Smart Home Monitoring in Node-Red Emulator using Flutter Mobile App and AI technology

By

Yap Jun Hong

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS AND NETWORKING
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my supervisor, Tan Lyk Yin, for providing me with the invaluable opportunity to work on an IC design project. This experience has been my first step toward a career in the IC design industry, and I am truly thankful for your guidance, mentorship, and encouragement.

I am also deeply grateful to my parents and family for their unwavering love, support, and encouragement throughout this journey.

ABSTRACT

The limitations of static automation in current IoT-based smart home systems highlight a lack of flexibility and personalization. To address this gap, this project develops an adaptive smart home control and monitoring system that integrates Internet of Things (IoT) and Artificial Intelligence (AI) technologies for dynamic automation. The system is emulated using Node-RED with an SVG-based interface for device visualization, while a Flutter mobile app serves as the user interaction platform. MQTT provides real-time communication, and InfluxDB supports time-series data management. An AI module leverages both historical and real-time data to enable predictive decision-making, enhancing functions such as temperature regulation, lighting management, and air purification. The prototype demonstrates that adaptive automation improves comfort, responsiveness, and energy efficiency compared to traditional static methods. Project deliverables include the partial development of the Flutter mobile UI, Node-RED flows, and the SVG emulator, with future work focusing on full integration of InfluxDB for advanced analytics and AI-driven predictive control through machine learning.

Area of Study (Maximum 2): Smart home automation and control system, Artificial Intelligence in IoT

Keywords (Maximum 5): Internet of Things, Smart Home Monitoring, Mobile App, Node-RED Simulation, Machine Learning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MQTT	Message Queuing Telemetry Transport
AI	Artificial Intelligence
ML	Machine Learning
IoT	Internet of Things
SVG	Scalable Vector Graphics
SDK	Software Development Kit
UI	User Interface
InfluxDB	Influx Database (Time-Series Database)
Py	Python Programming Language
IBM	International Business Machine Corporation
MVP	Minimum Viable Product
IP	Internet Protocol

Chapter 1

1.1 Background Information

Smart home systems have become an integral part of modern living, aiming to improve convenience, safety, and energy efficiency through the interconnection of household devices via the Internet of Things (IoT) [1]. These systems allow remote monitoring and control of appliances, environmental conditions, and security features through centralized interfaces such as mobile apps or web platforms [2]. Traditionally, smart homes have relied on static, rule-based automation, where devices operate according to fixed schedules or simple triggers. While this provides basic control, such systems lack personalization and adaptability to dynamic user behavior and environmental changes [3].

Recent advancements in IoT platforms, mobile applications, and artificial intelligence (AI) have enabled more adaptive smart home solutions. AI-driven predictive modeling can analyze user routines, environmental conditions, and historical sensor data to make proactive adjustments in devices like lighting, air conditioning, and air purifiers [4], [5]. Middleware platforms such as Node-RED allow seamless integration of multiple devices and IoT protocols, supporting real-time communication and visualization of the home environment [6]. These technologies enhance system efficiency, usability, and personalization, offering users greater comfort and convenience while providing developers and researchers with flexible and extensible testing frameworks.

Prior research has demonstrated the potential of adaptive smart home automation. Rashidi and Cook [3] explored activity recognition and context-aware decision-making in residential settings, while Mozer [3] investigated learning-based control strategies for energy-efficient and user-responsive environments. More recent studies by Khan et al. [4] and Alawadhi et al. [5] applied AI techniques to predict user behavior and environmental changes, though many of these works relied on simulated data rather than real user inputs. These studies highlight the importance of integrating adaptive control, predictive intelligence, and real-time monitoring to advance smart home systems beyond static automation.

1.2 Problem Statement and Motivation

The rise of Internet of Things (IoT) technologies has accelerated the adoption of smart home systems designed to improve convenience, energy efficiency, and lifestyle quality. Many commercial and open-source solutions such as Google Home, Amazon Alexa, and Samsung SmartThings have enabled users to automate home appliances and monitor environments remotely [2]. However, most existing systems rely on **static automation**, where devices act only on predefined rules and fixed schedules (e.g., turning lights on at 7 p.m. daily or maintaining air conditioning at a constant temperature). While effective for basic control, this rule-based approach lacks flexibility and adaptability to dynamic environmental conditions and user behavior, resulting in inefficiencies and limited personalization [9].

If these limitations remain unresolved, smart home technology will fail to deliver its full potential. Rigid control systems can cause **unnecessary energy consumption** (e.g., cooling empty rooms), **reduced comfort** (e.g., not adapting lighting to natural daylight), and **fragmented user experiences** due to isolated device management across different platforms [10]. More critically, in sensitive contexts such as homes with elderly residents or chronic illness patients, failure to adapt environmental conditions (e.g., temperature, humidity, or air quality) may increase health risks [11]. Over time, these shortcomings may discourage user adoption, undermine trust in smart home technologies, and contribute to higher costs and environmental impact.

To overcome these challenges, it is essential to shift from static, rule-based systems toward adaptive automation powered by artificial intelligence (AI). Addressing this problem requires knowledge of IoT integration, data processing from diverse home sensors, and AI models capable of prediction and decision-making [12]. By leveraging predictive AI for parameters such as temperature, lighting, and fragrance, smart homes can evolve into proactive, context-aware systems that anticipate user needs and optimize both comfort and energy efficiency [13]. This study is motivated to establish an adaptive, AI-driven control framework—implemented through a mobile app connected to a Node-RED-based virtual smart home emulator—that demonstrates the feasibility of intelligent and personalized home environments.

1.3 Project Objectives

The objectives of this project are aligned with addressing the challenges identified in the earlier problem statement, namely the limitations of static automation, the lack of adaptability to dynamic conditions, and the reduced user comfort and efficiency in conventional smart home systems. These objectives aim to establish a more intelligent, responsive, and user-centered smart home control and monitoring system that integrates simulation, adaptive automation, and real-time decision-making to enhance overall living experiences.

First Objective is to emulate a home environment using Node-RED and SVG graphics for temperature regulation, lighting management, and air purification with relevant sensors. The primary objective of this project is to create a functional and visualized smart home simulation platform that reflects real environmental conditions. This includes monitoring temperature, lighting, and air quality through connected sensors and visualizing device states using SVG graphics in Node-RED. By achieving this objective, the project addresses the problem of testing adaptive automation in real-world settings, providing a measurable and controllable simulation environment where scenarios can be analyzed before actual deployment.

Second Objective is to develop an adaptive and responsive control system for seamless interaction with the smart home system using Node-RED and Flutter. This objective focuses on integrating backend logic in Node-RED with a Flutter-based mobile app to deliver real-time device interaction and adaptive control. The aim is to ensure that users can manage and monitor their smart home environment dynamically while the system adapts to changing user preferences and environmental conditions. By completing this objective, the project directly addresses the problem of static, rule-based automation by providing flexibility, responsiveness, and improved user experience.

Third Objective **is t**o implement real-time decision-making for device control using AI-based analysis of environmental and user behavior data within the smart home simulation. The final objective introduces artificial intelligence to enhance decision-making capabilities within the system. By analyzing user behavior patterns and environmental data, the system will be capable of predictive and context-aware adjustments to devices such as temperature regulation, lighting, and air purification. Achieving this objective resolves the problem of inefficiency and lack of personalization in traditional automation, ensuring energy savings, optimized control, and improved comfort for the user.

In summary, the objectives of this project are designed to overcome the limitations of static automation in smart home systems by focusing on simulation, adaptive control, and AI-driven decision-making. By achieving these objectives, the project contributes to the development of a more intelligent, user-centric, and efficient smart home solution that improves comfort, personalization, and sustainability in everyday living environments.

1.4 Contribution

The "Adaptive Smart Home Control and Monitoring System" project aims to make significant contributions in addressing the critical problems associated with static automation in existing smart home platforms, namely inefficiency, lack of adaptability, and limited personalization. These contributions align closely with the project's objectives and the challenges outlined in the problem statement, ensuring both practical and research relevance in the context of IoT and smart home technologies.

The first contribution of this project is the advancement of an adaptive smart home simulation framework using Node-RED and SVG graphics. By emulating environmental conditions such as temperature, lighting, and air quality with sensor integration, the project provides a controllable and measurable platform for testing adaptive automation strategies. This directly addresses the challenge of limited flexibility in current static systems by enabling a safe and cost-effective way to experiment with intelligent automation before real-world deployment.

The second contribution is the development of a user-centric and responsive control system that integrates Node-RED for backend logic with a Flutter-based mobile app for user interaction. By combining Flutter's ability to deliver efficient, multi-platform mobile UIs with Node-RED's IoT logic, this contribution enhances the overall usability and adaptability of smart home environments by providing real-time responsiveness and seamless interaction between users and devices. In relation to the problem statement, this addresses the issue of rigid, rule-based systems that fail to adapt to user preferences or dynamic environmental changes, thereby improving personalization and user experience.

The third contribution is the integration of AI-driven real-time decision-making for smart home device control. By analyzing environmental and user behavior data, the system can make predictive and context-aware adjustments to appliances such as lighting, temperature regulation, and air purification. This contribution directly tackles the inefficiencies and energy waste associated with static automation, while also improving comfort and safety, particularly in sensitive contexts such as elderly care or households with health concerns.

In summary, the contributions of the "Adaptive Smart Home Control and Monitoring System" project align with its objectives and the problems identified earlier. By providing a robust simulation framework, a responsive and user-friendly control system, and AI-based real-time decision-making, the project advances the field of smart home automation from static, rule-based control to adaptive and intelligent environments. These contributions not only enhance energy efficiency and user comfort but also pave the way for more sustainable, personalized, and trustworthy smart home technologies in the future.

1.5 Project Scope

This project scope defines the boundaries and expected deliverables of the "Adaptive Smart Home Control and Monitoring System" project. It outlines the major components to be developed, including a Node-RED-based virtual smart home emulator, a Flutter mobile app, and an AI machine learning script, while also clarifying what is excluded to ensure a clear understanding of the project's focus and limitations.

The project is centered on demonstrating adaptive automation in smart homes by combining simulation, mobile control, and AI-driven decision-making. The system will emulate environmental conditions, provide a user-friendly mobile interface, and integrate AI for real-time decision-making, thereby showcasing the transition from static to adaptive automation in smart homes

Inclusions:

1. Node-RED and SVG-Based Virtual Smart Home Emulator: The project includes the development of a simulation platform using Node-RED with SVG graphics to emulate key environmental factors such as temperature regulation, lighting management, and air purification.
2. Flutter Mobile App: A user-centric mobile app will be developed using Flutter to enable seamless real-time control and monitoring of the virtual smart home system.

3. AI Machine Learning: The project scope includes the implementation of an AI-driven decision-making module using Python, capable of analyzing real user data along with environmental and user behavior data to support adaptive automation.

4. System Boundaries: The project focuses exclusively on indoor smart home devices and environmental factors, specifically covering temperature, lighting, and air quality. Other devices or outdoor automation systems are beyond the scope of this project.

Exclusions:

1. Physical Smart Home Devices: The project does not involve the development, purchase, or installation of physical IoT devices such as sensors, smart bulbs, or HVAC systems. The focus remains on simulation.

2. Commercial Deployment: The project does not include packaging the system for commercial release, marketing, or large-scale adoption. It is limited to a research and prototype context.

3. Advanced Security and Privacy Protocols: While basic communication between components will be implemented, comprehensive security measures (e.g., encryption, authentication frameworks) are beyond the scope of this project.

4. Extensive Hardware Integration: The project excludes integration with third-party platforms such as Google Home, Amazon Alexa, or Samsung SmartThings, focusing instead on a standalone prototype.

5. Excluded of Smart Home Device: The project does not cover additional smart home devices such as CCTV systems, automated gate controls, smart curtains, or other specialized home automation device.

In conclusion, the project scope encompasses the development of a Node-RED SVG-based smart home emulator, a Flutter-based mobile app, and an AI machine learning script to demonstrate adaptive and intelligent home automation. It emphasizes prototyping, simulation, and user interaction, while excluding physical hardware deployment, commercial activities, and extensive third-party integrations to maintain a clear and achievable research focus.

1.6 Report Organization

Chapter 1 Introduction presents the background and motivation of the project, outlining the key challenges in existing smart home automation systems. It describes the problem statement, research objectives, project contributions, and scope of work. The chapter establishes the foundation of the study by highlighting the need for adaptive, AI-driven solutions that go beyond static automation.

Chapter 2 Literature Review reviews previous studies and technologies related to smart home automation. It discusses the evolution of smart home systems from rule-based to adaptive models, emphasizing the importance of personalization, energy efficiency, health, and safety. Furthermore, it examines prior works in the field, identifying the research gap that this project aims to address. The chapter also evaluates enabling technologies such as Flutter for mobile application development and Node-RED as middleware for IoT automation, providing justification for their selection in this project.

Chapter 3 System Analysis and Design presents the analysis of system requirements and the design of the proposed smart home control system. It begins with the specification of hardware and software requirements, ensuring the necessary infrastructure for development and deployment. The chapter then details the functional requirements, mapping them to system workflows to illustrate the operational processes. Furthermore, it introduces the system architecture, use case diagrams, and activity diagrams, providing both structural and dynamic perspectives of the system's functionality.

Chapter 4 System Implementation explains the implementation of the proposed system based on the design established in Chapter 3. It describes the integration of the key components, including the Node-RED emulator, Flutter-based mobile application, InfluxDB database, and the Python AI module with Flask API. Communication between components through MQTT and API requests is also discussed, highlighting how user interactions are translated into system responses. The chapter concludes by presenting the complete architecture in operation, demonstrating how the system achieves real-time monitoring, control, and predictive decision-making.

Chapter 5 System Implementation is involved integrating multiple components, namely the Flutter mobile application, Node-RED SVG emulator, InfluxDB time-series database, and a Python-based AI prediction module. The Flutter app served as the user interface, enabling residents to control devices, monitor real-time status, and retrieve AI-driven predictions through an API. Node-RED functioned as the central middleware, processing MQTT messages, updating device states within the SVG emulator, and facilitating automation logic. InfluxDB was employed for structured storage of time-series data, with separate buckets created to manage user control inputs and AI predictions for both temperature and lighting-fragrance functions, ensuring efficient retrieval and analysis. The Python module implemented predictive modeling, generating adaptive control suggestions based on historical and contextual data, which were then exposed via a Flask API to the mobile application. The system was tested across multiple devices, validating stable MQTT communication with negligible latency, fast device responsiveness (50–100 ms), and acceptable delays in thermostat adjustments (1–3 s). Error handling mechanisms were integrated at both the mobile app and Node-RED layers to ensure reliability under varying conditions. Overall, the implementation successfully demonstrated a unified, adaptive, and data-driven smart home prototype that addressed the limitations of static automation systems and highlighted the potential of AI-enhanced personalization and efficiency in residential environments.

Chapter 6 System Evaluation and Discussion presents the evaluation of the adaptive smart home system in terms of performance, functionality, and achievement of objectives. It begins with system performance evaluation, covering load testing, response time analysis, network stability, and database efficiency under realistic operating conditions. Functional testing is then described through structured test cases, which validate device interactions, data storage, and AI-driven predictions. The chapter also highlights project challenges, particularly the limitations of SVG-based visualization in Node-RED, before assessing how well the project objectives were achieved. Finally, it concludes with a summary remark that the system delivered reliable, responsive, and intelligent smart home automation, while identifying areas for further improvement.

Chapter 7 Conclusion and Recommendation provides the concluding remarks of the project and proposes future directions. It summarizes the overall contribution of the study, reaffirming the successful design, implementation, and evaluation of an AI-driven smart home prototype that overcomes the limitations of static automation. The conclusion emphasizes how the integration of Flutter, Node-RED, Python, and InfluxDB produced a scalable and adaptive solution for user comfort, personalization, and efficiency. The chapter then offers recommendations for future work, focusing on the development of a physical demonstrator to showcase the system in a tangible environment and the enhancement of AI capabilities through advanced models and broader datasets. Together, these recommendations point toward expanding the system into a more sophisticated and deployable smart home solution.

Chapter 2

Literature Review

The purpose of this literature review is to provide a foundation for understanding the key technologies and research relevant to the development of an adaptive smart home system. Smart homes leverage IoT devices, mobile interfaces, and AI to automate household functions. Over time, the evolution has shifted from static, rule-based automation toward adaptive systems that respond to user behavior and environmental conditions in real time. This chapter reviews relevant frameworks, middleware, databases, and machine learning approaches, along with prior research and prototypes in the field.

2.1 Smart Home Automation and Adaptive Systems

2.1.1 Evolution of Smart Homes

Traditional smart homes primarily relied on static, rule-based automation, where devices like lights, thermostats, and security systems operated according to fixed schedules or pre-configured triggers. While these systems improved convenience and some energy efficiency, they often failed to respond to dynamic user needs or environmental changes.

With the advent of the Internet of Things (IoT), smart homes began incorporating connected sensors and actuators, enabling more granular monitoring of the indoor environment (Cook et al., 2013). These systems could gather data such as room occupancy, temperature, and energy usage, forming the foundation for data-driven automation.

The next stage involves adaptive smart home systems, which integrate AI and machine learning to dynamically adjust behavior based on real-time data and historical patterns (Mihailidis et al., 2018). Such systems go beyond pre-set rules by learning user preferences, predicting future actions, and optimizing device control for energy efficiency and comfort.

2.1.2 Importance of Adaptive Systems

Adaptive smart home systems offer several benefits over static automation:

1. Personalization – Modern systems can learn individual routines and adjust environmental settings automatically, improving user comfort and satisfaction (Demiris & Hensel, 2008).

2. Energy Efficiency – Predictive control algorithms can reduce energy consumption by optimizing device operation according to occupancy patterns and weather forecasts (Kofod-Petersen et al., 2015).
3. Health and Safety Apps – Sensors can detect indoor air quality, smoke, or unusual movements, enabling preventive interventions. Adaptive systems can adjust ventilation, lighting, and alerts in response to these signals (Wilson et al., 2018).
4. Context-Awareness – By integrating environmental, behavioral, and temporal data, adaptive systems can make informed decisions, such as dimming lights at night or pre-heating rooms before occupants arrive (Rashidi & Cook, 2009).

2.1.3 Prior Studies on Adaptive Automation

Several academic studies and prototypes have demonstrated the potential of adaptive smart homes:

- Rashidi & Cook (2009) developed a framework for activity-aware smart homes, which recognized user activities and adjusted environmental controls accordingly. While effective, the system relied on pre-collected sensor datasets rather than real-time user data.
- Mozer (2012) introduced The Adaptive Home, employing machine learning to predict and automate occupant behaviors for energy savings and comfort. Limitations included limited device coverage and dependency on simulated environments.
- Khan et al. (2020) explored predictive energy management using reinforcement learning, showing improvements in energy efficiency while maintaining user comfort. However, the study lacked integration with mobile control interfaces.
- Alawadhi et al. (2021) implemented a smart home assistant prototype using IoT sensors and AI for predictive decision-making, but datasets were largely simulated and not representative of diverse real user behavior.

These studies collectively highlight the feasibility of adaptive automation but also expose gaps in real-world integration, particularly in terms of mobile interaction and user-centered adaptive AI.

2.1.4 Research Gap

Despite the progress, current systems and prototypes frequently exhibit limitations:

- Heavy reliance on static rules or pre-defined scenarios rather than adaptive learning from real user behavior.
- Limited incorporation of real user data, reducing the effectiveness and personalization of predictive models.
- Fragmented system architectures: Existing solutions often lack integrated simulation, AI analysis, and mobile interface, preventing comprehensive testing and user feedback.
- Scalability and accessibility issues: Some prototypes are device-specific or too complex for broad adoption in research settings.

