current (A)** will be monitor from time to time, ensuring optimal steam conditions and motor performance throughout the milling process.

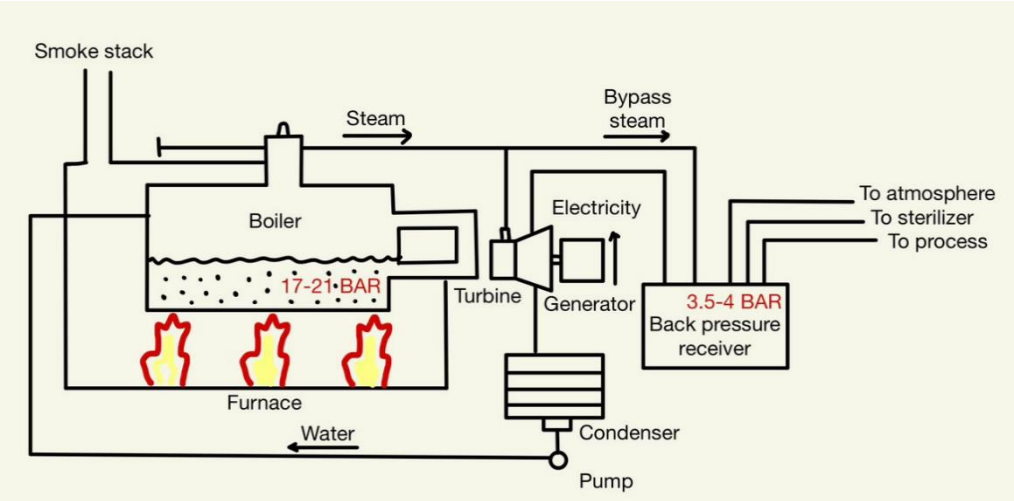


Figure 2.1: Co-Generation System in Palm Oil Mill.

The process as shown in Figure 2.1 start with the **boiler**, which generates steam by heating water. This steam is then supplied to different sections of the mill, including the **sterilizer**. At the meantime, **back pressure vessel (BPV)** also known as back pressure receiver (BPR), will be used to regulates and stores the excess steam that came from boiler, ensuring that the pressure remains stable and consistent for sterilization process. In this project, boiler and BPV will be grouped under sterilizer, as they are important components in supplying and regulating the steam for the sterilization process.

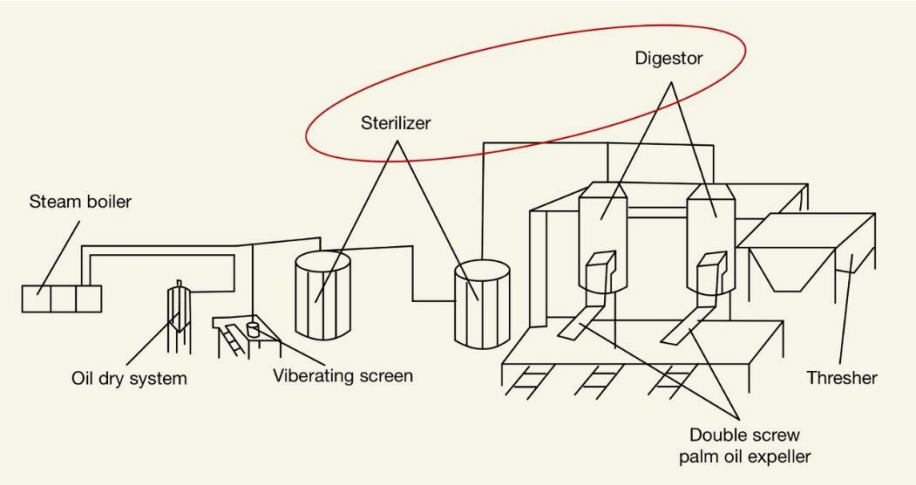


Figure 2.2: Palm Oil Press Line.

Next, the **sterilizer** as shown in Figure 2.2 used to cook the fresh fruit bunches under high pressure so that the fruits become soft and easier to separate from the bunch stalks (Kandiah et al., 2006). This process also helps deactivate the enzymes that could cause oil degradation. After sterilization, the fruits are stripped from the bunch stalks in the threshing process, then followed by digestion. During digestion, **digestor** is used to mash and break down the sterilized palm fruits. It normally uses rotating blades to stir and crush the fruit with heat, ensuring that the oil is extract efficiently. This step will soften the mesocarp and separates the oil from fruit fibres, so that it will be easier to extract crude palm oil in the following pressing stage.

Therefore, it is important to keep monitor the **electric current (A)** of digester's motor to ensure that it operates within the optimal range. By doing so, it helps to detect any abnormalities such as mechanical failures or insufficient energy earlier, which can prevent equipment breakdowns happened and minimize the disruptions in the milling process.

2.3 Cross-Platform POM Mobile Application

From Novaflow's perspective, developing a **cross-platform mobile application** for the POM monitoring system aims to maximize efficiency in development, reduce development costs, and ensure broader accessibility across different operating systems. By utilizing a single codebase without much more changes for both iOS and Android, Novaflow can streamline the development process, which eventually help in reducing the development time and labour costs. This is because shorter development time will be minimizing the need for additional manpower working on the same project.

Additionally, this cross-platform approach developed using Flutter framework ensures the **consistency in user experience** as both interface and functionality on both operating systems are almost same. This strategy also serves as a marketing advantage for Novaflow, allowing a **wider range of clients to access** and benefit from the application. Furthermore, any changes made after deployment and maintenance of the mobile application become more efficient, as changes can be implemented simultaneously across platforms, minimizing the system downtime and improving overall performance and reliability.

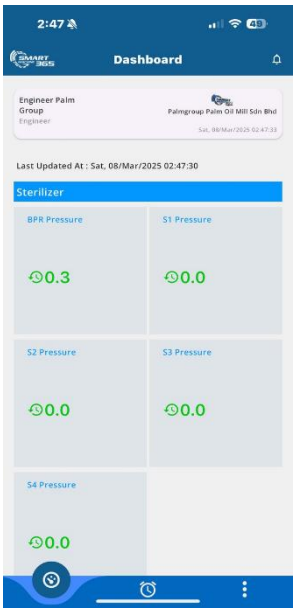


Figure 2.3: Dashboard of SmartMachine365.

In this project, Novaflow aims to enhance the monitoring system by addressing the limitations of existing application (**SmartMachine365)**. The

current Novaflow mobile application **lack of intuitive SVG graphics** on the dashboard as shown in Figure 2.3, making data visualization less interactive and harder to interpret. Besides that, manager that have different subgroups must **log in and out repeatedly to view each subgroup's dashboard** as the existing system only allows viewing one dashboard at one time. As a result, this process is time-consuming and inefficient, making it difficult for managers to monitor multiple subgroups simultaneously.

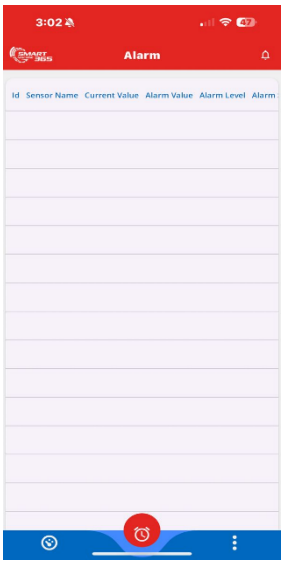


Figure 2.4: Alarm Monitoring System of SmartMachine365.

Besides that, the **alarm notification system** in existing Novaflow application is **not functioning** as shown in Figure 2.4 due to misconfiguration of the alarm settings. As a result, any critical conditions might be unnoticed by the operators or managers, leading to delay in taking corrective action.

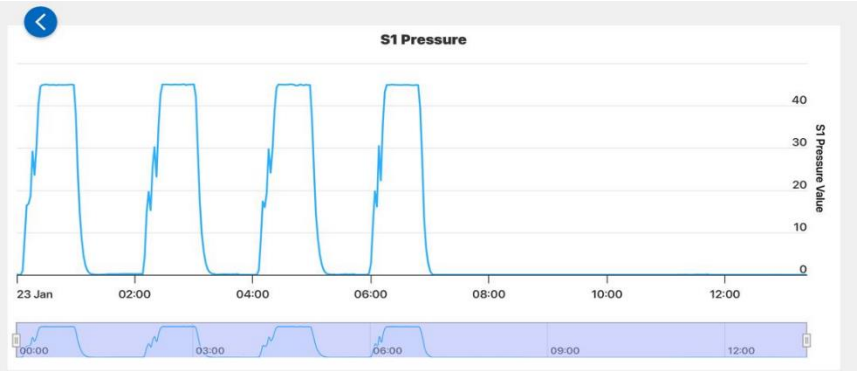


Figure 2.5: Real Time Channel Data of SmartMachine365.

Another limitation is that **only one channel of data able to view at a time** as shown in Figure 2.5. This restriction limits the ability for manager or operator to perform comparisons among different channel data for analysis purposes.

By overcoming these limitations, this project ensures a smoother and more efficient palm oil mill monitoring system. The **implementation of interactive SVG graphics** enhances data visualization, providing a more intuitive layout displaying the real-time channel data for users. Additionally, the **introduction of real-time alarm notification system** ensures that users able to receive instant alerts during any critical conditions, allowing for immediate corrective action and preventing machine breakdowns. Furthermore, **multiple-channel monitoring** allows users to view more than one graph simultaneously, improving data analysis and decision making. Besides that, the **implementation of multi-subgroup dashboards** allows managers to seamlessly access different dashboards under same company without the need to log in and out repeatedly. In short, this project significantly enhances the existing application by delivering a more robust, efficient and responsive monitoring system that aligns with the needs of Nova Flow's palm oil mill operations.

Table 2.1: Comparison of Features between Current Monitoring System.

	SmartMachine365 (existing system)	**SmartMills365** (system that will be implemented)
Platform Supported	iOS	Android and iOS
Single-graph view	✓	✓
Multi-graph view		✓
Multi-subgroup dashboards		✓
Alarm notification system		✓
SVG graphical dashboard		✓

Filtering channel's graph data		✓

2.4 Cross-Platform Framework Flutter

2.4.1 Introduction to Cross-Platform Framework

Figure 2.6: Most Popular Development SDK.

There are few cross-platform frameworks nowadays that can develop both iOS and Android applications without any cost. According to Stack Overflow (2019), the largest survey for software developers about their job preferences found that the **most popular cross-platform frameworks** are Flutter, React Native and Xamarin. Flutter framework achieved the most popular among the **cross-platform frameworks**, which is 75.4% as shown in Figure 2.6.

Flutter, a framework that mainly develop on high-performance applications, was developed by Google using its own programming language, Dart (Tashildar et al., 2020). Besides that, the second most popular cross-platform framework, React Native is developed by Facebook, mainly uses JavaScript and React for building mobile applications (Wu, 2018). Next, Xamarin, is a Microsoft-owned framework that uses C# and .NET to develop cross-platform applications with native performance (Lodhi, 2024).

2.4.2 Comparison among Performance

From the perspective of performance, React Native uses JavaScript bridge to communicate with the native operating system for rendering UI, which can lead to slower performance especially in complex applications

(Kishore et al., 2022). Besides that, Xamarin uses Just-in-Time (JIT) compiler for Android, while iOS apps must use ahead-of-time (AOT) compilation for Xamarin due to Apple's restrictions (Vishal and Kushwaha, 2018). In fact, switching between compiler can lead to an increase in app size and limit performance optimizations. Flutter, on the other hand, does not require a JavaScript bridge or JIT to interact with the native operating system as its **compiles directly into native machine code** using Dart's ahead-of-time (AOT) compilation for both Android and iOS, which can lead to faster performance and smoother the rendering process (Palumbo, 2021). As a result, Flutter helps reduce development time and labour costs as only single codebase is used for both iOS and Android platforms.

2.4.3 Comparison among User Interfaces

By looking at the user interface, Flutter uses its own rendering engine, Skia that provides platform-specific customizable widgets and plugins (Sattar et al., 2023). This allows Flutter to render UI components, images, and animations more smoothly and efficiently, ensuring a **consistent UI** across Android and iOS. In contrast, React Native uses native components, but styling and behaviour can be inconsistent across platforms (Penta, 2004). This can be taking more time during the testing phase to address these UI issues. Additionally, Xamarin.Forms offers a set of tools and components that allow developers to build cross-platform user interfaces for Xamarin framework, reducing the needs of writing separate UI code for different platforms (Ramadoss, 2023). However, it still has limitations in creating highly customized UI components compared to Flutter, as Flutter's rendering engine provides fully customizable widgets.

Thus, Novaflow has decided to use the **Flutter** framework to develop the cross-platform mobile application for the POM monitoring system, as it offers better performance, consistency in user interface across platforms, and a wide range of customizable widgets.

2.5 SVG Graphics

Scalable Vector Graphics (SVG) is an XML-based vector image format used to define 2D vector graphics for applications (Peng, 2000). It can be resized without losing quality, unlike other files such as PNG and JPEG are raster images that will become blurry when enlarged. SVG files normally smaller file size compared to other files as it stores data as code instead of store pixel data. Additionally, SVG supports animations and real-time data updates, therefore it is suitable for monitoring systems.

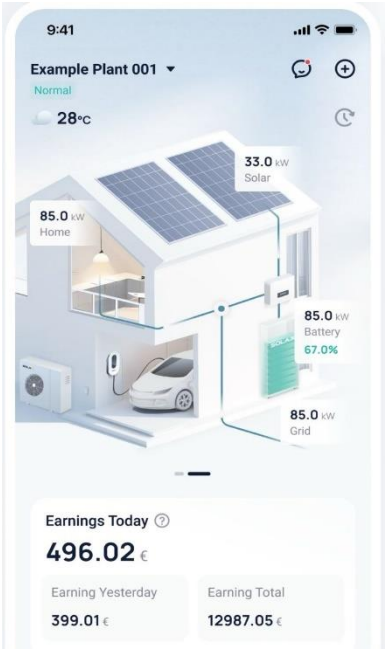


Figure 2.7: Solar Monitoring App (SolaXCloud).

One real-world example of SVG-based real-time monitoring system is solar monitoring app named as SolaXCloud as shown in Figure 2.7. This application displays real-time solar energy consumption, battery levels, and grid usage with an intuitive graphical interface. There are few SVG elements used in this application. For example, dynamic energy flow lines are used to show the real-time power distribution from solar panels to home appliances. Besides that, it also displays the current energy usage with numbers which will be update dynamically and allow users to click on it for further information.

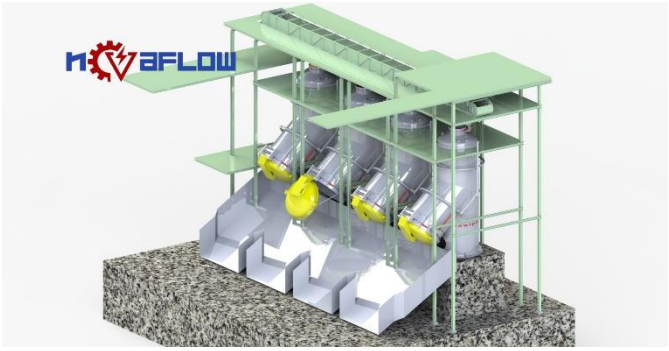


Figure 2.8: Sample 2D Vector-based Graphic.

Similarly, SVG can be used to visualize pipelines, machinery, and real-time channel data in this palm oil mill (POM) monitoring system. For example, a 2D vector-based graphic as shown in Figure 2.8 on the dashboard screen will provide an interactive layout of the mill's equipment, allowing users to monitor real-time channel data efficiently. The real-time channel data labels will be placed on specific areas corresponding to different machine types, ensuring an intuitive interface for monitoring the performance of each machine. These labels will display the real-time data with color-coded indicating the operational status, allowing users to quickly identify the abnormalities. For example, green for normal operation and red for critical alerts. When user clicked on specific label, the system will display the detailed channel data and relevant information for further monitoring and analysis.

2.6 Alarm Notification System

Alarm notification is important in monitoring systems as it uses to alert users in critical conditions that require **immediate attention and action taken**. These notifications help in maintaining stable operational processes, preventing potential equipment failures, and reducing production downtime. In this palm oil mills case, real-time alarms ensure that operators or managers can take corrective action **before the minor issues become serious problems**. By implementing this alarm notification system, users can enhance their reliability and safety of their operations, reducing the risk of failures occur.

The alarm notification implemented in this project will improve user awareness by **providing real-time alerts when predefined thresholds are exceeded.** For example, in palm oil mill, if the pressure in sterilizer exceeds a certain limit, an alarm is triggered to notify the operators or managers. In this

case, they can take **immediate corrective actions**, such as adjusting the volume or inspecting the machine to avoid any malfunctions happened. Without implementing this proper alarm system, these issues might not be noticed by them, which may lead to production downtime or delays.

In this POM monitoring app, alarm notification system will consist of **two types of alarm pages** which are **active alarms** and **alarms history**. The active alarm page displays real-time alerts when the channel data exceeds or below its predefined threshold. These **predefined high and low threshold values** along with the **alarm descriptions** are stored in the **cloud-based MySQL database** which is accessible to all authenticated users rather than stored locally. MySQL is chosen instead of Firebase for storing these values is because **Firebase has a daily request limit** where exceeding limit requires additional payment while MySQL don't have such restrictions. For example, if the high threshold for Sterilizer 1 (ch3) Pressure is set at 50 psi, an alarm will be triggered if exceeds this value. If its pressure is records 45 psi, no alarm is activated since it is below the threshold, whereas if the pressure records 55, the system triggers an alarm to alert users.

On the other hand, the **alarm history page** use to **keep record of past alarms**, including the **exact time** an alarm was triggered and when it was **cleared**. These history records **are retrieved via an API** that fetches the stored alarm data from the MySQL database when the user navigates to the alarm history page. For example, if an alarm for high pressure in Sterilizer 1 was triggered at 10:00 AM and cleared at 10:20 AM, the system will save these periods of time into a **cloud-based MySQL database** for future reference. By implementing this alarm history function, operators or managers can trace back the performance of each device on the app, identify any potential weaknesses and implement preventive measures to enhance the operational performance and stability.

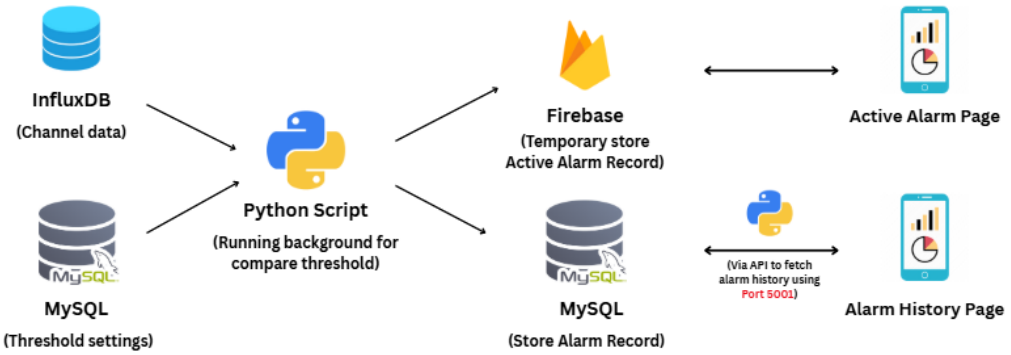


Figure 2.9: Alarm Notification System.

The continuous monitoring of threshold values is achieved through a **Python script** running under **systemd** as shown in Figure 2.9**,** ensuring that the process operates 24 hours. The script constantly checks whether devices' latest pressure data exceed or fall below the predefined thresholds set in the MySQL database. If a critical condition is detected, the alarm is logged into the **MySQL alarm history table**.

Active alarms records are temporarily stored in **Firebase** under the active_alarm collection. Once the alarm is cleared, it is removed from this collection. Additionally, the alarm history page includes an ACK (Acknowledge) button that allows user to confirm they have acknowledged that particular alarm. When the ACK button is pressed, the system logs the acknowledging user's email and acknowledgment time into the corresponding MySQL alarm record. This ensures that in the future able to trace back exactly who acknowledged the alarm and verify whether the issue raised by the alarm was addressed. Both the alarm history and ACK APIs run on an **HTTPS port created using a reverse proxy** instead of using a **Firebase Functions HTTP URL**. This significantly help to reduce latency, as Firebase Functions introduce a cold start delay of around 10 seconds for retrieving alarm records.

By combining real-time cloud data storage, continuous monitoring, and acknowledgment tracking, the alarm notification system in this POM monitoring app ensures prompt responses to critical conditions, accurate historical tracking, and improved operational reliability. The integration of both active and history alarms not only allows operators or managers to take immediate corrective actions when issues arise but also supports long-time

performance analysis. This combined approach improves overall monitoring efficiency, making the system more reliable and responsive in addressing any failures in palm oil mill operations.

Table 2.2: Comparison of MySQL and Firebase for Alarm Notification System.

Features	MySQL Cloud Database	Firebase
Storage of threshold values and alarm descriptions	Unlimited read/write access, no request limits.	Limited daily request quota, extra usage required payment.
Fetching alarm history API & ACK function	Reverse proxy HTTPS API provide faster/instant response.	Firebase Functions having ~10s cold start delay.
Cost	No extra charge for high request volume.	Pay per use if daily request quota is exceeded.
24/7 continuous monitoring	Well-suited for 24/7 systemd-based Python checks.	May be impacted by request limits during high traffic periods.

2.7 System Usability

2.7.1 Important of System Usability Test

System usability testing is important for evaluating how effectively users interact with a system, ensuring that its functionality and interface meet user expectations. This testing plays an important role especially for monitoring systems where users like operators spend long hours on tracking the real-time data. Unlike other consumer applications where interactive design is prioritized, monitoring systems focus more on usability and reducing strain on users. In this palm oil mill monitoring system, the designed interface must not only be functional but also designed to minimize eye strain and fatigue. There are some commonly used evaluation methods for system usability testing include the System Usability Scale (SUS), Computer System Usability Questionnaire (CSUQ), and Usability Metric for User Experience (UMUX).

2.7.2 System Usability Scale (SUS)

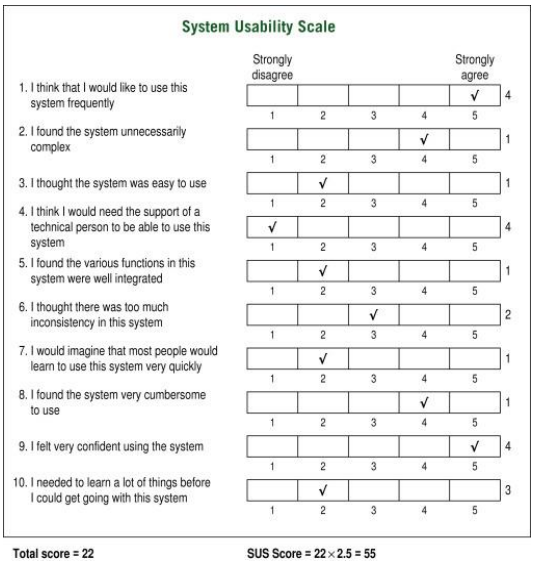


Figure 2.10: Sample of SUS.

System usability test (SUS) is a tool normally used for usability testing, mainly focuses on assessing overall system usability by providing a numeric usability score ranging from 0 to 100, allowing for easy benchmarking (Grier et al., 2013). The primary advantage is its simplicity as it consists of only 10 questions as shown in Figure 2.9, making it a fast and efficient method that save users' time. This is particularly beneficial for operators in palm oil mill factory, who are typically busy overseeing real-time data and have limited time to complete the evaluation test. The SUS score will then convert into a grading system of A to F, making it easier to interpret and compare usability levels in future assessments. However, SUS still has some limitations where it lacks detailed insights into specific usability problems. It does not offer qualitative feedback on why the usability issues occur, making developer hard to identify the precise areas of improvement.

2.7.3 Computer System Usability Questionnaire (CSUQ)

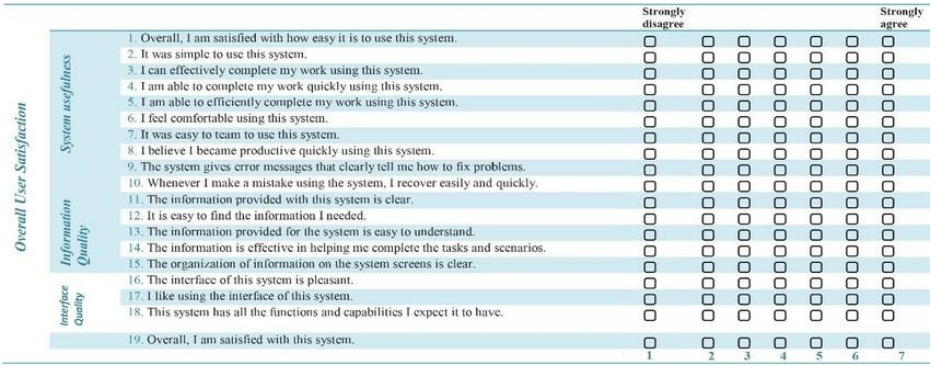
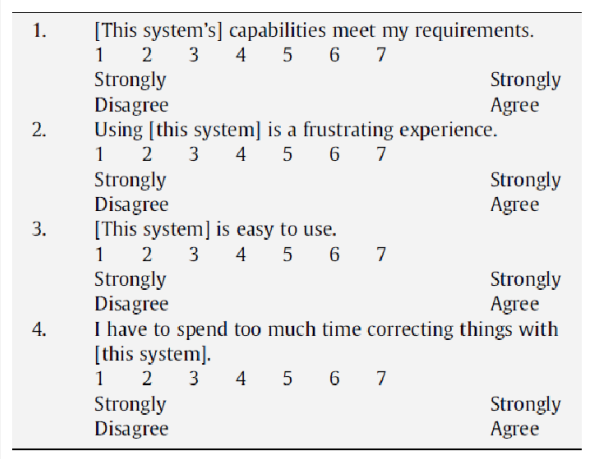


Figure 2.11: Sample of CSUQ.

Another widely used usability evaluation method is Computer System Usability Questionnaire (CSUQ), which mainly covers on system usefulness, information quality, and interface quality (Azami and Ibrahim, 2019). The questionnaire consists of 19 items which divided into four categories. The first category is overall user satisfaction covering all 19 items, system usefulness category covers 8 items, information quality covers 7 items, and interface quality covers with 3 items as shown in Figure 2.10. Unlike SUS, it allows for a more detailed user experience evaluation by gathering feedback on different aspects of the system.

However, it is more time-consuming as it consists of more questions, making it less suitable for situations where operators have limited time to answer. Additionally, CSUQ don't have a grading system, making it difficult to compare usability over time. It also more emphasis on user satisfaction rather than efficiency and ease of use, which may not align with the needs of a palm oil mill monitoring system where operators prioritize fast and accurate data interpretation over interactive design.

2.7.4 Usability Metric for User Experience (UMUX)



1.	[This system's] capabilities meet my requirements.						
	1	2	3	4	5	6	7
	Strongly Disagree						Strongly Agree
2.	Using [this system] is a frustrating experience.						
	1	2	3	4	5	6	7
	Strongly Disagree						Strongly Agree
3.	[This system] is easy to use.						
	1	2	3	4	5	6	7
	Strongly Disagree						Strongly Agree
4.	I have to spend too much time correcting things with [this system].						
	1	2	3	4	5	6	7
	Strongly Disagree						Strongly Agree

Figure 2.12: Sample of UMUX.

Usability Metric for User Experience (UMUX) is a usability evaluation tool consist of 4 questions, designed to provide a quick and reliable assessment of an application's usability (Varela-Aldás et al., 2023). Each question has 7 levels of selection, ranging from strongly disagree to strongly agree as shown in Figure 2.11. However, even though it is quick to answer, it provides limited insights into specific usability aspects. Additionally, UMUX primarily focuses on overall usability perception rather than assessing detailed usability components like system efficiency, effectiveness and user satisfaction. While UMUX is beneficial for quick assessments, it lacks the benchmarking capability and grading system like SUS, making it harder to make comparison on usability across different systems.

2.7.5 Justification for Choosing SUS in Palm Oil Monitoring System

Despite having some limitations on SUS, it still chosen for this palm oil monitoring project due to its quick administration and ease of implementation. Since operators' time often occupied with real-time data monitoring, a fast and simple usability assessment is more suitable in this case. As a result, SUS ensures that operators can complete the evaluation efficiently without disrupting their workflow while still providing valuable usability insights for system improvement. Moreover, the grading system in SUS allows for easier tracking of usability trends over time as it is measurable. In short, **SUS** is the

most appropriate choice for this project as it balances efficiency and ease of use.

2.8 Database

2.8.1 Influx DB

In this project, there are two main databases**, Influx DB and Firebase** are used to manage and support different aspects of the palm oil mill monitoring system efficiently. Firstly, **Influx DB** is a **time-series and large-scale database** specifically designed to handle high-frequency data logging, making it a best choice for storing and retrieving real-time channel data for various devices in palm oil industry (Zhu, Nie, and Liu, 2023). In this project, channel data includes pressure, current (A), and bar levels. Additionally, Influx DB is an **SQL-like query language**, which means it does not need deep study and easier to understand by non-tech people, making it a suitable choice for Nova Flow as they are more professional on automation industry (Naqvi, Yfantidou, and Zimanyi, 2017). As a result, Influx DB ensures that large volumes of time-stamped data are stored efficiently while allowing for fast query execution.

On the other hand, there are **some limitations on traditional relational databases** like **MySQL and PostgreSQL** when handling big time-series data, channel data in this case. Those traditional databases are mainly designed for structured and transactional data rather than continuous data streams. All the channel data need to update on Influx DB every 5 seconds. Hence, when the data volume increases significantly, performing queries on historical time-series data on traditional databases can lead to **high latency and slower performance** (Tahmassebpour, 2017). Additionally, relational databases store data in normalized tables to reduce redundancy, increasing the query complexity when retrieving historical data. In contrast, Influx DB use efficient **columnar storage with its compression algorithm** to store large and massive datasets more efficiently (Zhu, Nie, and Liu, 2023). Thus, it is proven that Novaflow has decided to use **Influx DB** for real-time monitoring and data management in the palm oil mill system.

2.8.2 Firebase

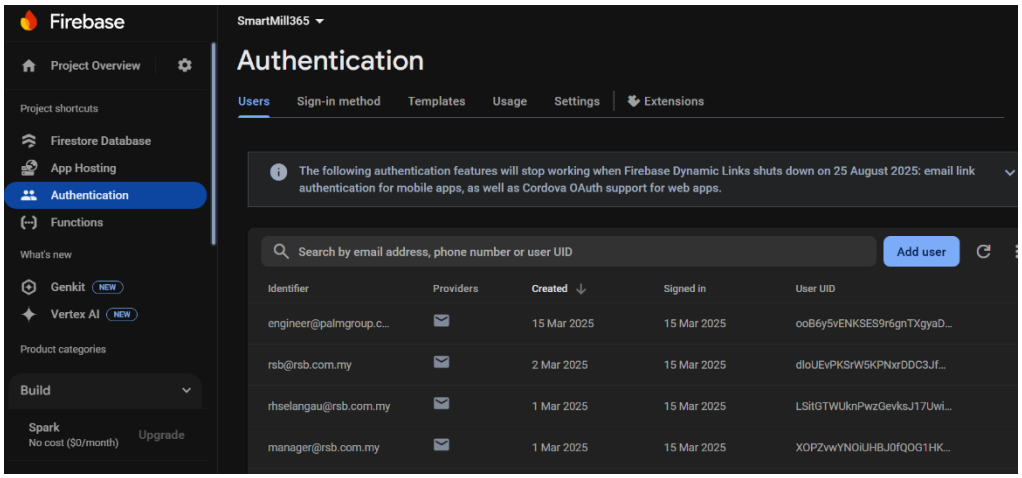


Figure 2.13: User Authentication on Firebase.