This research aims to address these gaps by developing a simulated smart home environment using Node-RED, integrating AI-driven predictive modeling based on real user data, and enabling real-time control via a Flutter mobile app. This integrated approach allows for testing and evaluation of adaptive automation strategies without the need for full physical IoT deployments.

2.2 Mobile App Framework – Flutter

Flutter is an open-source UI software development kit developed by Google, designed for creating natively compiled apps across mobile, web, and desktop platforms using a single codebase. It uses the Dart programming language and follows a widget-based architecture [17]. Research has shown that Flutter's cross-platform capabilities and rich widget system are particularly suitable for IoT and smart home apps where real-time monitoring and user interactivity are critical [23].

2.2.1 Characteristics

- Cross-platform development (iOS, Android, web, desktop) from one codebase [17]
- Rich widget system for modern UI design [23]
- Hot-reload for rapid testing and iteration [17]
- Strong ecosystem of plugins for device access and APIs [21]

- High rendering performance via Skia engine [23]

2.2.2 Function in Project

In this project, Flutter is used to develop the mobile app interface. It communicates with Node-RED via MQTT for sending user preferences, retrieving real-time sensor data, and displaying predictive results from the Flask API. Flutter’s performance ensures responsive display of real-time sensor values, which is essential for AI-assisted smart home monitoring systems [21].

2.2.3 Alternatives – Flutter

- React Native – An open-source JavaScript framework for building cross-platform mobile apps [23]. Strengths of Flutter is it has large community, fast development with hot-reload, good cross-platform support. And its weaknesses are slightly slower rendering, less “native feel” compared to Flutter.

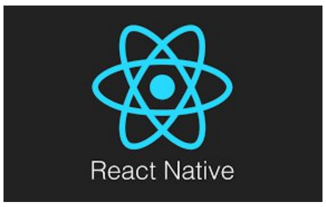


Figure 2.1 React Native

- Xamarin – A .NET-based framework for building apps for multiple platforms [21]. Strengths of is it has strong integration with Microsoft ecosystem, supports cross-platform development. And its weaknesses are produces larger apps with heavier runtime overhead, which can reduce efficiency in resource-constrained smart home scenarios.

Figure 2.2 Xamarin

- Native Development (Swift/Kotlin) – Platform-specific development for iOS (Swift) or Android (Kotlin) [23]. Strengths of is it has maximum performance and direct hardware access. And its weaknesses are requires separate codebases for each platform, slowing iterative development and updates.

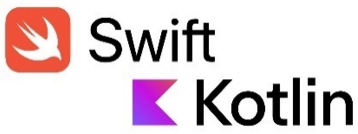


Figure 2.3 Native Development (Swift/Kotlin)

2.2.4 Summary

Flutter is chosen for this project because it offers an optimal balance between performance, cross-platform support, and rapid development, which is essential for a smart home app requiring real-time monitoring and responsive UI. Compared to alternatives, React Native provides good cross-platform support but slightly slower rendering and less native feel, Xamarin integrates well with Microsoft ecosystems but introduces heavier runtime overhead, and native development (Swift/Kotlin) achieves maximum performance but requires separate codebases, slowing iterative updates. Flutter's rich widget system, hot-reload capability, and strong ecosystem make it particularly suitable for integrating with IoT protocols and AI-driven predictive data in smart home applications.

Table 2.1 Comparison Table of Flutter vs Alternatives

Feature/Criteria	Flutter (Chosen)	React Native	Xamarin	Native (Swift/Kotlin)
Language	Dart	JavaScript	C# (.NET)	Swift (iOS) / Kotlin (Android)
Cross-platform	Yes (mobile, web, desktop)	Yes (mobile, web)	Yes (mobile, desktop)	No (separate per platform)
Performance	High (Skia engine)	Moderate (JS bridge)	Heavy runtime overhead	Maximum (native compiled)
Development Speed	Fast (hot-reload)	Fast (hot-reload)	Moderate	Slow (duplicate effort)

UI Customization	Rich widget system	Relies on native comp.	Limited customization	Full control
Ecosystem & Community	Strong, growing	Very large	Moderate	Strong but fragmented
Best Use Case	Modern real-time apps	Quick prototyping	Enterprise Microsoft	Performance-critical apps

2.3 Automation Middleware – Node-RED

Node-RED is a low-code, flow-based development tool designed for wiring together devices, APIs, and services. It is widely adopted in IoT and smart home projects due to its visual drag-and-drop interface and strong ecosystem of nodes [20][22].

2.3.1 Characteristics

- Visual programming with flow-based design. [22]
- Built-in nodes for IoT protocols (MQTT, HTTP, WebSocket). [21]
- Extensible with custom function nodes (JavaScript). (JavaScript) (Karakus et al., 2020).
- UI Dashboard and SVG support for visualization [20].
- Open-source, lightweight, and widely supported [22].

2.3.2 Function in Project

Node-RED acts as the central middleware in the smart home system. It handles:

- MQTT communication with the Flutter app [20].
- Collecting/storing sensor data into InfluxDB [19].
- Calling external APIs (e.g., weather) [19].
- Providing UI elements (gauges, templates, SVG graphics) for simulation [20].

- Passing predictions from Python/Flask API back to the mobile app [18].

2.3.3 Alternatives – Node-RED

- Home Assistant – Open-source home automation platform with wide device support [26]. Strengths of is it has highly extensible, strong community, integrates with many IoT devices. And its weaknesses is complex setup, steeper learning curve for custom flows.

Figure 2.4 Home Assistant

- OpenHAB – Open-source smart home platform with rule-based automation engine [26]. Strengths of is it has fine-grained control, strong interoperability with various devices. And its weaknesses are requires technical knowledge, slower for rapid prototyping, less intuitive for beginners.

-
- Figure 2.5 OpenHAB

- Blynk – Mobile-first IoT dashboard platform for visualizing device data [20]. Strengths of is it has Simple setup, excellent for creating dashboards quickly. And its weaknesses are limited flow-based logic and integration with AI or complex automation.

Figure 2.6 Blynk

2.3.4 Summary

Node-RED is chosen as the automation middleware for this project due to its ease of use, flexibility, and rapid prototyping capabilities. Its visual flow-based development style allows developers to intuitively understand and debug system logic, making it ideal for simulation-based smart home projects. Compared to alternatives, Home Assistant and OpenHAB rely on textual configuration or complex rule engines, which are less intuitive and slower for prototyping, while Blynk offers simple dashboards but lacks comprehensive flow-based logic and integration capabilities. Node-RED supports major IoT protocols (MQTT, HTTP, WebSocket), integrates easily with AI models and external APIs, and provides SVG-based visualization for emulating smart home environments. Overall, it balances simplicity and extensibility, making it well-suited for research prototypes involving simulation, real-time control, and AI-driven adaptive automation.

Table 2.2 Comparison Table of Node-RED vs Alternatives

Feature/Criteria	Node-RED (Chosen)	Home Assistant	OpenHAB	Blynk
Development Style	Visual flow (drag & drop)	Config + YAML automations	Config + rule engine	Mobile-first dashboard
Protocol Support	MQTT, HTTP, WebSocket	MQTT, Zigbee, Z-Wave	MQTT, KNX, Modbus	MQTT, HTTP
Ease of Use	Very easy (low-code)	Moderate (steeper setup)	Moderate/Advanced	Very easy (but limited)
Visualization	UI Dashboard + SVG	Built-in dashboard	HABPanel (basic)	Mobile app only
Community/Ecosystem	Strong, growing	Very large	Large but technical	Moderate
Best Use Case	IoT prototyping, emulation	Full home automation	Advanced smart homes	Quick IoT dashboards

2.4 Artificial Intelligence and Machine Learning in Smart Homes– Python

Artificial Intelligence (AI) plays a crucial role in modern smart home systems by enabling predictive analytics and adaptive decision-making. By analyzing environmental data (e.g., temperature, lighting, air quality) alongside real user behavior, AI models can anticipate user needs, optimize energy usage, and maintain comfort and safety within the home. For example, AI can predict when a room will be occupied and adjust lighting or temperature accordingly, or forecast air quality changes and proactively operate air purifiers.

Machine learning algorithms, such as regression models, decision trees, and ensemble methods, allow the system to learn patterns from historical and real-time data. These predictive insights are then used to make automated decisions, supporting adaptive automation beyond static rule-based controls. This approach enhances personalization, energy efficiency, and well-being, while reducing manual intervention.

Python is commonly used to implement these AI models due to its readability, extensive libraries (e.g., scikit-learn, TensorFlow, PyTorch), and strong integration capabilities with IoT pipelines [21]. In this project, Python serves as the development environment for training and deploying machine learning models that analyze both environmental and real user data, providing predictions that inform the smart home's adaptive automation system.

2.4.1 Characteristics

- Simple syntax and high readability [24].
- Strong libraries for ML (scikit-learn, TensorFlow, PyTorch) [18].
- Excellent ecosystem for data processing (pandas, numpy) [18].
- Easy API development with Flask or FastAPI [21].
- Large community and research support [24].

2.4.2 Function in Project

- Retrieve historical data from InfluxDB [19].
- Preprocess data with pandas and numpy [18].

- Train model (RandomForestRegressor via scikit-learn) [18].
- Store predictions back into InfluxDB [19].
- Expose results via Flask API for use in Flutter and Node-RED [21].

2.4.3 Alternatives – Python

- TensorFlow/PyTorch – Deep learning frameworks for complex neural network tasks. Strengths of is it has highly powerful for computer vision, NLP, and deep learning [27]. And its weaknesses are overkill for lightweight tabular data prediction, complex setup, and integration with IoT pipelines.

Figure 2.7 TensorFlow/PyTorch

- XGBoost/LightGBM – Gradient boosting frameworks for high-performance tabular data modelling [28]. Strengths of is it has very high accuracy and performance on tabular datasets. And its weaknesses are requires extensive hyperparameter tuning, less straightforward IoT integration.
- R Language – Programming language specialized in statistics and data analysis [29]. Strengths of is it has strong for statistical modeling and research. And its weaknesses are limited support for real-time IoT integration and API development compared to Python.

Figure 2.8 R Language

2.4.4 Summary

Python with RandomForestRegressor was chosen for this project because it provides a good balance of simplicity, interpretability, and integration with IoT pipelines, which is critical for real-time smart home monitoring and AI-driven predictions. Compared to alternatives, TensorFlow or PyTorch offer powerful deep learning capabilities but are more complex to set up and overkill for lightweight predictive tasks. XGBoost or LightGBM can deliver high performance on tabular data but require more hyperparameter tuning and are less straightforward to integrate with IoT APIs. R is strong for statistical analysis but has limited support for real-time IoT integration. Overall, Python with RandomForest ensures effective model deployment, ease of development, and seamless communication with Flutter and Node-RED, making it the most suitable choice for the system's adaptive automation needs.

Table 2.3 Comparison Table of Python vs Alternatives

Feature/Criteria	Python + RandomForest (Chosen)	TensorFlow/PyTorch	XGBoost/LightGBM	R Language
Learning Type	Supervised regression	Deep learning (NNs)	Gradient boosting trees	Statistical models + ML
Ease of Use	Easy (scikit-learn)	Harder (complex setup)	Moderate (tuning-heavy)	Easy for statistics, harder for APIs
Performance on Tabular Data	High	Overkill for simple tasks	Very high	Moderate
Interpretability	High	Low (black-box NN)	Moderate	High
Integration with IoT	Easy (Flask, APIs)	More complex	Moderate	Limited outside analytics
Best Use Case	Lightweight predictive tasks	Computer vision, NLP	Competitive ML tasks	Data analysis, research

2.5 Database – InfluxDB

InfluxDB is an open-source time-series database optimized for storing and querying timestamped data, especially for IoT and sensor-based apps [19][25]. Its design is particularly suitable for smart home monitoring systems where large volumes of real-time sensor data must be stored and analyzed efficiently.

2.5.1 Characteristics

- Time-series optimized (millions of inserts per second) [25]
- Schema-less design with buckets [19]
- Built-in queries for time windows, averages, and trends [19]
- Integration with popular IoT tools [25]
- Lightweight and scalable [19]

2.5.2 Function in Project

InfluxDB stores:

- User preferences (from mobile app via MQTT) [19]
- Environmental sensor data (from Node-RED) [20]
- Predicted values (from Python model) [18]

  Data can then be queried by both the mobile app and ML pipeline, enabling real-time monitoring and AI-driven prediction integration in smart home systems (Moghadam et al., 2022).

2.5.3 Alternatives

- MySQL/PostgreSQL – Traditional relational databases for general-purpose data storage[25]. Strengths of is it has strong SQL support, reliable and widely used. And its weaknesses are moderate write performance, limited suitability for high-frequency IoT time-series data.

Figure 2.9 MySQL/PostgreSQL

- MongoDB – Document-oriented NoSQL database for flexible schema design [19]. Strengths of is it has flexible data models, easy to scale horizontally. And its weaknesses are not optimized for time-series data, slower querying for high-frequency sensor inputs.

Figure 2.10 MongoDB

- TimescaleDB – PostgreSQL extension specialized in time-series data [25]. Strengths of is it has excellent support for time-series queries, integrates with SQL ecosystem. And its weaknesses are more complex setup and management compared to InfluxDB.

Figure 2.11 TimescaleDB

2.5.4 Summary

InfluxDB was selected for its native time-series optimization, high write performance, scalability, and seamless integration with IoT pipelines, making it ideal for real-time smart home monitoring and AI-based predictive analytics[19][21]. Alternatives such as

MySQL/PostgreSQL are general-purpose relational databases with moderate write performance and limited real-time suitability. MongoDB offers flexible schema design but is not optimized for high-frequency time-series data. TimescaleDB provides strong time-series support but has a higher setup complexity. Overall, InfluxDB strikes a practical balance of performance, ease of setup, and suitability for sensor data storage in adaptive smart home environments.

Table 2.4 Comparison Table of Influx DB vs Alternatives

Feature/Criteria	InfluxDB (Chosen)	MySQL/PostgreSQL	MongoDB	TimescaleDB
Data Model	Time-series optimized	Relational	Document-based	Relational + time-series
Write Performance	Very high (optimized)	Moderate	High	High
Querying	Time-series queries	SQL	JSON queries	SQL + time-series funcs
Schema Flexibility	Schema-less	Fixed schema	Flexible schema	Relational schema
IoT/Real-time Suitability	Excellent	Limited	Moderate	Excellent
Setup Complexity	Low	Moderate	Moderate	Higher
Best Use Case	Sensor/IoT data storage	General DB use	Flexible data models	Time-series + SQL apps

2.6 Related Work

2.6.1 Academic Studies

Adaptive Smart Home Automation: Prior research “Domain Selection and Adaptation in Smart Homes” by P. Rashidi and D. J. Cook is real, involving activity recognition and adapting smart home environmentsexplored adaptive control strategies in residential environments, focusing on activity recognition and personalized automation. Besides, machine Learning Integration: Studies by “Review on the Application of Artificial Intelligence in Smart Homes” by Xiao Guom Zhenjiang Shen and “Enhancing Smart Home Design with AI Models: A Case Study” by Rahman applied AI for predicting user behavior and environmental changes. However, many relied on simulated data rather than real user inputs.

2.6.2 Prototypes and Simulations

Node-RED-based simulations have been used for testing smart home logic flows, but few integrated AI-driven adaptive automation. Besides mobile apps for control often implemented static rules without predictive adjustments.

2.6.3 Commercial Attempts

Apple HomeKit, Google Home, Amazon Alexa: Widely adopted but limited to rule-based automation with minimal predictive functionality. Moreover specialized AI Home Solutions: Some proprietary platforms incorporate AI but are not research-friendly or customizable for simulation and testing.

2.6.4 Research Gap

Existing systems often lack integration of real user data into predictive models. On the other hand limited research combines simulation, mobile interfaces, and AI-based adaptive automation in one platform.

2.7 Summary of Literature Review

This chapter reviewed the evolution of smart home systems, highlighting the shift from static rule-based control to adaptive automation using AI and IoT. Key insights:

- Adaptive systems improve personalization, energy efficiency, and user safety.
- Flutter is suitable for cross-platform mobile applications in smart home contexts.

- Node-RED provides low-code middleware for IoT simulation and integration.
- Python enables predictive modeling using real user and environmental data.
- InfluxDB supports real-time time-series data storage critical for AI integration.
- Existing research and commercial solutions often lack real user data integration, adaptive AI, and full mobile + simulation workflow.

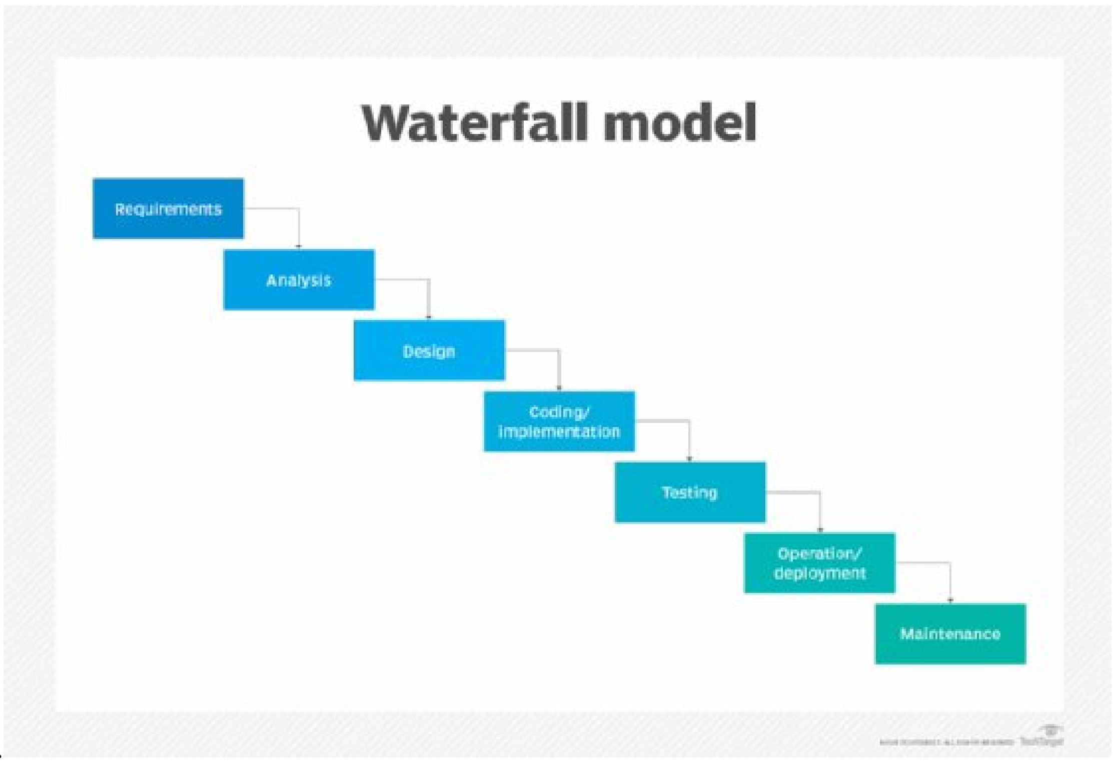
The literature review establishes the rationale for developing a simulated, AI-driven, mobile-enabled smart home system, addressing the identified gaps and supporting the research objectives.

Chapter 3
System Methodology/Approach OR System Model

3.1 System Development Models

3.1.1 Waterfall Model

The Waterfall model follows a linear and step-by-step approach, where each stage—such as requirement analysis, system design, coding, testing, deployment, and maintenance—must be completed before moving to the next. It is most effective in projects where the requirements are clearly defined, stable, and unlikely to change. Weaknesses**:** This model is inflexible to changing requirements, provides limited opportunities for user feedback during development, and errors discovered late can be costly to fix.



.