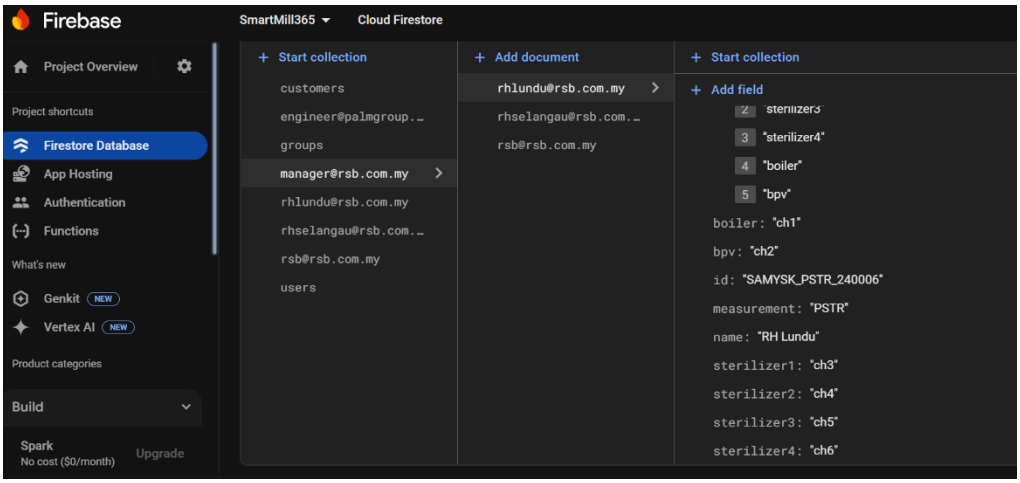


Figure 2.14: Mapping of Devices' Channel Data.

Secondly, **Firebase** is integrated into the monitoring system to manager user authentication and facilitate mapping between users, devices, with their corresponding channel data in Influx DB. Firebase, developed by Google, is a Backend-as-a-Service (BaaS) platform designed to **handle large amount of unstructured data**, which relational database management system (RDBMS) still unable to manager it effectively (Khawas and Shah, 2018). It also offering functions such as authentication, cloud storage and real-time databases. In this project, Firebase is utilized for **secure user authentication** as shown in Figure 2.12, ensuring that only authorized users can access specific device data. In this case, it prevents any unauthorized access of users to sensitive operational data. Additionally, Firebase plays an important role in **managing mapping of devices channels** within Influx DB as shown in Figure 2.13, allowing the

system to associate user accounts with specific devices and their respective real-time channel. This structured mapping ensures correct access to relevant channel data without requiring complex database queries from users.

By leveraging the capabilities of both databases, this project achieves a well-structured and secure data management system. The combination of Influx DB's time-series data handling and Firebase's authentication and data mapping capabilities ensure real-time monitoring, efficient data retrieval, and enhance security. Since **Firebase is still offering free services**, it can be integrated into application development without additional costs, making it a best choice for Novaflow to implement secure authentication and manage device-channel mappings efficiently in a simple way. This integration not only optimizes the performance of Novaflow monitoring system but also provides users with reliable experience when accessing and analysing operational data.

2.9 Summary

In short, this chapter has reviewed the essential components required to develop a palm oil mill monitoring system. The review covered the nature of POM processes, cross-platform mobile development frameworks, SVG graphics for real-time visualization, alarm notification systems, system usability evaluation methods, and database selection. These findings will then form a strong foundation for Chapter 3, where the chosen methodologies, development tools, and implementation strategies will be discussed in detail to demonstrate how the proposed system will be effectively implemented.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter discusses the software development methodology and project planning for this palm oil mill project. The chapter will include the software development methodology, work breakdown structure (WBS), Gann chart and discussion on the development tools that used for developing the monitoring system.

3.2 Software Development Methodology

In this project, **Evolutionary Prototyping Model** is chosen as the software development methodology because it supports continuous improvement, which is crucial when working with real-world industrial project. This approach allows for continuous refinement of the prototype through multiple iterations, ensuring that the final system is aligned with company needs and their operational requirements.

3.2.1 Evolutionary Prototyping Model

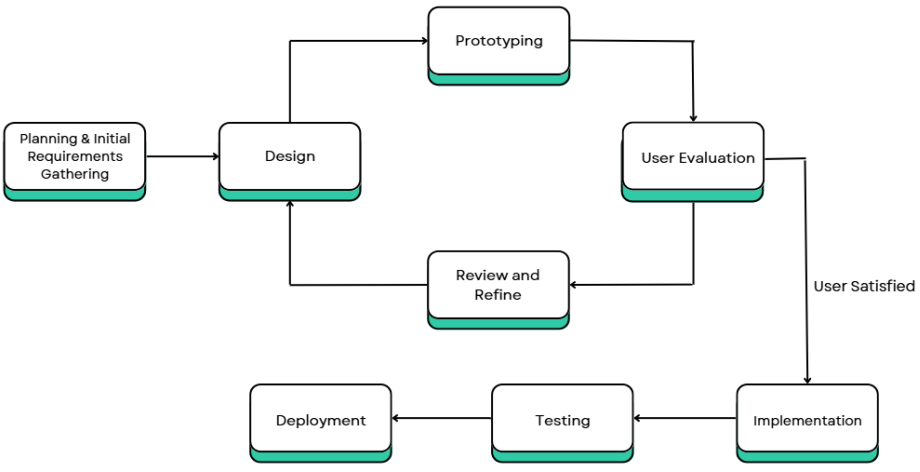


Figure 3.1: Overview of Evolutionary Prototyping Model.

The software development process for the palm oil mill monitoring system suits the evolutionary prototyping model as shown in Figure 3.1, which

emphasizes continuous refinement through iterative feedback. The development process begins with **planning and initial requirements gathering** for building the prototype, where essential system requirements and needs are identified through discussion and analysis. The **first prototype** is then submitted to the company supervisor for evaluation to obtain feedback, ensuring that the system's development is heading in the right direction. The development of **subsequent prototypes** with added functionalities and refinements are submitted again for further evaluation to ensure that the system aligns with user expectations and operational requirements. In this process, **company supervisor** may discover opportunities for additional features and request developer to add in those features. The prototype design will then be updated and modified based on the feedback received.

When the company is satisfied with the **refined prototype**, the development process moves into the **implementation phase**, where the code is developed based on **approved prototype**. The development process will then continue to other phase known as **testing** after all required functionalities and modules are developed. It ensures that all components functioning well, and no critical bugs or performance issues are found. Finally, the system will be **deployed** once it passes all the testing stages.

This prototyping model is appropriate for projects where not all requirements are clearly defined at the beginning stage. It allows developers to start by identifying the components they understand, focusing on specific parts of the system in each iteration rather than developing the entire system at once. As a result, this method helps reduce project risks as it avoids implementing features that are not understood.

A. Planning and Initial Requirement Gathering

The first phase of this prototyping model aims to identify the project objectives, scope and schedule. Initial requirements are collected through the analysis of existing palm oil mill monitoring system used by Novaflow and discussions with the company supervisor. Reviewing on existing systems can help identify the essential features that should be included, while the company supervisor provides insights on any additional features needed.

The following step is formulating a project plan outlining the necessary tasks needed to finish in the project. For example, WBS and Gantt chart help in planning the project. The WBS breaks down the project into smaller part, while the Gann chart visually displays those tasks with its timeline, showing the start and end dates for proper scheduling.

B. Design

In design phase, Unified Modelling Language (UML) diagrams are developed to provide a clear visualization of the palm oil mill monitoring system. These diagrams help the company understand how the system function. A use case diagram is created to represent overall system functionality, while detailed use case descriptions are written for each specific feature.

Additionally, an operational flow diagram is used to illustrate the step-by-step process of how data and actions flow through the system. This diagram provides a visual overview of how the system components interact, how task are processes, and how users engage with the system in real time. All these design elements are based on the requirements gathered during the initial planning phase.

C. Prototyping

During the prototyping phase, an initial version of the palm oil mill monitoring system is developed based on the gathered requirements. This prototype with limited functionality, allows company to visualize and interact with the system early in development. In this case, company can explore the prototype with better understanding on how the user interface might look like instead of relying on the written descriptions. This method provides valuable feedback for guiding on further development.

D. User Evaluation

The prototype developed in previous phase is presented to company for evaluation. Feedback and comments from the company are gathered and documented during this process. This evaluation is important as it helps developers to identify the requirements that not met and areas for

improvement. In this case, it ensures that the system aligns with the company needs and expectations.

E. Review and Refine

Company feedback collected during the evaluation phase is analysed to improve and refine the protype. Iteration will be repeated until the company is fully satisfied, and all the system requirements are met. After the prototype is approved, it serves as the foundation for developing the complete system in coming stage.

F. Implementation

The actual system is developed based on the design from prototype. All functionalities and modules are coded to reflect the refined prototype.

G. Testing

The testing will be conducted after the development of the system is completed. This includes unit testing, integration testing, system testing and user acceptance testing. The goal mainly is to identify and resolve bugs, ensure the system meets technical and design specifications.

H. Deployment

The system is prepared and ready for development after passes all testing phases. The finalized system is then launched and made operational for actual use.

3.3 Project Plan

3.3.1 Work Breakdown Structure (WBS)

1.0 Project Planning and Initial Requirements Gathering

- 1.1 Preliminary planning
  - 1.1.1 Study background of the project problem
  - 1.1.2 Define problem statements
  - 1.1.3 Define project objectives
  - 1.1.4 Define project proposed solution
  - 1.1.5 Define project proposed approach
  - 1.1.6 Define project scope
- 1.2 Literature review
  - 1.2.1 Review palm oil mill processes
  - 1.2.2 Review existing similar palm oil mill monitoring system
  - 1.2.3 Review cross platform framework
  - 1.2.4 Review SVG graphic
  - 1.2.5 Review alarm notification system
  - 1.2.6 Review system usability test
  - 1.2.7 Review databases
- 1.3 Methodology and work plan
  - 1.3.1 Identify suitable software development methodology
  - 1.3.2 Determine work plan
    - 1.3.2.1 Create a work breakdown structure (WBS)
    - 1.3.2.2 Create a Gantt Chart
  - 1.3.3 Identify development tools
- 1.4 Project and Design Specification
  - 1.4.1 Requirement specification
    - 1.4.1.1 Identify functional requirements
    - 1.4.1.2 Identify non-functional requirements
  - 1.4.2 Create UML diagram
    - 1.4.2.1 Develop a use case diagram
    - 1.4.2.2 Define use case descriptions

2.0 Design and Prototyping

2.1 First Iteration

2.1.1 Design user interface

2.1.2 Build low-fidelity prototype

2.1.2.1 Develop a prototype for Android-based monitoring system

2.1.2.2 Develop a prototype for iOS-based monitoring system

2.1.3 Evaluation and gathering feedback

2.1.4 Refine prototype

2.2 Second Iteration

2.2.1 Design

2.2.1.1 System architecture design

2.2.1.2 Database design

2.2.2 Prototyping

2.2.2.1 Develop essential features

2.2.2.2 Create database

2.2.3 Evaluation and gathering feedback

2.2.4 Refine prototype

2.3 Third Iteration

2.3.1 Functionality design

2.3.2 Mobile application prototyping

2.3.2.1 User Authentication and Authorization

2.3.2.2 User management

2.3.2.3 Role/Access management

2.3.2.4 Palm oil channel management

2.3.2.5 Database management

2.3.2.6 Dashboard

2.3.3 Evaluation and gathering feedback

2.3.4 Refine prototype

3.0 Implementation

3.1 Backend development

3.1.1 API development

3.1.2 Database integration

- 3.2 Frontend development
  - 3.2.1 Android platform
  - 3.2.2 iOS platform
- 3.3 Alarm and notification system integration

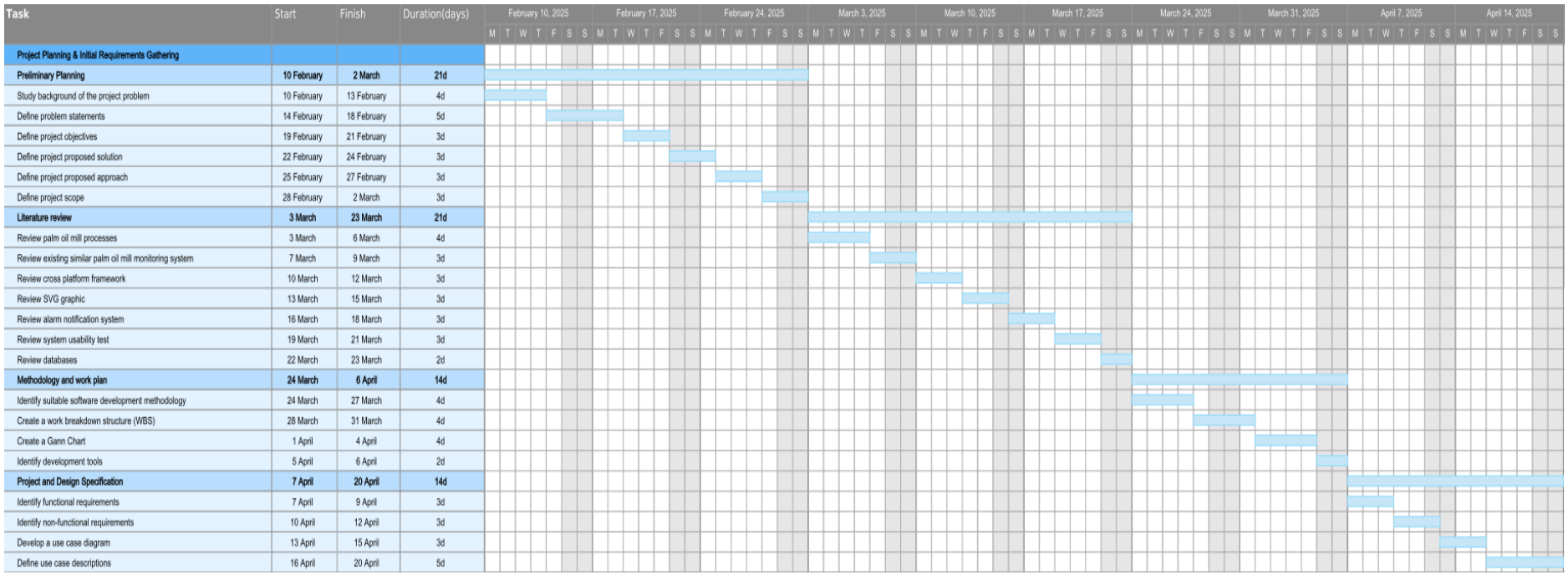
4.0 Testing

- 4.1 Unit testing
- 4.2 Usability testing
- 4.3 Alpha testing
- 4.4 Beta testing

5.0 Deployment

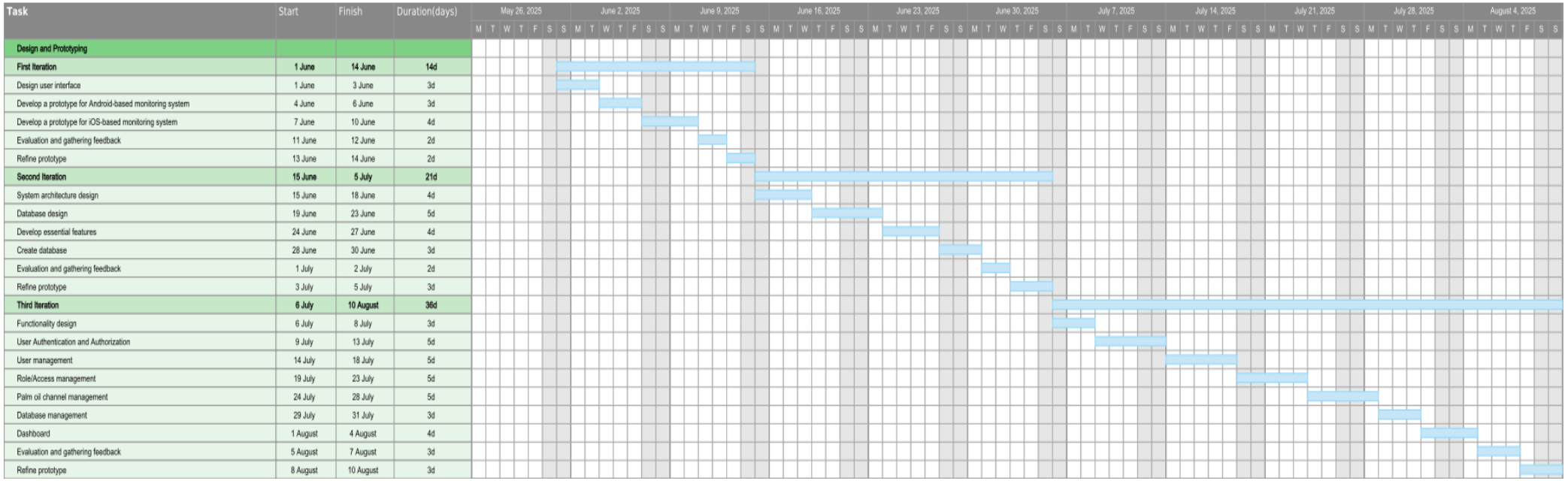
- 5.1 System Deployment
  - 5.1.1 Prepare Android APK for deployment
  - 5.1.2 Deploy Android APK
  - 5.1.3 Prepare iOS app for deployment
  - 5.1.4 Deploy iOS app to Apple App Store

3.3.2 Gantt Chart



Task	Start	Finish	Duration(days)
Project Planning & Initial Requirements Gathering			
Preliminary Planning	**10 February**	**2 March**	**21d**
Study background of the project problem	10 February	13 February	4d
Define problem statements	14 February	18 February	5d
Define project objectives	19 February	21 February	3d
Define project proposed solution	22 February	24 February	3d
Define project proposed approach	25 February	27 February	3d
Define project scope	28 February	2 March	3d
Literature review	**3 March**	**23 March**	**21d**
Review palm oil mill processes	3 March	6 March	4d
Review existing similar palm oil mill monitoring system	7 March	9 March	3d
Review cross platform framework	10 March	12 March	3d
Review SVG graphic	13 March	15 March	3d
Review alarm notification system	16 March	18 March	3d
Review system usability test	19 March	21 March	3d
Review databases	22 March	23 March	2d
Methodology and work plan	**24 March**	**6 April**	**14d**
Identify suitable software development methodology	24 March	27 March	4d
Create a work breakdown structure (WBS)	28 March	31 March	4d
Create a Gann Chart	1 April	4 April	4d
Identify development tools	5 April	6 April	2d
Project and Design Specification	**7 April**	**20 April**	**14d**
Identify functional requirements	7 April	9 April	3d
Identify non-functional requirements	10 April	12 April	3d
Develop a use case diagram	13 April	15 April	3d
Define use case descriptions	16 April	20 April	5d

Figure 3.2: Gantt Chart for Project Planning & Initial Requirement Gathering.



Task	Start	Finish	Duration(days)
Design and Prototyping			
First Iteration	**1 June**	**14 June**	**14d**
Design user interface	1 June	3 June	3d
Develop a prototype for Android-based monitoring system	4 June	6 June	3d
Develop a prototype for iOS-based monitoring system	7 June	10 June	4d
Evaluation and gathering feedback	11 June	12 June	2d
Refine prototype	13 June	14 June	2d
Second Iteration	**15 June**	**5 July**	**21d**
System architecture design	15 June	18 June	4d
Database design	19 June	23 June	5d
Develop essential features	24 June	27 June	4d
Create database	28 June	30 June	3d
Evaluation and gathering feedback	1 July	2 July	2d
Refine prototype	3 July	5 July	3d
Third Iteration	**6 July**	**10 August**	**36d**
Functionality design	6 July	8 July	3d
User Authentication and Authorization	9 July	13 July	5d
User management	14 July	18 July	5d
Role/Access management	19 July	23 July	5d
Palm oil channel management	24 July	28 July	5d
Database management	29 July	31 July	3d
Dashboard	1 August	4 August	4d
Evaluation and gathering feedback	5 August	7 August	3d
Refine prototype	8 August	10 August	3d

Figure 3.3: Gantt Chart for Design and Prototyping.

Task	Start	Finish	Duration(days)
Implementation			
Backend development	**11 August**	**20 August**	**10d**
API development	11 August	15 August	5d
Database integration	16 August	20 August	5d
Frontend development	**21 August**	**26 August**	**6d**
Android platform	21 August	23 August	3d
iOS platform	24 August	26 August	3d
Alarm and notification system integration	**27 August**	**31 August**	**5d**

Figure 3.4: Gantt Chart for Implementation.

Task	Start	Finish	Duration(days)
Testing	**1 September**	**8 September**	**8d**
Unit testing	1 September	2 September	2d
Usability testing	3 September	4 September	2d
Alpha testing	5 September	6 September	2d
Beta testing	7 September	8 September	2d
Deployment			
System Deployment	**9 September**	**13 September**	**5d**
Prepare Android APK for deployment	9 September	9 September	1d
Deploy Android APK	10 September	10 September	1d
Prepare iOS app for deployment	11 September	12 September	2d
Deploy iOS app to Apple App Store	13 September	13 September	1d

Figure 3.5: Gantt Chart for Testing and Deployment.

3.4 Development and Deployment Tools

Several tools were used throughout the development and deployment process, each playing an important role in building and managing different parts of the palm oil monitoring system. These tools were carefully selected based on their features during literature review stage. The following subsections are the key tools used and how they supported the development and deployment of the system:

3.4.1 Flutter

Flutter is a free to use UI software development kit that allows developers to build mobile applications using a single codebase. It was chosen for the development of the palm oil mill monitoring system mainly due to its cross-platform feature, which allows the system to run on different devices without the need to rewrite code for both Android and iOS platform. Additionally, Flutter having a rich set of pre-built widgets and flexible UI design tools, making the development process faster and more efficient. These features made Flutter a good choice for creating a consistent, responsive, and user-friendly interface, ensuring smooth functionality and accessibility for both operators and managers across different screen sizes.

3.4.2 Android Studio

Android Studio is the integrated development environment (IDE) for Android development. It offers robust features such as code editing, real time error checking and debugging tools. In this project, Android Studio served as the primary development environment for building and managing the codebase of the palm oil monitoring system.

Besides that, Android Studio also has an Android emulator for testing the application in a simulated Android environment, allowing developers to preview and debug the app from time to time without the need of physical device. Its built-in Git integration enabled direct code pushing to GitHub only with few commands, streamlining the process of version control.

3.4.3 Xcode

Xcode is another integrated development environment (IDE) for iOS application development. Since Android Studio was used as the primary environment for writing the Flutter code, Xcode was required to build, test and run the iOS version of the palm oil mill monitoring system. Xcode also used for compiling and deploying the iOS version of the application during the deployment phase.

3.4.4 Firebase

Firebase is a comprehensive backend-as-a-service platform provided by Google, offering features like authentication and real-time databases. In this project, Firebase is used for user authentication and device settings such as mapping channel data with Influx DB, ensuring secure login management and accurate data retrieval for different channels.

3.4.5 Influx DB

Influx DB is a time series database designed for storing large volumes of time-stamped data. In this project, Influx DB was used to store live channel data collected from the palm oil mill, such as pressure and temperature readings from various devices. By integrating Influx DB into the system, real-time data could be efficiently stored and retrieved without much delay.

3.4.6 MySQL

MySQL is an open-source relational database management system widely used for structured data storage. In this project, MySQL was used to store device threshold settings, alarm settings and maintain alarm history records. This ensured reliable storage, easy retrieval, and long-term access to alarm-related information.

3.4.7 Python

Python is a programming language used in this project for backend processing. It was responsible for comparing threshold values stored in MySQL with the current live readings from Influx DB, triggering alarms when thresholds were

exceeded. Besides that, it also provided APIs to retrieve alarm records from MySQL and update alarm acknowledgment.

3.4.8 TestFlight

TestFlight is an application testing platform by Apple that allows developers to distribute pre-release of iOS version applications to testers. In this project, TestFlight was used to test the iOS version of the palm oil mill monitoring system before deploying to App Store, allowing developers and testers to identify and resolve potential issues on real devices in a controlled environment.

3.4.9 App Store Connect

App Store Connect is Apple's platform for managing and deploying iOS applications. It was used in this project to submit and deploy the iOS version of the monitoring system to the App Store. Through App Store Connect, application metadata, screenshots, and build version were managed effectively, ensuring a smooth publishing process and availability of the app to end users.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

This chapter mainly focuses on the preliminary specifications of the project in view of **functional and non-functional requirements** of the system, followed by use case diagram along with each use case descriptions for the palm oil mill monitoring system. The purpose of this chapter is to define what the system should do and how should it behave so that company have a clear understanding on the system's scope and functionality.

The **use case diagram** gives an overview of the system's functionality by identifying the interactions or relationships between users and the system. Each use case represents the exact tasks that users can perform on the system, while the **use case descriptions** provide a details explanation of how these tasks are carried out in the system.

4.2 Requirement Specification

4.2.1 Functional Requirements

Functional requirements of the mobile application for Palm Oil Mill Monitoring System are outlined in Table 4.1 below:

Table 4.1: Functional Requirements.

ID	Functional Requirement
FR001	The mobile application shall allow the user to login account using a valid email and password.
FR002	The mobile application shall allow the user to view a dashboard displaying all available channels along with their live data.
FR003	The mobile application shall allow the manager to select a subgroup under their company to view the corresponding dashboard.
FR004	The mobile application shall allow the user to view single live channel data in graphical format.

FR005	The mobile application shall allow the user to select a specific time range to view the corresponding past channel data in graphical format.
FR006	The mobile application shall allow the user to view multiple available channel data on a single screen in graphical format.
FR007	The mobile application shall allow the user to view the active alarm that currently exceed or below the predefined thresholds.
FR008	The mobile application shall allow the user to receive an alarm notification when a critical condition is detected.
FR009	The mobile application shall allow the user to view the history of past alarm notifications.

4.2.2 Non-Functional Requirements

Non-functional requirements of the mobile application for Palm Oil Mill Monitoring System are outlined in Table 4.2 below:

.

Table 4.2: Non-Functional Requirements.

ID	Non-Functional Requirement	Category
NFR001	The mobile application shall be compatible with both Android and iOS devices.	Portability
NFR002	The mobile application shall retrieve and display **live** channel data not more than 3 seconds.	Performance
NFR003	The mobile application shall retrieve and display **pass** channel data not more than 5 seconds.	Performance
NFR004	The mobile application's response time shall be responsive when the user interacts with the system.	Performance
NFR005	The mobile application shall validate user input and prevent incorrect input formats by	Security

	displaying error messages to guide the user.	
NFR006	The mobile application's interface shall be easy to use, easy to navigate, and easy to understand by the user.	Reliability

4.2.3 Use Case Diagram

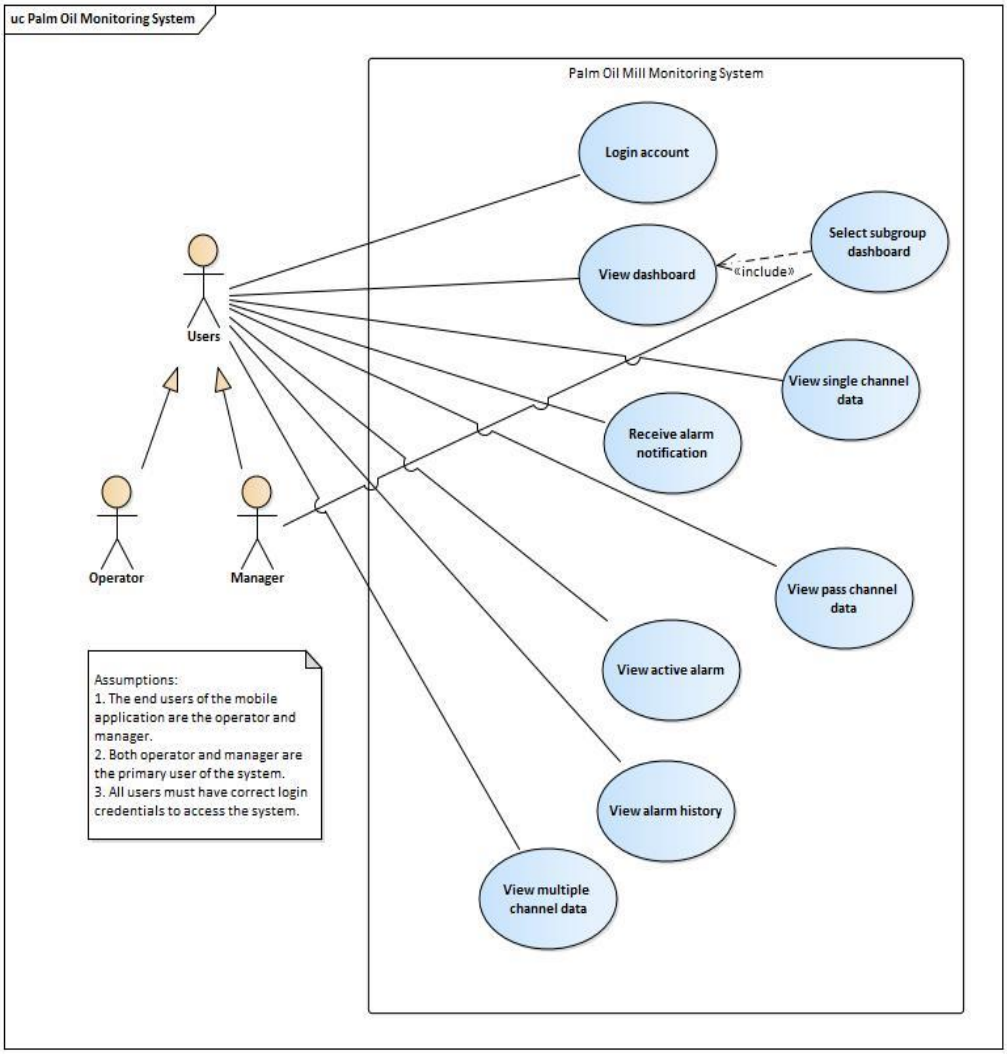


Figure 4.1: Use Case Diagram for Palm Oil Mill Monitoring System.

4.2.4 Use Case Description

<table>
<tr><td colspan="2">Use Case Name: Login account</td><td>ID: FR001</td><td>Importance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: Users</td><td colspan="2">Use Case Type: Detail, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
N/A</td></tr>
<tr><td colspan="4">Brief Description:
This use case describes the process of user login to the system to monitor the palm oil mill processes.</td></tr>
<tr><td colspan="4">Trigger: The user wants to log in to the system to access the palm oil mill monitoring mobile application.</td></tr>
<tr><td colspan="4">Relationships:
Association : Users
Include : N/A
Extend : N/A
Generalization: N/A</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The user accesses the system which shows a login screen and allows the user to enter email and password.
2. The user filled in the email and password.
3. The system retrieves the user information from the user account database in the firebase.
4. The system verifies the username and password. Perform 4.1 or 4.2
4.1 The username and password are correct. Continue to 5.
4.2 The username and password are incorrect. Continue to 6.
5. The user enters the system. The use case ended.
6. The system will prompt out a message to let the user renter the username or password. Continue to 4.</td></tr>
<tr><td colspan="4">Sub-flows: N/A</td></tr>
<tr><td colspan="4">Alternate/Exceptional Flows: N/A</td></tr>
</table>

Use Case Name: View Dashboard	ID: FR002	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of viewing a dashboard that displays all available channels along with their live data.		
Trigger: The user wants to monitor the available live channel data through dashboard.		
Relationships: Association : Users Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The user selects the "Dashboard" option from the bottom navigation. 2. The system queries the server to retrieve the list of all available channels. 3. The system retrieves the most recent live data for each channel. 4. The system displays the channel list along with its live data and presents in a well-organized dashboard. 5. The dashboard will automatically refresh the latest live data every 5 seconds.		
Sub-flows: N/A		
Alternate/Exceptional Flows: 1. Users must have valid login to access the system.		