Figure 3.1 Water Model Example

3.1.2 Agile Model

The Agile methodology is iterative and adaptive, promoting incremental development through short cycles called sprints. It emphasizes frequent testing, continuous user involvement, and flexibility to adjust to changing requirements, ensuring the system evolves in line with user needs and feedback. Weaknesses: Agile can be less predictable in terms of timeline and budget, requires active user participation, and may lead to scope creep if not properly managed.

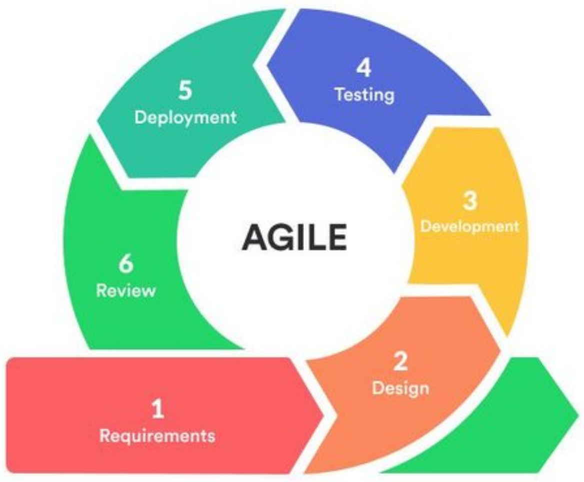


Figure 3.2 Agile Model Example

3.1.3 Prototype Model

The Prototype model focuses on rapid development of preliminary versions of the system to gather feedback at an early stage. These prototypes are refined repeatedly based on user input until they evolve into the final product, ensuring closer alignment with user expectations. Weaknesses**:** It can lead to incomplete or insufficient documentation, increased development cost if multiple iterations are needed, and users may mistake prototypes for finished products, causing unrealistic expectations.

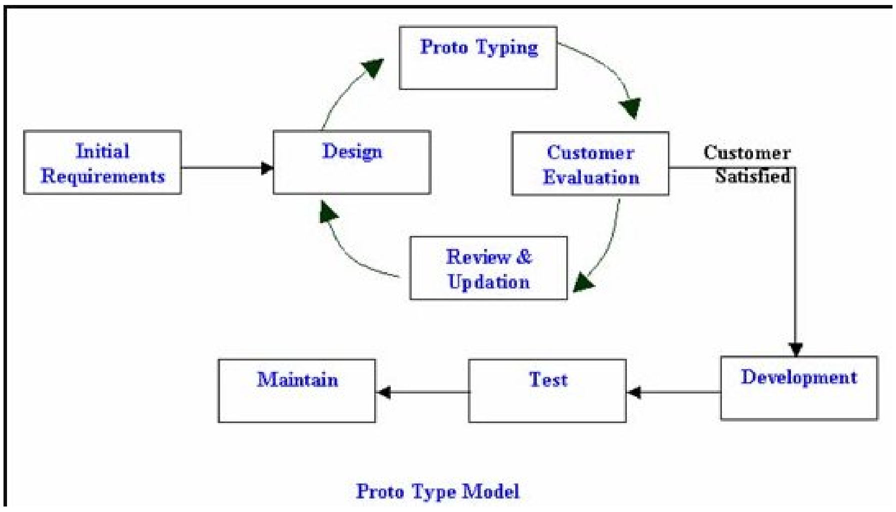


Figure 3.3 Prototype Model Example

3.1.4 Chosen Model: Agile Model

The Agile model was selected as the development methodology because it closely aligns with the project's dynamic and iterative workflow. Unlike rigid models such as Waterfall, Agile provided the flexibility to incrementally develop and refine the system components. Throughout this project, the Agile cycle was fully realized: the **development phase** was completed with the successful implementation of the Node-RED flows, SVG-based emulator interface, Flutter mobile app, Python-based AI prediction module, MQTT communication, and InfluxDB database integration. This was followed by a **comprehensive testing phase**, where the performance and functionality of the system were validated, including the accuracy of the AI machine learning predictions to ensure reliable operation across all modules.

The deployment phase, specifically the release of the mobile app on the Google Play Store, was not carried out, as the system was determined to be more suited for academic validation and prototype demonstration rather than immediate public distribution. Finally, a **review phase** was conducted to evaluate the overall outcomes and performance of the system, confirming that the objectives of developing an adaptive smart home control and monitoring solution were successfully met.

In summary, Agile proved to be the most effective methodology, as it supported step-by-step development, continuous testing, and adaptive refinement, ultimately ensuring that the project reached a complete and validated prototype stage within the scope of FYP2.

3.2 System Requirements

The hardware requirements define the essential devices and equipment needed to support the smart home control system. These components provide the necessary computing power, connectivity, and user interaction to ensure smooth operation and reliable communication between the system modules.

3.2.1 Hardware Requirements

The hardware requirements define the essential devices and equipment needed to support the smart home control system. These components provide the necessary computing power, connectivity, and user interaction to ensure smooth operation and reliable communication between the system modules.

Table 3.1 Hardware Requirements for Smart Home Control System

Component	Function	Specifications
Personal Computer (PC/Laptop)	Hosts Node-RED server (emulator), InfluxDB database server (local storage), Python ML server (prediction + Flask API).	Windows 10 or above, 8 GB RAM, 20 GB free storage.
Mobile Device (Smartphone)	Runs Flutter-based mobile app for monitoring/control; provides hotspot for MQTT same-network communication.	Android 8.0+, 3 GB RAM minimum, stable connectivity.
Wi-Fi Router	Provides internet access for Node-RED HTTP requests (e.g., Weather API); maintains communication between components.	Standard router with stable broadband/Wi-Fi connection.

Table 3.1 listed the hardware requirements that is used for the project. The hardware setup ensures smooth execution and integration of the system. The PC acts as the central server, managing Node-RED for automation flows, InfluxDB for storing time-series data, and Python ML for predictions and API services. The smartphone serves as the client-side interface where users can monitor and control devices using the Flutter app. Additionally, it provides a hotspot, enabling MQTT communication between devices on the same network.

Finally, the Wi-Fi router supports internet connectivity, particularly for handling external API requests and facilitating overall communication stability.

3.2.2 Software Requirements

The software requirements specify the tools, frameworks, and platforms used to develop, run, and manage the smart home control system. Each software component plays a critical role in enabling application development, data processing, system integration, and visualization.

Table 3.2 Software Requirements for Smart Home Control System

Software/Tool	Function	Specifications/Notes
Visual Studio Code (VS Code)	IDE for Flutter, Python, and Node-RED development.	Lightweight, cross-platform editor with extensions.
Flutter & Dart	Framework and language for mobile app development.	Flutter SDK + Dart installed, 2.8 GB storage, 64-bit OS, 8 GB RAM.
Python Language	Runs ML algorithms, Flask API, and handles data processing.	Python 3.8+, libraries: scikit-learn, pandas, numpy, matplotlib, Flask.
Node.js & Node-RED	Node.js runtime for Node-RED; Node-RED manages flows, MQTT, InfluxDB integration, and API handling.	Node.js LTS version, browser support for Node-RED dashboard.
InfluxDB	Time-series database for user preferences, IoT data, and ML predictions.	64-bit OS, 3 GB RAM, 1.5 GB storage.
Inkscape	Creates SVG graphics for smart home emulator integrated in Node-RED.	Windows 10/macOS/Linux, 3 GB RAM, 1 GB storage.
FlutterFlow	No-code/low-code platform for designing and scaffolding Flutter mobile apps.	Web-based, requires browser support, internet connection, exports Flutter code.

Table 3.2 listed the software requirements that is used for the project. The software stack was carefully selected to balance development flexibility, compatibility, and performance. Visual Studio Code serves as the primary development environment, supporting Flutter, Python, and Node-RED coding. Flutter and Dart are chosen for mobile app development due to their ability to deliver cross-platform apps efficiently. Python powers the machine learning models and Flask API, enabling intelligent predictions and communication with the Flutter app. Node.js provides the runtime environment required by Node-RED, which is responsible for orchestrating device flows, integrating MQTT messaging, and linking with InfluxDB for real-time data storage. InfluxDB is essential for handling large volumes of time-series IoT data. Inkscape is used to design SVG graphics that bring a visual emulator of the smart home into the Node-RED dashboard. Finally, FlutterFlow is employed as a no-code/low-code platform to accelerate UI design and app scaffolding, allowing rapid prototyping and seamless integration with the Flutter codebase.

3.3 Functional Requirements

A functional requirement is a specific description of a system's behavior or action that the system must perform. In the context of the Smart Home Control and Monitoring System, functional requirements define the exact operations and tasks that each component must execute to achieve the project's objectives of smart home emulation, adaptive automation, monitoring, and predictive intelligence. These requirements ensure that the system can control devices, process data, perform AI predictions, and enable communication between all modules.

Functional Requirements of the Proposed System

Table 3.3 Component Functional Requirements Table

ID	Component	Functional Requirement Description
FR-01	Node-RED SVG Smart Home Emulator	Must provide a visual interface of the virtual smart home using SVG graphics.
FR-02	Node-RED SVG Smart Home Emulator	Must allow users to observe real-time device states (lights, AC, purifier, doors, fragrance).

FR-03	Node-RED SVG Smart Home Emulator	Must allow users to interact with devices (e.g., turn ON/OFF, adjust settings) through the emulator.
FR-04	Python Script (AI Module)	Must execute machine learning algorithms for predicting temperature, lighting, and fragrance preferences based on historical and contextual data.
FR-05	Python Script (AI Module)	Must expose AI predictions through a Flask API for integration with the mobile app.
FR-06	Flutter Mobile App	Must allow users to control virtual home devices (ON/OFF, adjust settings).
FR-07	Flutter Mobile App	Must publish user-side data (temperature, lighting, fragrance) via MQTT to Node-RED and InfluxDB.
FR-08	Flutter Mobile App	Must display AI predictions retrieved via Flask API.
FR-9	InfluxDB Database	Must store user-side data, including preferences and control inputs.
FR-10	InfluxDB Database	Must store predicted data generated by the AI module.
FR-11	InfluxDB Database	Must allow data retrieval by both Python scripts and Flutter mobile app for analysis and display.

The functional requirements highlight the main system capabilities and describe how each component contributes to achieving the overall goal. The Node-RED emulator provides a visual and interactive environment, the Python script with machine learning ensures predictive intelligence, the Flutter app provides user control and monitoring, and InfluxDB acts as the backbone for storing and retrieving both real and predicted data. Together, these requirements ensure that the system can operate effectively as an integrated smart home solution.

3.4 Non-Functional Requirements

Non-functional requirements describe how the system should perform in terms of quality attributes, such as reliability, scalability, and performance. While functional requirements define what the system must do, non-functional requirements determine *how well the system must do it*. For the Smart Home Control and Monitoring System, these requirements ensure the system not only performs its functions but does so consistently, efficiently, and with user satisfaction.

Non-Functional Requirements of the Proposed System

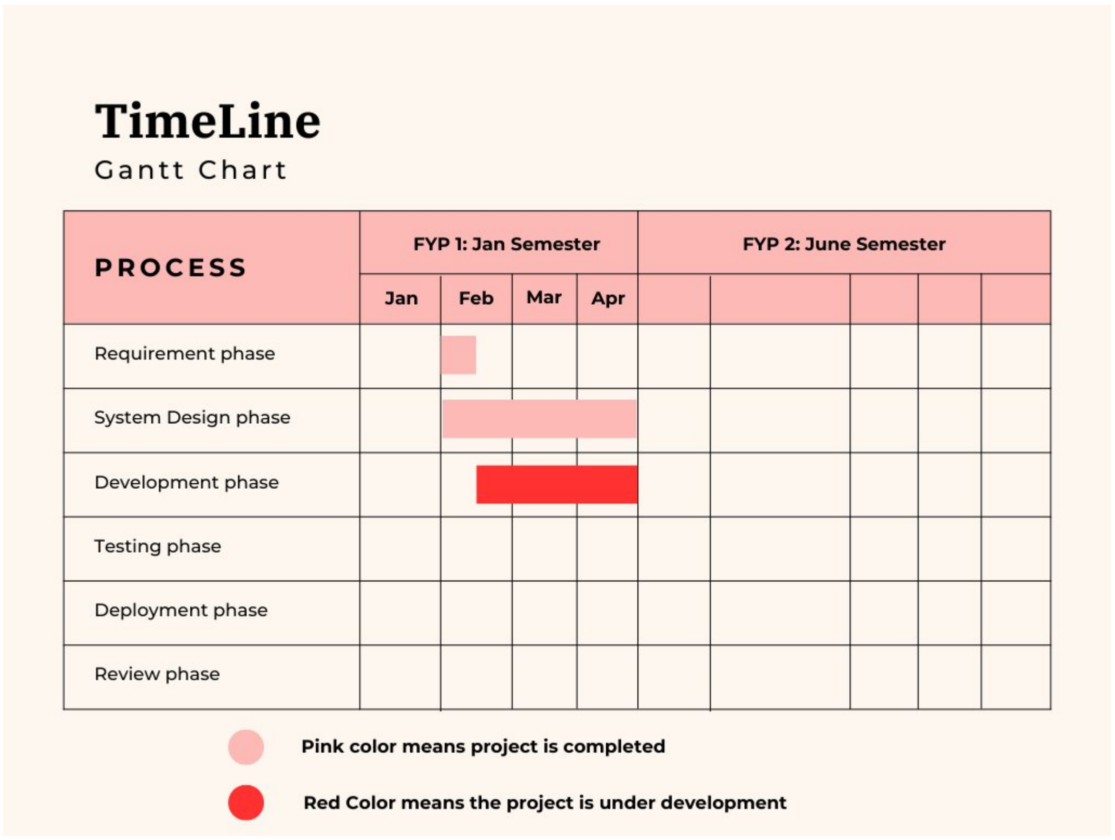
Table 3.4 Component Non-Functional Requirements Table

Attribute	Description in System Context
Reliability	The system should ensure consistent communication between components (Flutter → MQTT → Node-RED → InfluxDB → Python) with minimal data loss. If one module fails, other components should resume communication once the service is restored.
Scalability	The architecture should allow expansion, such as adding new IoT devices (e.g., humidity or motion sensors) or extending the AI model to include more prediction features, without redesigning the entire system.
Performance	The system should provide near real-time communication with minimal latency (ideally <1 second for MQTT message delivery). Database queries should remain efficient even as the dataset grows to thousands of time-series records.
Usability	The mobile app interface should be simple, user-friendly, and intuitive, allowing users to control and monitor devices without technical expertise.
Maintainability	The system should adopt a modular design so that updates, such as upgrading the AI model or adding new visualization features in Node-RED, can be applied without affecting the entire system.
Security (Future)	The MQTT communication should support authentication (username/password) and potentially SSL/TLS encryption in future iterations to protect data transmission.

The non-functional requirements establish the quality standards that guide system operation. Reliability ensures that communication and data handling are consistent, while scalability guarantees that the system can grow with additional devices or features. Performance addresses speed and responsiveness, ensuring real-time interactivity. Usability focuses on providing a smooth user experience through the Flutter app, while maintainability highlights the importance of modularity for future updates. Security, although optional at this stage, provides a roadmap for enhancing system trustworthiness in future developments.

3.5 Project Milestone

3.5.1 Project I Timeline



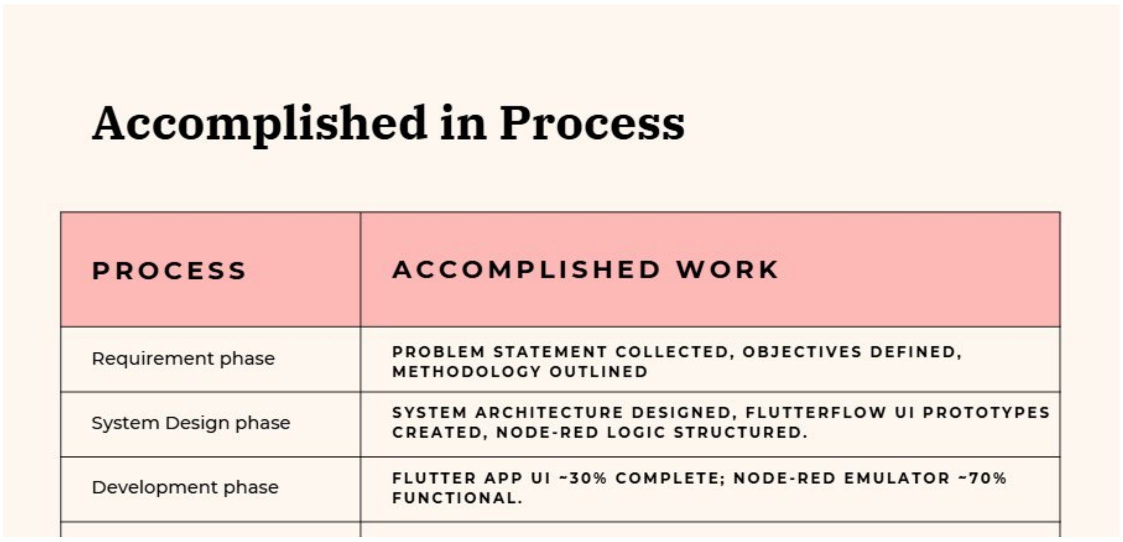
TimeLine
Gantt Chart

PROCESS	FYP 1: Jan Semester				FYP 2: June Semester				
	Jan	Feb	Mar	Apr					
Requirement phase		▨							
System Design phase		▨	▨	▨					
Development phase		🟥	🟥	🟥					
Testing phase									
Deployment phase									
Review phase									

▨ Pink color means project is completed

🟥 Red Color means the project is under development

Figure 3.4 Gannt Chart for the TimeLine for project I



Accomplished in Process

PROCESS	ACCOMPLISHED WORK
Requirement phase	PROBLEM STATEMENT COLLECTED, OBJECTIVES DEFINED, METHODOLOGY OUTLINED
System Design phase	SYSTEM ARCHITECTURE DESIGNED, FLUTTERFLOW UI PROTOTYPES CREATED, NODE-RED LOGIC STRUCTURED.
Development phase	FLUTTER APP UI ~30% COMPLETE; NODE-RED EMULATOR ~70% FUNCTIONAL.

Figure 3.5 Accomplish in Process for project I

Project I, conducted in the current semester, concentrated on the initial phases of the system development lifecycle. These included:

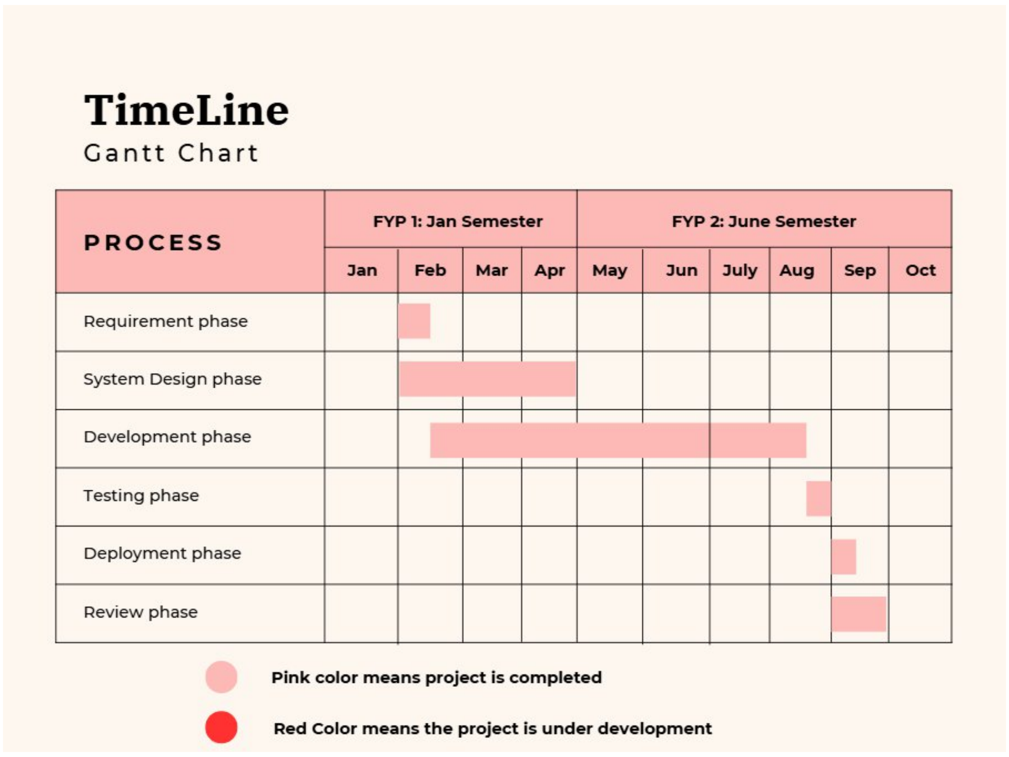
- Requirement Analysis
- System Design
- Partial Development

Together, these phases accounted for approximately 40% of the overall system progress. The focus during this stage was on establishing the system framework, core communication setup, and prototyping essential functionalities.

Accomplish In Project I:

- **Requirement Phase – Completed:** Collected problem statement details, reviewed related literature, defined objectives, and developed the system methodology design.
- **Design Phase – Completed:** Designed the system architecture, created FlutterFlow-based UI prototypes, and outlined Node-RED logic structures.
- **Development Phase (Partial) – In Progress:**
    - Flutter mobile app: Around 30% completed, excluding MQTT-based device control.
    - Node-RED and SVG UI: Approximately 70% functional, supporting dynamic updates and real-time status visualization.

3.5.2 Project II Timeline



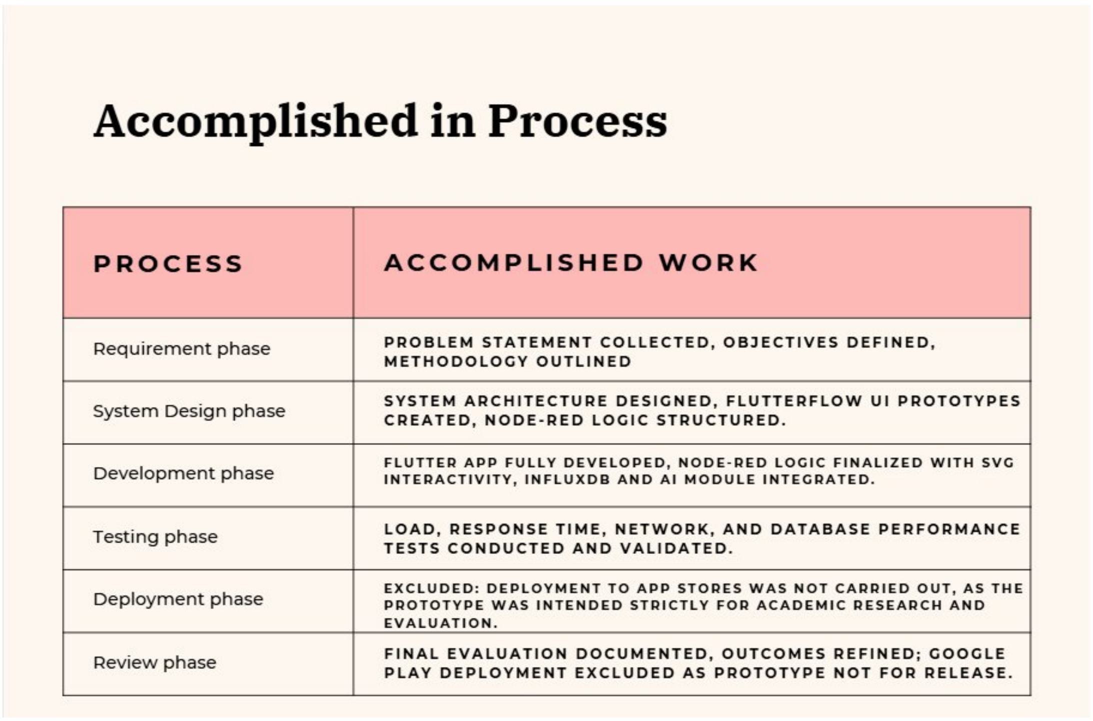
TimeLine
Gantt Chart

PROCESS	FYP 1: Jan Semester				FYP 2: June Semester					
	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sep	Oct
Requirement phase		🟥								
System Design phase		🟥	🟥	🟥						
Development phase		🟥	🟥	🟥	🟥	🟥	🟥	🟥		
Testing phase								🟥		
Deployment phase									🟥	
Review phase									🟥	

● Pink color means project is completed

● Red Color means the project is under development

Figure 3.6 Gannt Chart for the TimeLine for project II



Accomplished in Process

PROCESS	ACCOMPLISHED WORK
Requirement phase	PROBLEM STATEMENT COLLECTED, OBJECTIVES DEFINED, METHODOLOGY OUTLINED
System Design phase	SYSTEM ARCHITECTURE DESIGNED, FLUTTERFLOW UI PROTOTYPES CREATED, NODE-RED LOGIC STRUCTURED.
Development phase	FLUTTER APP FULLY DEVELOPED, NODE-RED LOGIC FINALIZED WITH SVG INTERACTIVITY, INFLUXDB AND AI MODULE INTEGRATED.
Testing phase	LOAD, RESPONSE TIME, NETWORK, AND DATABASE PERFORMANCE TESTS CONDUCTED AND VALIDATED.
Deployment phase	EXCLUDED: DEPLOYMENT TO APP STORES WAS NOT CARRIED OUT, AS THE PROTOTYPE WAS INTENDED STRICTLY FOR ACADEMIC RESEARCH AND EVALUATION.
Review phase	FINAL EVALUATION DOCUMENTED, OUTCOMES REFINED; GOOGLE PLAY DEPLOYMENT EXCLUDED AS PROTOTYPE NOT FOR RELEASE.

Figure 3.7 Accomplish in Process for project II

In Project I, the scope for Project II was defined to complete the remaining 60% of the system development lifecycle, covering development, system testing, deployment, and final review. The following key tasks were planned and carried out in Project II:

Planned Work (from Project I):

- Complete the Flutter app with enhanced UI, device status display, and user feedback integration.
- Finalize Node-RED automation logic with advanced SVG interactivity (e.g., animation, dynamic controls).
- Implement InfluxDB for historical time-series data storage.
- Develop the AI module for adaptive control suggestions based on environmental and behavioral data.
- Conduct system-level testing to validate performance under simulated smart home scenarios.
- Deploy the prototype into the Google Play Store.
- Perform a final review and evaluation of the system.

Accomplished in Project II:

- Flutter app fully developed with enhanced UI, device status display, and user feedback integration.
- Node-RED automation logic finalized with advanced SVG interactivity.
- InfluxDB successfully implemented for storing and analyzing historical time-series data.
- AI module developed and integrated for adaptive control using environmental and behavioral data.
- Comprehensive system-level testing conducted, validating performance and functionality.
- Final review and evaluation performed, documenting outcomes and areas for future improvement.
- Deployment to the Google Play Store excluded, As the prototype served academic validation purposes rather than public distribution.

Project I established 40% of the system with requirements, design, and partial development, while Project II completed the remaining 60% through full development, AI integration, system testing, and review, excluding public deployment.

3.6 Estimated Cost

The development of this project relies on the developer's personal hardware, including a mobile phone and laptop, which cover the primary device requirements. In addition, all software tools and platforms used—such as Flutter SDK, Dart, Visual Studio Code, Python, Node-RED, Node.js, InfluxDB, and Inkscape—are open-source or freely downloadable from the Internet. Similarly, the MQTT broker (Mosquitto) and MQTT client tool (MQTTX) are available at no charge. As a result, no additional costs were incurred for the development of the system.

3.7 Concluding Remark

This chapter outlined the development methodology, requirements, and milestones undertaken for the Smart Home Control and Monitoring System. The Agile model was adopted as the primary development approach, providing the flexibility and adaptability necessary to manage iterative design, development, and testing cycles. Both hardware and software requirements were defined to ensure seamless integration between system components, including Flutter, Node-RED, InfluxDB, Python, and MQTT.

Furthermore, the chapter detailed the system's functional and non-functional requirements, ensuring that the project not only fulfills its intended operations—such as device control, predictive intelligence, and real-time monitoring—but also upholds quality attributes like reliability, scalability, performance, and usability. The milestone breakdown highlighted the phased progress across Project I and Project II, demonstrating a structured approach from requirement analysis to testing and final evaluation, with deployment excluded for prototype-focused validation.

In summary, Chapter 3 established the foundation for implementing and validating the proposed system. By clearly defining the methodology, requirements, and project plan, it ensures that the subsequent implementation and testing stages can be executed in a systematic and goal-oriented manner, ultimately supporting the achievement of the project objectives.

Chapter 4
System Design

4.1 System Architecture

The system architecture of the proposed smart home prediction and control system is designed to integrate data storage, machine learning, automation, and user interaction into a cohesive framework. It defines how the various components—InfluxDB, Node-RED, Python-based machine learning, and the mobile application—interact to achieve intelligent decision-making and seamless device control. This architecture ensures that real-time data from the user is collected, processed, stored, and transformed into actionable insights that can be used to improve the overall smart home experience. By combining predictive intelligence with user-driven actions, the system provides a reliable, adaptive, and user-friendly platform for managing devices within the home environment.

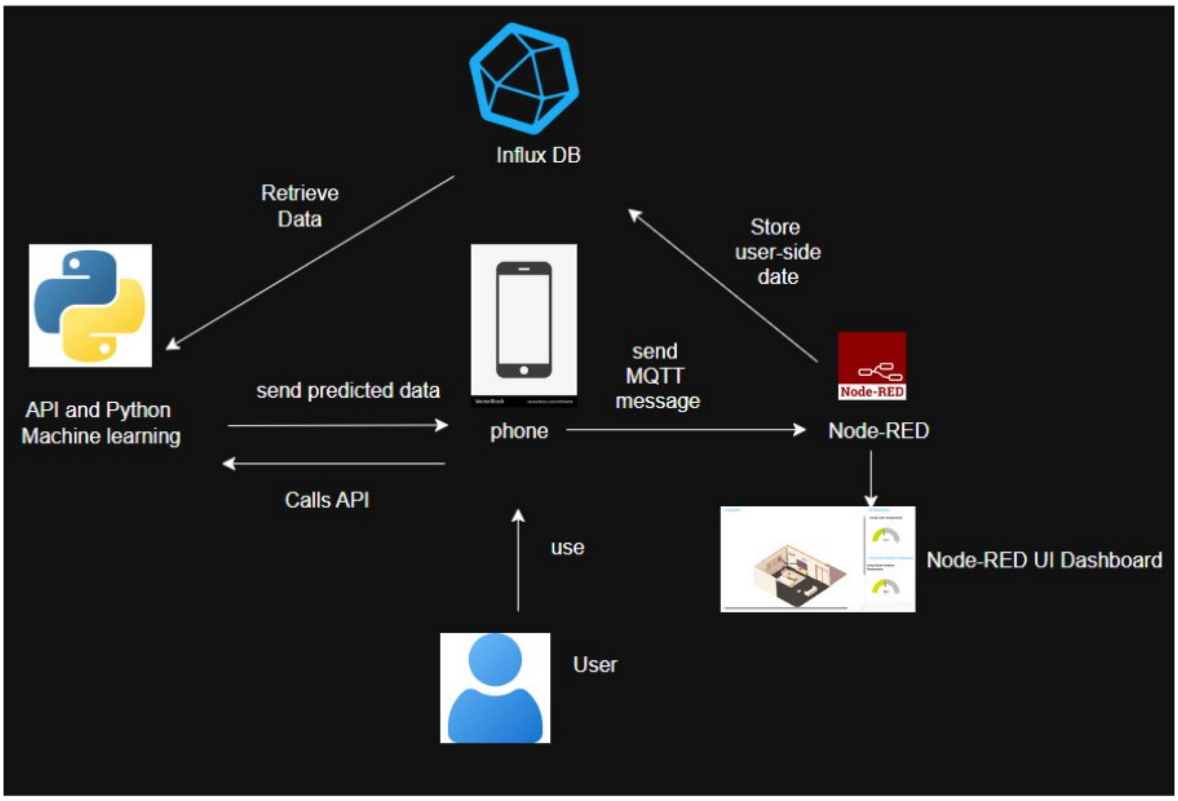


Figure 4.1 System Architecture

The system architecture of the proposed smart home prediction and control system is designed to provide a reliable, intelligent, and user-friendly platform for managing devices and enhancing user interaction with the home environment. The architecture is organized into several key components, each serving a specific purpose within the system. The system architecture is illustrated in Figure 4.1.

InfluxDB (Data Storage): InfluxDB serves as the primary database for storing and retrieving data. It is responsible for maintaining both real-time user-side information and historical records. Data generated by Node-RED, such as device commands and sensor readings, is stored in InfluxDB. Additionally, predicted values from the Python machine learning module are also stored here, ensuring that all information is accessible for monitoring and future analysis.

Node-RED (Automation and Messaging): Node-RED functions as the central automation and communication hub within the architecture. It processes MQTT messages received from the mobile application, executes workflows, and updates device states accordingly. Node-RED also stores user-side data into InfluxDB to maintain consistency. Its integration ensures seamless coordination between devices, the database, and the prediction system.

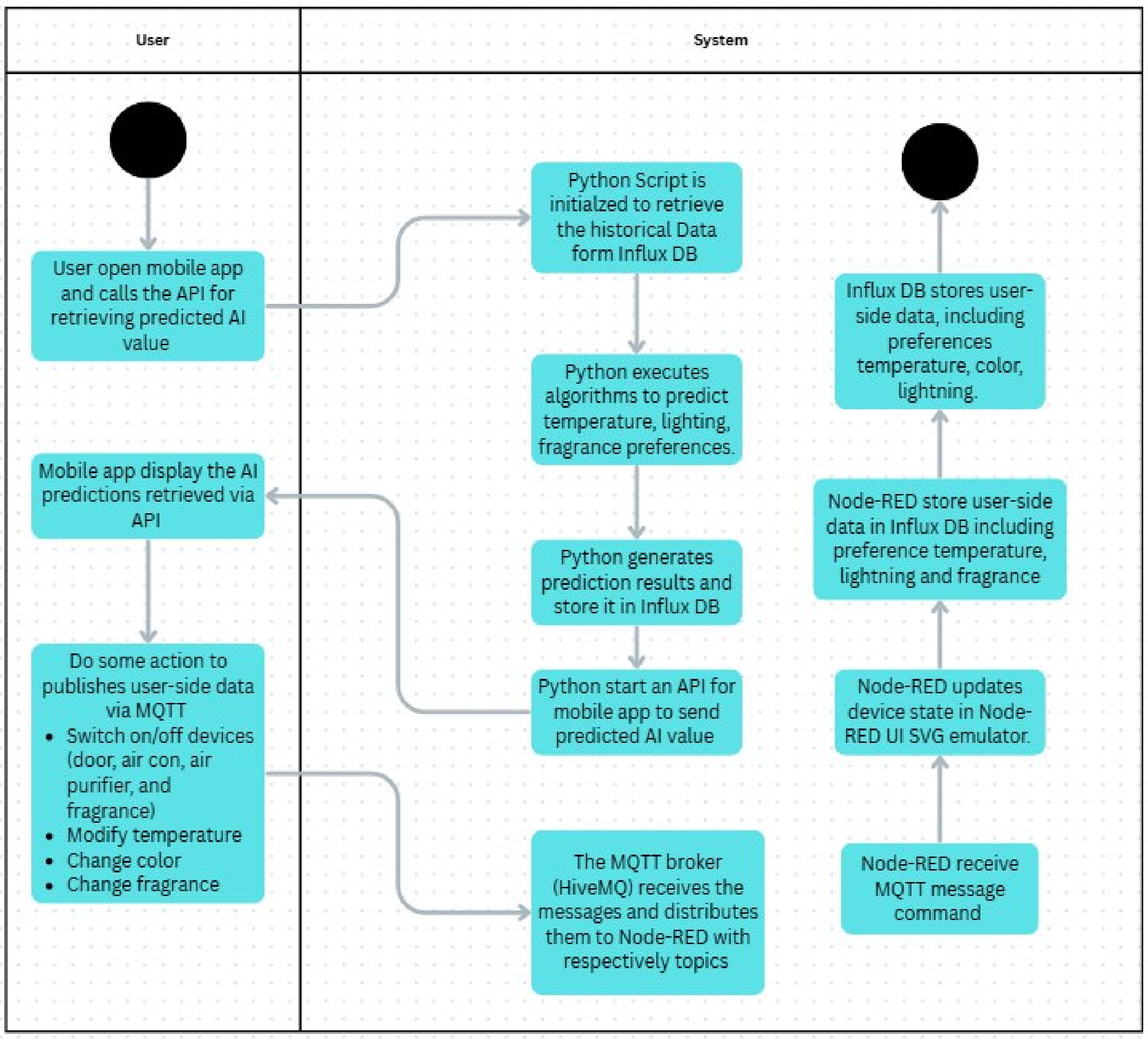
API and Python Machine Learning (Prediction Engine): The Python-based machine learning module is responsible for handling prediction tasks and intelligent decision-making. It retrieves historical and real-time data from InfluxDB, processes the data using trained machine learning models, and generates predicted results. These predicted values are then sent to the mobile application, enabling proactive control and recommendations for the user.

Mobile Application (User Interaction): The mobile application serves as the user's interface with the system. It receives predicted values from the Python ML API and displays them to the user. Through the app, users can issue commands such as turning devices on or off, adjusting temperature, or modifying other environmental settings. These commands are transmitted to Node-RED via MQTT messages, where they are executed accordingly.

User (End Interaction): The user is the final actor in the architecture. By interacting with the mobile application, users gain control over their smart home environment while also benefiting from the system's predictive insights. This ensures a balance between user-driven actions and intelligent, automated decision-making.

In conclusion, the system architecture integrates data management (InfluxDB), automation and messaging (Node-RED), predictive intelligence (Python ML API), and user interaction (mobile app) into a cohesive framework. Together, these components enable efficient communication, accurate predictions, and user-friendly control, forming a robust and adaptive smart home system.

4.2 Use Case Diagram



User
System
User open mobile app and calls the API for retrieving predicted AI value
Mobile app display the AI predictions retrieved via API
Do some action to publishes user-side data via MQTT
• Switch on/off devices (door, air con, air purifier, and fragrance)
• Modify temperature
• Change color
• Change fragrance
Python Script is initialzed to retrieve the historical Data form Influx DB
Python executes algorithms to predict temperature, lighting, fragrance preferences.
Python generates prediction results and store it in Influx DB
Python start an API for mobile app to send predicted AI value
The MQTT broker (HiveMQ) receives the messages and distributes them to Node-RED with respectively topics
Influx DB stores user-side data, including preferences temperature, color, lightning.
Node-RED store user-side data in Influx DB including preference temperature, lightning and fragrance
Node-RED updates device state in Node-RED UI SVG emulator.
Node-RED receive MQTT message command

Figure 4.2 Use Case Diagram

The use case diagram illustrates the interactions between two primary actors—the Home Resident and the System Administrator—with the smart home system. The Home Resident primarily engages with the system through usage-related functions. They can publish commands to control connected devices, subscribe to system updates, and view dashboards that present the current status of their home environment. In addition, they receive predictive insights such as temperature or energy usage forecasts and can also execute adaptive control functions, enabling automated adjustments for improved comfort and efficiency.

On the other hand, the System Administrator plays a supervisory and maintenance role. Their responsibilities include managing MQTT topics to ensure seamless communication between devices, monitoring system data to maintain performance and reliability, and training or updating AI models to improve predictive analytics. Furthermore, they are responsible for maintaining the overall security of the system, safeguarding both user data and operational integrity.