Use Case Name: Select subgroup dashboard	ID: FR003	Importance Level: High
Primary Actor: Manager	Use Case Type: Detail, Essential	

Stakeholders and Interests:

N/A

Brief Description:

This use case describes the process of a manager selecting a specific subgroup dashboard and displays real-time data related to the selected subgroup.

Trigger:

The manager wants to view real-time data and monitor the performance of a specific subgroup.

Relationships:

Association : N/A

Include : View dashboard

Extend : N/A

Generalization: N/A

Normal Flow of Events:

1. The system displays a list of available subgroups on the dashboard after manager login.
2. The manager selects a specific subgroup from the list.
3. The system queries the server to retrieve the list of all available channels for the selected subgroup.
4. The system retrieves the most recent live data for each channel.
5. The system displays the channel list along with its live data and presents in a well-organized dashboard.
6. The dashboard will automatically refresh the latest live data every 5 seconds.

Sub-flows: N/A

Alternate/Exceptional Flows:

1. Manager must have valid login to access the system.

Use Case Name: View single channel data	ID: FR004	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of users viewing a single detailed live data for monitoring its performance.		
Trigger: The user wants to view detailed live data of specific channel by tapping on the channel from the dashboard.		
Relationships: Association : Users Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The user navigates to the dashboard page. 2. The user taps on a specific channel from the available list. 3. The system retrieves and processes the channel's most recent data. 4. The system displays the channel's detailed data in graphical format.		
Sub-flows: N/A		
Alternate/Exceptional Flows: 1. Users must have valid login to access the system.		

Use Case Name: View pass channel data	ID: FR005	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of a user viewing historical data for a specific channel.		
Trigger: A user wants to view historical data for a specific channel to analyse past performance and investigate issues that occurred in the past.		
Relationships: Association : User Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The user navigates to the dashboard page. 2. The user taps on the desired channel from the list to view its data. 3. The system shows an option to select a time range for the past data. 4. The system retrieves the pass data from the database based on the selected time range. 5. The system displays the past data in graphical format.		
Sub-flows: N/A		
Alternate/Exceptional Flows: 1. Users must have valid login to access the system.		

<table>
<tr><td colspan="2">Use Case Name: View multiple channel data</td><td>ID: FR006</td><td>Importance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: Users</td><td colspan="2">Use Case Type: Detail, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
N/A</td></tr>
<tr><td colspan="4">Brief Description:
This use case describes the process of a user viewing the live data for multiple available channels on a single page for analysis purposes.</td></tr>
<tr><td colspan="4">Trigger:
The user wants to view live data for multiple channels on one page to track overall channel performance and identify any issues or trends.</td></tr>
<tr><td colspan="4">Relationships:
Association : Users
Include : N/A
Extend : N/A
Generalization: N/A</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The user navigates to the dashboard page.
2. The user clicks the “View Graphs” button on the top of the dashboard page.
3. The users select the channel according to their preferences through Filter option.
4. The system retrieves the most recent data for all selected channels.
5. The system processes and organizes the data for each channel.
6. The system displays the data for selected channels on one page.
7. The users can choose either view the selected graphs separately or combine together.</td></tr>
<tr><td colspan="4">Sub-flows: N/A</td></tr>
<tr><td colspan="4">Alternate/Exceptional Flows:
1. Users must have valid login to access the system.</td></tr>
</table>

Use Case Name: View active alarm	ID: FR007	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of a user viewing the active alarm record triggered by available channels.		
Trigger: The user wants to view the active alarm that currently exceed or below the predefined thresholds.		
Relationships: Association : N/A Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The user navigates to the active alarm page through the “Alarm” icon on the header. 2. The system retrieves the current active alarm records for all available channels from the database. 3. The system displays the active alarm history in a list format with its alarm code, alarm triggered time and description.		
Sub-flows: N/A		
Alternate/Exceptional Flows: 1. Users must have valid login to access the system.		

Use Case Name: Receive alarm notification	ID: FR008	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	

Stakeholders and Interests:

N/A

Brief Description:

This use case describes the process of a user receiving alarm notifications when a monitored channel exceeds predefined threshold values.

Trigger:

A monitored channel detects a condition that exceeds the predefined threshold, triggering an alarm.

Relationships:

Association : Users

Include : N/A

Extend : N/A

Generalization: N/A

Normal Flow of Events:

1. The system continuously monitors all channel data in real-time.
2. A channel's data exceeds the predefined threshold stored in the database.
3. The system generates an alarm event based on the configured rules.
4. The system immediately sends an alarm notification to the user via push notification.
5. The user receives an alarm notification through the system.

Sub-flows: N/A

Alternate/Exceptional Flows:

1. Users must have valid login to access the system.

Use Case Name: View alarm history	ID: FR009	Importance Level: High
Primary Actor: Users	Use Case Type: Detail, Essential	

Stakeholders and Interests:

N/A

Brief Description:

This use case describes the process of a user viewing the historical alarm events triggered by available channels.

Trigger:

The user wants to review previous alarm notifications and investigate past incidents.

Relationships:

Association : Member

Include : N/A

Extend : N/A

Generalization: N/A

Normal Flow of Events:

1. The user selects the “Alarm” option from the bottom navigation.
2. The user can filter the alarm history based on criteria such as time range and date.
3. The system retrieves the alarm history from the database based on the filter criteria.
4. The system displays the alarm history in a list format with its alarm code, alarm triggered time, description and acknowledgement.
5. The user can acknowledge the particular alarm record by clicking the Ack button. Once Ack button is pressed, it will turn into green colour.

Sub-flows: N/A

Alternate/Exceptional Flows:

1. Users must have valid login to access the system.

4.3 Low-fidelity Prototypes

Low-fidelity prototypes with **user interface (UI) representations** were developed for the **mobile application** of the palm oil mill monitoring system. The prototype includes key interfaces such as login page, dashboard page, channel data monitoring page, active alarm page, alarm history page, and welcoming page. The purpose of creating this prototype was to provide ideas for visualizing a palm oil monitoring system and illustrate how it would look and work. It served as reference point for both design and functionality, allowing the company supervisor to make any changes or improvements when necessary. In addition, feedback will be collected after the low-fidelity prototype is presented to the company supervisor, which will be used for further refinement. The low-fidelity prototype is shown in **Appendix A**.

CHAPTER 5

SOLUTION

5.1 Introduction

This chapter presents the solutions that implemented in the system to solve the problems mentioned in Chapter 1. The system design has progressed from a low-fidelity prototype to a high-fidelity interface, with the implementation on usability and functionality. Each problem identified in the Chapter 1 is resolved through the modules and features developed in this project. This solution not only addresses the highlighted issues but also serves as the actual implementation adopted by the company in their daily operations.

5.2 Problem-Solution Mapping

5.2.1 Limited Android Access and Discontinuation of iOS Subscription

The previous company application was constrained to limited Android access and the discontinuation of the iOS subscription, where managers and operators were unable to monitor the palm oil mill operation through the mobile app. With the implementation of this proposed system, both Android and iOS platforms are supported, allowing broader accessibility for all users.

The application is developed using a cross-platform framework, Flutter, which ensures the updates and new features are released simultaneously for both Android and iOS. As a result, users are no longer restricted by their device type. For example, manager using iPhones and operators using Android phones can both access the same system without compatibility issues. In addition, the app also supports iPads and Android tablets, providing a larger screen option for monitoring operations. This is useful since long time viewing of graphs on smaller phone screens can cause eye strain. To verify this cross-platform accessibility, screenshots of the app icons on real iOS and Android phones are provided as proof of successful deployment as shown in Figure 5.1 and Figure 5.2. The compatibility on iPads and Android tablets are shown in Figure 5.3 and Figure 5.4.

Figure 5.2: SmartMill365 on iOS.

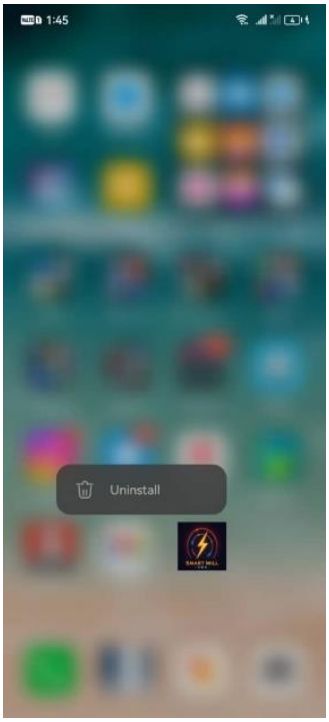


Figure 5.1: SmartMill365 on Android.

Figure 5.3: SmartMill365 on iPads.

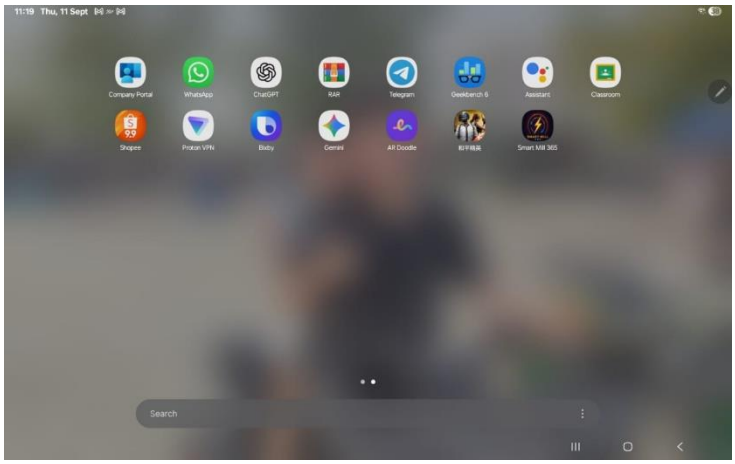


Figure 5.4: SmartMill365 on Android Tablet.

5.2.2 Limited Visual Representation of Real-Time Processes

The earlier application lack of intuitive graphical interface for monitoring, as real-time processes were only represented in raw data form without a graphical layout dashboard. Thus, this system introduces a **high-fidelity SVG-based dashboard** as shown in Figure 5.5 and Figure 5.6 that dynamically displays device information, including device name, live data from Influx DB, and its unit from Firebase. Each device is represented by an icon with real-time status indicators where green for online and red for offline. For disconnected devices, it will show "**Offline"** text instead of device information. This visualization improves situational awareness and makes monitoring more intuitive and efficient.

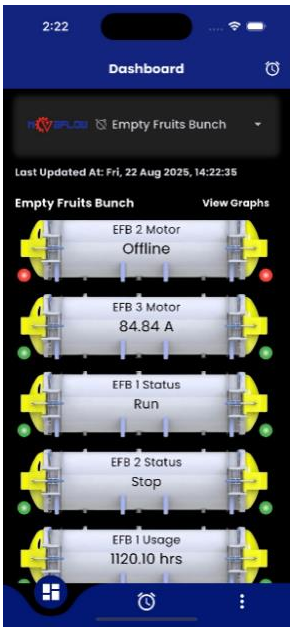


Figure 5.5: Graphical Layout Dashboard on iOS.

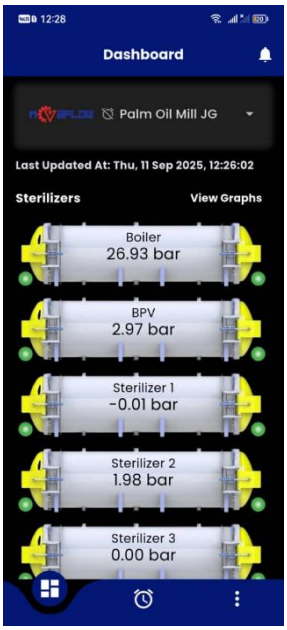


Figure 5.6: Graphical Layout Dashboard on Android.

As this project is currently developed under **Phase 1** for the company, the current dashboard design serves as a **temporary implementation** to validate system feasibility. A more refined and suitable design will be considered for **future improvement**, as the final graphical layout has not yet been finalized by the company. Additionally, users can click on the device icon to view its corresponding live graph data, enabling deeper insights into real-time performance.

5.2.3 Delayed Response to Critical Conditions

Previously, palm oil mills relied on operator physically checking devices on site, which increased the risk of delayed responses to sudden critical conditions that could cause production downtime or equipment damage. Therefore, this system implements a comprehensive alarm notification system consisting of two main components which are **Active Alarm page** and **Alarm History page**. The **Active Alarm page** displays real-time records of devices that currently exceed of fall below predefined thresholds, where the record automatically cleared once values return to normal. Each subgroup has its own active alarm records, presented in a table with key information including the alarm code, trigger time, and a dynamically generated description as shown in Figure 5.7 and Figure 5.8 below. The threshold values and alarm description for each device are set by the Novaflow IT team based on client or user specifications, with a detailed explanation in Chapter 6 later on.

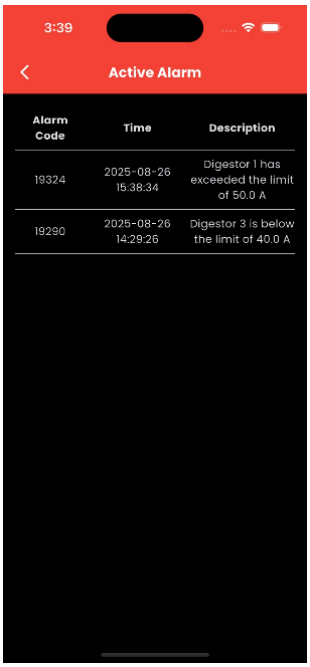


Figure 5.8: Active Alarm Page on iOS.

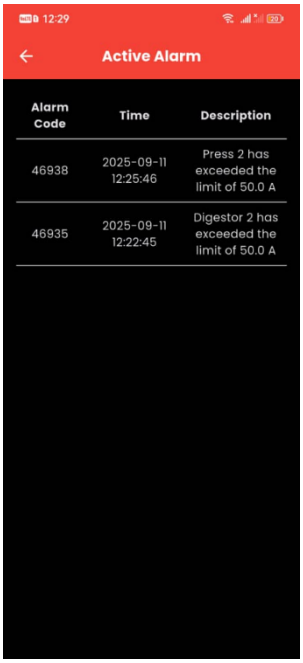


Figure 5.7: Active Alarm Page on Android.

The **Alarm History page** show past alarm records for the selected subgroup, presented in a table form with four columns including alarm code, alarm trigger time, description and an Acknowledge (Ack) button as shown in Figure 5.9 and Figure 5.10. When a user acknowledges an alarm, the system logs their email, ensuring the corrective action has been taken while also providing a clear audit trail for future reference. Once an alarm is acknowledged, the Ack button **turns green** and can only be clicked once per record. To support usability in sites with many devices which might contain large volumes of alarm records, the system also offers filtering options **by specific date or by days** (1, 7, 30) as shown in Figure 5.11 to Figure 5.14.

Figure 5.10: Alarm History Page on iOS.

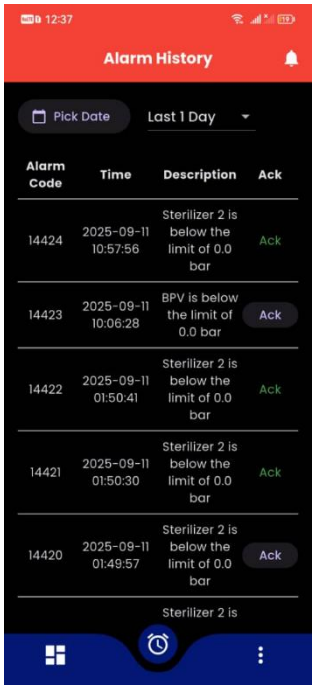


Figure 5.9: Alarm History Page on Android.

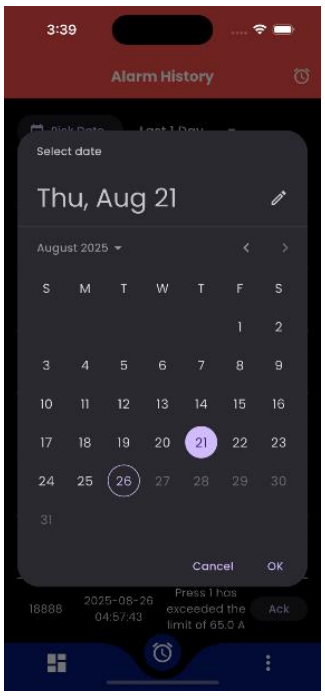


Figure 5.12: Alarm Record Filtered by Date on iOS.

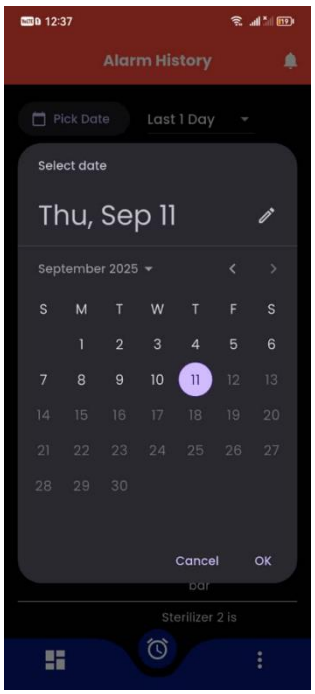


Figure 5.11: Alarm Record Filtered by Date on Android.

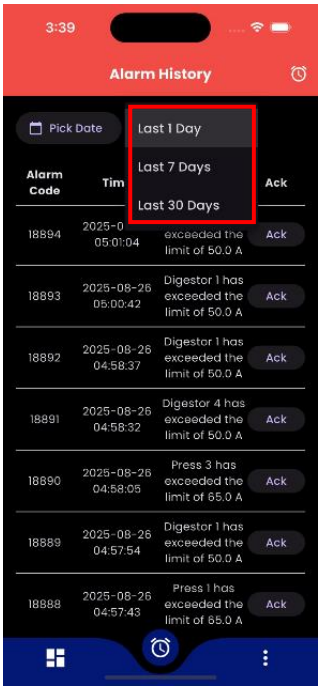


Figure 5.13: Alarm Record Filtered by Days on iOS.

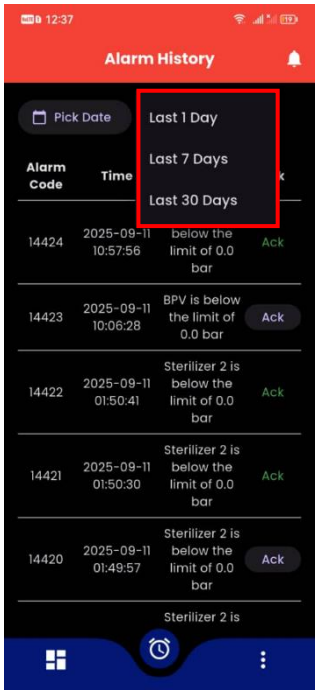


Figure 5.14: Alarm Record Filtered by Days on Android.

Additionally, the system provides **real-time pop-up notifications** as shown in Figure 5.15 and Figure 5.16 below whenever a new active alarm record is triggered, ensuring managers or operators are immediately aware of the critical conditions without needing to continuously monitor the Active Alarm page. On the Dashboard page, there are also having a **small active alarm icon** shown in Figure 5.17 and Figure 5.18 that indicates the status of active alarms for that particular subgroup. A red alarm icon signifies that an active alarm is present, while a grey alarm icon indicates currently no active alarms. This feature is especially useful for company that managing multiple subgroups, as it eliminates the need for managers or operators to manually switch between each subgroup's Active Alarm page to check for issues.

Figure 5.16: Pop-Up Notification on iOS.

Figure 5.15: Pop-Up Notification on Android.

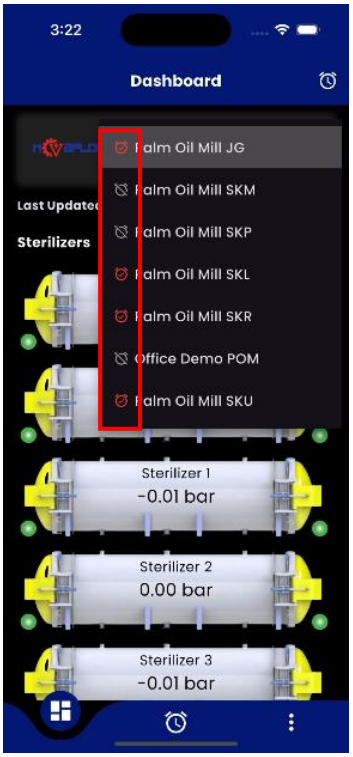


Figure 5.18: Active Alarm Icon on iOS.

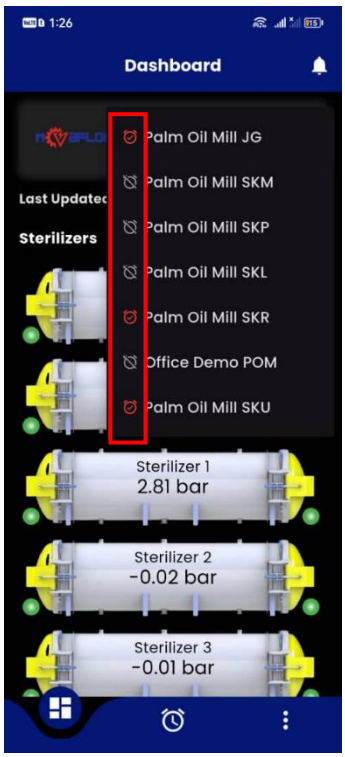


Figure 5.17: Active Alarm Icon on Android.

5.2.4 Limited Usability and Interface Constraints

In previous monitoring system has some usability constraints where user can only view one channel's data at a time, which cause real-time monitoring and analysis inefficient, especially in critical conditions. Furthermore, managers that have multiple subgroups face difficulty in monitoring since they must log in and out of different accounts to access each subgroup's dashboards and device data, which cause time-consuming and impractical.

To overcome the issues mentioned above, the new system introduces two major improvements on usability. First, the graph module allows users to view channel data in two ways which are Separate Graphs and Combine Graphs. **Separate Graphs view** shown in Figure 5.19 and Figure 5.20 allow multiple graphs can be displayed simultaneously, enable users to analyse each channel individually. Meanwhile, the **Combine Graphs view** shown in Figure 5.21 and Figure 5.22 allow selected channel data displayed in a single graph for easy comparison and analysis. Additionally, users can filter the devices they wish to monitor and adjust the time range (Last 3, 6, 12 hours) as shown in Figure 5.23 to Figure 5.26 according to their objectives or preferences.

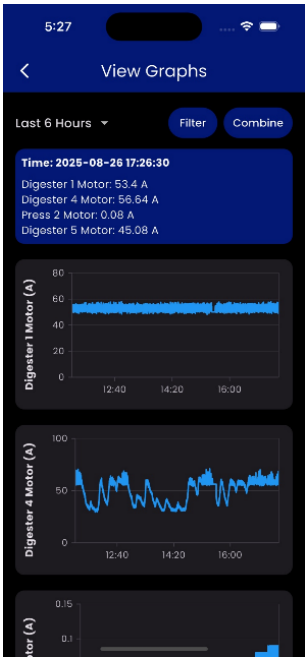


Figure 5.20: Separate Graphs View on iOS.

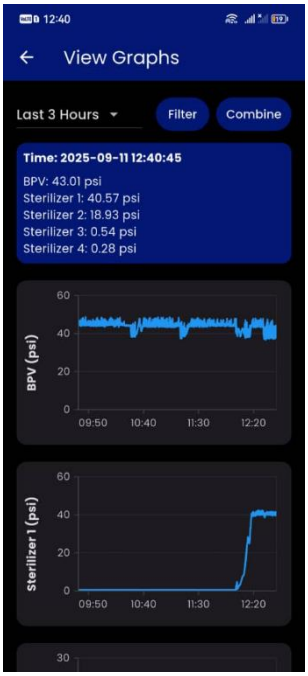


Figure 5.19: Separate Graph View on Android.

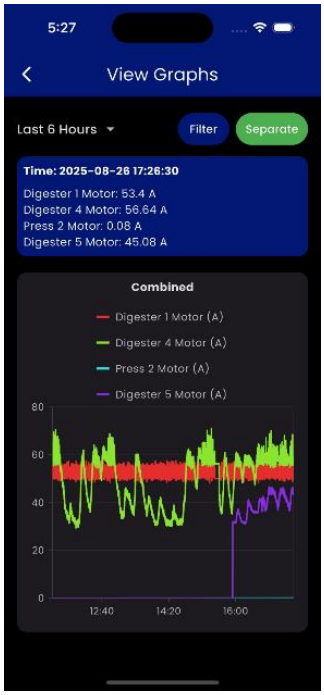


Figure 5.21: Combine Graphs View on iOS.

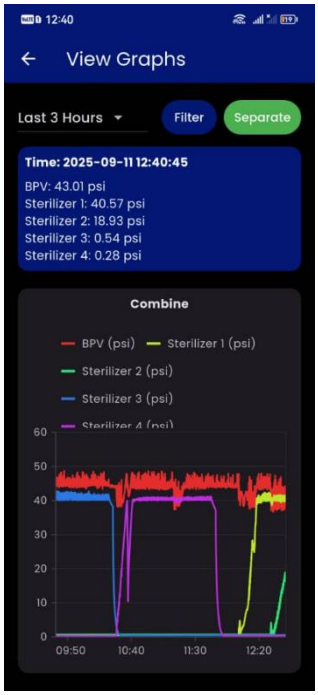


Figure 5.22: Combine Graphs View on Android.

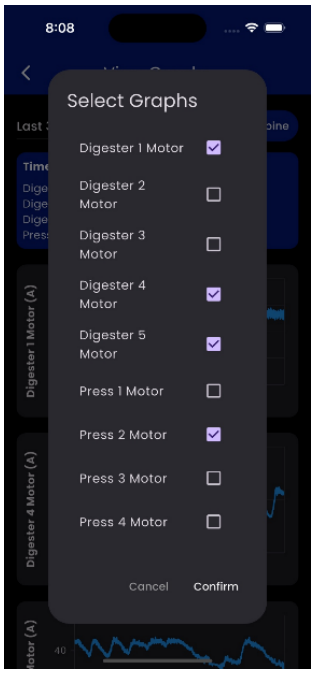


Figure 5.23: Graphs Filtered by Devices on iOS.

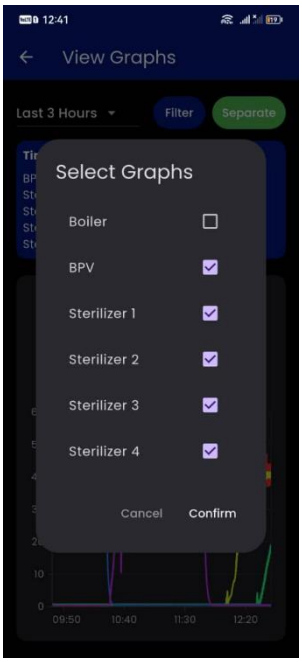


Figure 5.24: Graphs Filtered by Devices on Android.

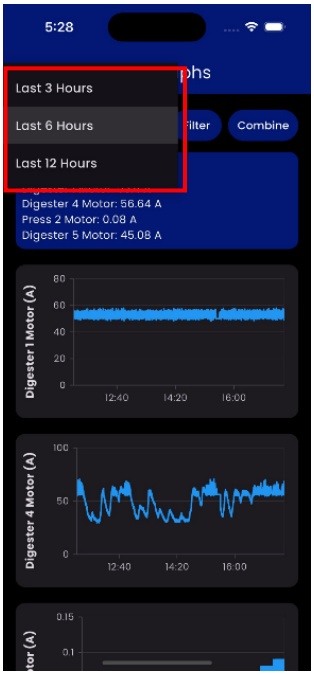


Figure 5.26: Graphs Filtered by Time Range on iOS.

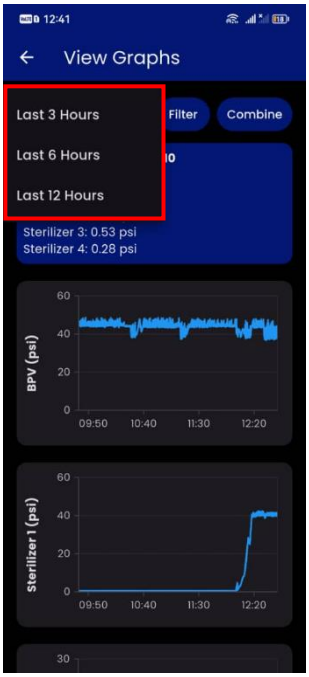


Figure 5.25: Graphs Filtered by Time Range on Android.

Second, managers with multiple subgroups can now seamlessly **switch between subgroups dashboards without repeated logins** as shown in Figure 5.27 and Figure 5.28, as all subgroups are configured under a single account. Once a subgroup is selected on the dashboard, all components including device data, active alarms and alarm history update simultaneously, ensuring smooth monitoring across multiple sites.

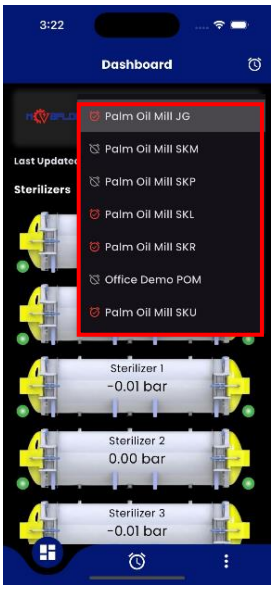


Figure 5.28: Switching between Subgroups on Single Account for iOS.

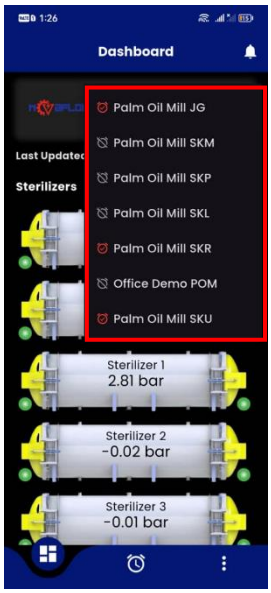


Figure 5.27: Switching between Subgroups on Single Account for Android.

In addition to Separate Graphs and Combine Graphs view, the system also provides **Single Graph view** as shown in Figure 5.29 and Figure 5.30, which remain the previous implementation where line spot values (tooltip) are displayed on each graph point when the user tap on it. The **tooltip** displays the time of the selected point, device name and the corresponding value. Several improvements have been introduced in this view. The **device unit** is now clearly displayed on graph, making the data more informative and reducing ambiguity. A **time range selection** (3, 6, or 12 hours) is implemented at the top of the graph. These time ranges are specifically designed based on the palm oil mill process, where one complete operational cycle typically take at least 3 hours. This ensures that users can analyse the data in more meaningful ways that algin with the mill's workflow.

From UI perspective, the graph has been expanded to its **maximum size**, giving users a larger and clearer visualization for a better monitoring experience. Additionally, a **blank space** has been reserved beside the Y-axis values. This is specially designed to accommodate devices with screen island layouts, ensuring that the Y-axis values remain visible and not blocked by the island. As many modern devices no matter Apple or Android now adopting this design, the enhancements improve accessibility and delivers a seamless monitoring experience across different screen types.