This diagram emphasizes the distinction between roles: while residents focus on interacting with and benefiting from the system's features, administrators ensure that the system operates reliably, securely, and intelligently in the background.

4.3 Activity Diagram

An activity diagram is a visual representation of the flow of activities or actions within a system or process. It is a type of UML (Unified Modeling Language) diagram commonly used in software engineering and systems modeling to describe the dynamic aspects of a system. Activity diagrams are particularly useful for modeling business processes, software workflows, and other processes that involve a sequence of activities.

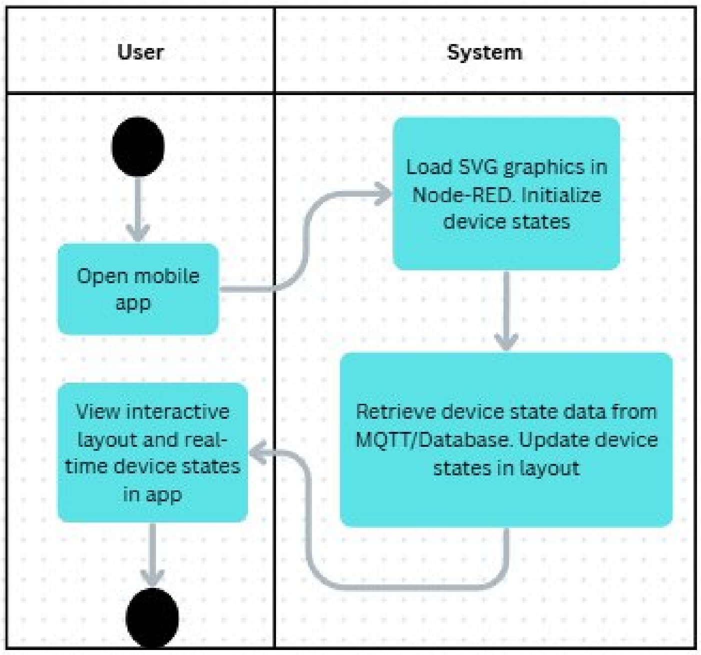
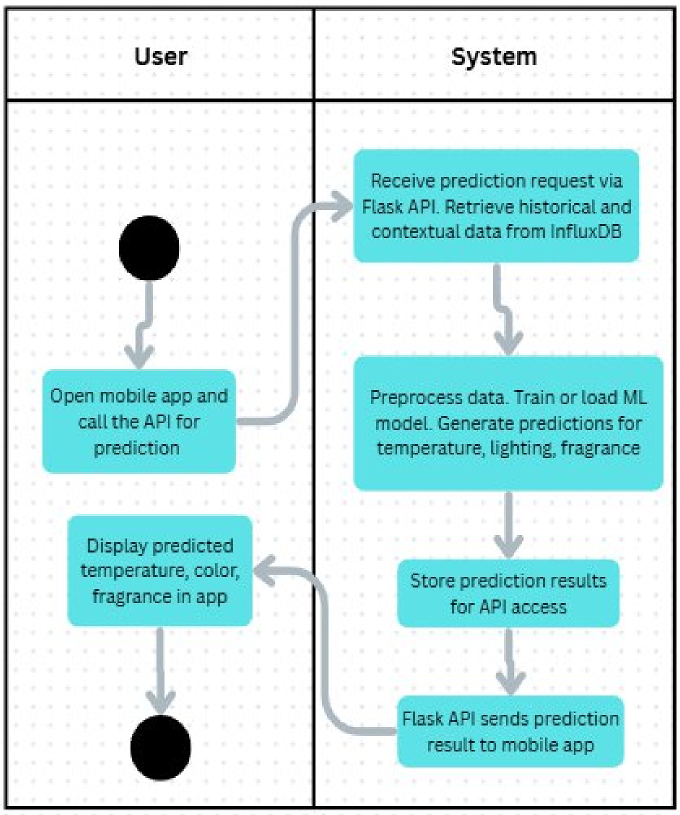

User
System
Open mobile app
Load SVG graphics in Node-RED. Initialize device states
Retrieve device state data from MQTT/Database. Update device states in layout
View interactive layout and real-time device states in app

Figure 4.3 Activity Diagram 1


User
System
Receive prediction request via Flask API. Retrieve historical and contextual data from InfluxDB
Open mobile app and call the API for prediction
Preprocess data. Train or load ML model. Generate predictions for temperature, lighting, fragrance
Display predicted temperature, color, fragrance in app
Store prediction results for API access
Flask API sends prediction result to mobile app

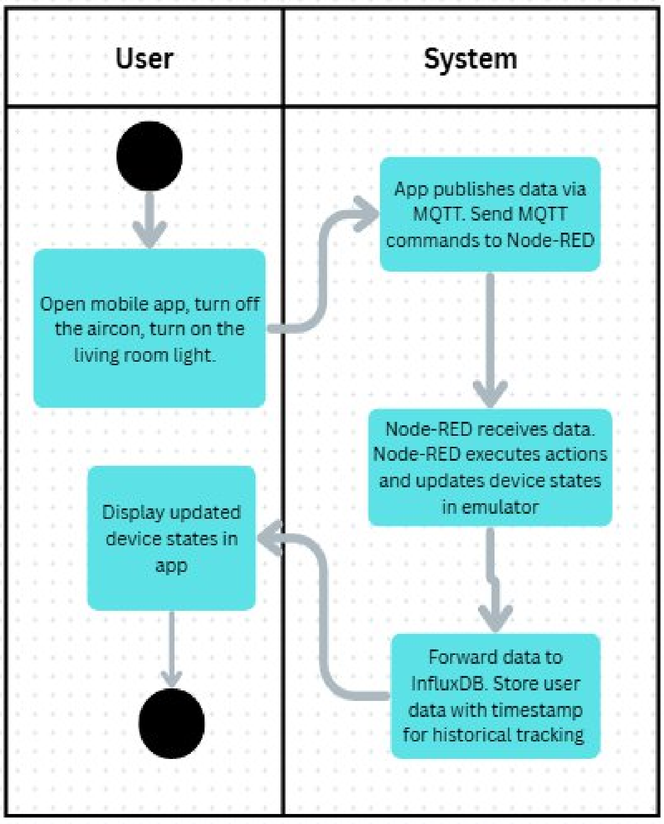

User
System
App publishes data via MQTT. Send MQTT commands to Node-RED
Open mobile app, turn off the aircon, turn on the living room light.
Node-RED receives data. Node-RED executes actions and updates device states in emulator
Display updated device states in app
Forward data to InfluxDB. Store user data with timestamp for historical tracking

Figure 4.4 Activity Diagram 2

Figure 4.3 and 4.4 illustrates the sequence of interactions between the user and the system when operating the smart home prediction and control application. It demonstrates how user actions, such as switching devices on/off or modifying environmental conditions, are processed by the system through Node-RED, InfluxDB, and the Python machine learning module. The diagram highlights the flow of data, decision-making processes, and the delivery of predicted AI values to the mobile application.

Diagram Description: The process begins when the user opens the mobile application and receives the latest predicted AI value. The user may then perform a series of actions such as turning devices on or off (e.g., air conditioner, purifier, door), changing temperature, color, or fragrance settings. These commands are transmitted as MQTT messages to Node-RED.

Node-RED receives the MQTT message, executes the command, and updates the visualization interface while also storing relevant data into InfluxDB. A Python script then retrieves the stored data and applies the prediction model to generate a new AI value. The system stores this predicted value back into InfluxDB for record-keeping and also makes it accessible via the Python API. Finally, the predicted value is sent to the mobile application, completing the feedback loop and allowing the user to view updated insights in real time.

This continuous cycle ensures that both user inputs and system intelligence work together to create an adaptive and automated smart home environment.

4.4 Database Design

Database design is a crucial aspect of the *Adaptive Smart Home Automation* project, as it defines the structure and organization of the data stored in InfluxDB buckets. These buckets contain sensor readings, user-side inputs, and AI-generated predictions for controlling devices such as lighting, fragrance, and temperature. Proper database design ensures data integrity, efficient retrieval, and scalability, allowing the system to handle real-time inputs and predictive automation. Below is a detailed description of the database design for this project:

Organization Table

The organization table represents the central configuration of the InfluxDB instance, storing information common to all buckets.

Table 4.1 Organization table

Field Name	Data Type	Description	Field Size
ID	String	Unique identifier for the organization	VARCHAR(50)
Name	String	Name of the organization	VARCHAR(100)
URL	String	The InfluxDB server URL (shared across all buckets)	VARCHAR(255)

Bucket: colorfrag

This bucket is used to store prediction of Lighting & Fragrance

Table 4.2 colorfrag Bucket

Field Name	Data Type	Description	Field Size
Bucket_Name	String	Unique bucket identifier	VARCHAR(50)
Organization_ID	String	References the parent organization	VARCHAR(50)
API_Token	String	Authentication token specific to this bucket	VARCHAR(255)
_measurement	String	Measurement name (e.g., “light”, “fragrance”)	VARCHAR(100)
_field	String	Specific field recorded (e.g., “intensity”, “status”)	VARCHAR(100)
_time	String	Timestamp of data entry	VARCHAR(100)
value	Double	Recorded or predicted value	-

Bucket: colortest

This bucket is used to store user input of Lighting & Fragrance

Table 4.3 colortest Bucket

Field Name	Data Type	Description	Field Size
Bucket_Name	String	Unique bucket identifier	VARCHAR(50)
Organization_ID	String	References the parent organization	VARCHAR(50)
API_Token	String	Authentication token specific to this bucket	VARCHAR(255)
_measurement	String	Measurement name (same as *colorfrag*)	VARCHAR(100)
_field	String	Specific field recorded (same as *colorfrag*)	VARCHAR(100)
_time	String	Timestamp of data entry	VARCHAR(100)
value	Double	Recorded value from user input	-

Bucket: fyp2

This bucket is used to store user input of Temperature

Table 4.4 fyp3 Bucket

Field Name	Data Type	Description	Field Size
Bucket_Name	String	Unique bucket identifier	VARCHAR(50)
Organization_ID	String	References the parent organization	VARCHAR(50)
API_Token	String	Authentication token specific to this bucket	VARCHAR(255)
_measurement	String	Measurement name (e.g., “temperature”)	VARCHAR(100)

_field	String	Specific field recorded (e.g., “value”, “unit”)	VARCHAR(100)
_time	String	Timestamp of data entry	VARCHAR(100)
value	Double	Actual sensor reading	-

Bucket: fypy

This bucket is used to store prediction of Temperature

Table 4.5 fypy Bucket

Field Name	Data Type	Description	Field Size
Bucket_Name	String	Unique bucket identifier	VARCHAR(50)
Organization_ID	String	References the parent organization	VARCHAR(50)
API_Token	String	Authentication token specific to this bucket	VARCHAR(255)
_measurement	String	Measurement name (e.g., “temperature_prediction”)	VARCHAR(100)
_field	String	Specific field recorded (e.g., “predicted_value”)	VARCHAR(100)
_time	String	Timestamp of prediction	VARCHAR(100)
value	Double	Predicted temperature value	-

Relationship

The relationship between each bucket is they are linked to a single **Organization** through the Organization_ID. **Colorfrag** (prediction) is logically related to **colortest** (user input) for lighting and fragrance data. F**ypy** (prediction) is logically related to **fyp2** (sensor readings) for temperature.

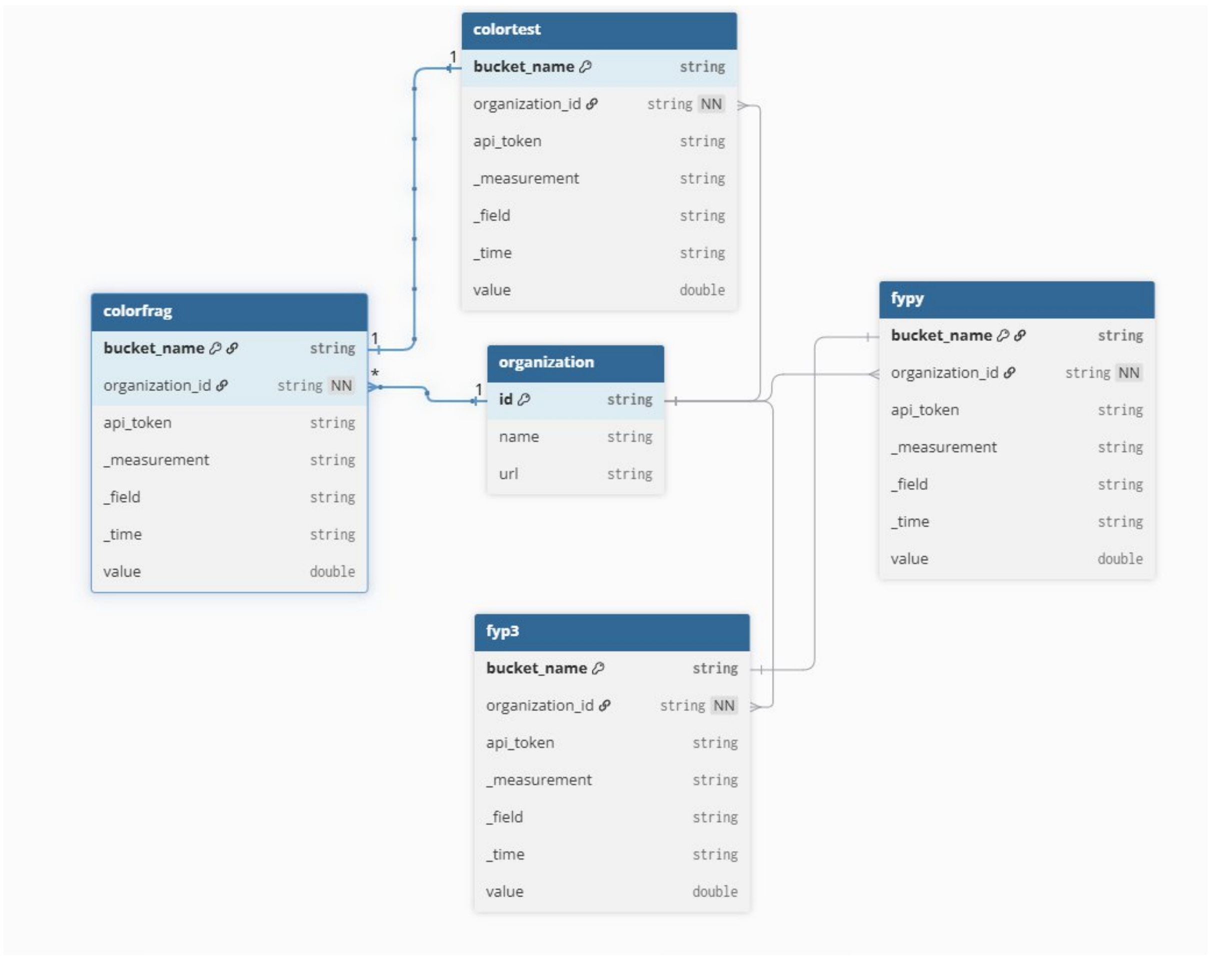


Figure 4.5 Database Diagram

4.5 Concluding Remark

This chapter presented the overall design framework of the Smart Home Control and Monitoring System, emphasizing its architecture, use case, activity flow, and database design. The four-layer system architecture demonstrated how data management, logic processing, user interaction, and communication protocols integrate to create a cohesive and adaptive environment. The use case diagram clarified the roles of both the Home Resident and the System Administrator, while the activity diagram outlined the end-to-end operational workflow that combines user interaction with AI-driven predictions. Additionally, the database design ensured structured handling of time-series data for efficient monitoring and predictive analysis.

In summary, Chapter 4 established a clear blueprint for how the system functions and interacts across different layers, users, and processes. This foundation not only supports seamless implementation but also ensures scalability, adaptability, and intelligent control in addressing the challenges of conventional smart home systems.

Chapter 5
System Implementation

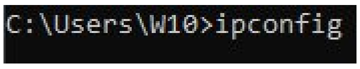
5.1 Software/Hardware Setup and Configuration

This chapter outlines the preparation steps, installation processes, and environment setup required to ensure the smooth development and operation of the AI-driven smart home monitoring and control system. The section covers the installation and configuration of the Flutter mobile app environment, Node-RED logic system, SVG emulator using Inkscape, MQTT messaging protocol, AI module, time-series database, and external API integration.

5.1.1 Hardware Setup

The hardware setup primarily involves a laptop (for mobile app development, Node-RED server, and AI server) and a mobile device (for running the Flutter-based mobile app). Both devices must be connected to the same IP network (e.g., the same Wi-Fi router) to enable seamless communication between the system components. To test whether laptop and mobile phone are in the same network.

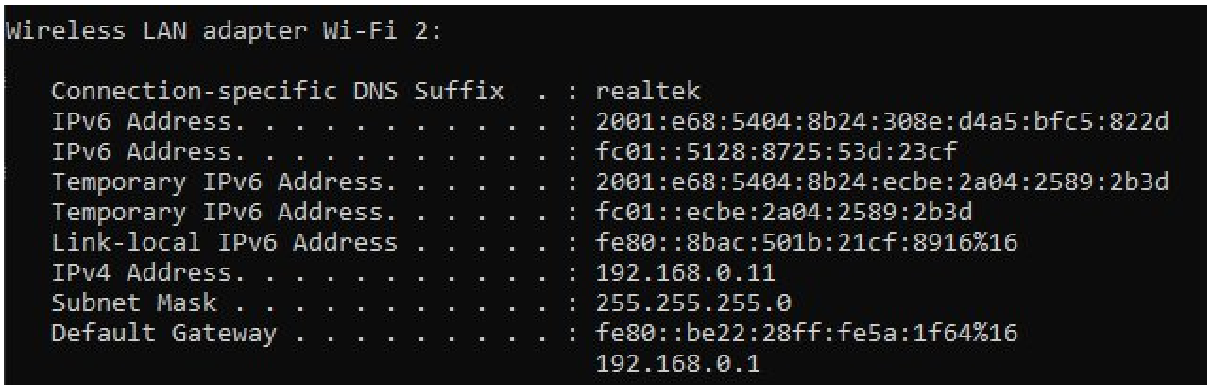
- First make sure they both connect with a same WIFI network.
- Second, go to command prompt of the laptop and type ipconfig



```
C:\Users\W10>ipconfig
```

Figure 5.1

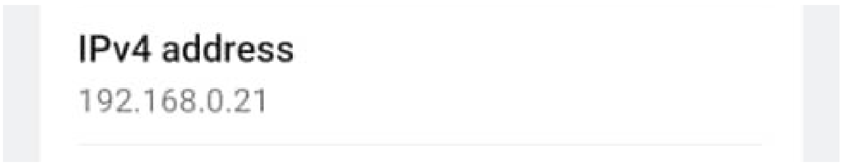
- Third locate the section titled "Wireless LAN adapter Wi-Fi 2". Under this section, find the IPv4 Address. For example, if the IPv4 address is 192.168.0.11 (as shown in the Figure 5.2), the corresponding IP network will be 192.168.0.0 with a subnet mask of 255.255.255.0



```
Wireless LAN adapter Wi-Fi 2:

   Connection-specific DNS Suffix  . : realtek
   IPv6 Address. . . . . . . . . . . : 2001:e68:5404:8b24:308e:d4a5:bfc5:822d
   IPv6 Address. . . . . . . . . . . : fc01::5128:8725:53d:23cf
   Temporary IPv6 Address. . . . . . : 2001:e68:5404:8b24:ecbe:2a04:2589:2b3d
   Temporary IPv6 Address. . . . . . : fc01::ecbe:2a04:2589:2b3d
   Link-local IPv6 Address . . . . . : fe80::8bac:501b:21cf:8916%16
   IPv4 Address. . . . . . . . . . . : 192.168.0.11
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::be22:28ff:fe5a:1f64%16
                                       192.168.0.1
```

Figure 5.2 IP address in Command Prompt

- Fourth, go to the mobile phone's WIFI setting and navigate to "About this network", below their find IPv4 address, and this will be the IP address. The IP network will also be the same as 192.168.0.0 with a subnet mask of 255.255.255.0.