As Single Graph view serves as the main monitoring tool, it also allows to **zoom in or out** as shown in Figure 5.31 and Figure 5.32 for a closer analysis of the data and **pan** across the timeline for better navigation. When a different time range is selected, the graph resets to its normal size, maintaining consistency in the viewing experience.

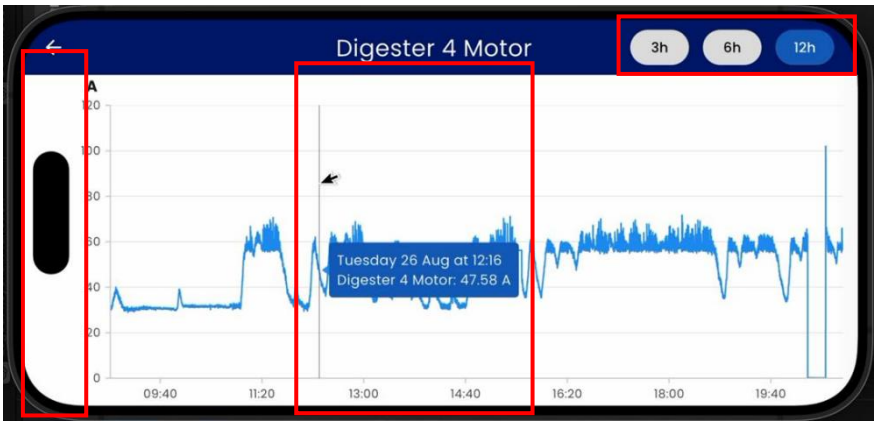


Figure 5.29: Single Graph View on iOS.

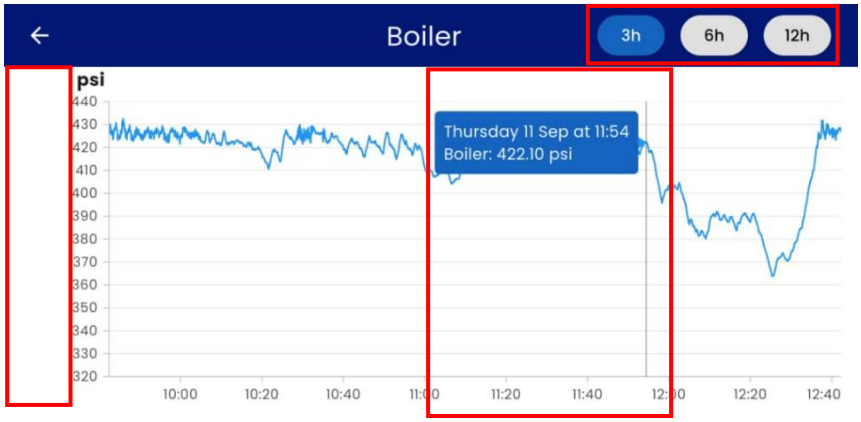


Figure 5.30: Single Graph View on Android.

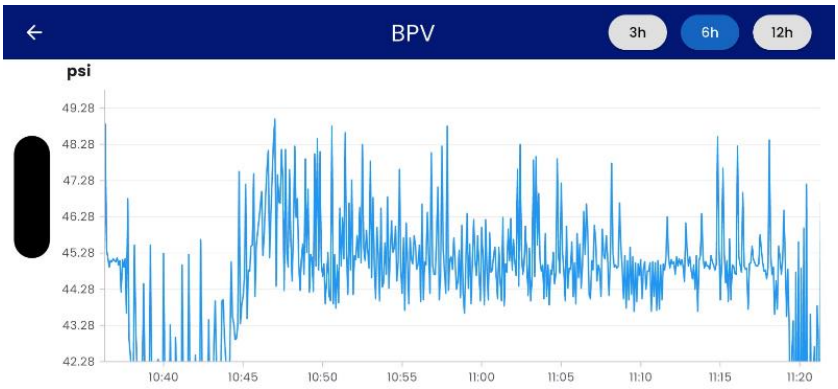


Figure 5.31: Zoom in Graph on iOS.

Figure 5.32: Zoom in Graph on Android.

Another critical usability feature is the implementation of **theme selection** as shown in Figure 5.33 and Figure 5.34. Since managers or operators may monitor the device performance for extended periods of time, **reducing eye strain** is essential. To address this issue, the system allows users to choose

between dark theme and light theme based on their preferences or situational needs. Furthermore, users can set the application theme to automatically follow their device settings, offering an additional option for user convenience. This ensures a consistent visual experience across the system and other applications on the device, whether in light or dark mode.

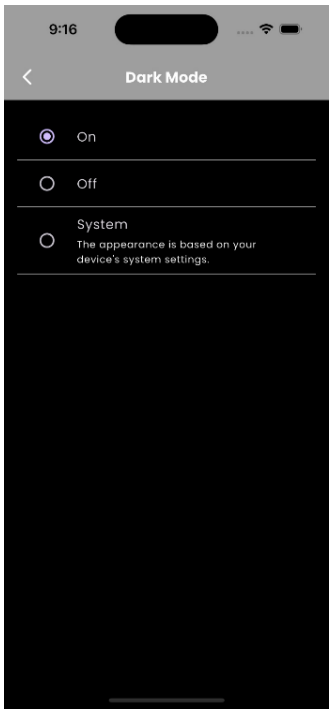


Figure 5.33: Theme Selection in Settings on iOS.

Figure 5.34: Theme Selection in Settings on Android.

Lastly, the More page shown in Figure 5.35 and Figure 5.36 contains several general app functionalities such as **About Us, Frequently Asked Questions (FAQ), Privacy Policy, and Settings**. At the top of this page, the current logged-in user's account email is displayed for clear identification. In addition, the page includes a **Data Plotter** feature as shown in Figure 5.37 and Figure 5.38 requested by the company. It allows users to upload CSV files for plotting graphs and export the generated graphs as PDF files. Users can also **log out** through this page to securely end their session.

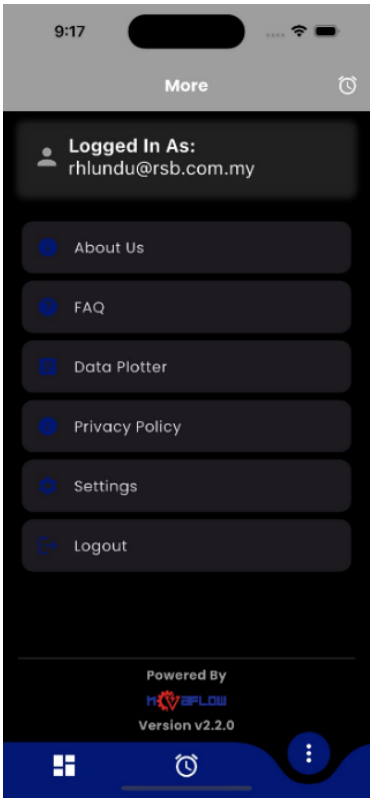


Figure 5.36: More Page on iOS.

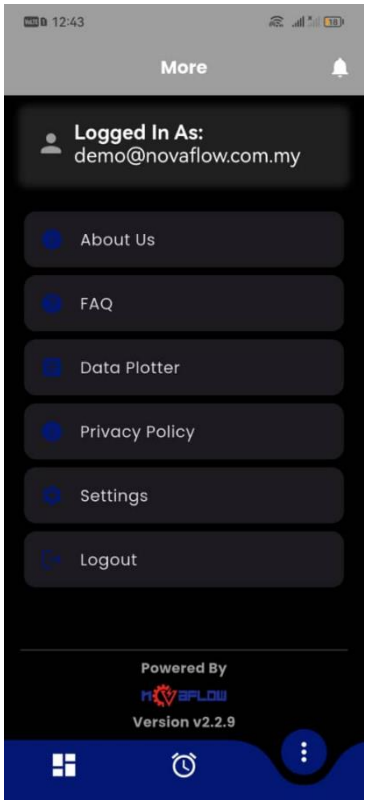


Figure 5.35: More Page on Android.

Figure 5.37: Data Plotter on iOS.

Figure 5.38: Data Plotter on Android.

The About Us, FAQ, Privacy Policy and Settings pages are shown below. All information was provided by the company. These pages are important because the **About Us** page helps users understand the company's background and mission, the **FAQ** page addresses common questions to improve user experience and reduce support requests, and the **Privacy Policy** page informs users about how their personal data is collected, used, and protected, ensuring transparency and building trust.

Figure 5.40: About Us Page on iOS.

Figure 5.39: About Us Page on Android.

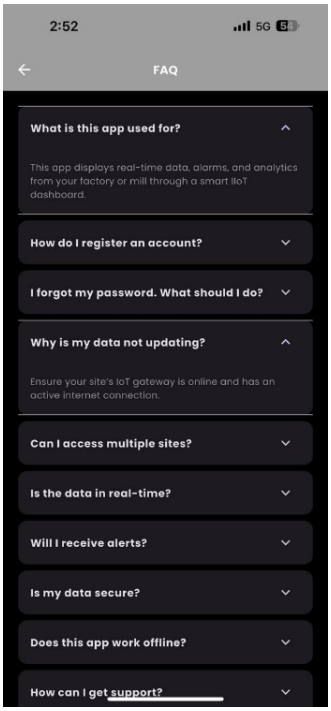


Figure 5.42: FAQ Page on iOS.

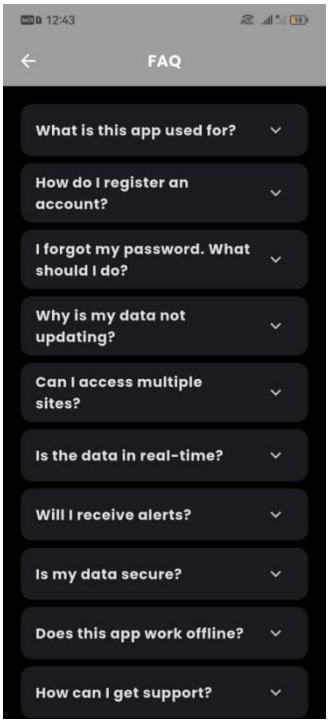


Figure 5.41: FAQ Page on Android.

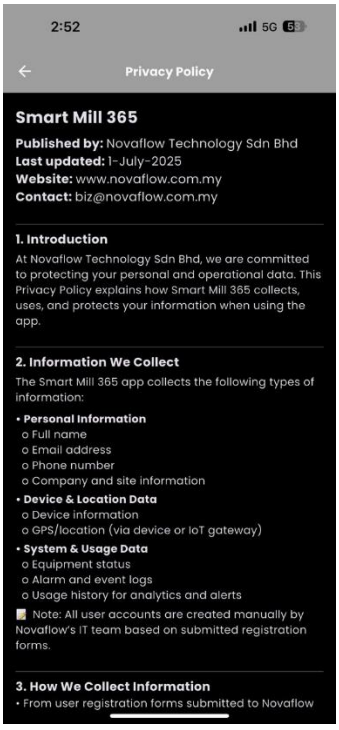


Figure 5.44: Privacy Policy Page on iOS.

Figure 5.43: Privacy Policy Page on Android.

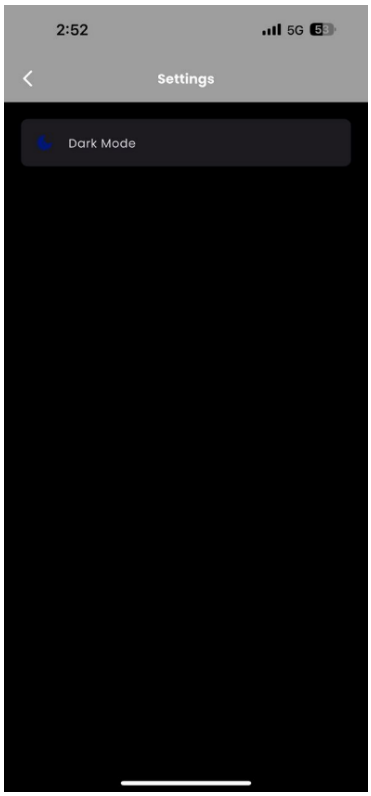


Figure 5.46: Settings Page on iOS.

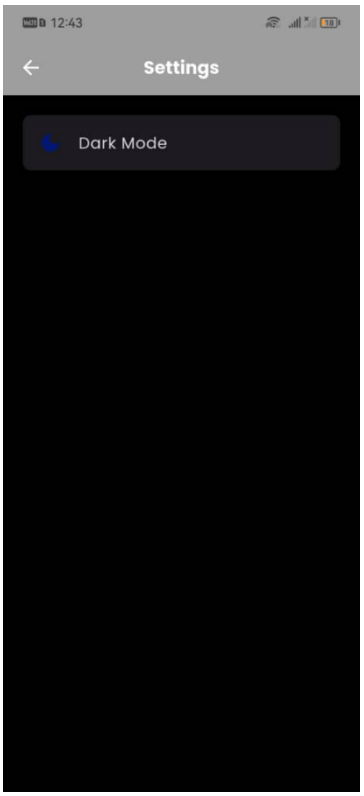


Figure 5.45: Settings Page on Android.

5.3 Deployment of Solution

For the iOS version, the application was successfully deployed to the **Apple App Store** as shown in Figure 5.47 below, which showing the completion of the development and deployment cycle for the iOS platform. Prior to the official release, **TestFlight** was used extensively for both internal testing by the development team and external testing by selected testers and stakeholders to identify and resolve the potential issues, ensuring that the application met company requirements and Apple's quality standards. This testing process helped verify that all functionalities operated as intended under different conditions before making the application publish to public.

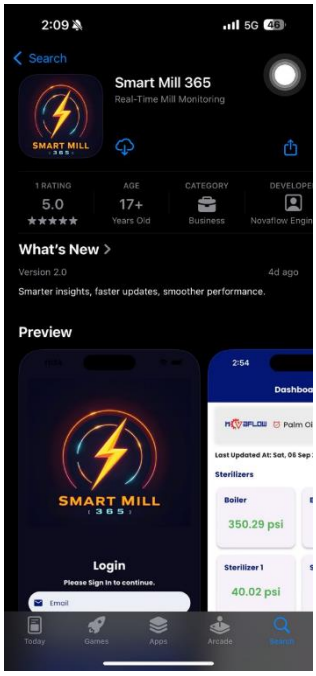


Figure 5.47: Apple App Store Listing of Smart Mill 365.

For the Android version, the company is in the process of preparing the Google Play developer account. While the functionality is being finalized, the application is temporarily distributed in **APK format** as shown in Figure 5.48 below, enabling direct installation on Android devices for testing and usage. This approach ensures that testers and stakeholders can continue evaluating the app's performance, identifying improvements, and providing feedback. The company will decide on the official deployment to the Google Play Store once the developer account is fully active and any final refinements are completed.

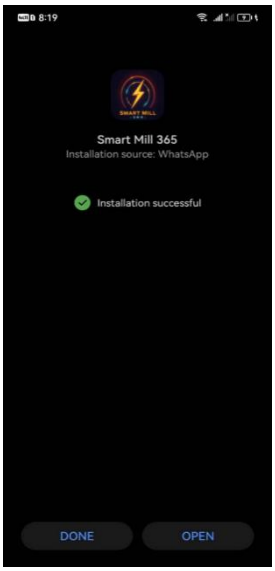


Figure 5.48: Android Version APK Download for Smart Mill 365.

All project folders and source code have been uploaded to a **GitHub** repository shared by Novaflow. This ensures centralized version control and easy collaboration among developers. By hosting the code on GitHub, the project can be easily maintained and improved in the future. The repository has also been formally handed over to the company, allowing Novaflow to take full ownership and continue for further development or deployment in future.

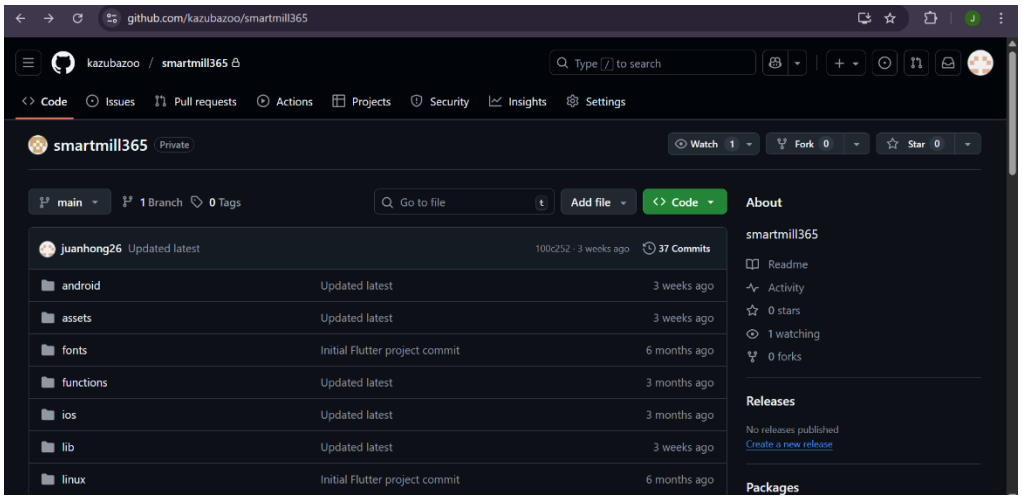


Figure 5.49: Project Upload to GitHub Repository.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter presents the implementation of the SmartMill365 system, detail out the setup of Firebase and MySQL cloud database, and the development of its five main modules such as Login, Dashboard, Graph Monitoring, Alarm Management, and User Profile & Settings. It explains how each component was built and integrated to achieve real-time monitoring, alarm handling, and user interaction.

6.2 Project Setup

6.2.1 Firebase Setup

This section describes the steps to set up Firebase for both Android and iOS applications used in the Smart Mill 365 system:

1. **A google account** is needed to set up a Firebase project.
2. Once a Google account is ready, go to https://console.firebase.google.com/ to **create a new Firebase project** by clicking the "Create a Firebase project" button in the Firebase console as shown in Figure 6.1 below:

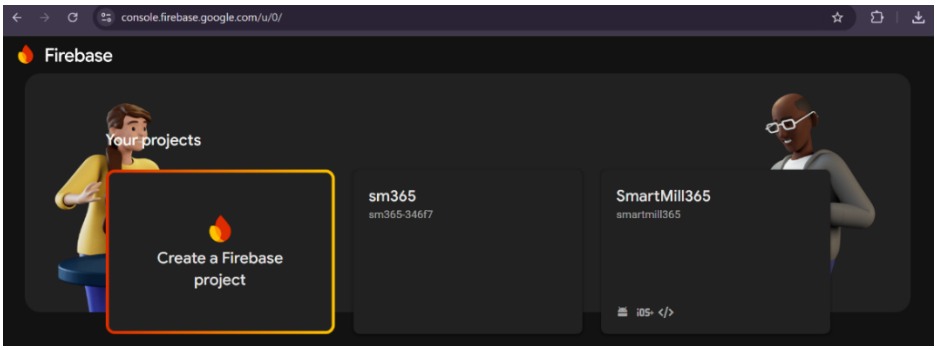


Figure 6.1 Firebase Website to Create Project.

3. Link the Firebase Project that created in previous step with the Flutter App. Each app will generate its own **SDK setup file** as shown in Figure 6.2 and Figure 6.5 respectively, which must be placed in the corresponding project folder as shown in Figure 6.3 and Figure 6.6.

- **Android:**
    - Downloads the google-services.json file from Firebase.

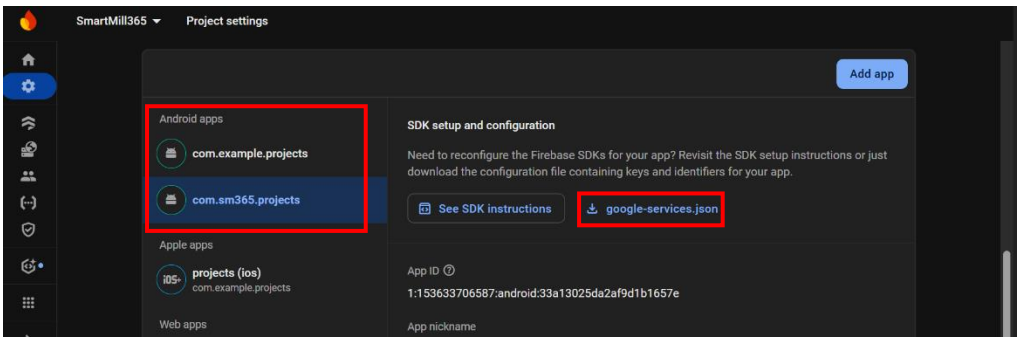


Figure 6.2 Firebase Configuration File for Android Mobile Application.

    - Place it in the **android/app/** directory of the Flutter project.

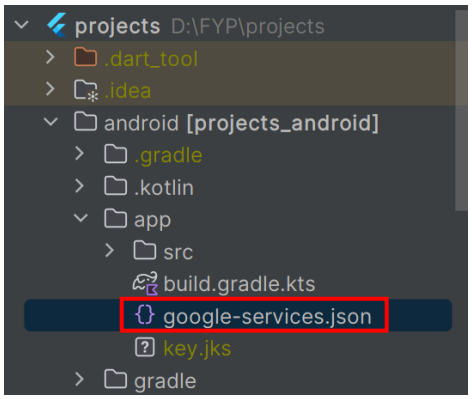


Figure 6.3 Location to Place the google-services.json File.

    - Then, add the Google-services Gradle plug-in as a dependency in the project-level **build.gradle.kts** file to enable processing of the google-services.json configuration, which is required for the Firebase SDK to function properly.

Figure 6.4 Add Google-services Plug-In at build.gradle.kts File.

- **iOS:**
    - Downloads GoogleService-Info.plist file.

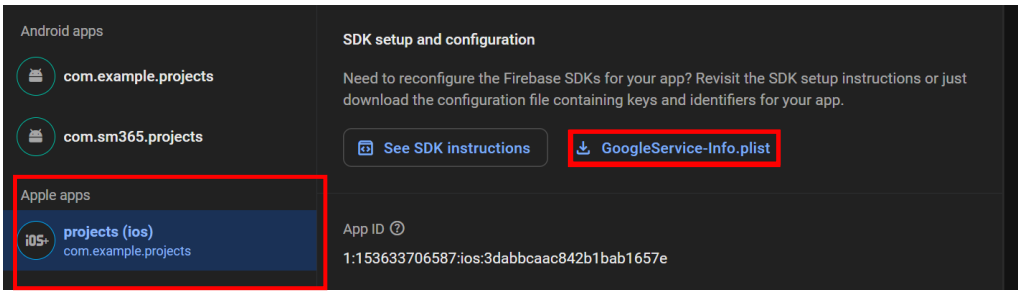


Figure 6.5 Firebase Configuration File for iOS Mobile Application.

- Place it in the **ios/Runner/** directory of the Flutter project.

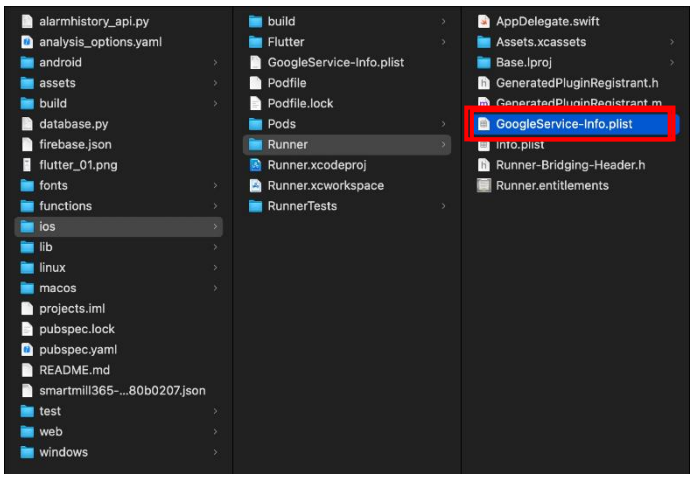


Figure 6.6 Location to Place the GoogleService-Info.plist File.

- Then, add the Google-services Gradle plug-in as a dependency in the project-level **build.gradle.kts** file to enable processing of the google-services.json configuration, which is required for the Firebase SDK to function properly.

Figure 6.7 Add Google-services Plug-In at build.gradle.kts File.

4. Navigate to **Authentication** in Firebase Console as shown in Figure 6.8 below to set up user account. Click the "Add user" button to create a new account by entering their desired Login Email and Password as shown in Figure 6.9. These credentials will be used later for logging into the application.

   Note: All user accounts for the Smart Mill 365 app are created by the Novaflow IT team.

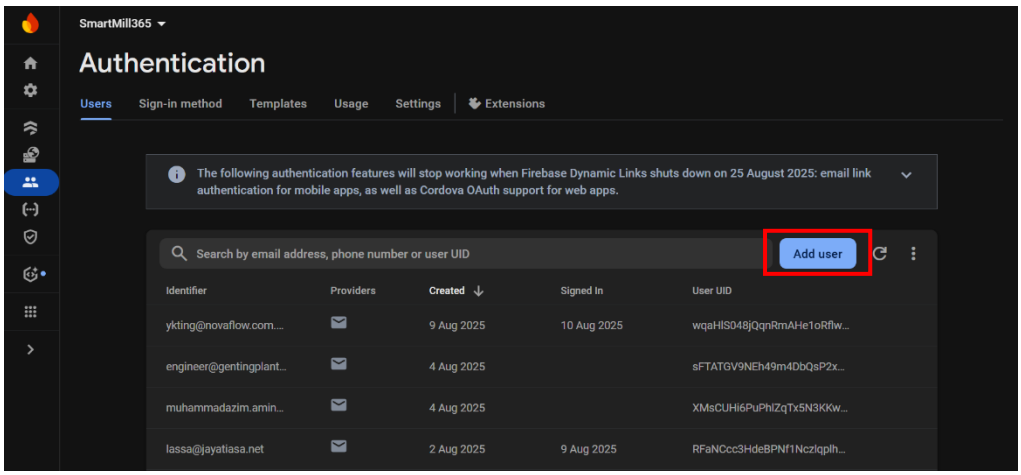


Figure 6.8: Firebase Authentication Page.

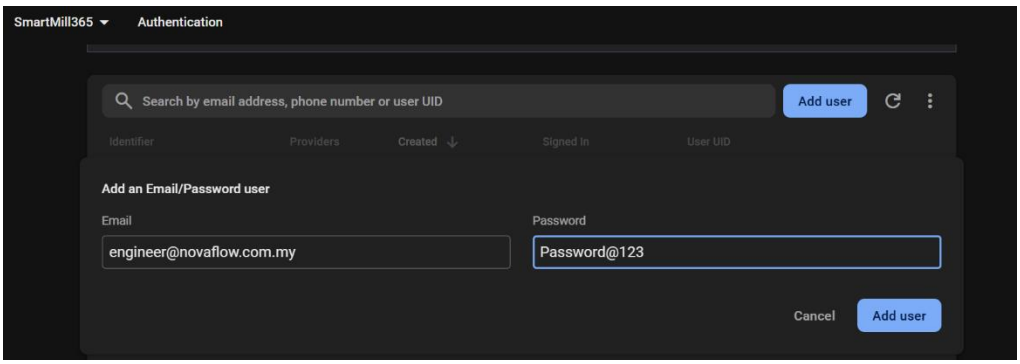


Figure 6.9: Create New Account.

5. Navigate to the **Firestore Database / Cloud Firestore** in Firebase Console. Create a new collection where the collection name must be same as the user's login email address as shown in Figure 6.10 below.

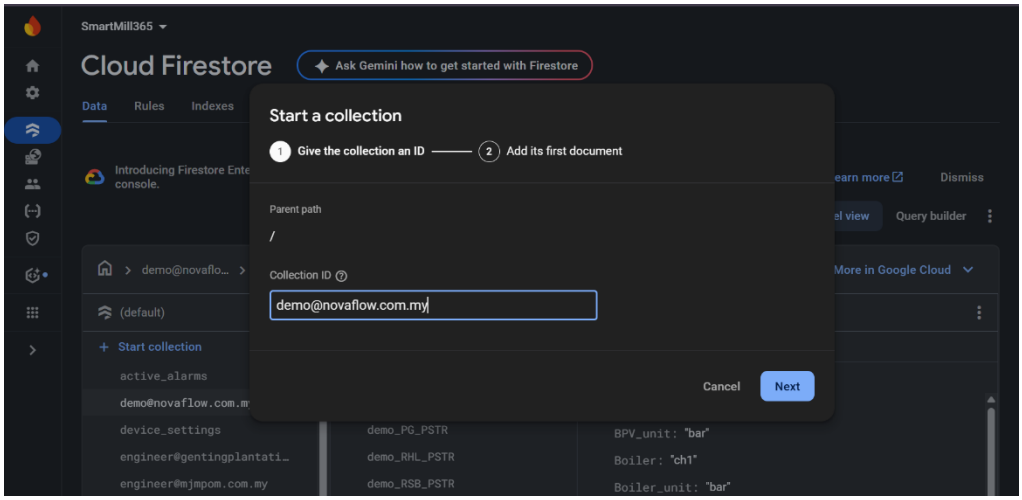


Figure 6.10 Create New User Collection.

6. Next step is to create Document(s) inside the Collection as shown in Figure 6.11. For users with multiple subgroups/sites need to create a separate document for each subgroup/site, while for users with only 1 subgroup/site, only 1 document need to be created.

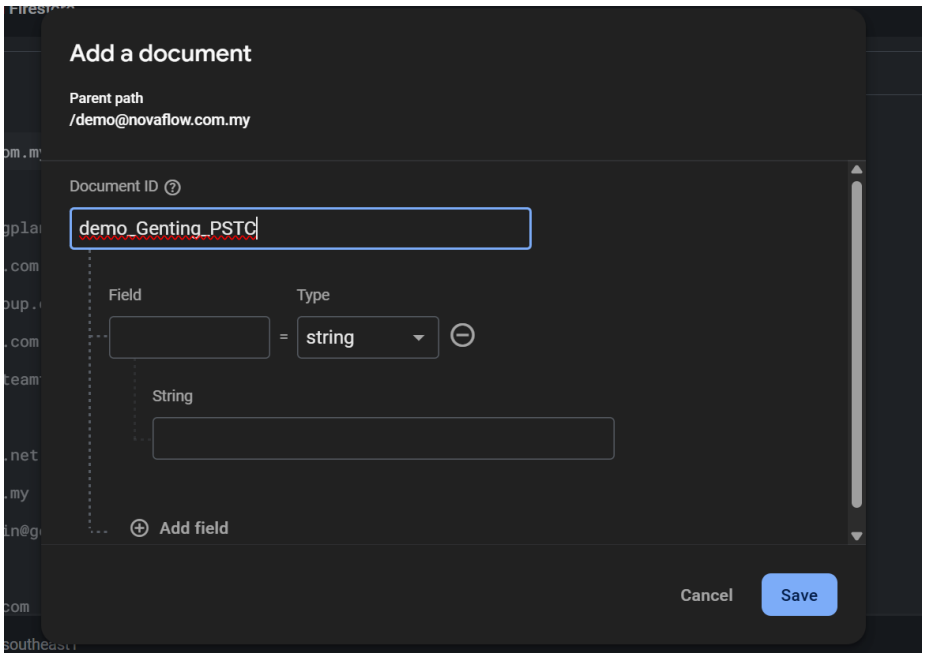


Figure 6.11 Create New Document for Subgroup/Site.

7. Then add all required fields in each document. Each document must include the following fields in Table 6.1:

Table 6.1 Required Field in Each Document.