IPv4 address
192.168.0.21

Figure 5.3 IP address in mobile phone

By ensuring both devices are on the same IP network as both Figure 5.2 and Figure 5.3 shows they are in a same IP network 192.168.0.0 with a subnet mask of 255.255.255.0. Therefore they are able to do real-time communication, control, and data synchronization between the mobile app, Node-RED, and Python AI modules .

5.1.2 Environment Setup

5.1.2.1 Visual Studio Code Setup

Download and install Visual Studio Code by visiting https://code.visualstudio.com/. Choose the compatible version for your operating system and follow the installation instructions

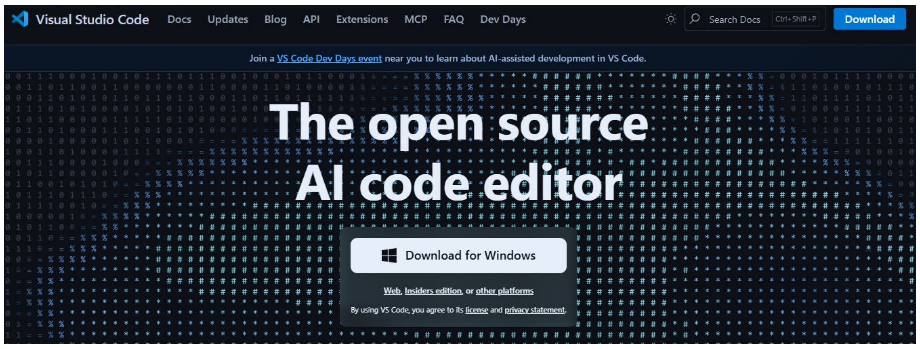
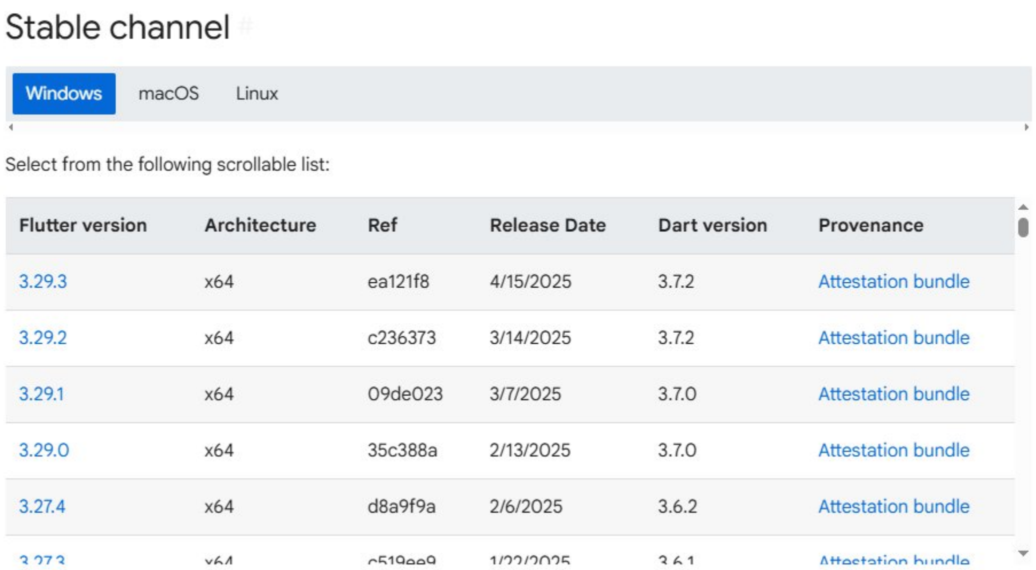

Visual Studio Code
Docs
Updates
Blog
API
Extensions
MCP
FAQ
Dev Days
Search Docs
Ctrl+Shift+P
Download
Join a VS Code Dev Days event near you to learn about AI-assisted development in VS Code.
The open source AI code editor
Download for Windows
Web, Insiders edition, or other platforms
By using VS Code, you agree to its license and privacy statement.

Figure 5.4 Install Visual Studio Code

5.1.2.2 Flutter Mobile App Setup

Flutter SDK

1. Download the Flutter SDK from the official Flutter website: https://docs.flutter.dev/release/archive



Stable channel

Windows macOS Linux

Select from the following scrollable list:

Flutter version	Architecture	Ref	Release Date	Dart version	Provenance
3.29.3	x64	ea121f8	4/15/2025	3.7.2	Attestation bundle
3.29.2	x64	c236373	3/14/2025	3.7.2	Attestation bundle
3.29.1	x64	09de023	3/7/2025	3.7.0	Attestation bundle
3.29.0	x64	35c388a	2/13/2025	3.7.0	Attestation bundle
3.27.4	x64	d8a9f9a	2/6/2025	3.6.2	Attestation bundle
3.27.3	x64	c519ee9	1/22/2025	3.6.1	Attestation bundle

Figure 5.5 Flutter website

2. Extract the SDK to a preferred location on your system, recommended to pick the latest version of SDK.

Configure Flutter Environment Path

Configuring the Flutter environment path allows your system to recognize and run Flutter commands from any terminal or command prompt. This ensures you can build, run, and manage Flutter projects without typing the full path to the Flutter SDK each time. It also enables tools like flutter doctor to verify dependencies and check the setup.

1. Open "Edit the system environment variables" on Windows.

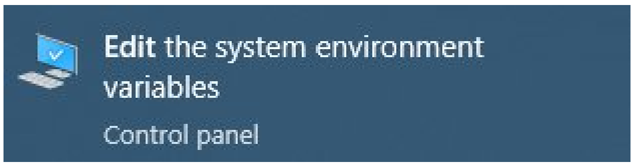


Figure 5.6 Flutter Environment Path

2. Click on "Environment Variables."

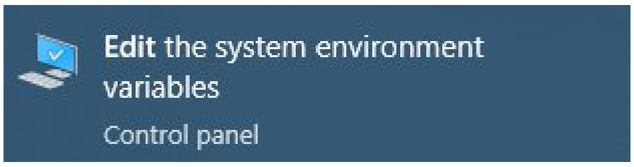
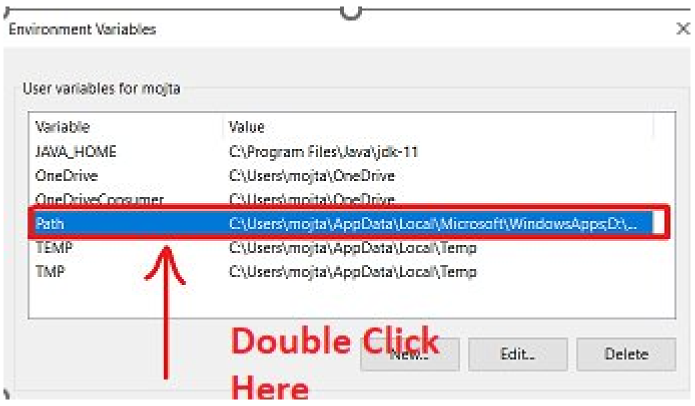


Figure 5.7 Environment Variable

3. Under System Variables, find and double-click the **Path** variable.



Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk-11
OneDrive	C:\Users\mojta\OneDrive
OneDriveConsumer	C:\Users\mojta\OneDrive
Path	C:\Users\mojta\AppData\Local\Microsoft\WindowsApps;D:\...
TEMP	C:\Users\mojta\AppData\Local\Temp
TMP	C:\Users\mojta\AppData\Local\Temp

Figure 5.8 System Variables

4. Click **New** and add the full path to the bin folder inside the Flutter SDK directory.

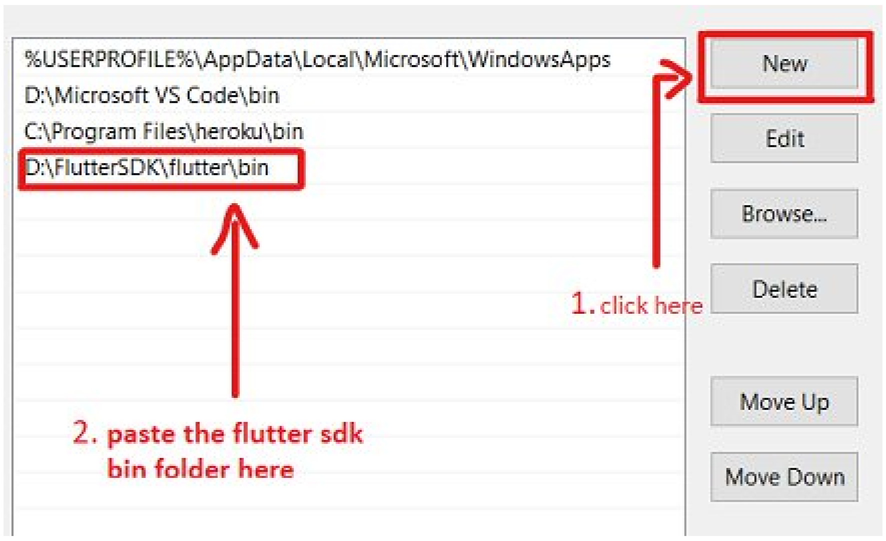

%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
D:\Microsoft VS Code\bin
C:\Program Files\heroku\bin
D:\FlutterSDK\flutter\bin

Figure 5.9 SDK directory

5. Click **OK** to save changes.
6. Open Command Prompt and type flutter doctor or flutter --version to verify the installation.

Set Up Visual Studio Code for Flutter

1. Launch the Visual Studio Code app once the installation is complete.
2. Install the Flutter extension by clicking the Extensions icon on the sidebar or pressing Ctrl+Shift+X. Search for "Flutter" in the Extensions marketplace, select the one published by Dart Code, and click **Install**. This will also automatically install the Dart extension.
3. (Optional) If Dart was not installed along with Flutter, it can be installed separately. In the Extensions view, search for "Dart" and click **Install** on the official Dart extension.
4. Set the Flutter SDK path. Press Ctrl+Shift+P (or Cmd+Shift+P on macOS) to open the Command Palette. Type and select **Flutter: Choose Flutter SDK**, then browse to and select the folder where Flutter was installed (e.g., D:\FlutterSDK\flutter).
5. Open a Flutter project by selecting **File > Open Folder** in Visual Studio Code and browsing to the project's folder on your PC. Alternatively, a new Flutter project can be created directly from the command prompt by typing flutter create projectname and

pressing Enter. This generates the necessary project files, which can then be opened in VS Code.

6. If the project folder is not already open, go to **File > Open Folder** and select the directory of your Flutter project.
7. Open the terminal from the menu in VS Code. Ensure the project folder is selected (navigate using cd if necessary), then type flutter run and press Enter. This will build and launch the app on the connected device or emulator.

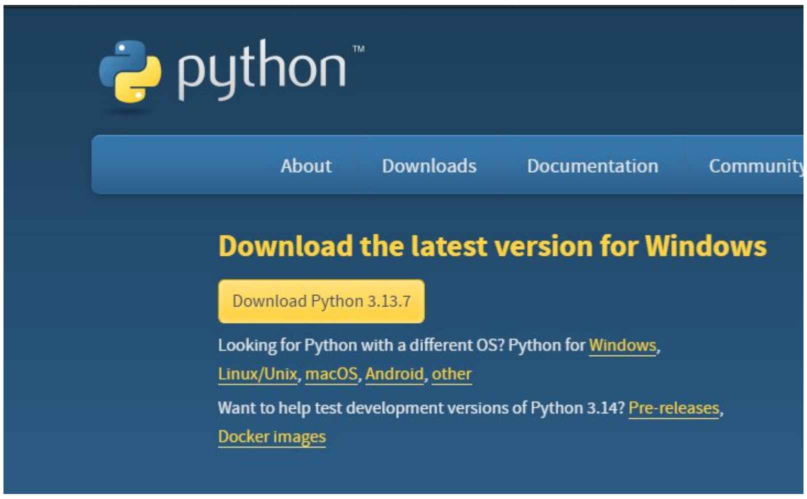
Project Template Initialization and Setup

1. Browse repositories (e.g. FlutterFlow) for smart home or IoT UI templates.
2. Use templates to guide the structure of pages, navigation, and widgets.

5.1.2.3 Python AI/ML Setup

Download the Installer

The latest version of Python is obtained from the official Python website: https://www.python.org/downloads/



Download the latest version for Windows

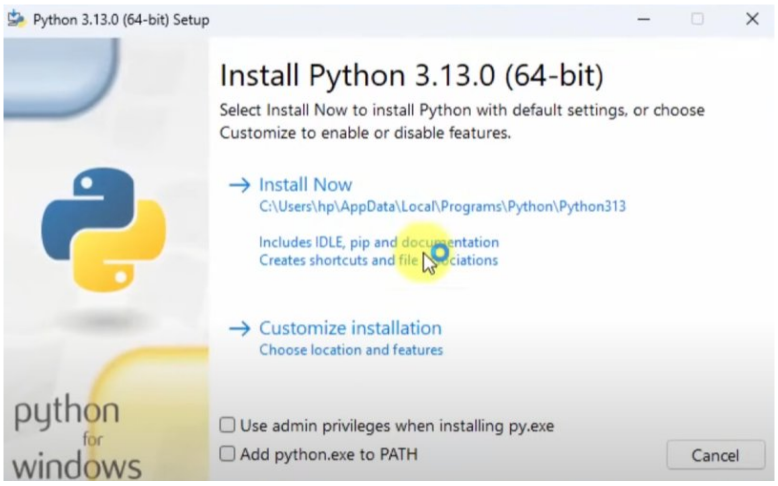
Looking for Python with a different OS? Python for Windows, Linux/Unix, macOS, Android, other

Want to help test development versions of Python 3.14? Pre-releases, Docker images

Figure 5.10 Download Python

Run the Installer

Start the program after download and click install now. Note: Figures are just a demonstration, please download the latest version.



Python 3.13.0 (64-bit) Setup

Install Python 3.13.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\hp\AppData\Local\Programs\Python\Python313

Includes IDLE, pip and documentation
Creates shortcuts and file associations

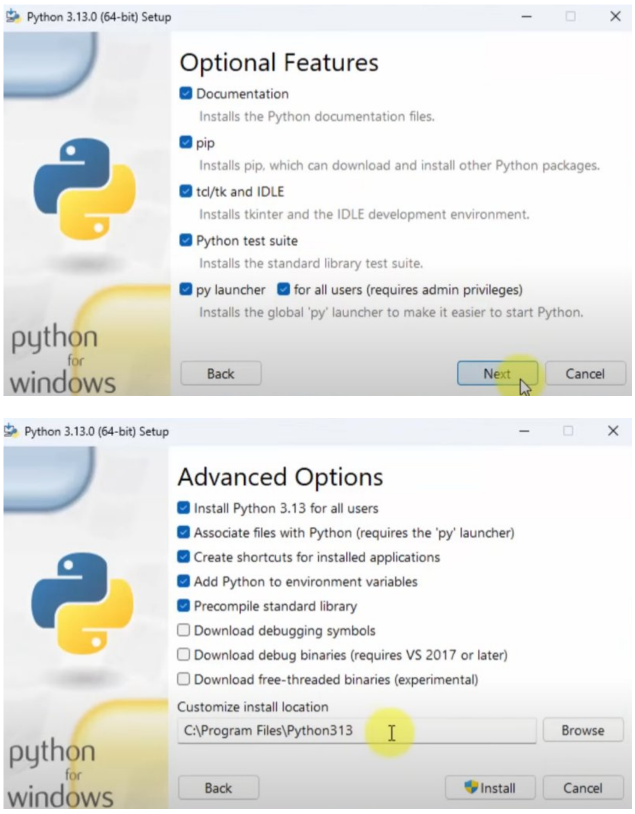
→ Customize installation
Choose location and features

☐ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

Figure 5.11 Python Installer

Select Installation Options

Next, check every checkboxes in the Optional Features and Adanced Options, every options are needed.



Python 3.13.0 (64-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation files.

☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires admin privileges)
Installs the global 'py' launcher to make it easier to start Python.

Python 3.13.0 (64-bit) Setup

Advanced Options

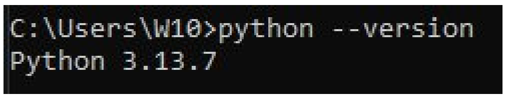
☑ Install Python 3.13 for all users
☑ Associate files with Python (requires the 'py' launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2017 or later)
☐ Download free-threaded binaries (experimental)

Customize install location
C:\Program Files\Python313

Figure 5.12 Installation Options

Verify Installation

After installation, the command python –version (or python3 –version depending on the system) is executed in the terminal/command prompt to confirm that Python is installed correctly as shown as Figure 5.1.13.



```
C:\Users\W10>python --version
Python 3.13.7
```

Figure 5.13 Verify Installation Python

Install Python Libraries

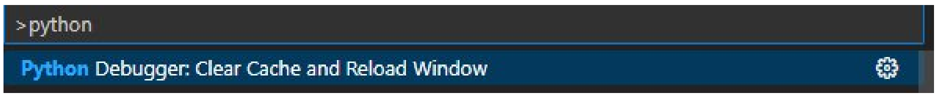
Install required libraries for prediction and database operations:

Table 5.1 Python Libraries

Category	Library	Installation Command
Data Processing & ML	pandas, numpy, scikit-learn, matplotlib, joblib	pip install pandas numpy scikit-learn matplotlib joblib
Web Framework & API Handling	flask, flask-cors, requests	pip install flask flask-cors requests
Database Client	influxdb-client	pip install influxdb-client

Set Up Visual Studio Code for Python

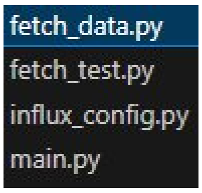
1. Install the Python extension. Click the Extensions icon on the sidebar or press Ctrl+Shift+X. In the Extensions marketplace, search for "Python," select the one published by Microsoft, and click Install. This extension provides Python language support, linting, IntelliSense, debugging, and code navigation features.
2. (Optional) For working with notebooks, install the **Jupyter** extension. Open the Extensions Marketplace in VS Code, search for "Jupyter," and click Install. This enables interactive notebook functionality.
3. Set the Python interpreter by opening the Command Palette in VS Code. Search for **"Python: Select Interpreter"** and choose the desired version or virtual environment (for example, Python 3.12 installed at C:\Users\YourName\AppData\Local\Programs\Python\Python312).



```
>python
Python Debugger: Clear Cache and Reload Window
```

Figure 5.14 Python in Command Prompt

4. Open your Python project or create a new one. To open a folder, go to File > Open Folder and select the directory where your Python scripts are located. To create a new file, click File > New File and save it with a .py extension.



```
fetch_data.py
fetch_test.py
influx_config.py
main.py
```

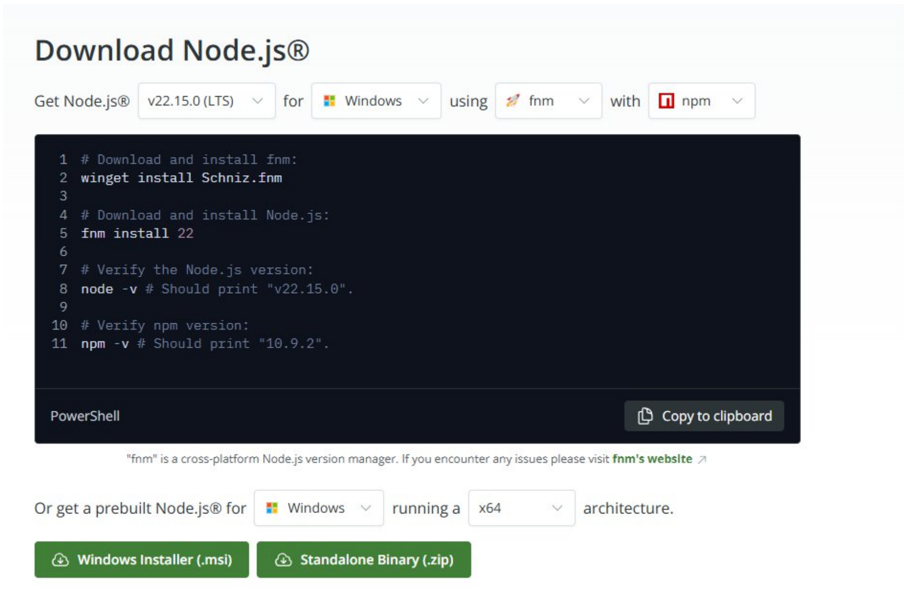
Figure 5.15 Python extension

5. To run a Python script, open the integrated terminal via View > Terminal or Ctrl+`` (backtick). Confirm the correct project directory. Type python filename.py (replacing filename.py with your script's name) and press Enter. The program will then execute, and the output will appear in the terminal window.