Field Name	Type	Description
availablefield	Array	List of available devices **(Boiler, BPV, Sterilizer 1)**
id	String	Influx DB's ID **(SAMYSK_PSTR_240006)**
measurement	String	Influx DB's measurement **(PSTR, PSDG)**
measurement_name	String	Category name displayed on dashboard **(Sterilizers/Digestors)**
name	String	Site/subgroup name displayed on dashboard **(POM Genting Group)**
device_channel	Map	Mapping of device to channels **(BPV: ch1, Boiler: ch2)**
device_unit	Map	Mapping of devices to units **(BPV: bar, Boiler: psi)**
device_event	Map	Mapping of devices to event **(0: Stop, 1: Run, 2: E-stop)**

A sample output is shown in Figure 6.12 below:

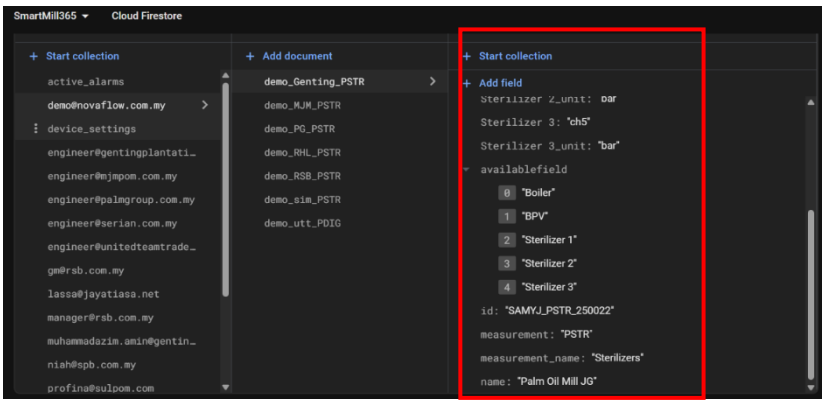


Figure 6.12 Sample Fields inside Document.

6.2.2 MySQL Cloud Database Setup

Note: Since the database operations and structure inside the phpMyAdmin can only be created by Novaflow (as my side is blocked from performing CRUD operations due to company's concern that I might accidentally remove data or amend table structures), I will not demonstrate the process step by step. I will only identify and mention the steps that need to be taken.

1. Prepare a MySQL cloud database account.
2. Create a Group named POM (Palm Oil Mill) so that POM-related databases are categorized properly and won't accidentally mix with other groups.
3. Inside this group, create different databases for different users. For example, POM_BorneoAgro and POM_Genting.
4. Inside each user database, create 3 structures/tables:

- ID_alarm_history
- ID_description_template
- ID_threshold_settings

The ID in the table name must match with the Firebase / Influx DB id, otherwise the system will not be able to retrieve or store data. For structures/table names example,

- SAMYJ_PSTR_250022_alarm_history
- SAMYJ_PSTR_250022_description_template
- SAMYJ_PSTR_250022_threshold_settings

A sample of Step 2 to Step 4 is shown in the Figure 6.13 below:

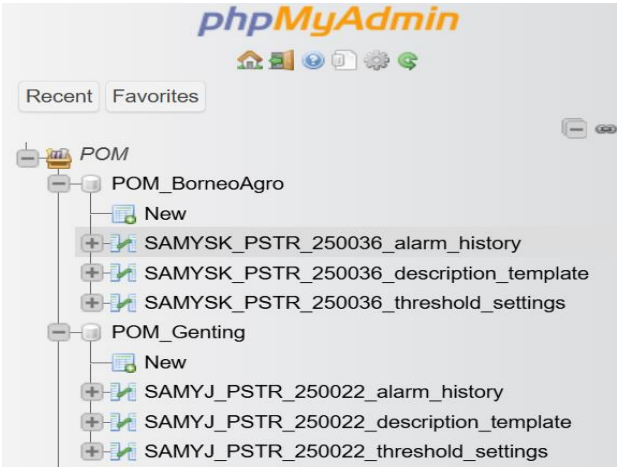


Figure 6.13 Sample MySQL Databases.

5. Each table contains its own specific data structure. The data structures of the **ID_alarm_history, ID_description_template, and ID_threshold_settings** tables are described in the following sections:

Table 6.2 Data Structures of ID_alarm_history Table.

Field Name	Data Type	Description
id	int	Unique identifier for each alarm record
channel_name	varchar (255)	Devices **(BPV, Boiler, Sterilizer 1)**
variable	varchar (255)	Devices unit **(pressure, bar, amp)**
start_time	datetime	Time when the alarm was triggered
end_time	datetime	Time when the alarm ended
description	text	Alarm message **(Sterilizer 1 is below the limit of 20.0 psi)**
last_value	float	Value recorded at the moment the alarm was triggered
acknowledge	enum ('Yes', 'No')	Acknowledgment status, default is "No"
ack_user	varchar (255)	Email of the user who acknowledged the alarm **(demo@novaflow.com.my)**

The sample record of Alarm History Table is shown below:

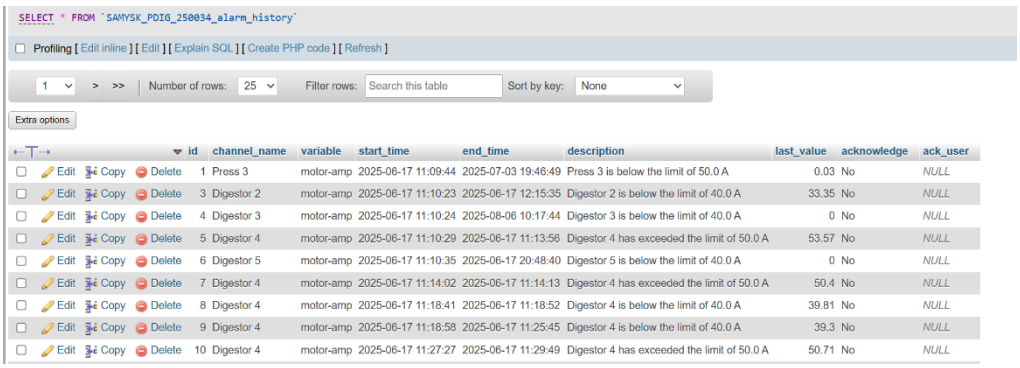


Figure 6.14: Sample Data in ID_alarm_history Table.

Table 6.3 Data Structures of ID_description_template.

Field Name	Data Type	Description
id	int	Unique identifier of the description
alarm_type	enum ('below', 'exceed')	Indicates whether the alarm condition is triggered by being below or exceeding the limit
description	text	Alarm description text **(is below the limit of / has exceeded the limit of)**

Table 6.4 Data Structures of ID_threshold_settings.

Field Name	Data Type	Description
id	int	Unique identifier for threshold settings
site_id	varchar (255)	Influx DB's ID **(SAMYSK_PSTR_250002)**
channel_key	varchar (255)	Device channel **(ch1, ch2)**
channel_name	varchar (255)	Device name **(Boiler, BPV)**
variable	varchar (255)	Device variable **(pressure)**
unit	varchar (255)	Device unit **(bar, psi)**
bucket	varchar (255)	Bucket name **(Mill)**
measurement	varchar (255)	Measurement type **(PSTR, PDIG)**
alarm	enum ('Yes', 'No')	Alarm status **(On/Off)**
threshold_high	float	Upper threshold limit **(50)**
threshold_low	float	Lower threshold limit **(20)**

6. After creating the tables based on the defined data structures, filled in some predefined threshold values and device configurations into the databases, **except for the alarm history table**. The sample data for ID_description_template and ID_threshold_settings tables is shown below:

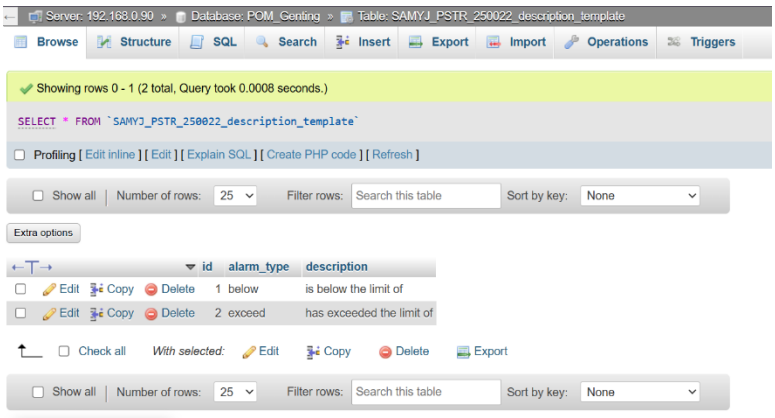


Figure 6.15 Sample Data in ID_description_template Table.

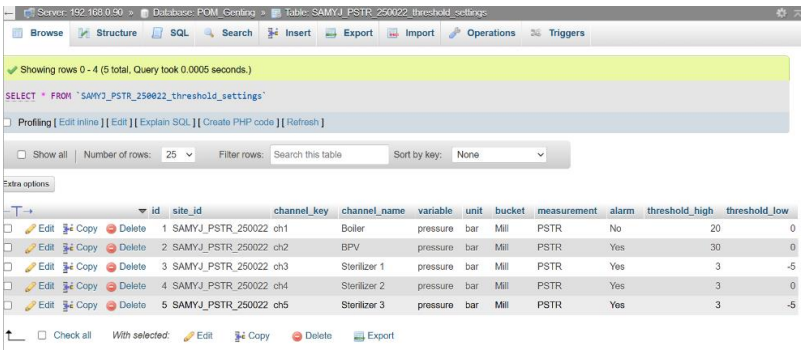


Figure 6.16 Sample Data in ID_threshold_settings Table.

6.3 System Modules

Modules
6.3.1 Log In
6.3.2 Dashboard
6.3.3 Graph Monitoring
6.3.4 Alarm Management
6.3.5 User Profile & Settings

6.3.1 Log In Module

The login module is used for authenticating users before granting access to the application. This process is handled using Firebase Authentication, which securely verifies the provided email and password against stored user credentials as shown in Figure 6.17.

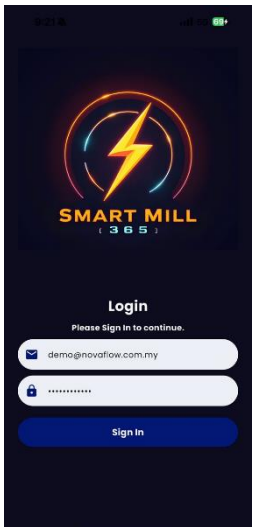


Figure 6.17 Login Page

The modules included the following validation steps before authentication:

1. **Empty input fields.** If the email or password are left blank, the system prevents login and display an error message shown in Figure 6.18.
2. **Invalid email format.** If the entered email does not match the standard email format, the system prompts the user with an error message shown in Figure 6.19.
3. **Incorrect login credentials.** If the email or password does not match any registered account in Firebase, the system displays “Wrong email or password” as shown in Figure 6.20.

If all the validation checks above pass, Firebase Authentication verifies the credentials. Upon **successful authentication**, the user is redirected to the dashboard (home page) to start using the application.

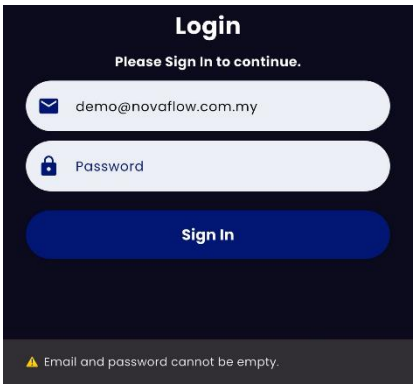


Figure 6.18: Login Page with Empty Input Field Error Message.

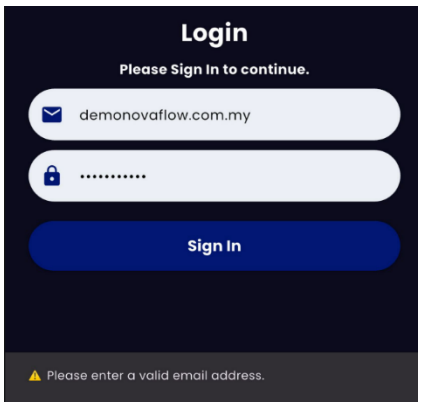


Figure 6.19: Login Page with Invalid Email Format Error Message.

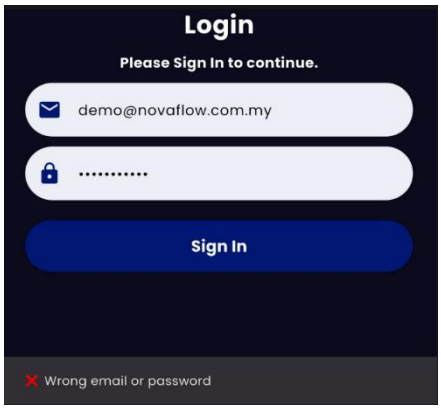


Figure 6.20: Login Page with Incorrect Login Credentials Error Message.

The implementation of input validation for login credentials is shown in Figure 6.21 below.

```
// Validate empty fields
if (email.isEmpty || password.isEmpty) {
  _showMessage("⚠ Email and password cannot be empty.");
  return;
}

// Validate email format
if (!isValidEmail(email)) {
  _showMessage("⚠ Please enter a valid email address.");
  return;
}
```

Figure 6.21: Implementation of Input Validation.

To handle the user login session as implemented in Figure 6.22, **FirebaseAuth.instance.authStateChanges()** continuously listens to the user's authentication state and automatically restores the session if the user has logged in previously, where **snapshot.hasData** returns true and navigates to the Home Page; otherwise, if the user has logged out, it returns false and shows the Login Page. This check runs every time the app starts or when the login state changes, and because Firebase Authentication use its own client SDK with tokens (not sessions or cookies-based), the user remains logged in across app restarts until explicitly call **signOut()** in More Page.

```
class AuthWrapper extends StatelessWidget {
  const AuthWrapper({super.key});

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Scaffold(body: Center(child: HeartbeatLoading(
            size: 60.0,
          ),));  // HeartbeatLoading, Center, Scaffold
        }
        if (snapshot.hasData) {
          return const HomePage();
        }
        return LoginPage();
      },
    );  // StreamBuilder
  }
}
```

Figure 6.22: Implementation of Firebase Authentication State Management.

6.3.2 Dashboard Module

The dashboard module serves as the central or home of the system, providing users with a real-time overview of site conditions, device statuses, and alarm notifications. It is considered as an overview that allows users to easily monitor and analyse the palm oil mill operations and performance.

First, the header of the dashboard screen having an alarm button as highlighted in Figure 6.23 which allow users to quickly navigate to the Active Alarm page. By looking into Figure 6.24, the id (eg., SAMKYSK_PSTR_250022) is required to pass into the Active Alarm Page for fetching the relevant active alarm record in Firebase, which later on discussed in Section 6.3.4 Alarm Management Module. This provides a direct access for users to review critical alerts without having complicated steps.

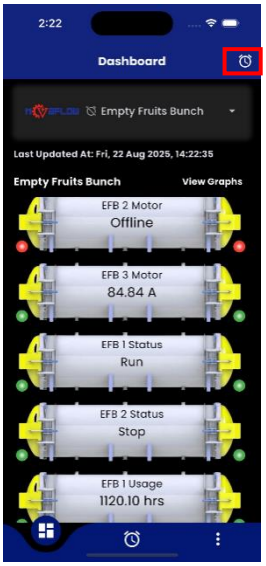


Figure 6.23: Dashboard Page.

```
IconButton(
  icon: const Icon(Icons.alarm, color: Colors.white),
  onPressed: () {
    if (id != null && id!.isNotEmpty) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => ActiveAlarmPage(id: id!),
        ), // MaterialPageRoute
      );
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text("Please select a site first. ID is still loading."),
        ), // SnackBar
      );
    }
  },
), // IconButton
```

Figure 6.24: Implementation of Active Alarm Button.

On top of the dashboard page, there are a container having selection with available subgroup name as shown in Figure 6.25 where user can select from different subgroups or site to view its own dashboard. Once the particular subgroup is selected, it will dynamically update the dashboard to display only the available devices and measurements for the chosen subgroup.

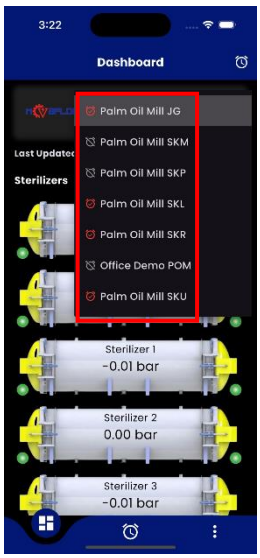


Figure 6.25: Subgroup / Site Selection for Switching Dashboard.

The list of available subgroups is configured and stored in the **Firebase document section** as shown in Figure 6.26 below, ensuring that administrators can easily add, update, or remove subgroup configurations without modifying the application code.

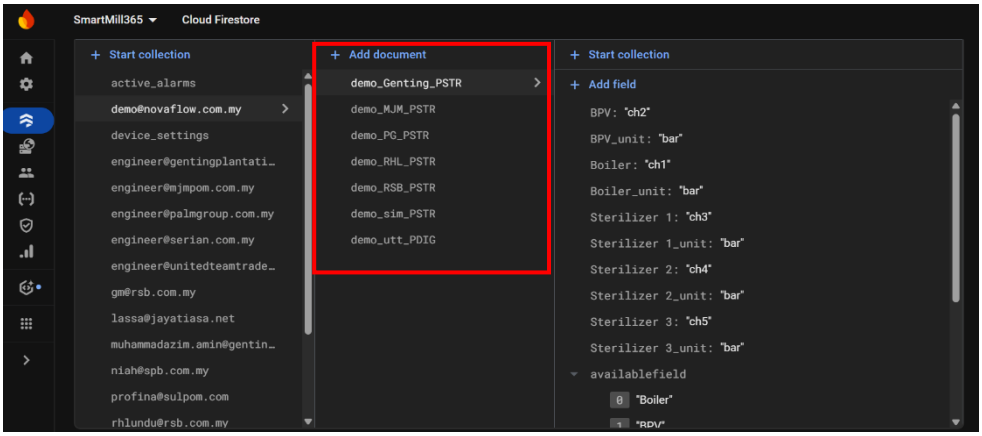


Figure 6.26: List of Available Subgroups in Firebase Document.

When the app is running, the application retrieves these subgroups by querying the Firebase collection associated with the current user's login email as shown in Figure 6.27 below. Each subgroup document contains its own name, which then extracted and displayed in the dashboard's selection menu. This ensures that the dashboard dynamically reflects the latest subgroup structure defined in Firebase without requiring manual update in source code.

```
Future<void> _fetchSubgroupsForUser(String userEmail) async {
  try {
    setState(() {
      isLoading = true;
      _isInitialLoad = true;
    });

    QuerySnapshot querySnapshot = await _firestore.collection(userEmail)
        .get();

    if (querySnapshot.docs.isNotEmpty) {
      subgroups = querySnapshot.docs.map((doc) {
        Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
        return {
          'email': doc.id,
          'id': '${data['id']}',
          'name': '${data['name'] ?? doc.id}',
        };
      }).toList().cast<Map<String, String>>();
```

Figure 6.27: Implementation of Fetching Available Subgroup.

Beside each subgroup selection, a **small alarm status icon** indicates whether there are active alarms for the selected subgroup as shown in Figure 6.28. A red alarm icon indicates that active alarms are detected, while a grey alarm icon indicates no active alarms. The system determines this by checking against the **active_alarms** collection in Firebase. If the collection contains an alarm record with the current subgroup's ID, the icon will turn red immediately without having to refresh the screen; otherwise, it remains grey. This implementation allows users to respond more quickly to critical conditions, especially for users that manage multiple subgroups or site simultaneously.

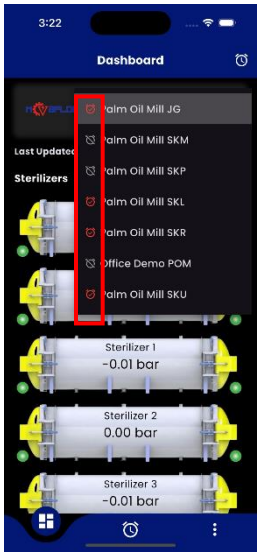


Figure 6.28: Active Alarm Icon.

```
void listenToActiveAlarms() {
  alarmSubscription?.cancel(); // Cancel old subscription if any

  alarmSubscription = FirebaseFirestore.instance
      .collection('active_alarms')
      .where('status', isEqualTo: 'active')
      .snapshots()
      .listen((snapshot) {
    Map<String, bool> tempMap = {};

    // Mark all site_ids that currently have active alarms
    for (var doc in snapshot.docs) {
      final data = doc.data() as Map<String, dynamic>?;
      if (data == null) continue;

      final siteId = (data['site_id'] as String?)?.trim();
      if (siteId != null) {
        tempMap[siteId] = true;
      }
    }
```

Figure 6.29: Implementation of Active Alarm Icon Logic.

```
final hasActiveAlarm = alarmStatus[id] ?? false;

return DropdownMenuItem<String>(
  value: email,
  child: Row(
    children: [
      Icon(
        hasActiveAlarm ? Icons.alarm_on : Icons.alarm_off,
        color: hasActiveAlarm ? Colors.red : Colors.grey,
        size: 18,
      ),  // Icon
      const SizedBox(width: 6),
      Flexible(
        child: Text(
          name,
          overflow: TextOverflow.ellipsis,
        ),  // Text
      ),  // Flexible
    ],
  ),  // Row
);  // DropdownMenuItem
```

Figure 6.30: Implementation of Active Alarm Icon.

Next, to make the dashboard more alive, a **timestamp indicator** (eg., Last Updated At: 22 Aug 2025, 14:22:35 as shown in Figure 6.31 below) is implemented. The timestamp is synchronized with the batch data pull from Influx DB, reflecting the most recent live data update. Since the site data from Influx DB is updated in batch form and each site contains a channel **ch1**, the timestamp will always take the ch1's timestamp as the representative time as shown in Figure 6.33. The timestamp format is then set using the **_formatInfluxTimestamp** function in Figure 6.32 and displayed out. In most cases, it shows the current time, but in some scenarios, it may display the last recent update time instead. If the device is totally offline, the dashboard will display **"Offline"** in the place of timestamp.

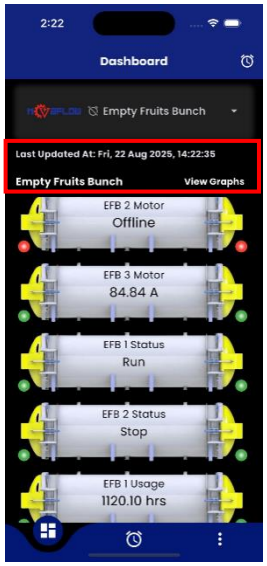


Figure 6.31: Timestamp, Measurement Name, and View Graphs button.

```
String _formatInfluxTimestamp(String influxTime) {
  try {
    final dateTime = DateTime.parse(influxTime).toLocal();
    return DateFormat('EEE, dd MMM yyyy, HH:mm:ss').format(dateTime);
  } catch (e) {
    print("Error formatting timestamp: $e");
    return "Invalid timestamp";
  }
}
```

Figure 6.32: Implementation of Timestamp Format.

```
if (channelName == "ch1" && timestamp != null && mounted) {
  setState(() {
    _latestCh1Timestamp = _formatInfluxTimestamp(timestamp);
  });
```

Figure 6.33: Implementation of Timestamp Logic.

The Empty Fruits Bunch as shown in Figure 6.31 above, known as **measurement name** is dynamically updated based on settings retrieved from Firebase site configuration using the **_fectchMeasurementName** function in Figure 6.34. This allows flexible in managing device measurement naming without requiring hardcoded values. On the same row, a **"View Graphs"** text button is provided as implemented in Figure 6.35. Users can click this button to navigate to the Graph Monitoring Module (discussed in Section 6.3.3). This feature enables users to view multiple devices' graphs simultaneously, making it easier to perform comparisons and analysis.

```
Future<void> _fetchMeasurementName(String subgroupEmail) async {
  try {
    final doc = await _firestore
        .collection(currentUserEmail!)
        .doc(subgroupEmail)
        .get();

    if (doc.exists) {
      final data = doc.data() as Map<String, dynamic>;
      setState(() {
        measurementName = data['measurement_name'];
        print("✅ Loaded measurement_name: $measurementName");
      });
    } else {
      print("❌ Document for $subgroupEmail not found.");
    }
  } catch (e) {
    print("❌ Error fetching measurement_name: $e");
  }
}
```

Figure 6.34: Implementation of Fetch Measurement Name.

```
TextButton(
  onPressed: () {
    final fieldToView = availableFields.isNotEmpty ? availableFields[0] : '';
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => ViewAllScreen(
          subgroupEmail: selectedSubgroupEmail!,
          availableFields: availableFields,
          fieldMap: fieldMap,
          deviceName: fieldToView,
        ),  // ViewAllScreen
      ),  // MaterialPageRoute
    );
  },
  child: Text(
    'View Graphs',
    style: TextStyle(
      fontSize: 14.0,
      fontWeight: FontWeight.bold,
      color: labelColor,
    ),  // TextStyle
```

Figure 6.35: Implementation of View Graphs Button.

The implementation of the main body of the dashboard shown in Figure 6.36 is discussed below.

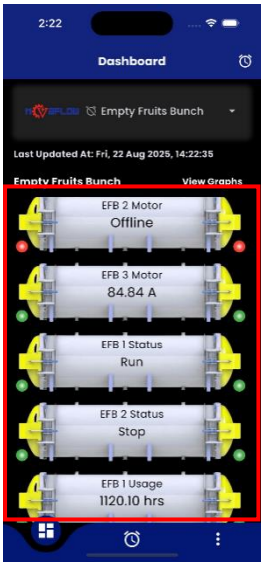


Figure 6.36: SVG-based Graphical Layout.

When user open the application, the system will first initialize and load all first subgroup's device data such as device's Influx DB ID and measurement, measurement name and unit that configured in Firebase. During this loading process, instead of showing an incomplete dashboard, the application displays the company brand logo shown in Figure 6.38 with a heartbeat-style animation as a loading indicator. This design ensures a smooth user experience by maintaining branding consistency and preventing confusion. Once all the relevant data has been fully retrieved and initialized, the application proceeds to render and display the dashboard with the live device data.

```
@override
void initState() {
  super.initState();

  WidgetsBinding.instance.addPostFrameCallback((_) {...});

  _startTimer();
  listenToActiveAlarms();

  _getCurrentUserEmail().then((_) {
    if (selectedSubgroupEmail != null) {
      _fetchMeasurementName(selectedSubgroupEmail!);
      _fetchIdForSubgroup(selectedSubgroupEmail!);
      _fetchUnitsForFields(selectedSubgroupEmail!);
      _fetchMeasurement(selectedSubgroupEmail!);
    } else {
      print("❌ selectedSubgroupEmail is null");
    }
  });
}
```

Figure 6.37: Implementation of Dashboard Initialization.

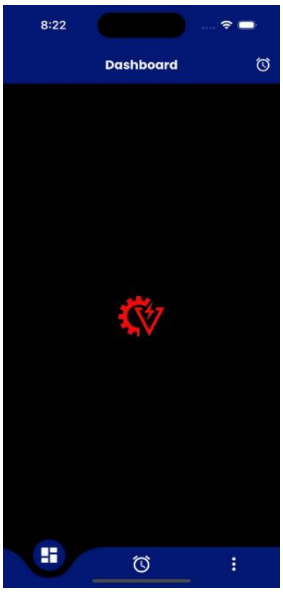


Figure 6.38: Brand Logo Loading Indicator.

When user switches the dashboard between different subgroups or site, the current dashboard data will clear and set to null as implemented in Figure 6.39. This implementation prevents any possibility of passing incorrect data to the newly selected subgroup. During this transition, a loading indicator shown in Figure 6.38 above is displayed to inform user about the system is retrieving the new subgroup's data. Once all the device data has been fully fetched from Firebase and Influx DB, the dashboard for the new selected subgroup will be rendered and displayed.

```
void onSubgroupChanged(String? newValue) async {
  if (newValue != null && newValue != selectedSubgroupEmail) {
    setState(() {
      _isSubgroupLoading = true; // Show loading state
      // Clear ALL previous data immediately
      selectedSubgroupEmail = newValue;
      availableFields = [];
      latestPressureData = {};
      fieldUnits = {};
      _latestCh1Timestamp = null;
      measurementName = null;
      measurement = null;
      id = null;
    });

    try {
      // Fetch all required data for new subgroup
      await _fetchIdForSubgroup(newValue);
      await _fetchMeasurementName(newValue);
      await _fetchMeasurement(newValue);
      await _fetchAvailableFields(newValue);
      await _fetchUnitsForFields(newValue);
```

Figure 6.39: Implementation of Switching Subgroup Logic.

When the account only created in Firebase Authentication, but no corresponding Firebase collection exists for that account, the system will display the message in Figure 6.40. For accounts that already have their own Firebase collection, but no device settings are configured (no available fields found), the system will return the message as shown in Figure 6.41.

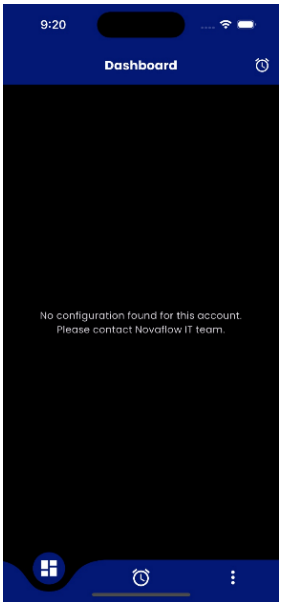


Figure 6.40: Dashboard Page with No Account Configuration.

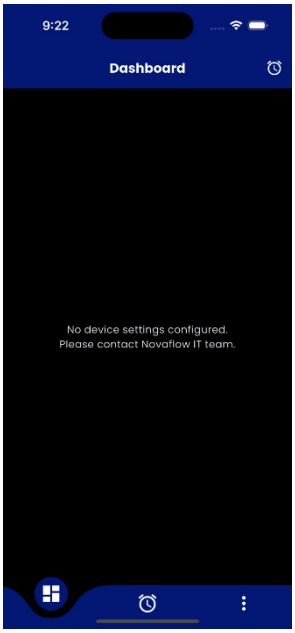


Figure 6.41: Dashboard Page with No Device Configuration.

```
    : hasError
    ? const Center(child: Text("No configuration found for this account.\nPlease contact Novaflow IT team.", textAlign: TextAlign.center,
)) // Text, Center
    : availableFields.isEmpty
    ? const Center(child: Text("No device settings configured.\nPlease contact Novaflow IT team.", textAlign: TextAlign.center,
)) // Text, Center
```

Figure 6.42: Implementation of Configuration Check Logic on Dashboard.

At the bottom of the dashboard, navigation options are provided for **Alarm History** and **More pages as shown in Figure 6.43** below. These give user a quick access to past alarm records and additional system features. The design implemented for the navigation is a **curved navigation bar** as this is the one of the company's design requirements, providing a modern look and smooth user interaction.

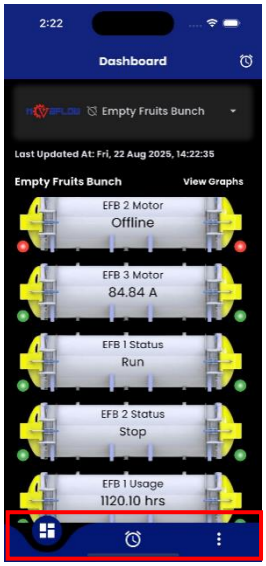


Figure 6.43: Dashboard Page Bottom Navigation.

```
bottomNavigationBar: CurvedNavigationBar(
  index: _selectedIndex,
  height: 60,
  items: <Widget>[
    Icon(Icons.dashboard, size: 30, color: Colors.white),
    Icon(Icons.alarm, size: 30, color: Colors.white),
    Icon(Icons.more_vert, size: 30, color: Colors.white),
  ],  // <Widget>[]
  color: Color(0xFF031774),
  buttonBackgroundColor: Color(0xFF031774),
  backgroundColor: Colors.transparent,
  animationCurve: Curves.easeInOut,
  animationDuration: Duration(milliseconds: 300),
  onTap: _onItemTapped,
),  // CurvedNavigationBar
```

Figure 6.44: Implementation of Bottom Navigation.

6.3.3 Graph Monitoring Module

The Graph Monitoring Module is one of the core components of the palm oil mill monitoring system, where the **Single Graph View** serves as the main monitoring tool. This view provides a clear and interactive visualization of device's channel data, enabling users to analyse performance efficiently.

Each device graph is generated using a single reusable codebase (genericscreen.dart). This design ensures consistency across devices and eliminates the need to manually create and modify separate codebase for different devices. Any changes made to the code are automatically reflected in all single graph views, significantly improving maintainability and scalability.

When navigating to the single graph view screen, the screen automatically rotates to horizontal orientation for a better graph viewing experience. During initialization, the app also enables full-screen mode and sets up the zoom and pan behaviours for the graph. After that, it retrieves the device's data through **fetchPressureData()** and loads its corresponding unit using **_fetchUnitForDevice()**.

```
@override
void initState() {
  super.initState();
  SystemChrome.setEnabledSystemUIMode(
    SystemUiMode.edgeToEdge,
    overlays: SystemUiOverlay.values,
  );
  _enterFullScreen();
  _zoomPanBehavior = ZoomPanBehavior(
    enablePinching: true,
    enableDoubleTapZooming: true,
    enablePanning: true,
    enableSelectionZooming: false,
  );
  _getCurrentUserEmail().then((_) {
    fetchPressureData();
    _fetchUnitForDevice();
  });
}
```

Figure 6.45: Implementation of initState.

The device channel data is fetched from **Influx DB**, using the **given authorization token and URL**, with correct passing of the Influx DB measurement, id, and field for the selected channel that configured in Firebase. Meanwhile, the unit is also directly retrieved from Firebase.

```
final query = '''
from(bucket: "Mill")
|> range(start: $selectedRange)
|> filter(fn: (r) => r["_measurement"] == "${data["measurement"]}")
|> filter(fn: (r) => r["id"] == "${data["id"] ?? widget.subgroupEmail}")
|> filter(fn: (r) => r["_field"] == "$field")
''';

try {
  final response = await http.post(
    Uri.parse('http:/[redacted]'),
    headers: {
      'Authorization': 'Token [redacted]',
      'Content-Type': 'application/vnd.flux',
      'Accept': 'application/csv',
    },
    body: query,
  );
```

Figure 6.46: Fetching Channel Data from Firebase.

```
Future<void> _fetchUnitForDevice() async {
  try {
    final firestore = FirebaseFirestore.instance;
    final user = FirebaseAuth.instance.currentUser;
    final userEmail = user?.email;
    if (userEmail == null) return;

    final doc = await firestore.collection(userEmail).doc(
        widget.subgroupEmail).get();

    if (doc.exists) {
      final data = doc.data()!;
      final unitKey = '${widget.deviceName}_unit';
      setState(() {
        pressureUnit = data[unitKey] ?? '';
      });
    }
  } catch (_) {}
}
```

Figure 6.47: Fetching Device Unit from Firebase.

To optimize the screen space, the graph does not display values on the graph header. Instead, it uses an **interactive tooltip** as **s**how in Figure 6.48 that appears only when the user taps on a data point. The tooltip is implemented using **TrackballBehavior.builder** as shown in Figure 6.49. It shows the time of the selected point, device name, value and unit as implemented in Figure 6.50, ensuring that the data remains informative while maximizing the visible graph area.

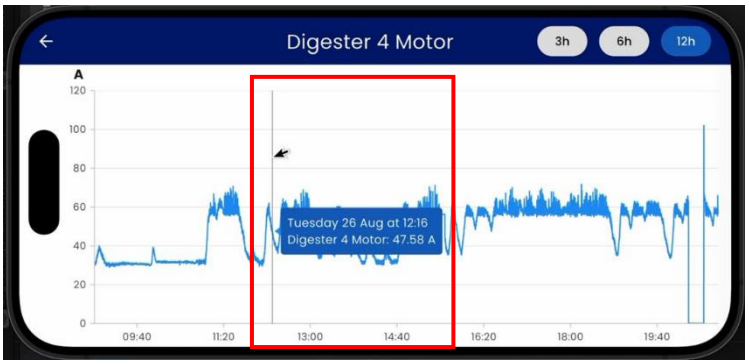


Figure 6.48: Tooltip.

```
trackballBehavior: TrackballBehavior(
  enable: true,
  lineWidth: 0.5,
  activationMode: ActivationMode.singleTap,
  tooltipDisplayMode: TrackballDisplayMode.floatAllPoints,
  builder: (BuildContext context, TrackballDetails details) {
    try {
      final point = details.point ??
          (details as dynamic).chartPointInfo?.first ??
          (details as dynamic).points?.first;

      if (point == null) return const SizedBox();

      final DateTime dateTime = point.x as DateTime;
      final double pressureValue = point.y as double;

      String displayValue;
      if (_statusMappings.isNotEmpty) {
        // For status fields, use the mapped text
        final intValue = pressureValue.toInt();
        displayValue = _statusMappings[intValue] ?? pressureValue.toStringAsFixed(2);
      } else {
        // For numeric fields
        displayValue = pressureValue.toStringAsFixed(2);
      }
```

Figure 6.49: Implementation of Tooltip.

```
      return Container(
        padding: const EdgeInsets.all(8),
        decoration: BoxDecoration(
          color: Colors.blue[800],
          borderRadius: BorderRadius.circular(4),
        ),  // BoxDecoration
        child: Text(
          '${_formatDisplayTime(dateTime.toString())}\n'
              '${widget.deviceName}: $displayValue ${_statusMappings.isNotEmpty ? '' : pressureUnit ?? ''}',
          style: const TextStyle(color: Colors.white),
        ),  // Text
      );  // Container
    } catch (e) {
      debugPrint('Tooltip error: $e');
      return const SizedBox();
    }
```

Figure 6.50: Implementation of Tooltip Display Format.

Besides that, users can choose to view the graph within specific time ranges (3h, 6h, and 12h). The selection buttons are placed in the header section for quick access. These buttons are implemented through the **_buildTimeRangeButton** function, where each button dynamically updates

its colour to indicate the active selection and triggers **fetchPressureData()** to reload the graph according to the chosen range.

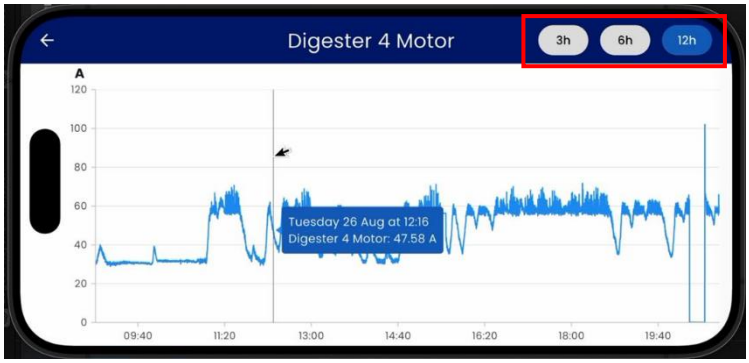


Figure 6.51: Time Range Selection.

```dart
Widget _buildTimeRangeButton(String value, String label) {
  final isSelected = selectedRange == value;
  return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 4),
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: isSelected ? Colors.blue[800] : Colors.grey[300],
        foregroundColor: isSelected ? Colors.white : Colors.black,
        minimumSize: const Size(60, 36),
        padding: EdgeInsets.zero,
      ),
      onPressed: () {
        setState(() => selectedRange = value);
        fetchPressureData();
      },
      child: SizedBox(
        width: 40, // consistent button width
        child: Center(
          child: Text(label),
        ), // Center
      ), // SizedBox
    ), // ElevatedButton
  ); // Padding
}
```

Figure 6.52: Implementation of Time Range Selection Button Logic.

The graph also supports interactive zooming and panning, implemented through the **ZoomPanBehavior** function. Whenever a different time range is selected, the graph automatically **resets to its normal size** as implemented in Figure 6.53, ensuring consistency and preventing distorted views across different ranges.

```
Future<void> fetchPressureData() async {
  if (currentUserEmail == null) return;

  // Reset zoom state when fetching new data
  _zoomPanBehavior?.reset();
```

Figure 6.53: Graph Reset to Normal Size.

For **Separate Graph View** and **Combine Graph View**, the implementation of fetching device channel data is almost same as the logic implemented in **Single Graph View** screen by using the **fetchFieldData()** function.

```
Future<void> fetchFieldData(String field, String measurement,
    String id) async {
  final channelName = widget.fieldMap[field];
  if (channelName == null) return;

  final query = '''
    from(bucket: "Mill")
      |> range(start: $selectedRange)
      |> filter(fn: (r) => r["_measurement"] == "$measurement")
      |> filter(fn: (r) => r["id"] == "$id")
      |> filter(fn: (r) => r["_field"] == "$channelName")
  ''';

  try {
    final response = await http.post(
      Uri.parse(influxUrl),
      headers: {
        'Authorization': influxToken,
        'Content-Type': 'application/vnd.flux',
        'Accept': 'application/csv',
      },
      body: query,
    );
```

Figure 6.54: Data Fetching from Influx DB.

The raw data fetched from Influx DB is first processed through the **convertToChartData()** function before being displayed on the graph. This function maps each data entry in a PressureData object by parsing the pressure values and converting timestamps. The **convertUTCToLocal()** function help convert UTC timestamps into local time format, ensuring that the displayed data aligns with the user's local time zone.

```dart
List<PressureData> _convertToChartData(List<Map<String, dynamic>> data) {
  return data.map((entry) {
    final time = convertUTCToLocal(entry["_time"]);
    final pressure = double.tryParse(entry["_value"] ?? '0') ?? 0;
    return PressureData(time, pressure);
  }).toList();
}

String convertUTCToLocal(String utcTime) {
  try {
    final dt = DateTime.parse(utcTime);
    return dt.toLocal().toString().split('.')[0];
  } catch (_) {
    return "Invalid Time";
  }
}
```

Figure 6.55: Convert to Chart and Local Time Zone.

From the UI implementation, the **Combined Graph View** shown in Figure 6.56 displays all selected graphs in a single chart with a share timeline. Each graph line is assigned a unique colour using HSV colour space for clear differentiation as shown in Figure 6.58. Additionally, users can temporarily hide the corresponding line by just clicking on the device name that shown inside the graph section.

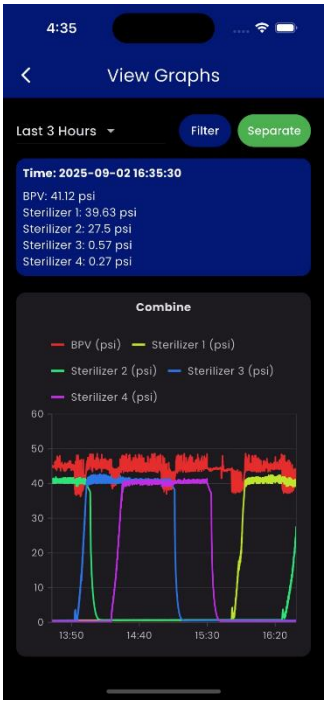


Figure 6.56: Combine Graph View.

```
Widget _buildCombinedGraph() {
  return Card(
    margin: const EdgeInsets.symmetric(vertical: 10),
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
        children: [
          const Text("Combined",
              style: TextStyle(fontWeight: FontWeight.bold)),  // Text
          const SizedBox(height: 10),
          SizedBox(
            height: 400,
            child: SfCartesianChart(
              primaryXAxis: DateTimeAxis(
                intervalType: DateTimeIntervalType.minutes,
                edgeLabelPlacement: EdgeLabelPlacement.shift,
                majorGridLines: const MajorGridLines(width: 0),
              ),  // DateTimeAxis
              primaryYAxis: NumericAxis(minimum: 0),
              legend: Legend(
                  isVisible: true,
                  position: LegendPosition.top,
                  overflowMode: LegendItemOverflowMode.wrap
              ),  // Legend
              series: _buildCombinedSeries(),
            ),  // SfCartesianChart
          ),  // SizedBox
```

Figure 6.57: Implementation of Combine Graph.

```
List<LineSeries<PressureData, DateTime>> _buildCombinedSeries() {
  return selectedGraphs.asMap().entries.map((entry) {
    final index = entry.key;
    final field = entry.value;
    final data = chartDataMap[field] ?? [];
    final unit = fieldUnits[field] ?? '';

    // Generate distinct colors using HSV
    final hue = (index * 360.0 / selectedGraphs.length) % 360.0;
    final color = HSVColor.fromAHSV(1.0, hue, 0.8, 0.9).toColor();

    return LineSeries<PressureData, DateTime>(
      name: '$field ($unit)',
      dataSource: data,
      color: color,
      width: 2,
      xValueMapper: (PressureData p, _) => DateTime.parse(p.time),
      yValueMapper: (PressureData p, _) => p.pressure,
    );
  }).toList();
}
```

Figure 6.58: UI for Combine Graph.

The **Separate Graph View** show each selected graph in a separate card shown in Figure 6.59 with a vertically rotated label as implemented in Figure 6.60. This view is useful for detailed analysis of specific device channels.

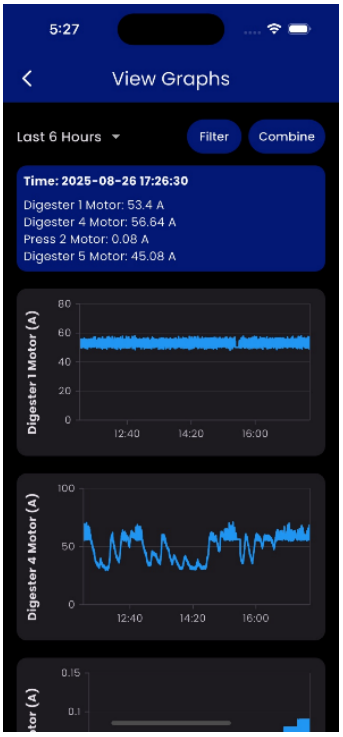


Figure 6.59: Separate Graph View.

```
Widget _buildIndividualGraph(String field) {
  final data = chartDataMap[field] ?? [];
  final unit = fieldUnits[field] ?? '';

  return Card(
    margin: const EdgeInsets.symmetric(vertical: 10),
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Row(
        children: [
          RotatedBox(
            quarterTurns: 3,
            child: Text(
              "$field ($unit)",
              style: const TextStyle(fontWeight: FontWeight.bold),
            ), // Text
          ), // RotatedBox
          const SizedBox(width: 10),
          Expanded(
            child: SizedBox(
              height: 180,
              child: SfCartesianChart(
                primaryXAxis: DateTimeAxis(
                  intervalType: DateTimeIntervalType.minutes,
                  edgeLabelPlacement: EdgeLabelPlacement.shift,
                  majorGridLines: const MajorGridLines(width: 0),
                ), // DateTimeAxis
                primaryYAxis: NumericAxis(minimum: 0),
                series: <CartesianSeries>[
                  LineSeries<PressureData, DateTime>(
                    color: Colors.blue,
```

Figure 6.60: Implementation of Separate Graph.

When users click the **Filter** button, an **AlertDialog** is shown, displaying a list of available channels with checkboxes. Users can select or deselect the checkboxes to choose which graphs they want to display. The dialog uses a **StatefulBuilder** to manage the temporary selection (tempSelected) independently, so changes inside the dialog do not

immediately affect the main state. When Confirm is clicked, it will update the selected graphs with the temporary selection, closes the dialog, and refreshes the chart dart via **fetchPressureData()**. If users clicked Cancel button, it will close the dialog without saving any changes.

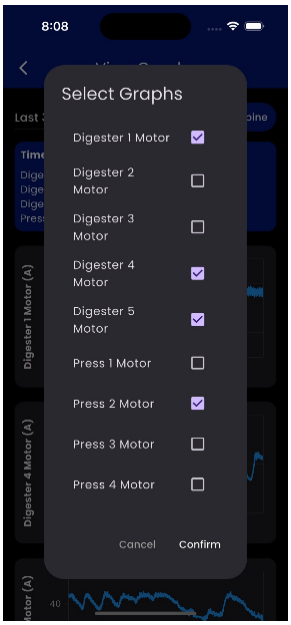


Figure 6.61: Dialog for Select Graphs.

```
void _showMultiSelectDialog(BuildContext context) {
  List<String> tempSelected = List.from(selectedGraphs);

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text("Select Graphs"),
        content: SingleChildScrollView(
          child: ListBody(
            children: widget.availableFields.map((field) {
              return StatefulBuilder(
                builder: (context, setState) {
                  return CheckboxListTile(
                    title: Text(field),
                    value: tempSelected.contains(field),
                    onChanged: (value) {
                      setState(() {
                        value! ? tempSelected.add(field) : tempSelected
                            .remove(field);
                      });
                    },
                  ); // CheckboxListTile
                },
              ); // StatefulBuilder
            }).toList(),
          ), // ListBody
        ), // SingleChildScrollView
        actions: [
```

Figure 6.62: Implementation of Dialog.

After selecting the desired graphs, users can filter the data by choosing a time range from the dropdown menu (Last 3,6,12 Hours) as shown in Figure 6.63. This allows them to view and analyse graph data within the selected period.

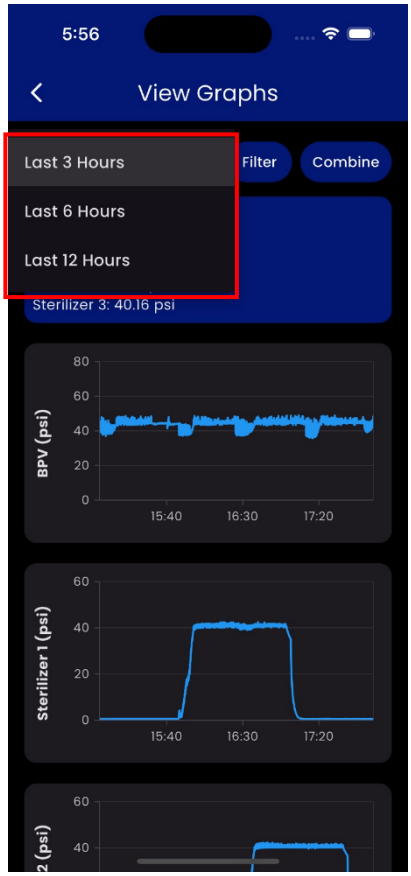


Figure 6.63: Tim Range Selection.

```
child: DropdownButton<String>(
  value: selectedRange,
  items: const [
    DropdownMenuItem(value: '-3h', child: Text('Last 3 Hours')),
    DropdownMenuItem(value: '-6h', child: Text('Last 6 Hours')),
    DropdownMenuItem(value: '-12h', child: Text('Last 12 Hours')),
  ],
  onChanged: (value) {
    if (value != null) {
      setState(() => selectedRange = value);
      fetchPressureData();
    }
  },
),  // DropdownButton
```

Figure 6.64: Implementation of Time Range Selection.

On top of the selected graphs, there is a **data summary panel** that shows the latest readings from all the device channel the user selected. First, it displays the timestamp (commonTime) to indicate when the data was last updated. Below the time, it lists each selected device (from selectedGraphs) along with its latest value and corresponding unit (retrieved from fieldUnits). If a device has no recent data, it shows “Offline” instead.

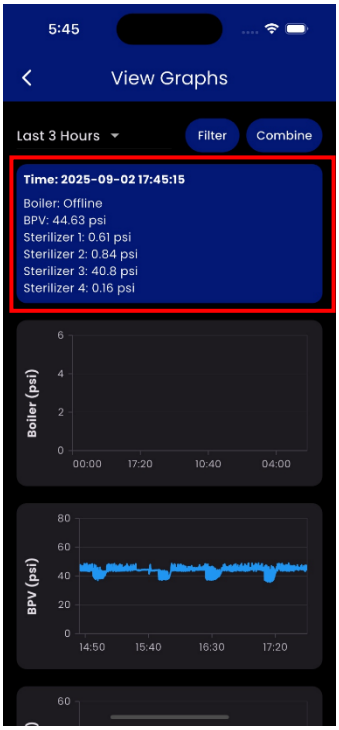


Figure 6.65: Data Summary Panel.

```
Card(
  color: const Color(0xFF031774),
  margin: const EdgeInsets.symmetric(vertical: 10),
  child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text("Time: $commonTime",
            style: const TextStyle(
                color: Colors.white, fontWeight: FontWeight.bold)),
        const SizedBox(height: 8),
        ...selectedGraphs.map((field) {
          final data = chartDataMap[field];
          final unit = fieldUnits[field] ?? '';
          final value = data?.isNotEmpty == true
              ? "${data!.last.pressure} $unit"
              : "Offline";
          return Text(
            "$field: $value",
            style: const TextStyle(color: Colors.white),
          );  // Text
        }).toList(),
      ],
    ),  // Column
  ),  // Padding
),  // Card
```

Figure 6.66: Implementation of Data Summary Panel.

6.3.4 Alarm Management Module

As discussed in section ***6.2.2 MySQL Cloud Database Setup***, all device threshold settings are configured in the MySQL database through phpMyAdmin. The monitoring script is written in Python and is responsible for continuously reading each channel data received from the Influx DB. The script is deployed as a **systemd** service, ensuring that it runs 24/7 in the background without manual intervention.

Once channel data is collected, the script performs a comparison between the real-time values from Influx DB and the threshold limits stored in MySQL. If any parameter crosses its configured boundary, the alarm logic is triggered. The triggered alarm is then logged back into the database with essential details such as alarm code, alarm triggered time, alarm triggered value and its description.

Firstly, the Python script will dynamically retrieve all active threshold rules **(alarm= 'yes')** for each device from tables ending with **_threshold_settings**.

```
threshold_cursor = mysql_conn.cursor()
threshold_cursor.execute(f"SELECT * FROM {threshold_table} WHERE LOWER(alarm)='yes'")
thresholds = threshold_cursor.fetchall()
threshold_cursor.close()
mysql_conn.commit()
```

Figure 6.67: Check alarm = 'yes'.

For each threshold, the script queries the most recent data within last 5 minutes.

```
# === Function to query InfluxDB ===
def query_influx(bucket, measurement, site_id, channel_key):
    query = f'''
    from(bucket: "{bucket}")
    |> range(start: -5m)
    |> filter(fn: (r) => r["_measurement"] == "{measurement}")
    |> filter(fn: (r) => r["id"] == "{site_id}")
    |> filter(fn: (r) => r["_field"] == "{channel_key}")
    |> last()
    '''
    result = query_api.query(org="Novaflow", query=query)
    for table in result:
        for record in table.records:
            return record.get_value()
    return None
```

Figure 6.68: Query Data from Influx DB.

The real-time value is compared with the low and high limits that configured in MySQL.

```
alarm_triggered = False
alarm_type = None
if value < threshold_low:
    alarm_triggered = True
    alarm_type = "below"
elif value > threshold_high:
    alarm_triggered = True
    alarm_type = "exceed"
```

Figure 6.69: Check Low and High Thresholds.

If an alarm is triggered and no active alarm exists for the channel, a new record is inserted into the corresponding **_alarm_history table** with details such as channel name, variable, timestamps, description, and last value.

```
mysql_cursor.execute(f"""
    INSERT INTO {alarm_table}
    (channel_name, variable, start_time, description, `last_value`, `acknowledge`)
    VALUES (%s, %s, %s, %s, %s, %s)
""", (channel_name, variable, now_time, alarm_text, value, "No"))
mysql_conn.commit()
```

Figure 6.70: Insert Alarm Record into Database.

The active alarms are temporary inserted into the **active_alarms collection** in Firebase, ensuring that the active alarms are immediately visible in the mobile application.

```
firebase_db.collection("active_alarms").add({
    "mysql_id": alarm_id,
    "site_id": site_id,
    "channel_name": channel_name,
    "variable": variable,
    "description": alarm_text,
    "start_time": now_time,
    "last_value": value,
    "status": "active",
})
print(f"[{db_name}] ✅ Alarm inserted to Firebase.")
```

Figure 6.71: Adding Active Alarm to Firebase.

If the value returns to the normal range and an active alarm exists, the system updates the alarm record with an **end_time** in MySQL and removes the corresponding entry from Firebase.

```
mysql_cursor.execute(f"""
    UPDATE {alarm_table} SET end_time=%s WHERE id=%s
""", (now_time, alarm_id))
mysql_conn.commit()

query = firebase_db.collection("active_alarms")\
    .where("mysql_id", "==", alarm_id).limit(1).stream()

for doc in query:
    doc.reference.delete()
```

Figure 6.72: Remove Active Alarm Record from Firebase.

This loop run continuously, ensuring that alarms are check every seconds. By deploying the script as a **systemd service**, as shown in Figure 6.74 below, the process operates in the background 24/7 without the need for manual intervention and automatically restarts if it crashes.

```
# === Monitoring Loop ===
while True:
    print(f"\n[INFO] Running monitoring loop at {datetime.now()}")

    try:
        mysql_conn.ping(reconnect=True)
    except Exception as e:
        print(f"[ERROR] MySQL reconnect failed: {e}")
        time.sleep(5)
        continue
```

Figure 6.73: Monitoring Loop.

```
ubuntu@milldb: ~
ubuntu@milldb:~$ sudo systemctl status alarms2db
● alarms2db.service - Alarms to Database Monitor Service
     Loaded: loaded (/etc/systemd/system/alarms2db.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-09-05 10:42:58 +08; 1 week 1 day ago
   Main PID: 1177 (python3)
      Tasks: 16 (limit: 18791)
     Memory: 105.6M (peak: 107.6M)
        CPU: 1h 52min 37.449s
     CGroup: /system.slice/alarms2db.service
             └─1177 /home/ubuntu/Scripts/almenv/bin/python3 /home/ubuntu/Scripts/alarms2db.py

Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Thresholds - Low: 0.0, High: 90.0, Value: 73.52
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] ✔ Normal value, no active alarm.
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Checking Press 1 (ch5)...
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Influx value for ch5 = 0.0
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Thresholds - Low: 0.0, High: 50.0, Value: 0.0
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] ✔ Normal value, no active alarm.
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Checking Press 2 (ch6)...
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Influx value for ch6 = 0.0
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] Thresholds - Low: 0.0, High: 50.0, Value: 0.0
Sep 14 09:40:03 milldb python3[1177]: [POM_UnitedTradeTeam] 🔲 Clearing alarm with ID 47253
ubuntu@milldb:~$
```

Figure 6.74: Comparing Thresholds Python running on Systemd.

For the frontend part, the active alarm record that displays on the Active Alarm page is implemented through the **fetchActiveAlarms** function. This function is used to load and display the latest active alarms for a specific site. When called, it queries the Firebase collection **active_alarms**, filtering records by the current **site's ID (site_id)** and only retrieving alarms with a status of active. If alarms are found, each documents' data is processed to extract the MySQL **alarm code (mysql_id), start time, and description**. The start time is parsed into a DateTime object, formatted into a readable string (yyyy-MM-dd HH:mm:ss). The alarms are sorted in descending order by their start time so that the **latest alarm appears on top** first. If no active alarms are found, a short message **"No active alarm found"** is returned as shown in Figure 6.76.

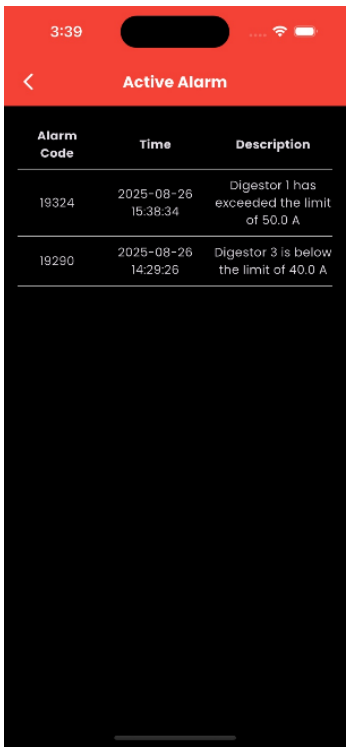


Figure 6.75: Active Alarm Page.

Figure 6.76: No Active Alarm Record.

```
Future<void> fetchActiveAlarms() async {
  setState(() {
    isLoading = true;
    hasError = false;
  });

  try {
    QuerySnapshot snapshot = await FirebaseFirestore.instance
        .collection('active_alarms')
        .where('site_id', isEqualTo: widget.id)
        .where('status', isEqualTo: 'active')
        .get();

    if (snapshot.docs.isNotEmpty) {
      activeAlarms = snapshot.docs.map((doc) {
        final data = doc.data() as Map<String, dynamic>;

        DateTime parsedDate = DateTime.tryParse(data['start_time'] ?? '') ?? DateTime.now();
        String formattedTime = DateFormat('yyyy-MM-dd\nHH:mm:ss').format(parsedDate);

        return {
          'code': data['mysql_id']?.toString() ?? '',
          'time': formattedTime,
          'description': data['description']?.toString() ?? '',
          'parsedTime': parsedDate,
        };
      }).toList();
```

Figure 6.77: Implementation of Fetching Active Alarm.

In Alarm History page, the **fetchAlarms()** function is implemented to retrieve alarm history data from the **backend API (/api/alarms/{site_id}).** Once the JSON responses are received, it validates the format by parses each alarm's **start_time** into local time, and prepares a clean record with alarm code, formatted timestamps known as alarm triggered time, description

message, and acknowledgement status. The alarms are then sorted in descending order by time (latest first) and passed to **_applyFilter()** for filtering before updating the UI.

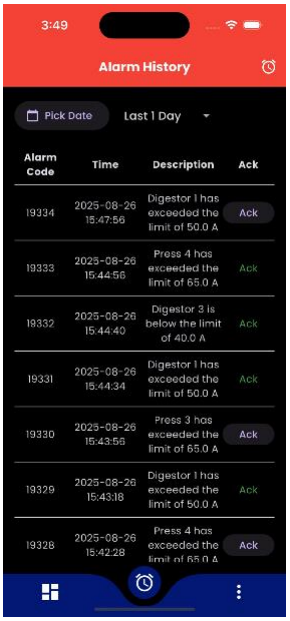


Figure 6.78: Alarm History Page.

```
Future<void> fetchAlarms() async {
  try {
    final url = 'https://sm365.novaplus.my/api/alarms/${widget.id}';
    print('🌐 Fetching alarms from: $url');

    final response = await http.get(Uri.parse(url));
    print('📦 Status code: ${response.statusCode}');

    if (response.statusCode == 200) {
      final decoded = json.decode(response.body);
      print('✅ Raw JSON decoded: $decoded');

      if (decoded is! List) {
        throw Exception('Unexpected format: Expected a list, got ${decoded.runtimeType}');
      }

      final List<dynamic> data = decoded;

      final filtered = data.map((alarm) {
        final parsedDate = DateFormat("EEE, dd MMM yyyy HH:mm:ss zzz")
            .parse(alarm['start_time'], true)
            .toLocal()
            .subtract(const Duration(hours: 8));
        final formattedDate = DateFormat("yyyy-MM-dd HH:mm:ss").format(parsedDate);

        return {
          'id': alarm['id'],
          'start_time': formattedDate,
          'description': alarm['description'].toString(),
          'acknowledged': alarm['acknowledge'] == true || alarm['acknowledge'] == 1,
          'parsedDate': parsedDate,
```

Figure 6.79: Implementation of Fetching Alarm History.