5.1.2.4 Node-RED Setup

Install Node.js

1. Download the LTS version of Node.js from https://nodejs.org.



Download Node.js®

Get Node.js® v22.15.0 (LTS) for Windows using fnm with npm

```
# Download and install fnm:
winget install Schniz.fnm

# Download and install Node.js:
fnm install 22

# Verify the Node.js version:
node -v # Should print "v22.15.0".

# Verify npm version:
npm -v # Should print "10.9.2".
```

PowerShell · Copy to clipboard

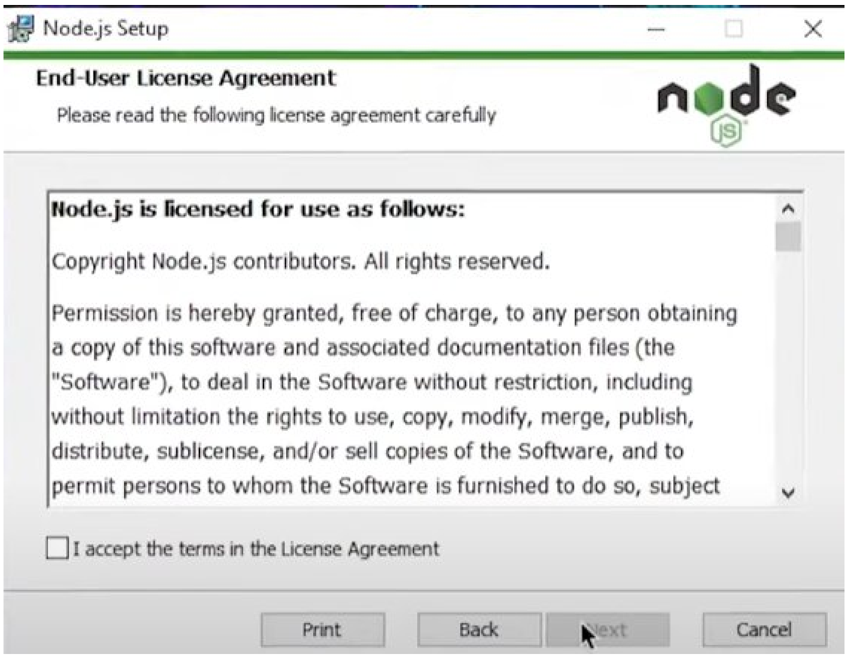
"fnm" is a cross-platform Node.js version manager. If you encounter any issues please visit fnm's website

Or get a prebuilt Node.js® for Windows running a x64 architecture.

Windows Installer (.msi) · Standalone Binary (.zip)

Figure 5.16 Download Nodejs

2. Run the installer and follow prompts.


Node.js Setup
End-User License Agreement
Please read the following license agreement carefully

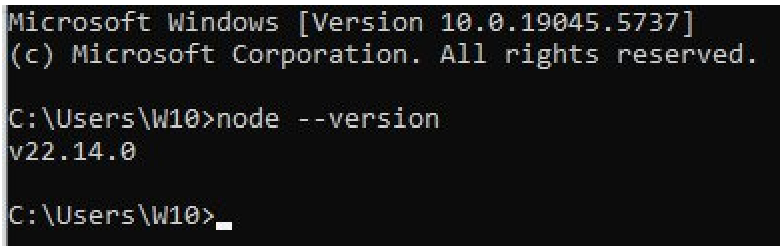
Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

Figure 5.17 Nodejs Installer

3. Verify installation by running node --version in Command Prompt. If it is successfully installed, the expected outcome would be the same to Figure 5.18

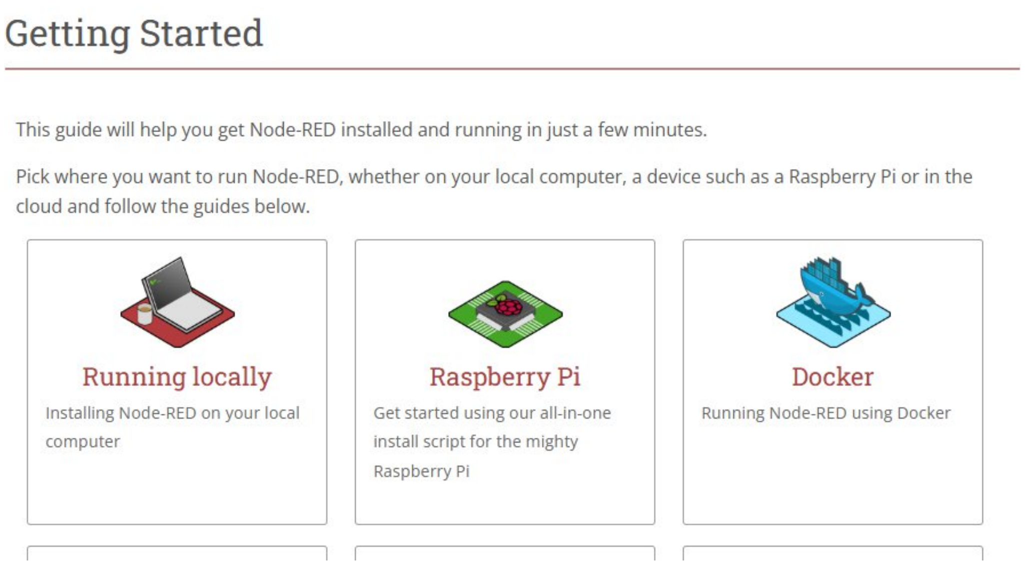


```
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\W10>node --version
v22.14.0

C:\Users\W10>_
```

Figure 5.18 Verify Installation Nodejs

Install Node-RED

1. Download Node-RED. Go to this website https://nodered.org/docs/getting-started/, and click on "running locally".


Getting Started

This guide will help you get Node-RED installed and running in just a few minutes.

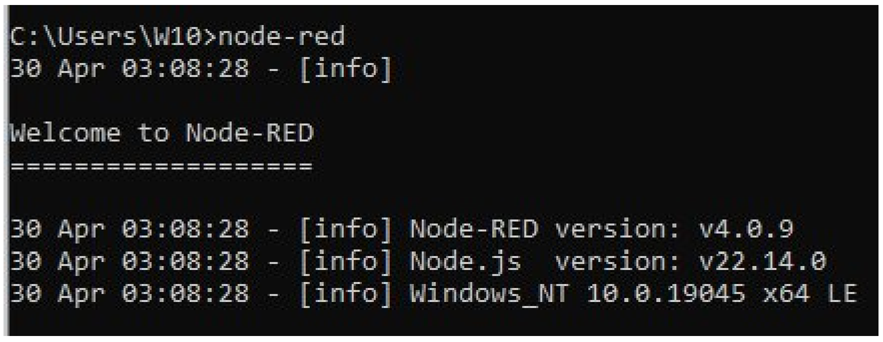
Pick where you want to run Node-RED, whether on your local computer, a device such as a Raspberry Pi or in the cloud and follow the guides below.

Running locally
Installing Node-RED on your local computer
Raspberry Pi
Get started using our all-in-one install script for the mighty Raspberry Pi
Docker
Running Node-RED using Docker

Figure 5.19 Download Node-RED

Run Node-RED

1. Open Command Prompt and type node-red.



```
C:\Users\W10>node-red
30 Apr 03:08:28 - [info]

Welcome to Node-RED
===================

30 Apr 03:08:28 - [info] Node-RED version: v4.0.9
30 Apr 03:08:28 - [info] Node.js  version: v22.14.0
30 Apr 03:08:28 - [info] Windows_NT 10.0.19045 x64 LE
```

Figure 5.20 Run Node-RED

2. Access the flow editor via http://localhost:1880.

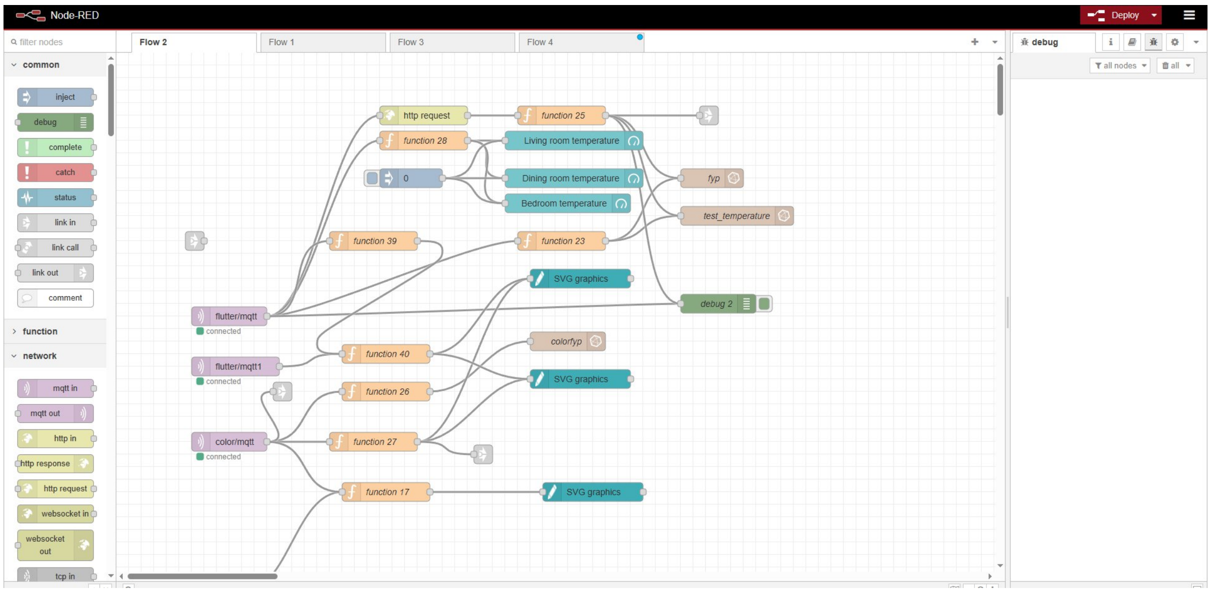


Figure 5.21 flow editor of Node-RED

3. Access dashboard UI via http://localhost:1880/ui.

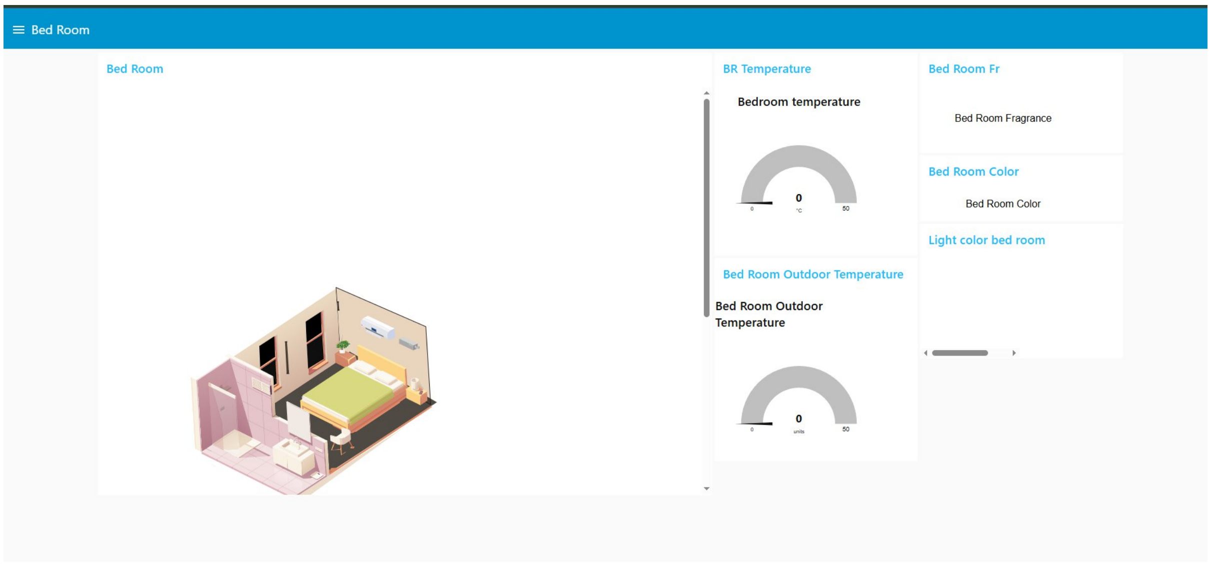


Figure 5.22 dashboard UI of Node-RED

4. For the development of Node-RED SVG, it is not enough for merely using of the default modulus that is give, some additional modules were needed to be installed due to additional function including node-red-dashboard for the visual user interface, node-red-node-ui-svg for SVG-based emulation, and node-red-contrib-influxdb for InfluxDB integration. Modules can be installed though the Manage Palette.

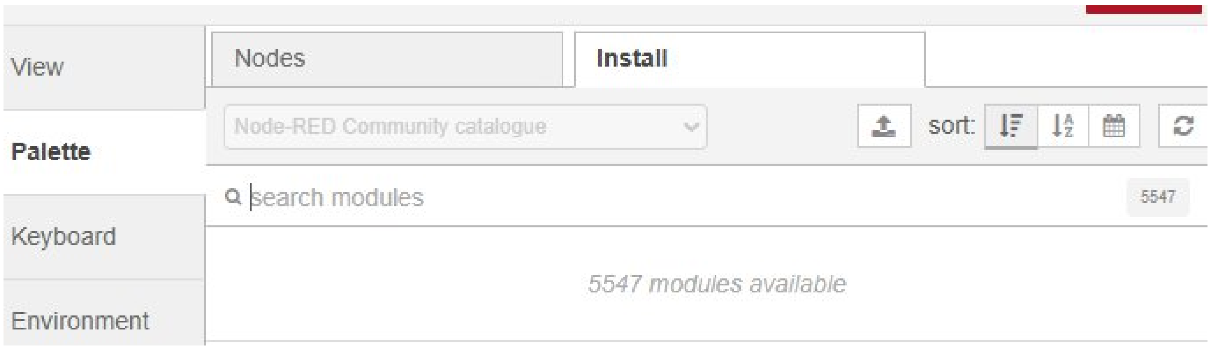
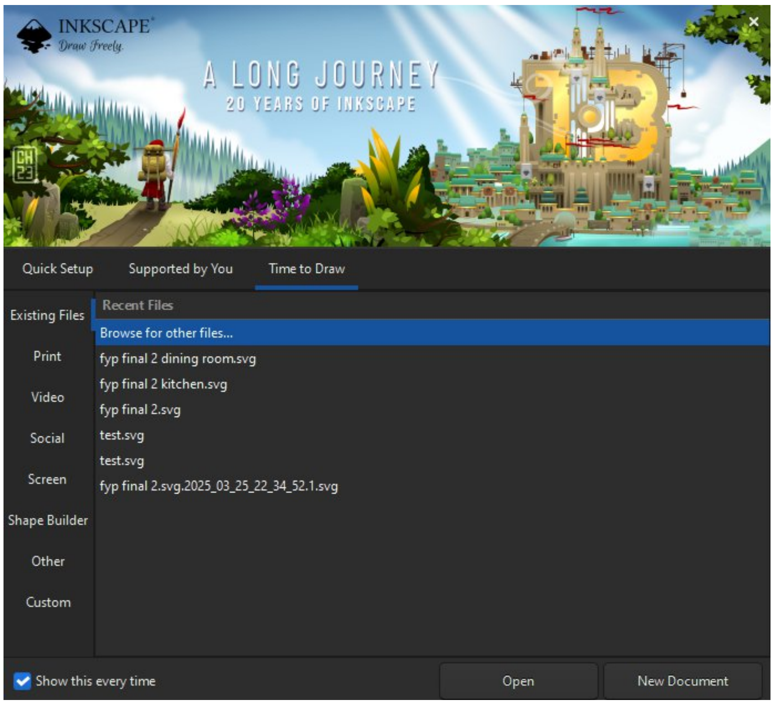


Figure 5.23 Manage Palette

5.1.2.5 Inkscape Setup (SVG Emulator)

Install Inkscape

1. Download and install from https://inkscape.org/.



fyp final 2 dining room.svg
fyp final 2 kitchen.svg
fyp final 2.svg
test.svg
test.svg
fyp final 2.svg.2025_03_25_22_34_52.1.svg

Figure 5.24 Install Inkscape

2. Open Inkscape and design SVG elements (floor plans, device layouts, dynamic dashboards).

Integration with Node-RED

This section is to integrate SVG graphic into Node-RED to form the visual foundation of the smart home emulator on the dashboard.

1. Import SVG graphic into Node-RED using SVG graphics node from the module node-red-node-ui-svh.

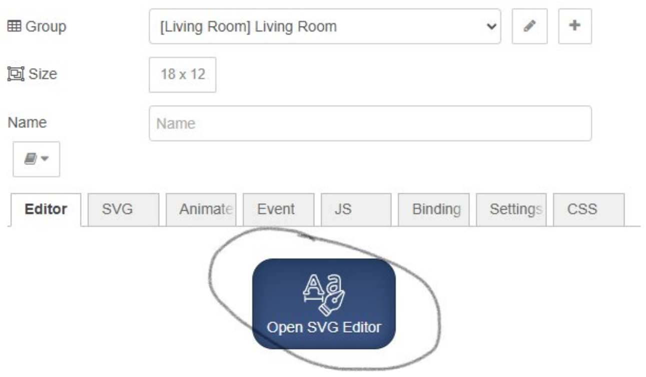


Figure 5.25 SVG node in Node-RED

5.1.2.6 Library and Packages Setup and Installation

Add all dependencies listed in pubspec.yaml (e.g., mqtt_client, flutter_svg, provider, sqflite) to the project.