Users can filter the loaded alarms based on their preferences through the **_applyFilter()** function. If the user has picked a specific date, only alarms from the exact date are shown. Otherwise, it applies a time-range filter such as “Last 1,7, or 30 Days”.

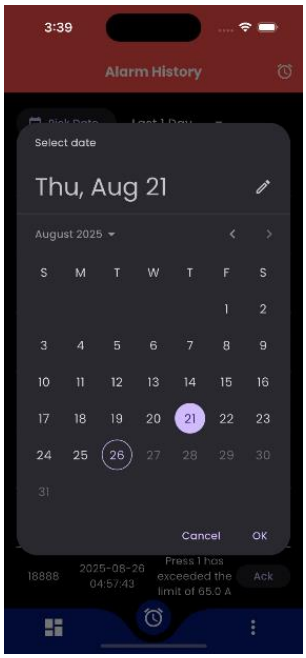


Figure 6.80: Alarm Record Filter by Date.

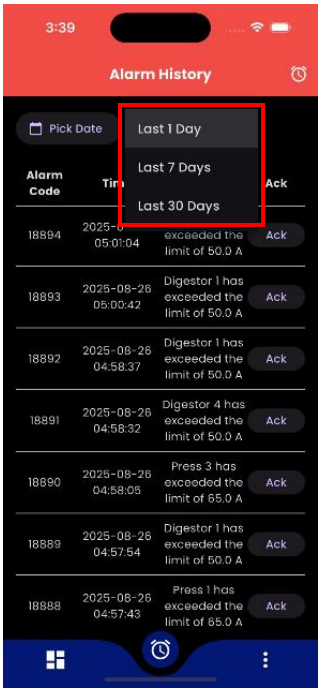


Figure 6.81: Alarm Record Filter by Time Range.

```
void _applyFilter() {
  List<Map<String, dynamic>> filtered;

  if (pickedDate != null) {
    filtered = allAlarms.where((alarm) {
      DateTime alarmDate = DateTime.parse(alarm['start_time']);
      return alarmDate.year == pickedDate!.year &&
          alarmDate.month == pickedDate!.month &&
          alarmDate.day == pickedDate!.day;
    }).toList();
  } else {
    DateTime now = DateTime.now();
    DateTime cutoff;

    if (selectedTimeRange == 'Last 1 Day') {
      cutoff = now.subtract(const Duration(days: 1));
    } else if (selectedTimeRange == 'Last 7 Days') {
      cutoff = now.subtract(const Duration(days: 7));
    } else if (selectedTimeRange == 'Last 30 Days') {
      cutoff = now.subtract(const Duration(days: 30));
    } else {
      cutoff = now.subtract(const Duration(days: 1));
    }

    filtered = allAlarms.where((alarm) {
      DateTime alarmDate = DateTime.parse(alarm['start_time']);
      return alarmDate.isAfter(cutoff);
    }).toList();
  }
```

Figure 6.82: Implementation of Filtering Alarm History.

The **_clearDateFilter()** function resets the alarm history filter back to its default state by clearing any previously chosen date (pickedDate = null) and reselecting the default time range which is Last 1 Day.

```
void _clearDateFilter() {
  setState(() {
    pickedDate = null;
    selectedTimeRange = 'Last 1 Day';
  });
  _applyFilter();
}
```

Figure 6.83: Implementation of Clearing Filter.

The Ack function located in the fourth column of alarm history table is implemented through **_acknowledgeAlarm** function. This function is responsible for marking a specific alarm as acknowledged in both the MySQL database and in the mobile application. It retrieves the alarm' ID, the logged-in user's email, and the site ID from the alarms list, then send this information as a **JSON** format in an **HTTP POST request** to the **backend API endpoint (/ack_alarm/{alarmId}).** If the API responds with a **200 status code**, the function updates the alarm's state in the app via setState() by setting its

acknowledged field to true. As a result, the Ack button turn green, indicating that the alarm has been acknowledged by the user. At the same time, the **ack_user column in MySQL database** is updated with the email of user who performed the acknowledgement, ensuring that the action is recorded for reference in future.

```
void _acknowledgeAlarm(int index) async {
  final alarmId = alarms[index]['id'];
  final userEmail = widget.userEmail;
  final siteId = alarms[index]['site_id']; // ✅ now exists in each alarm

  print("📩 Acknowledging alarm $alarmId by $userEmail at site $siteId");

  try {
    final response = await http.post(
      Uri.parse('https://sm365.novaplus.my/api/ack_alarm/$alarmId'),
      headers: {'Content-Type': 'application/json'},
      body: jsonEncode({
        'user_email': userEmail,
        'site_id': siteId,
      }),
    );

    if (response.statusCode == 200) {
      setState(() {
        alarms[index]['acknowledged'] = true;
      });
    } else {
      print('🔴 Ack failed: ${response.body}');
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Failed to acknowledge alarm')),
      );
    }
```

Figure 6.84: Implementation of Ack Function.

Since **Flutter cannot directly connect to a MySQL** database, **Python with Flask** is used as the backend layer to handle data retrieval and updates. Both backend APIs for the **Alarm History** (/api/alarms/{site_id}) and the **Acknowledgement** (/ack_alarm/{alarmId}) are implemented. The implementation of **Alarm History API** shown in Figure 6.85 queries the MySQL databases to locate the corresponding **{site_id} _alarm_history table**, retrieved all alarms sorted by start time, and returns them as JSON to the mobile app.

The implementation **Acknowledgement API** shown in Figure 6.86 is triggered when a user acknowledges an alarm in the app by clicking the Ack button, it locates the alarm record in the MySQL table, then updates it acknowledge column to ‘Yes’ and saves the acknowledging user’s email in the

ack_user column. Both API are deployed as **systemd services** as shown in Figure 6.87.

```python
@app.route('/alarms/<site_id>', methods=['GET'])
def get_alarm_history_by_site(site_id):
    table_name = f"{site_id}_alarm_history"
    debug_log = []

    try:
        base_conn = get_base_connection()
        base_cursor = base_conn.cursor()
        base_cursor.execute("SHOW DATABASES")
        databases = [db[0] for db in base_cursor.fetchall() if db[0].startswith("POM_")]
        base_cursor.close()
        base_conn.close()
        debug_log.append(f"Databases found: {databases}")

        for db_name in databases:
            conn = get_db_connection(db_name)
            cursor = conn.cursor()
            cursor.execute("SHOW TABLES")
            tables = [t[0] for t in cursor.fetchall()]
            debug_log.append(f"In {db_name}, tables: {tables}")
            if table_name in tables:
                cursor.close()
                conn.close()

                # Connect again with dict cursor to fetch actual records
                conn2 = get_db_connection(db_name)
                cursor2 = conn2.cursor(dictionary=True)
                cursor2.execute(f"""
                    SELECT * FROM {table_name}
                    ORDER BY start_time DESC
                """)
                rows = cursor2.fetchall()
                cursor2.close()
                conn2.close()

                for row in rows:
                    if 'acknowledge' in row:
                        row['acknowledge'] = (str(row['acknowledge']).lower() == 'yes')

                return jsonify(rows)

            cursor.close()
            conn.close()

        return jsonify({
            "error": f"Table {table_name} not found in any POM_ database",
            "debug": debug_log
        }), 404

    except Exception as e:
        return jsonify({"error": str(e), "debug": debug_log}), 500
```

Figure 6.85: Alarm History API.

```python
@app.route('/ack_alarm/<alarm_id>', methods=['POST'])
def acknowledge_alarm(alarm_id):
    data = request.json
    user_email = data.get('user_email')
    site_id = data.get('site_id')

    if not user_email or not site_id:
        return jsonify({"error": "user_email and site_id required"}), 400

    try:
        alarm_id = int(alarm_id)
    except ValueError:
        return jsonify({"error": "Invalid alarm_id"}), 400

    try:
        base_conn = get_base_connection()
        base_cursor = base_conn.cursor()
        base_cursor.execute("SHOW DATABASES")
        databases = [db[0] for db in base_cursor.fetchall()]
        base_cursor.close()
        base_conn.close()

        for db_name in databases:
            if not db_name.startswith("POM_"):
                continue  # Only process relevant databases

            print(f"🔍 Checking database: {db_name}")
            conn = get_db_connection(db_name)
            cursor = conn.cursor()

            cursor.execute("SHOW TABLES")
            tables = [t[0] for t in cursor.fetchall() if t[0].endswith("_alarm_history")]

            for table in tables:
                if site_id not in table:
                    continue  # Skip unrelated site

                print(f"🔍 Searching table: {table}")
                cursor.execute(f"SELECT 1 FROM `{table}` WHERE id = %s", (alarm_id,))
                if cursor.fetchone():
                    # Found the correct record
                    cursor.execute(f"""
                        UPDATE `{table}`
                        SET acknowledge = 'Yes', ack_user = %s
                        WHERE id = %s
                    """, (user_email, alarm_id))
                    conn.commit()
                    print(f"✅ Updated alarm {alarm_id} in table {table}")
                    cursor.close()
                    conn.close()
                    return jsonify({"success": True}), 200

                cursor.close()
```

Figure 6.86: Acknowledgment API.

```
ubuntu@mill365: ~
ubuntu@mill365:~$ sudo systemctl status apialarm
● apialarm.service - Gunicorn instance to serve apialarm Flask app
     Loaded: loaded (/etc/systemd/system/apialarm.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-09-04 14:04:29 UTC; 1 week 2 days ago
   Main PID: 1185 (gunicorn)
      Tasks: 4 (limit: 38075)
     Memory: 134.1M (peak: 184.3M)
        CPU: 3min 9.630s
     CGroup: /system.slice/apialarm.service
             ├─1185 /opt/mill365/v_apialarm/bin/python3 /opt/mill365/v_apialarm/bin/gunicorn --workers 3 --bind 0.0.0.0>
             ├─1221 /opt/mill365/v_apialarm/bin/python3 /opt/mill365/v_apialarm/bin/gunicorn --workers 3 --bind 0.0.0.0>
             ├─1236 /opt/mill365/v_apialarm/bin/python3 /opt/mill365/v_apialarm/bin/gunicorn --workers 3 --bind 0.0.0.0>
             └─1238 /opt/mill365/v_apialarm/bin/python3 /opt/mill365/v_apialarm/bin/gunicorn --workers 3 --bind 0.0.0.0>

Sep 04 14:04:29 mill365 systemd[1]: Started apialarm.service - Gunicorn instance to serve apialarm Flask app.
Sep 04 14:04:29 mill365 gunicorn[1185]: [2025-09-04 14:04:29 +0000] [1185] [INFO] Starting gunicorn 23.0.0
Sep 04 14:04:29 mill365 gunicorn[1185]: [2025-09-04 14:04:29 +0000] [1185] [INFO] Listening at: http://0.0.0.0:5001 (11>
Sep 04 14:04:29 mill365 gunicorn[1185]: [2025-09-04 14:04:29 +0000] [1185] [INFO] Using worker: sync
Sep 04 14:04:29 mill365 gunicorn[1221]: [2025-09-04 14:04:29 +0000] [1221] [INFO] Booting worker with pid: 1221
Sep 04 14:04:29 mill365 gunicorn[1236]: [2025-09-04 14:04:29 +0000] [1236] [INFO] Booting worker with pid: 1236
Sep 04 14:04:29 mill365 gunicorn[1238]: [2025-09-04 14:04:29 +0000] [1238] [INFO] Booting worker with pid: 1238
lines 1-20/20 (END)
```

Figure 6.87: APIs Python Running on Systemd.

The pop-up notification shown in Figure 6.88 is triggered when a new alarm with matching site_id is added into the Firestore **active_alarms** collection. This function prevents duplicate notifications by checking against a global list of already-notified alarms (globalNotifiedAlarmIds). When a new alarm with a matching site_id is detected, it triggers **showAlarmNotification** with the alarm's description, ensuring users are alerted in real time about critical events.

Figure 6.88: Pop Up Notification.

```
for (var docChange in snapshot.docChanges) {
  if (docChange.type == DocumentChangeType.added) {
    final data = docChange.doc.data() as Map<String, dynamic>?;
    if (data == null) continue;

    final siteId = (data['site_id'] as String?)?.trim();
    final description = data['description'] ?? 'New Alarm Triggered';
    final alarmId = docChange.doc.id;

    if (siteId == id && !globalNotifiedAlarmIds.contains(alarmId)) {
      showAlarmNotification('Active Alarm', description);
      globalNotifiedAlarmIds.add(alarmId);
    }
  }
}
```

Figure 6.89: Implementation of Pop-Up Notification.

6.3.5 User Profile & Settings Module

Users can navigate to “More” Page to perform additional system settings. This page serves as a general menu hub in the app, showing the currently logged-in users’ email at the top. It provides navigation options to different sections like **About Us, FAQ, Data Plotter, Privacy Policy, Settings**, and a **Logout** button, each implemented using **buildMenuOption** to wrap icons, labels, and navigation logic. Most options navigate to internal pages using **Navigator.push**, while the **Data Plotter** options open an external URL in a browser via launchURL. The **About Us, FAQ, and Privacy Policy** pages are implemented by hardcoding the information provided by the company directly into the pages.

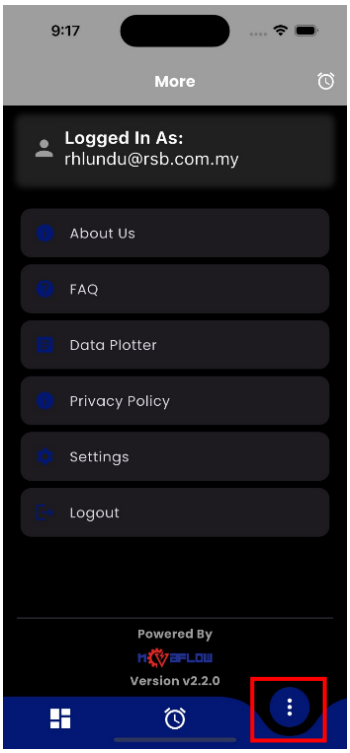


Figure 6.90: More Page.

```
text: TextSpan(
  text: 'Logged In As: ',
  style: TextStyle(
    fontWeight: FontWeight.bold,
    fontSize: 20,
    color: textColor,
  ),  // TextStyle
  children: [
    TextSpan(
      text: _userEmail ?? 'Logging in...',
      style: TextStyle(
        fontWeight: FontWeight.normal,
        fontSize: 20,
        color: textColor,
      ),  // TextStyle
    ),  // TextSpan
```

Figure 6.91: Implementation of Current Login Email.

```
buildMenuOption(Icons.info, 'About Us', () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const AboutPage()),
  );
}),
buildMenuOption(Icons.help, 'FAQ', () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const FAQPage()),
  );
}),
buildMenuOption(Icons.article, 'Data Plotter', () {
  launchURL('https://novaflow-dataplotter-a495e.web.app/');
}),
buildMenuOption(Icons.info, 'Privacy Policy', () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const PrivacyPolicyPage()),
  );
}),
buildMenuOption(Icons.settings, 'Settings', () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const SettingsPage()),
  );
}),
buildMenuOption(Icons.logout, 'Logout', () {
  Logout.confirmLogout(context);
}),
```

Figure 6.92: Implementation of Navigation to Different Sections.

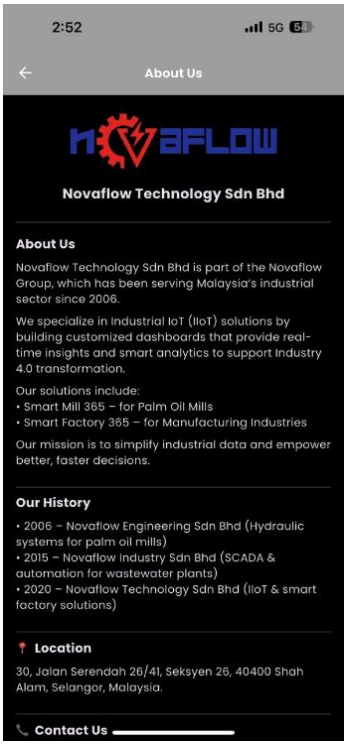


Figure 6.93: About Us Page.

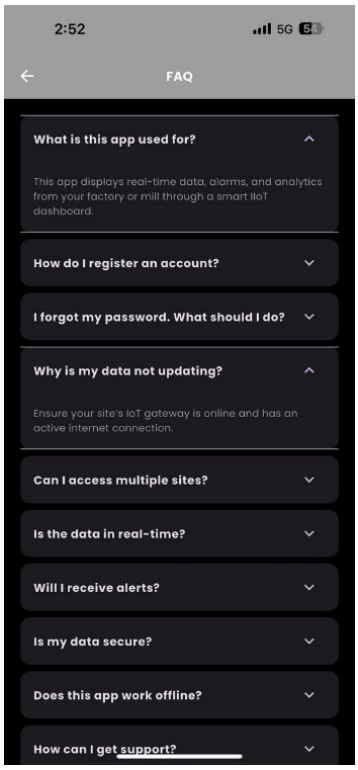


Figure 6.94: FAQ Page.

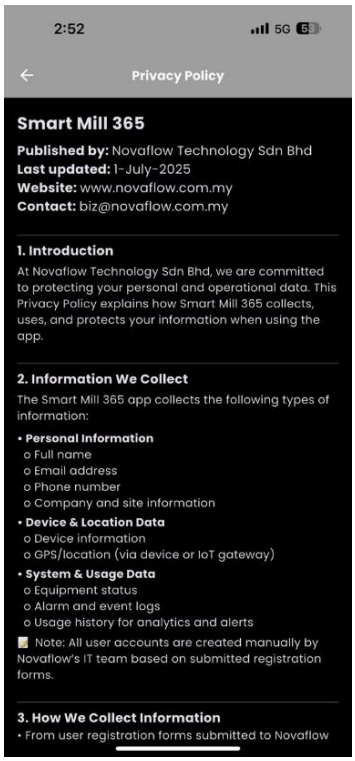


Figure 6.95: Privacy Policy Page.

In the **settings** page, there is a **Dark Mode** section. This section creates three radio buttons such as Dark, Light, and System, allowing users to select the app's theme. Each option uses **value** to identify the theme, **groupValue** to show the currently selected option, and **onChanged**: **themeProvider.setThemeMode** to update the app's theme in real time when a selection is made, with the System option following the device's theme settings. The reason of placing Dark Mode in the Settings tab instead of directly on the More page is to keep the More page clean and organized, while also allowing flexibility for adding more settings in the future without cluttering the main menu.

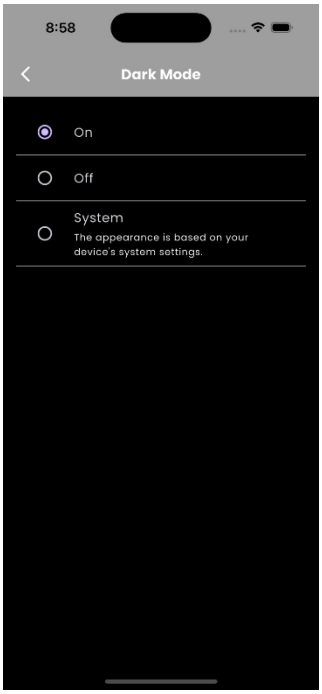


Figure 6.96: Dark Mode Setting Page.

```
buildRadioOption(
  title: 'On',
  value: ThemeModeOption.dark,
  groupValue: currentOption,
  onChanged: themeProvider.setThemeMode,
),
buildRadioOption(
  title: 'Off',
  value: ThemeModeOption.light,
  groupValue: currentOption,
  onChanged: themeProvider.setThemeMode,
),
buildRadioOption(
  title: 'System',
  subtitle: 'The appearance is based on your device\'s system settings.',
  value: ThemeModeOption.system,
  groupValue: currentOption,
  onChanged: themeProvider.setThemeMode,
),
```

Figure 6.97: Implementation of Dark Mode Setting.

The **Data Plotter** option in the More page provides an additional function that navigates to an **external URL** using **launchURL ('https://novaflow-dataplotter-a495e.web.app/')** as shown in Figure 6.99.

Figure 6.99: Data Plotter.

```
buildMenuOption(Icons.article, 'Data Plotter', () {
  launchURL('https://novaflow-dataplotter-a495e.web.app/');
}),
```

Figure 6.99: Implementation of Data Plotter.

Lastly is the Logout function. Users can logout the system by pressing the **Logout button** at the More page. The implemented function provides a secure and user-friendly way for users to sign out of the app. It first shows a confirmation dialog as shown in Figure 6.100 to prevent accidentally logouts, and if user confirms (Yes), it signs them out from **Firebase Authentication** and redirects them to the Login page.

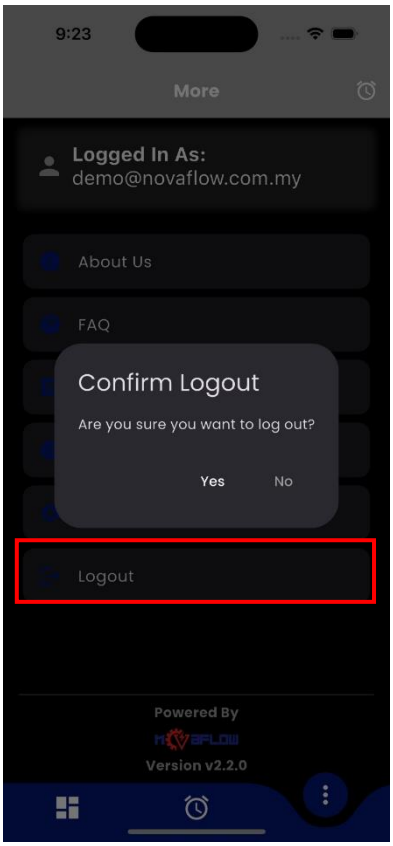


Figure 6.100: Confirmation Dialog for Logout.

```
static Future<void> _logout(BuildContext context) async {
  try {
    await FirebaseAuth.instance.signOut();

    // Navigate back to login page and remove previous screens
    Navigator.pushAndRemoveUntil(
      context,
      MaterialPageRoute(builder: (context) => LoginPage()),
          (route) => false,
    );
```

Figure 6.101: Implementation of Logout.

6.4 Conclusion

In conclusion, this chapter detailed the **implementation** of the **SmartMill365 system**, from Firebase and MySQL setup to the development of five modules such as Login, Dashboard, Graph Monitoring, Alarm Management, and User Profile & Settings. The system is now fully functional, supporting real-time monitoring and alarm notifications. The following chapter will focus on system testing.

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

In this chapter, system testing is carried out to ensure that both functional and non-functional requirements of the system are fulfilled. The testing process includes **unit testing**, which verifies the correctness of individual modules, and **usability testing** used to evaluate overall user experience and ease of use of the system. In addition to unit and usability testing, **alpha testing** was conducted with internal teams, while **beta testing** was carried out with external users on deployed builds for both Android and iOS platforms.

The unit testing process described in section 7.2 is conducted module by module. An overview of the test cases for each module and their corresponding results are shown below.

7.2 Unit Testing

Table 7.1: Unit Testing of Login Module.

Test Module	Log In Module		**Test Title**	Log in to the user account	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-001	Log in with valid credentials	1. Enter the registered email and password 2. Click Sign In button	i. Registered email ii. Valid password	Redirected to the dashboard page	**Pass**
UNT-002	Log in with invalid email	1. Enter the wrong email format and password 2. Click Sign In button	i. Invalid email ii. Password	Display re-enter valid email address	**Pass**
UNT-003	Log in with valid email and invalid password	1. Enter the registered email but invalid password 2. Click Sign In button	i. Registered email ii. Invalid password	Display incorrect email or password	**Pass**
UNT-004	Log in with empty	1. Click Sign In button	No test data	Display email and password cannot be	**Pass**

	email and password			empty	

Table 7.2: Unit Testing of the Dashboard Module.

Test Module	Dashboard Module		**Test Title**	Displaying relevant device channel data on the dashboard	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-005	Registered email with Firebase collection and device settings configuration	1. Navigate to the dashboard page	Account with correct device settings	All the available device channel data are displayed accordingly in the dashboard page	**Pass**
UNT-006	Registered email without Firebase collection	1. Navigate to the dashboard page	No test data	Display no configuration found for this account	**Pass**
UNT-007	Registered email with Firebase collection	1. Navigate to the dashboard page	No test data	Display no device settings configured	**Pass**

	but no device settings configuration				
UNT-008	Switch subgroup and verify correct dashboard	1. Navigate to the dashboard page 2. Click on the subgroup dropdown selector 3. Select different subgroup from the list	A manager account with few subgroups	The dashboard updates the device channel data relevant to the newly selected subgroup	**Pass**
UNT-009	Channel data live update (most recent data)	1. Navigate to the dashboard page 2. Observe all value display on each device	Account with active device	The value on each device updates and display the most recent value (without manual page refresh) with green colour dot indicator	**Pass**
UNT-010	Inactive device shows “Offline” status	1. Navigate to the dashboard page 2. Observe inactive device	Account with inactive device	The device’s status indicator is shown as “Offline” with red colour dot indicator	**Pass**
UNT-011	Subgroup alarm icon	1. Navigate to the	1. A subgroup with at	1. For subgroup with at least one	**Pass**

	indicates active alarm status	dashboard page 2. Observe the alarm icon next to each subgroup name in the selection list	least one active alarm 2. A subgroup with no active alarm	active alarm, the alarm icon is red 2. For subgroup with no active alarm, the alarm icon is grey	
UNT-012	Correct quantity and name of subgroup is shown	1. Navigate to the dashboard page 2. Observe the list of subgroups in the dropdown menu	A manager account with few subgroups	The dropdown list displays the exact quantity of subgroups that configured in Firebase. Each subgroup's name is displayed correctly and matches the configured data.	**Pass**
UNT-013	"Last Updated At" timestamp shows most recent data update time	1. Navigate to the dashboard page 2. Observe the "Last Update At" timestamp	Account with active device	The "Last Updated At" timestamp updates follow the time of the most recent device channel data	**Pass**

Table 7.3: Unit Testing of the Graph Monitoring Module.

Test Module	Graph Monitoring Module		**Test Title**	Displaying device channel data in graph view	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-014	Display graph with correct device channel data	1. Navigate to the dashboard page 2. Select a device channel from the dashboard	Device with valid channel data in Firebase	Graph displays the correct data points according to the selected channel	**Pass**
UNT-015	Data range filter shows correct graph data	1. Navigate to the dashboard page 2. Select a device channel from the dashboard 3. Apply time range filter	Device with historical data	Graph updates and only shows data within the selected time range	**Pass**
UNT-016	Graph zoom and pan function works correctly	1. Navigate to the dashboard page	Active device channel with data	Graph zooms and pans as expected. When time range is changed, graph	**Pass**

		2. Select a device channel from the dashboard 3. Zoom into a section of the graph 4. Pan the graph left or right 5. Change the time range filter		resets back to normal view	
UNT-017	Tooltip displays correct info when clicking on data point	1. Navigate to the dashboard page 2. Select a device channel from the dashboard 3. Click on any data point in the graph	Active device channel with data	Tooltip appears showing correct timestamp, device name, value and unit	**Pass**
UNT-017	Correct device name and	1. Navigate to the	Device settings	Graph title, axis and tooltip show the	**Pass**

	unit shown from Firebase configuration	dashboard page 2. Select a device channel from the dashboard 3. Observe device name and unit on graph labels	configured in Firebase	correct device name and unit	
UNT-018	Multi-channel selection (combine graph view)	1. Navigate to View Graphs page 2. Select 2 or more channels	Multiple active device channels	Graph displays multiple lines correctly distinguished by legend	**Pass**
UNT-019	Multi-channel selection (separate graph view)	1. Navigate to View Graphs page 2. Select 2 or more channels	Multiple active device channels	Each selected channel is displayed in its own graph panel, with correct title and axis	**Pass**
UNT-020	Data summary panel displays timestamps and latest values for each	1. Navigate to View Graphs page 2. Select 2 or more	Multiple active device channels	Data summary panel shows the most recent timestamp and each selected channel's latest value with correct unit	**Pass**

	selected channel correctly	channels 3. Observe the data summary panel above the graph			
UNT-020	Data summary panel shows "Offline" for selected channel of an offline device	1. Navigate to View Graphs page 2. Select 2 or more channels 3. Observe the data summary panel above the graph	Device configured in Firebase but currently inactive/offline	Data summary panel displays the device name with status shown as "Offline" instead of value	**Pass**
UNT-021	Selected device channel is removed from graph after unchecking checkbox in filter option (combine and separate graph view)	1. Navigate to View Graphs page 2. Select 2 or more channels 3. Uncheck 1 of the previously selected channels	Multiple device channels	The unchecked channel is removed from both the graph and the data summary panel immediately, while other selected channels remain visible	**Pass**

Table 7.4: Unit Testing of the Alarm Management Module.

Test Module	Alarm Management Module		**Test Title**	Displaying, updating and managing device alarms	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-022	Display list of active alarms	1. Navigate to Active Alarm page 2. Observe alarm list	Device that exceeds or below predefined thresholds	Alarm list displays all currently active alarms with correct alarm code, timestamp, and description	**Pass**
UNT-023	Display "No active alarm found" when no alarms exist	1. Navigate to Active Alarm page	Device without active alarms (normal condition)	Active Alarm page shows "No active alarm found" instead of empty list	**Pass**
UNT-024	Alarm status changes to active when threshold exceeded	1. Configure a threshold 2. Simulate channel	Device with threshold configured	Alarm appears in active list with correct details (alarm code, time, description)	**Pass**

		data exceed threshold 3. Refresh Active Alarm page			
UNT-025	Alarm status changes to cleared when condition returns to normal	1. Trigger an alarm 2. Simulate channel data returning to normal	Device with threshold configured	Alarm is removed from active list and only show in history page	**Pass**
UNT-026	Alarm history log displays cleared information	1. Navigate to Alarm History Page 2. Review past alarms	Device with previously cleared alarms	History page lists all relevant past alarms with correct alarm code, timestamp, description and acknowledge status	**Pass**
UNT-027	Alarm acknowledgement works correctly	1. Navigate to Alarm History Page 2. Select an alarm record and click the Ack button	Record in alarm history list	Ack button immediately turns into green colour	**Pass**
UNT-028	Alarm filtering works	1. Navigate to Alarm	Record in alarm	Only alarms on the selected date are	**Pass**

	correctly (Filter by Date)	History Page 2. Apply filter by date	history list	displayed	
UNT-029	Alarm filtering works correctly (Filter by Time Range)	1. Navigate to Alarm History Page 2. Apply filter by time range	Record in alarm history list	Only alarms on the selected time ranges are displayed	**Pass**
UNT-030	No alarm history for selected time range / date	1. Navigate to Alarm History Page 2. Apply filter by time range / date 3. Ensure no alarm exits on the selected time range / date 4. Observe history list	No test data	Alarm history page displays “No alarm history found” for the selected time range / date	**Pass**

Table 7.5: Unit Testing of the User Profile & Settings Module.

Test Module	User Profile & Settings Module		**Test Title**	Displaying user profile and settings	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-031	Display user email correctly	1. Navigate to More page 2. Observe the current log in email	Registered user with email	Profile page shows correct email	**Pass**
UNT-032	Able to navigate to About Us page	1. Navigate to More page 2. Select About Us	No test data	System redirects to About Us page successfully	**Pass**
UNT-033	Able to navigate to FAQ page	1. Navigate to More page 2. Select FAQ	No test data	System redirects to FAQ page successfully	**Pass**
UNT-034	Able to navigate to Privacy Policy page	1. Navigate to More page 2. Select Privacy Policy	No test data	System redirects to Privacy Policy page successfully	**Pass**

UNT-035	Able to change theme mode	1. Navigate to More page and select Settings 2. Select Dark Mode and choose either on/off or follow system theme	No test data	System update theme mode according to user selection	**Pass**
UNT-036	Able to remember theme mode	1. Set theme to Dark mode 2. Quit the app and open again 3. Observe the current theme	No test data	Dark mode remains applied after close the app	**Pass**
UNT-037	Able to navigate to external URL (Data Plotter)	1. Navigate to More page 2. Select Data Plotter	No test data	Browser successfully open external Data Plotter URL	**Pass**
UNT-038	Able to logout successfully	1. Navigate to More page	No test data	User is logged out and redirect to Login page	**Pass**

		2. Click Logout button 3. Confirm action			

7.3 System Usability Testing

In this project, System Usability Scale (SUS) is chosen to evaluate the usability of the palm oil mill system. SUS is a widely used method that provides a quick and reliable measure of system usability through a standardized 10 questionnaire. This help to capture users' perceptions of the system in terms of effectiveness, efficiency and satisfaction. The SUS evaluation in this project was conducted with selected respondents by Nova flow to assess how intuitive and user-friendly the implemented system so that the 4th objective in this project is achieved.

Table 7.6: Template of System Usability Scale (SUS) Survey.

Participant No: Name:					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to use this app frequently.					
2. I found the app unnecessarily complex.					
3. I thought the app was easy to use.					
4. I think that I would need the support of a technical person to use this app.					
5. I found the various functions in this					

app were well integrated.					
6. I thought there was too much inconsistency in this app.					
7. I would imagine that most people would learn to use this app very quickly.					
8. I found the app very cumbersome to use.					
9. I felt very confident using the app.					
10. I needed to learn a lot of things before I could get going with this app.					

1. What do you like most about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

7.3.1 Test Scenario of Usability Testing

Table 7.7: Usability Testing Scenario for Operator or Manager.

Test Scenarios to act as an operator or manager
Scenario 1 – Login to an account created by the admin to access the system
Imagine you are an operator or a manager and you have been provided with login credentials to access the palm oil mill monitoring system's mobile application. Your task is to log in to the application using the provided email and password. What would you do to access the system? **Email: demo@novaflow.com.my** **Password: Password@123**
Scenario 2 – View the dashboard
Imagine you are using the system to monitor devices and alarms. Your task is to access the dashboard and review the information displayed, such as device status and channel values. How would you check whether the information shown matches your expectations?
Scenario 3 – View graph of a device data
Imagine you are tasked to analyse a device's historical performance. Your task is to open the dashboard page, select a device channel, and observe the graph such as zoom in, pan, or view tooltips. How would you interact with the graph to get the required insights?
Scenario 4 – View multiple graphs for analysis
Imagine you needs to analyse device performance across multiple channels. Your task is to open the view graphs page, select multiple device channels, and view the results in both combined graph view and separate graph view. How would you interact with the system to switch between these views and use them for analysis purposes?
Scenario 5 – Check active alarms
Imagine you are responsible for monitoring alarms. Your task is to navigate to the active alarm page and review the current activate alarms for particular subgroup. Identify whether there are any active alarms, and if so, view the details such as device name, timestamp, and description. How would you access and confirm this information in the mobile application?

Scenario 6 – Check alarm history
Imagine you are responsible for monitoring alarms. Your task is to access the alarm history page, review the alarm list for particular subgroup, and filter by a time range or date. How would you identify whether there are alarms and view their details?
Scenario 7 – Acknowledge an alarm record
Imagine you are monitoring device alarms. Your task is to review the list of alarm records and acknowledge one of them so that other users know it has been handled. How would you perform the acknowledge action, and how would you verify that the alarm status is updated in the mobile application?
Scenario 8 – Update theme settings
Imagine you want to customize your app experience. Your task is to navigate to the settings page, change the theme mode, and confirm that the system remembers your selection after re-login or re-open the app. How would you verify that your settings are applied correctly?
Scenario 9 – Switch between subgroups
Imagine you are responsible for monitoring multiple subgroups' devices. Your task is to navigate to the dashboard page, use the subgroup dropdown selector, and switch to a different subgroup. How would you confirm that the dashboard updates to display the correct device channel data (dashboard) for the newly selected subgroup in the mobile application?
Scenario 10 – Logout from the system
Imagine you has completed monitoring tasks or want to login using another account. Your task is to logout of the application securely. How would you perform the logout action, and how would you confirm that you are redirected back to the login page in the mobile application?

7.3.2 Results of Usability Testing

Five respondents were selected to provide feedback on the 10 scenarios during the usability testing process as shown in section 7.3.1. The responses from each tester are placed in Appendix B.

The respondents' answers are analysed to calculate the SUS score by assigning a corresponding number score to each response. The SUS score is calculated using the following framework:

1. Scoring Positive Items for odd-number questions (1,3,5,7,9): Subtract 1 from the user's response. For example, if the user selected 4, the adjusted score is 3 (4-1).
2. Scoring Negative Items for even-number questions (2,4,6,8,10): Subtract the user's response from 5. For example, if the user selected 2, the adjusted score is 3 (5-2).
3. Add all adjusted scores together to obtain the raw SUS score for each respondent and multiplied by 2.5 to convert it to a usability score out of 100.
4. Summed up all the percentage scores for each respondent and divided by the total number of respondents. In this case, the total percentage is divided by 5.

The SUS score for each respondent can be determined by the method shown above. The average SUS score for a project is 68 as shown in Table 7.8 below, which means that a score of 68 represents the 50^{th} percentile, indicating average usability when compared to other systems. Scores above 68 suggest that the system has better than average usability, while score below 68 reflect poorer usability performance.

Table 7.8: General Guideline on the Interpretation of SUS Score.

SUS Score	Grade	Adjective Rating
>80.3	A	Excellent
68 – 80.3	B	Good
68	C	Okay

51 - 68	D	Poor
<51	F	Awful

Based on the results obtained from SUS as shown in Table 7.9 below, the system received an average system usability score of **94.5**, representing Grade A rating. This indicates that the mobile application is highly usable and user-friendly.

Table 7.9: Summary of SUS Survey Results.

Participants Name	**Usability score for each question**										**Total**	**Percentage (%)**
	1	2	3	4	5	6	7	8	9	10		
Loi Teck Cheu	4	4	4	4	4	4	4	4	4	4	40	100
Kenny Lai	4	4	3	3	4	4	3	3	4	4	36	90
Lim Heng Lai	4	3	4	4	4	3	4	4	3	4	37	92.5
Pan Sieng Hua	3	4	4	4	3	4	3	4	4	3	36	90
Ting Yuen Kiong	4	4	4	4	4	4	4	4	4	4	40	100
Average SUS Score												**94.5**
Grade												**A**

In addition to the System Usability Scale (SUS), a set of open-ended questions was also prepared to allow respondents provide brief comments on the palm oil mill monitoring system. This approach helped capture some valuable feedback from the users' thoughts and feelings towards the implemented system. The open-ended questions are shown below:

1. What do you like most about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

The Table 7.10 below show the summary of the most like features based on the participant feedback. However, no least-liked features or functionality were identified by the participant during the usability testing.

Table 7.10: Summary of Participants' Most Liked Features of the System.

Summary of Participants' Most Liked Features of the System
The dashboard is very clean and easy to understand. I can see the real-time device updates clearly.
The graph monitoring function is very useful, especially the zoom and tooltip feature.
The alarm notification is clear and accurate.
The system is able to change the theme mode.
The system is very stable and responsive. I felt confident using it without any training.

Despite no least-liked feedback from the participants, there are some suggestions from participants for improving the system as shown in Table 7.11 below. These recommendations are valuable and useful for future improvement on phase 2 later on to enhance the overall system usability.

Table 7.11: Summary of Suggestions for Improving the system from Participants.

Summary for Improving the System Recommended by Participants
Allow alarm threshold settings able to configure in the mobile app.
Have auto analysis in the system based on the alarm monitoring system.
Provide more detailed documentation or video guides inside the system for new users.
Able to change password in the system.
Include a search function in the FAQ to quickly find solutions.

7.4 Alpha Testing

In addition to unit testing and usability testing described in previous sections, **alpha testing** was also conducted on the deployed mobile application for both **Android and iOS** platforms to **internal team**. This testing ensured that all released features worked as expected across platforms and **met company requirements**.

To ensure traceability of changes made, **version control** was recorded and maintained throughout the development cycle. Each release version was tested for functionality, usability and performance by the internal team. The alpha version history is shown in Table 7.12 below:

Table 7.12: Alpha Version Control History.

Version	Release Date	Changes / Notes
SM365_V2.0.1 Alpha	28/04/2025	- Initial release - Graph monitoring function
SM365_V2.1.0 Alpha	14/06/2025	- Implemented alarm notification system
SM365_V2.1.1 Alpha	24/06/2025	- Updated Ack function for newer Python version - App name changed to Smart Mill 365 - Updated app icon
SM365_V2.1.2 Alpha	25/06/2025	- Updated new influx DB URL and token - Updated new login page design
SM365_V2.1.3 Alpha	27/06/2025	- Adjusted light and dark mode colour - Updated UI for chart screen - Unique device unit retrieve from Firebase - Created a generic screen for all devices

SM365_V2.1.4 Alpha	29/06/2025	- Created theme settings in app
SM365_V2.1.5 Alpha	30/06/2025	- Updated app info in About Us, FAQ and Privacy Policy. - Updated Logout button UI.
SM365_V2.1.6 Alpha	3/07/2025	- Updated new influx DB URL
SM365_V2.1.7 Alpha	10/07/2025	- Updated applicationId

Figure 7.1 below shows the latest build history of the mobile application in TestFlight for **internal testing** purposes. These records provide a trace of deployed versions that were verified by the internal team to ensure that all features met company requirements before proceeding to external beta testing.

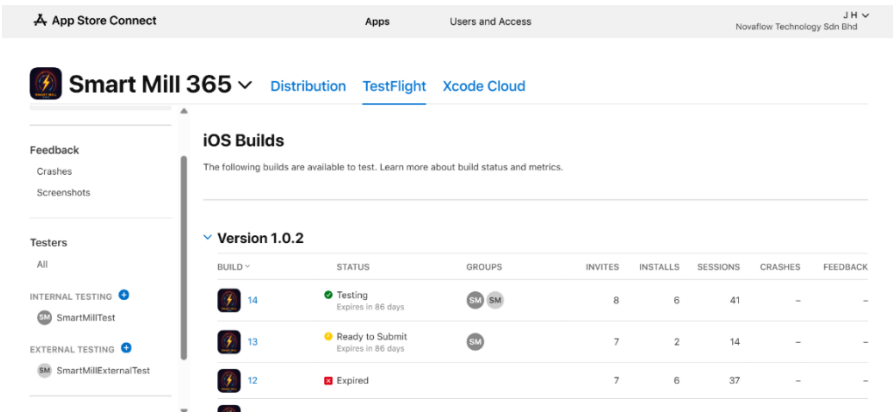


Figure 7.1: Build History for TestFlight in App Store Connect.

7.5 Beta Testing

After the completion of internal alpha testing, beta testing was conducted to evaluate the application under **real-world conditions**. Unlike alpha testing, which is limited to internal teams, beta testing involves **selected external users** who provide valuable feedback on usability, performance, and overall user experience. This stage is crucial before deployment as it helps identify issues that may not be found during alpha testing.

Figure 7.2 below shows the list of external users in **TestFlight** on **App Store Connect**. Each build version was distributed to selected external users (Novaflow clients) through TestFlight for beta testing before the official release on the App Store. Feedback gathered from these external testers help validate functionality, usability, and performance in real-world conditions. This serves as evidence of structured beta testing carried out before the deployment process. The beta version history is outlined in Table 7.13 below:

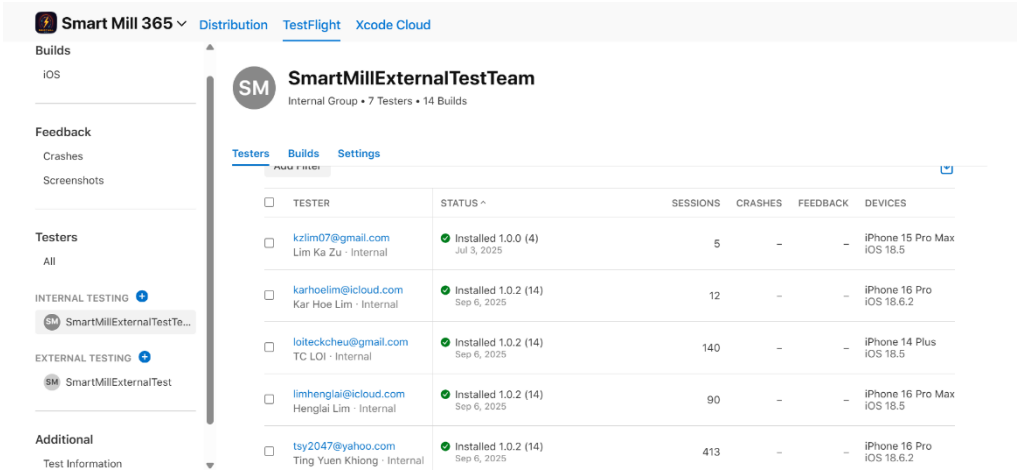


Figure 7.2: List of External Users for Beta Testing.

Table 7.13: Beta Version Control History.

Version	Release Date	Changes / Notes
SM365_V2.2.0 Beta	1/08/2025	- Graph update from every 5 seconds to 1 minute to avoid lagging in Android - Last Updated As: change from current timestamps to follow influx DB's last

		timestamps - Show loading indicator before selected subgroup's data fully retrieve - Fixed screens rotate issue
SM365_V2.2.1 Beta	5/08/2025	- Click graph can show line spot value - Graph able to zoom in and out - Removed live graph to avoid lagging
SM365_V2.2.2 Beta	7/08/2025	- Updated graph screen UI
SM365_V2.2.3 Beta	7/08/2025	- Initial subgroup loading state performance
SM365_V2.2.4 Beta	20/08/2025	- UI for loading indicator (Novaflow logo) - Run/Stop label for xxx_event in Firebase - UI for Confirm button
SM365_V2.2.5 Beta	20/08/2025	- UI for bottom navigation - Update timestamps and run/stop in view graphs page
SM365_V2.2.6 Beta	22/08/2025	- Fix blank screen issue
SM365_V2.2.7 Beta	10/09/2025	- Update UI for active alarm icon - Update version number in app - Update error message - Added hide/unhide password function - Add Remember Me function during login

CHAPTER 8

CONCLUSION AND RECOMMENDATION

8.1 Conclusion

This last chapter is to provide a conclusion and recommendation for future improvement on this project. All objectives stated in Chapter 1 were successfully achieved including:

1. To develop a cross-platform mobile application using Flutter that can replace the existing company's iOS app and ensures accessibility for both Android and iOS users in monitoring POM processes.
2. To integrate SVG graphics for displaying process layouts along with real-time data display.
3. To implement an alarm notification system to alert users about critical process conditions.
4. To enhance system usability and interface for more efficient real-time monitoring and analysis.

Since the project was successful achieving all stated objectives, this resulting a fully functional cross-platform mobile application for monitoring palm oil mill processes. The system developed not only replaced the company's existing iOS app but also improved its accessibility, real-time data visualization and overall usability. Through systematic testing, deployment and feedback from company, the application has been validated as a reliable and practical solution. The application now is fully launched and actively used by company's clients.

The official evaluation letter from **Novaflow Technology Sdn. Bhd.** is shown in **Appendix C**, which acknowledges the successful completion and handover of the project, as well as my contributions and performance during the development.

8.2 Limitations and Recommendations for Future Works

During the development and testing phases of the system, some limitations were identified by the company, the participants that involved in the usability testing and myself. These limitations along with their recommendations are summarized in Table 8.1 below serve as guidance for future improvement in Phase 2 of the project by the company.

Table 8.1: Limitations and its Recommendations of the System.

Limitations	**Recommendation**
Lack of device performance analysis in the system.	Implement advanced performance analytics features to provide deeper insights for operators and managers.
Dashboard UI basic and not fully suited for complex palm oil mill processes.	Enhance the dashboard with customizable widgets and process-specific visualizations to better support operational needs.
Device and alarm settings cannot be configured directly from the mobile app.	Enable in-app configuration of device and alarm settings to reduce dependency on Firebase and reliance on Novaflow IT team.
Change and reset password function is not available in the mobile app.	Integrate change and reset password feature to improve user account management and security.
Limited documentation and learning support for new users.	Provide more detailed documentation or video guides in the system to help new users learned quickly.
FAQ section is basic and not searchable.	Include a search function in the FAQ module to allow users to quickly find solutions to common issues.

REFERENCES

Foong, S.Z., Lam, Y.L., Andiappan, V., Foo, D.C. and Ng, D.K., 2018. A systematic approach for the synthesis and optimization of palm oil milling processes. *Industrial & Engineering Chemistry Research*, *57*(8), pp.2945-2955.

Camburn, B., Viswanathan, V., Linsey, J., Anderson, D., Jensen, D., Crawford, R., Otto, K. and Wood, K., 2017. Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science*, *3*, p.e13.

United Nations (2025). *The 17 Sustainable Development Goals*. [online] United Nations. Available at: https://sdgs.un.org/goals.

Kishore, K., Khare, S., Uniyal, V. and Verma, S., 2022, October. Performance and stability comparison of react and flutter: Cross-platform application development. In *2022 International Conference on Cyber Resilience (ICCR)* (pp. 1-4). IEEE.

Stack Overflow. (2019). *Stack Overflow Developer Survey 2019*. [online] Available at: https://survey.stackoverflow.co/2019.

Tashildar, A., Shah, N., Gala, R., Giri, T. and Chavhan, P., 2020. Application development using flutter. *International Research Journal of Modernization in Engineering Technology and Science*, *2*(8), pp.1262-1266.

Palumbo, D., 2021. *The Flutter framework: Analysis in a mobile enterprise environment* (Doctoral dissertation, Politecnico di Torino).

Vishal, K. and Kushwaha, A.S., 2018, August. Mobile application development research based on xamarin platform. In *2018 4th International Conference on Computing Sciences (ICCS)* (pp. 115-118). IEEE.

Sattar, A.M., Soni, P., Ranjan, M.K., Kumar, A., Sahu, C., Saxena, S. and Chaudhari, P., 2023. Accelerating Cross-platform Development with Flutter Framework.

Penta, H., 2004. *A COMPREHENSIVE TESTING APPROACH USING JEST FOR REACT NATIVE MOBILE APPLICATIONS* (Doctoral dissertation, CALIFORNIA STATE UNIVERSITY, NORTHRIDGE).

Ramadoss, G., 2023. Choosing Xamarin Platform for App Development. *North American Journal of Engineering Research*, *4*(3).

Wu, W., 2018. React Native vs Flutter, Cross-platforms mobile application frameworks.

Lodhi, M.K., 2024. *Comparison and Evaluation of Cross-Platform Frameworks* (Doctoral dissertation, Hochschule für Angewandte Wissenschaften Hamburg).

Kandiah, S., Basiron, Y., Suki, A., Taha, R.M., Tan, Y.H. and Sulong, M., 2006. Continuous sterilization: The new paradigm for modernizing palm oil milling. *J. Oil Palm Res*, pp.144-152.

Peng, C., 2000. Scalable vector graphics (svg). In *Research Seminar on Interactive Digital Media*.

Zhu, X., Nie, X. and Liu, J., 2023, September. Time Series Database Optimization Based on InfluxDB. In *2023 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEC)* (pp. 879-885). IEEE.

Naqvi, S.N.Z., Yfantidou, S. and Zimányi, E., 2017. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles*, *12*, pp.1-44.

Tahmassebpour, M., 2017. A new method for time-series big data effective storage. *Ieee Access*, *5*, pp.10694-10699.

Khawas, C. and Shah, P., 2018. Application of firebase in android app development-a study. *International Journal of Computer Applications*, *179*(46), pp.49-53.

Grier, R.A., Bangor, A., Kortum, P. and Peres, S.C., 2013, September. The system usability scale: Beyond standard usability testing. In *Proceedings of the human factors and ergonomics society annual meeting* (Vol. 57, No. 1, pp. 187-191). Sage CA: Los Angeles, CA: SAGE Publications.

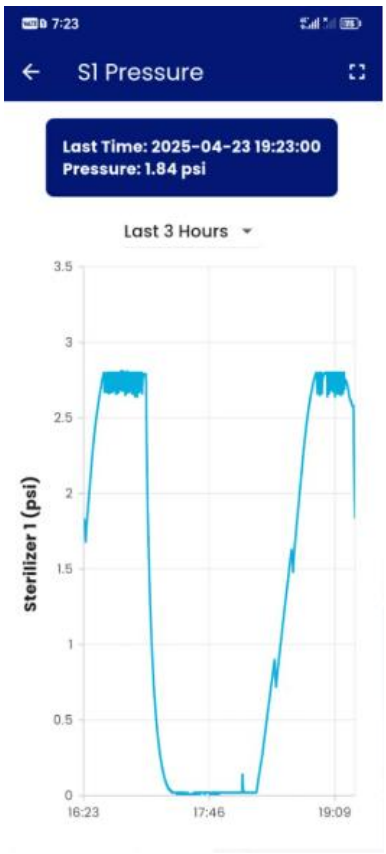
Azami, H.H.R. and Ibrahim, R., 2019. Development and evaluation of massive open online course (MOOC) as a supplementary learning tool: An initial study. *International Journal of Advanced Computer Science and Applications*, *10*(7).

APPENDICES

Appendix A: Low-fidelity prototype for SmartMill365 Mobile Application.

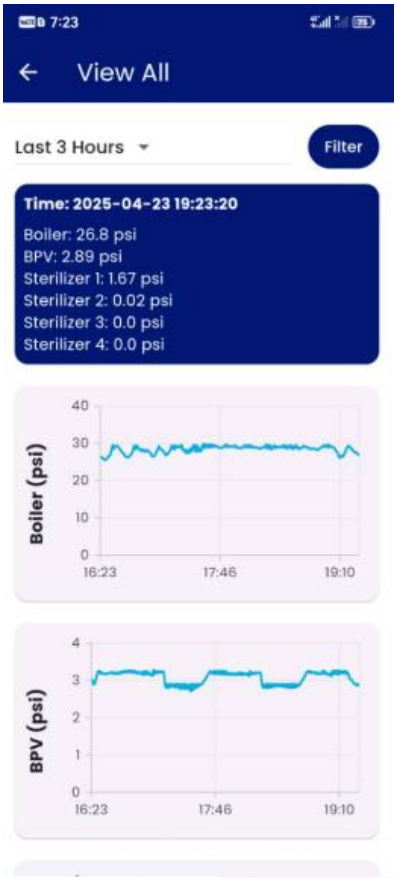
Login Page

Dashboard Page

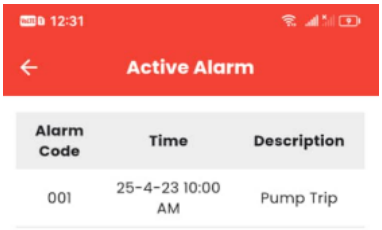


Single Live Channel Data Monitoring Page

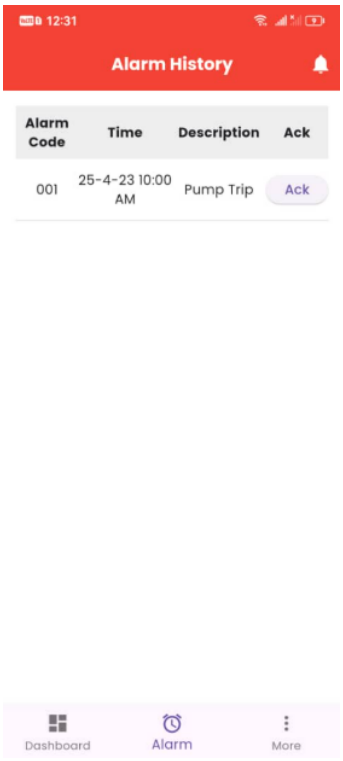
Full Screen Live Channel Data Monitoring Page



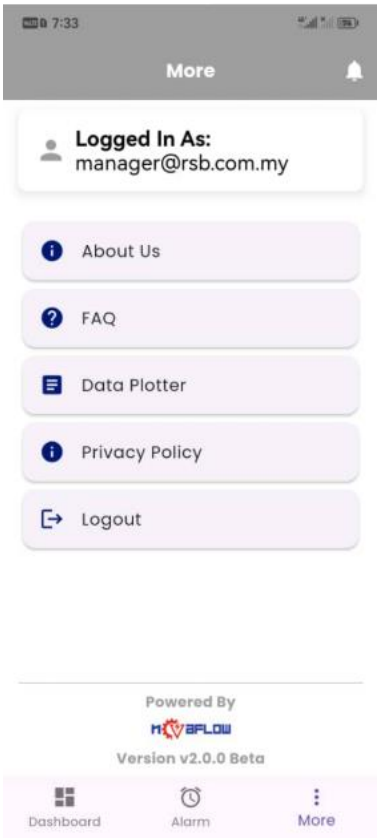
All / Filtered Live Channel Data Monitoring Page



Active Alarm Page



Alarm History Page



More Page

Welcoming Page

Appendix B: SUS Usability Test Responses.

Participant No: 1 Name: Loi Teck Cheu					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to use this app frequently.					✓
2. I found the app unnecessarily complex.	✓				
3. I thought the app was easy to use.					✓
4. I think that I would need the support of a technical person to use this app.	✓				
5. I found the various functions in this app were well integrated.					✓
6. I thought there was too much inconsistency in this app.	✓				
7. I would imagine that most people would learn to use this app very					✓

quickly.					
8. I found the app very cumbersome to use.	✓				
9. I felt very confident using the app.					✓
10. I needed to learn a lot of things before I could get going with this app.	✓				

1. What do you like most about the system?
   The dashboard is very clean and easy to understand. I can see the real-time device updates clearly.

2. What do you like least about the system?
   -

3. Do you have any suggestions for improving the current system?
   Allow alarm threshold settings able to configure in the mobile app.

Participant No: 2 Name: Kenny Lai					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to					✓

use this app frequently.					
2. I found the app unnecessarily complex.	✓				
3. I thought the app was easy to use.				✓	
4. I think that I would need the support of a technical person to use this app.		✓			
5. I found the various functions in this app were well integrated.					✓
6. I thought there was too much inconsistency in this app.	✓				
7. I would imagine that most people would learn to use this app very quickly.				✓	
8. I found the app very cumbersome to use.		✓			
9. I felt very confident using the app.					✓

10. I needed to learn a lot of things before I could get going with this app.	✓				

1. What do you like most about the system?
   The graph monitoring function is very useful, especially the zoom and tooltip feature.

2. What do you like least about the system?
   -

3. Do you have any suggestions for improving the current system?
   Have auto analysis in the system based on the alarm monitoring system.

Participant No: 3 Name: Lim Heng Lai					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to use this app frequently.					✓
2. I found the app unnecessarily complex.		✓			
3. I thought the app was easy to use.					✓
4. I think that I would need the	✓				

support of a technical person to use this app.					
5. I found the various functions in this app were well integrated.					✓
6. I thought there was too much inconsistency in this app.		✓			
7. I would imagine that most people would learn to use this app very quickly.					✓
8. I found the app very cumbersome to use.	✓				
9. I felt very confident using the app.				✓	
10. I needed to learn a lot of things before I could get going with this app.	✓				

1. What do you like most about the system?
   The alarm notification is clear and accurate.

2. What do you like least about the system?
   -

3. Do you have any suggestions for improving the current system?
   Provide more detailed documentation or video guides inside the system for new users.

Participant No: 4 Name: Pan Sieng Hua					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to use this app frequently.				✓	
2. I found the app unnecessarily complex.	✓				
3. I thought the app was easy to use.					✓
4. I think that I would need the support of a technical person to use this app.	✓				
5. I found the various functions in this app were well				✓	

integrated.					
6. I thought there was too much inconsistency in this app.	✓				
7. I would imagine that most people would learn to use this app very quickly.				✓	
8. I found the app very cumbersome to use.	✓				
9. I felt very confident using the app.					✓
10. I needed to learn a lot of things before I could get going with this app.		✓			

1. What do you like most about the system?
   The system able to change the theme mode.

2. What do you like least about the system?
   -

3. Do you have any suggestions for improving the current system?
   Able to change password in the system.

Participant No: 5 Name: Ting Yuen Kiong					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think that I would like to use this app frequently.					✓
2. I found the app unnecessarily complex.	✓				
3. I thought the app was easy to use.					✓
4. I think that I would need the support of a technical person to use this app.	✓				
5. I found the various functions in this app were well integrated.					✓
6. I thought there was too much inconsistency in this app.	✓				
7. I would imagine that most people would learn to use this app very quickly.					✓

8. I found the app very cumbersome to use.	✓				
9. I felt very confident using the app.					✓
10. I needed to learn a lot of things before I could get going with this app.	✓				

1. What do you like most about the system?
   The system is very stable and responsive. I felt confident using it without any training.

2. What do you like least about the system?
   -

3. Do you have any suggestions for improving the current system?
   Include a search function in the FAQ to quickly find solutions.

Appendix C: Official Evaluation Letter from Novaflow.

Novaflow Technology Sdn Bhd

Date: 06-Sep-2025

To: Dr Khor Kok Chin
Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

This letter serves as an official evaluation of Lim Juan Hong, who completed his Final Year Project (FYP) at Novaflow Technology Sdn Bhd on Developing a Mobile Apps for Monitoring Palm Oil Mill Process.

Throughout the course of the project, Lim Juan Hong demonstrated strong initiative and dedication in his work. His contributions to the development of the mobile application were meaningful, and he exhibited the following key strengths:

1. Initiative & Independence
   Lim Juan Hong showed great initiative by actively asking what needed to be done in order to fulfill Novaflow's requirements. With minimal supervision, he was able to quickly pick up Flutter programming and successfully deliver a working mobile application for both Android and iOS platforms.

2. Collaboration & Creativity
   He was collaborative with both the IT and automation teams at Novaflow, ensuring smooth communication and alignment with project goals. Additionally, he displayed creativity in designing and delivering several mobile app pages, contributing to a user-friendly and effective solution.

3. Areas for Improvement
   While Lim Juan Hong demonstrated excellent technical and collaborative skills, he currently has limited exposure to real-world IT market requirements. We encourage him to be involved in more projects, as this will help him gain broader industry knowledge and enable him to deliver tasks and projects with greater professionalism.

4. Overall Performance
   In summary, Lim Juan Hong's overall performance during the Final Year Project has been excellent. His initiative, adaptability, and ability to work both independently and collaboratively made him a valuable contributor to the team. We are confident that with continued exposure to industry projects, he will further develop into a highly capable professional.

Novaflow Technology Sdn Bhd

We highly recommend Lim Juan Hong for any future opportunities, as he has shown great potential and dedication throughout his project work.

Should you require any further information regarding his performance, please feel free to contact me at tcloi@novaflow.com.my or +6019-6113569.

Yours faithfully,

.....

Loi Teck Cheu
(Automation & IIOT Development Manager)
Novaflow Technology Sdn Bhd
(Registration No: 1379177K / 202001022857)
Address: No 30 Jalan Serendah 26/41, Sekitar 26 40400 Shah Alam, Selangor Darul Ehsan.