Table 5.2 Library and Packages Setup

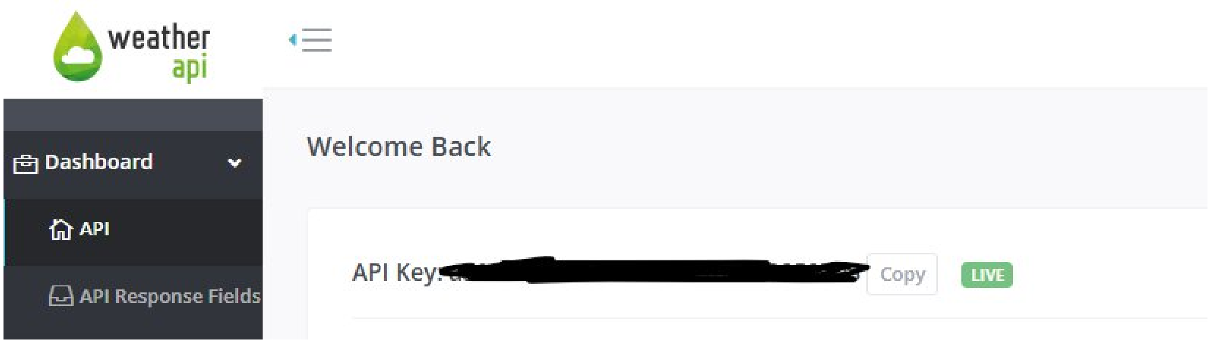
Component	**Essential Libraries / Packages**	**Purpose**
Flutter (Mobile App)	flutter (SDK)	Core framework
	flutter_localizations	Multi-language support
	flutter_web_plugins	Web platform integration
	cupertino_icons	iOS-style icons
	google_fonts	Custom fonts
	flutter_svg	SVG graphics display
	flutter_colorpicker	Color selection UI
	auto_size_text	Dynamic text scaling
	page_transition	Page navigation animations
	provider	State management
	mqtt_client	MQTT messaging communication
	http	REST API requests

	shared_preferences	Local key-value storage
	sqflite	Local SQLite database
	path_provider	File paths for storage
	cached_network_image	Image caching
	flutter_cache_manager	Cache management
	flutter_animate	UI animations
	timeago	Relative time display
	url_launcher	Open URLs externally
Python (AI / Backend)	pandas	Data handling
	numpy	Numerical computing
	scikit-learn	Machine learning models
	joblib	Model saving/loading
	matplotlib	Data visualization
	flask	Backend API server
	flask_cors	Cross-origin requests
	requests	API requests
	datetime, pytz	Date/time handling
	influxdb-client	Connect to InfluxDB
Node-RED	function	Custom JavaScript logic
	inject	Trigger flows
	debug	Debug output
	mqtt in/out	MQTT messaging
	http request	API calls
	ui_gauge	Dashboard gauge UI
	ui_text	Dashboard text display
	ui_template	Custom UI components
	node-red-node-ui-svg	SVG integration from Inkscape

5.1.3 External Services Integration Configuration and Setup

5.1.3.1 Weather API

The Weather API was integrated into the system to provide real-time and forecasted environmental data, including temperature, weather conditions, and location-based information. Its setup required registering at the official WeatherAPI website (https://www.weatherapi.com) and obtaining an API key. This API key was later used to perform HTTP requests, allowing the system to fetch weather information. The collected data supported the AI module in making predictive adjustments to indoor conditions, such as temperature regulation and device scheduling.

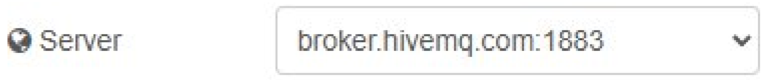


API Key: [redacted]

Figure 5.26 weatherapi

5.1.3.2 MQTT Broker: HiveMQ

HiveMQ was employed as the MQTT broker to handle messaging and communication between system components. It is a cloud-based broker that allowed seamless interaction between the Flutter mobile app, Node-RED server, InfluxDB, and the Python AI script. No additional configuration was required beyond installing the MQTT node in Node-RED, as HiveMQ provides an open and ready-to-use broker service. The broker acted as the central hub for publishing and subscribing messages, ensuring smooth two-way communication across the system.



Server: broker.hivemq.com:1883

Figure 5.27 HiveMQ broker in MQTT node

5.1.3.3 Python Flask

Python Flask was used to transform the machine learning model into a web-accessible API. Its setup involved installing Flask through pip and enabling Cross-Origin Resource Sharing (CORS) to allow requests from both Node-RED and the Flutter app. The Flask server was also connected to the InfluxDB client, enabling it to read sensor data and write back prediction results into the database. By acting as a lightweight backend service, Flask made AI-driven predictions available to other system components, particularly the Flutter mobile app, which retrieved prediction outputs through API requests.

5.1.3.4 FlutterFlow

FlutterFlow was adopted to streamline the creation of the mobile app's user interface. It provided a visual, low-code platform to design the Flutter-based app quickly, reducing the time and complexity of writing UI code manually. In addition, FlutterFlow offers a variety of pre-built templates that can be used as idea bases for creating new mobile apps. These templates serve as starting points, allowing developers to customize layouts, components, and styles to fit project-specific needs. This feature not only accelerates the design process but also helps ensure consistency and usability across the app's interface.

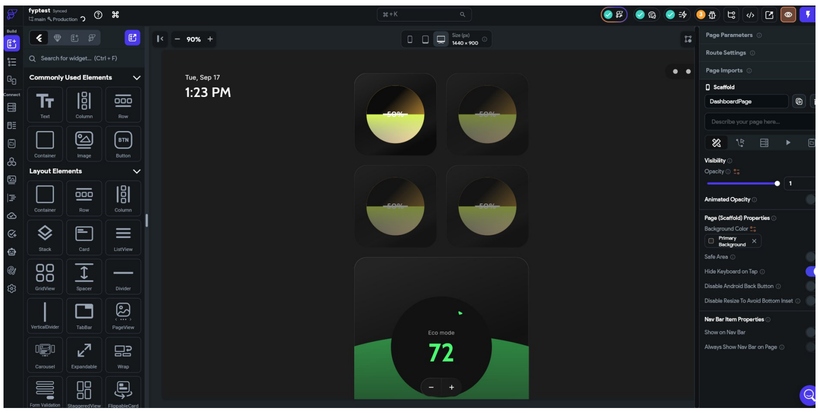


Figure 5.28 FlutterFlow UI

Note: No push notifications, authentication, or cloud storage is used.

5.1.4 Database Configuration and Setup

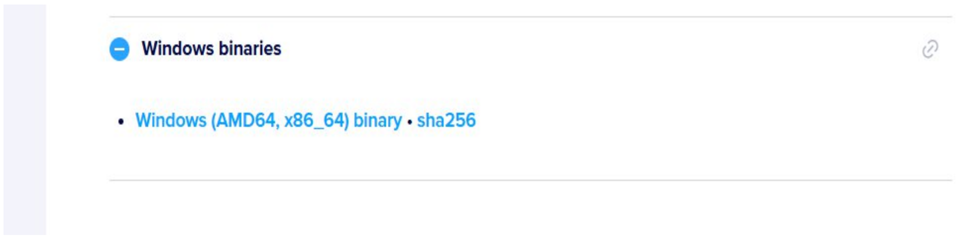
5.1.4.1 InfluxDB Database Configuration

Install InfluxDB

InfluxDB was installed locally on the PC to act as the main time-series database for storing both raw sensor readings and prediction results. Installing it locally provided greater flexibility and control over data handling without relying on external hosting services. To install InfluxDB, the installation script was obtained directly from the official InfluxData website. (https://www.influxdata.com/downloads/) This was done by running the following command in the terminal:

Steps:

1. Visit the official InfluxData website . (https://docs.influxdata.com/influxdb3/core/install/#install) and download the **Windows binary package** for InfluxDB.



Windows binaries

- Windows (AMD64, x86_64) binary • sha256

Figure 5.29 Install InfluxDB

2. Once downloaded, extract the contents of the **ZIP file**.
3. Locate the executable file named **influxd.exe** inside the extracted folder.

```
influxd                                5/21/2025 6:53 AM
```

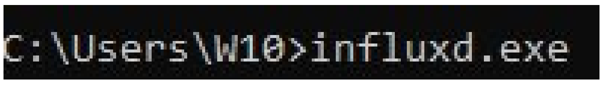
Figure 5.30 influxd.exe file

4. Move **influxd.exe** to a directory of your choice (e.g., C:\InfluxDB\) for easier access.

5. Launch the InfluxDB service to verify that it is running correctly.

6. Open the **Command Prompt** and navigate to the selected directory using the cd command.

7. Run **influxd.exe** in the Command Prompt to start the InfluxDB server.

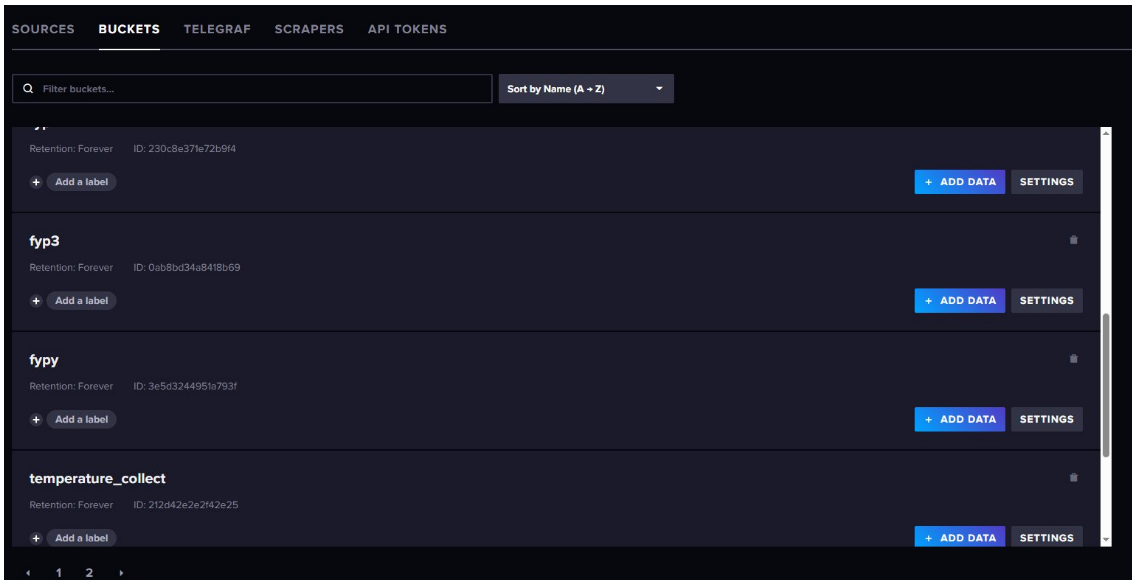


```
C:\Users\W10>influxd.exe
```

Figure 5.31 Run Influx DB

Create Buckets

Following the installation, four dedicated buckets were created to organize the data efficiently. The first bucket stores color and fragrance records obtained from sensor inputs, while the second bucket is reserved for predicted color and fragrance values generated by the AI module. The third bucket maintains temperature records, and the fourth bucket stores predicted temperature values. Each bucket can be accessed using a combination of the organization code, the bucket name, and the API token, ensuring secure and structured data management.



Retention: Forever ID: 230c8e371e72b9f4

fyp3
Retention: Forever ID: 0ab8bd34a8418b69

fypy
Retention: Forever ID: 3e5d3244951a793f

temperature_collect
Retention: Forever ID: 212d42e2e2f42e25

Figure 5.32 Influx DB's bucket

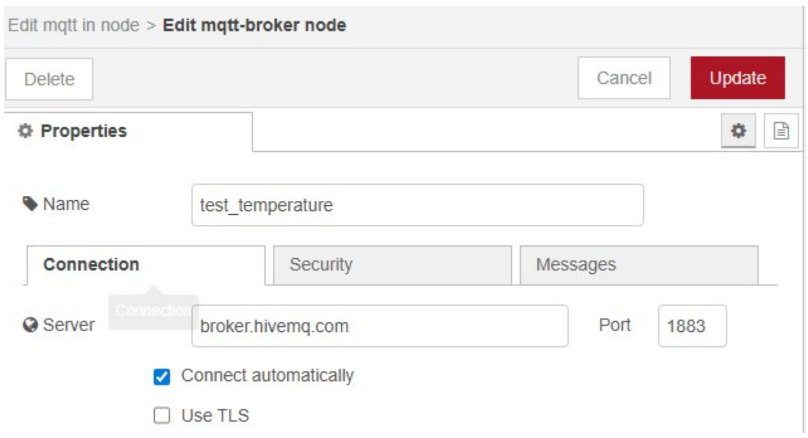
Integration

Integration between InfluxDB and the rest of the system was achieved through multiple channels. Node-RED pushes user input data into the appropriate buckets via MQTT, ensuring real-time updates of environmental data. Meanwhile, the Python AI scripts interact with InfluxDB by retrieving the data store by Node-RED. After making decision write the prediction results directly into their respective buckets. This integration allows both real-time monitoring and historical data analysis to support adaptive automation within the smart home environment.

5.1.5 Cloud Services Tools Configuration and Setup

In this project, the only cloud service used is the **HiveMQ MQTT broker**, which serves as the central communication layer connecting the mobile app (Flutter/Dart), Node-RED, InfluxDB, and the Python AI script. HiveMQ provides a reliable **publish–subscribe messaging system**, enabling real-time data exchange among all components.

To configure this, ensure that **HiveMQ is set as the server** when editing the mqtt-in node in Node-RED. As shown as Figure 5.33


Edit mqtt in node > Edit mqtt-broker node

Delete | Cancel | Update

Properties

Name: test_temperature

Connection | Security | Messages

Server: broker.hivemq.com   Port: 1883

☑ Connect automatically

☐ Use TLS

Figure 5.33 MQTT node node's server

5.1.6 Deployment and Hosting Tools Configuration

5.1.6.1 Mobile App Tools

The mobile app is deployed as an APK build, intended primarily for internal testing purposes. This allows the app to be installed directly on Android devices without requiring distribution through app stores.

5.1.6.2 Backend Services Tools

The backend services, which include Node-RED, the Python-based AI prediction script, and InfluxDB, are hosted locally on a personal computer. This setup ensures that the core logic, data processing, and storage remain within a controlled local environment during development and testing.

5.1.6.3 Server Configurations

The system operates using local ports for communication between the components. No SSL/TLS encryption or MQTT authentication mechanisms have been implemented, as the system is designed for a local testing environment rather than a production-grade deployment.

5.1.7 Monitoring and Analytics tools Configuration and Setup

5.1.7.1 Node-RED Dashboard

The Node-RED Dashboard is utilized to visualize smart home device data and current status. It serves as the central interface where users can observe real-time system performance and interact with devices when needed.

5.1.7.2 Metrics Tracked

The system continuously monitors and tracks key environmental and device-related metrics. These include **temperature, lighting, and fragrance** levels, ensuring that the environment adapts to user preferences.

5.1.7.3 Data Logging

All monitored data is stored in InfluxDB. This allows for efficient time-series data storage, enabling both real-time analytics and historical record keeping for future analysis.

5.2 Implementation Details

5.2.1 Flutter Implementation Details

Flutter serves as the mobile interface of the smart home system, providing cross-platform development capabilities and a widget-based structure for building responsive user interfaces. In this system, Flutter is responsible for:

- Connecting to the MQTT broker for device communication.
- Integrating UI widgets with device commands.
- Allowing real-time control and monitoring of smart home appliances.

5.2.1.1 MQTT Client Setup

To enable real-time communication between the Flutter mobile app, Node-RED, and the Python AI module, an MQTT client is configured. This client manages the publish–subscribe mechanism, ensuring that device commands and sensor data are reliably exchanged across all components. The following tables summarize the libraries, methods, and workflows used for MQTT communication.

Table 5.3 Libraries and Methods for MQTT Client Setup

Library / Method	Function / Description
mqtt_client	Provides MqttServerClient to establish MQTT connections.
MqttConnectMessage()	Configures session parameters for the MQTT connection.
publishMessage()	Sends messages to specific MQTT topics.

Functions and Workflow:

The MQTTService class manages MQTT communication. A unique clientId is generated for each session to identify the client. The connect() method establishes a connection to the broker (broker.hivemq.com, port 1883).

Table 5.4 Function and Workflow of MQTT Client

Function / Method	Description
publish(String message)	Sends a message to the default MQTT topic.
publishToTopic(String topic, String message)	Sends a message to a specified MQTT topic.
_send(String topic, String message)	Helper function that checks the connection state, builds the payload with MqttClientPayloadBuilder().addString(), and publishes via client.publishMessage().
disconnect()	Safely terminates the MQTT connection.

Table 5.5 Example MQTT code Function

Code	Function
mqttService.publish("thermostat:25");	Publish to default topic
mqttService.publishToTopic("home/light/set", "ON");	Publish to custom topic

5.2.1.2 UI Integration with MQTT (Thermostat Example)

The Flutter frontend integrates with MQTT to enable interactive control of smart home devices. In this example, a thermostat slider captures user input, synchronizes with the backend via a Flask API, and publishes updates to the MQTT broker. The table 5.6 summarizes the key widgets and functions involved in this integration.

Table 5.6 Widgets and Functions for thermostat MQTT

Widget / Function	**Description**
CircularTempSliderWidget	Captures user input via the onChanged callback.
setState()	Updates the local UI state to reflect changes.
http.get()	Retrieves the latest recorded temperature from the backend Flask API for initial synchronization.

Workflow:

When the slider value changes, onChanged triggers:

1. The UI state updates (setState(() => currentTemp = newValue)).
2. The new value is sent via mqttService.publish().

Initialization:

During initialization, the widget fetches the most recent value from the API:

final response = await http.get(Uri.parse("http://<server-ip>:5000/latest"));

The API response is retrieved using the http.get() method, and the JSON payload is parsed via json.decode(). The latest temperature value is extracted from the response and converted to a numeric type using double.tryParse(). This value is then applied to update the UI state through the setState() function and by calling a state management method (e.g., FFAppState().updateThermostatStruct()), ensuring the displayed temperature matches the last recorded state from the backend.

Debounced Publishing:

To prevent flooding the broker with rapid updates, a debounce mechanism delays publishing using Timer(Duration(milliseconds: 1500)). The converted Celsius value is sent via MQTT:

```
final celsiusValue = (value - 32) * 5 / 9;

_mqttService.publish('room temperature is $celsiusValue');
```

5.2.1.3 Device Control Example – Thermostat

This section demonstrates how the thermostat widget is used for device control in the smart home system. By combining Flutter UI elements with MQTT messaging, the user can interactively adjust device settings and synchronize with the backend in real-time.

Table 5.7 of Widgets and Methods Thermostat Device Control

Widget / Method	Function / Description
Slider	Provides interactive adjustment of the target temperature.
onChanged()	Captures new slider values and triggers MQTT publishing.
setState()	Updates the UI to reflect the slider position.

5.2.2 Node-RED Implementation Details

Node-RED serves as the middleware and visualization layer of the smart home system. It processes MQTT messages exchanged between the Flutter mobile app and the devices, while also providing a dashboard interface for real-time monitoring and emulation.

5.2.2.1 MQTT Integration

Node-RED connects to the same MQTT broker as the Flutter app (broker.hivemq.com) and subscribes to topics ("flutter/mqtt" for thermostat controlling and main topic for general message, "flutter/mqtt1" for devices and switch controlling, and "color/mqtt" for lightning andn fragrance controlling). It also republishes status updates by through debug node to feedback channels, ensuring synchronization between the user interface and device states.

5.2.2.2 Emulation with Dashboard

The Node-RED Dashboard is used to emulate the smart home environment through SVG-based graphics. Each room and device is visually represented in the interface, creating a more realistic simulation compared to simple toggle switches.

- Thermostat – A gauge widget is used to display the current room temperature, updated in real-time via MQTT messages.
- Lighting – A text-based UI shows the current lighting status, while a UI template dynamically renders the exact color of the light for accurate visualization.
- Fragrance Dispenser – A text indicator displays the active fragrance mode (e.g., lavender, citrus, or off).
- Air Purifier and Other Devices – Icons embedded in the SVG graphic change state to represent whether the device is active or idle.

This graphical emulation ensures that system behavior is both intuitively visible and synchronized with the Flutter app's controls, allowing users to confirm device responses at a glance.

5.2.2.3 Data Logging to InfluxDB

Node-RED forwards selected device data to InfluxDB for historical storage. This enables long-term monitoring of device usage, environmental conditions, and energy efficiency, which can later support AI-based analysis and adaptive automation. In addition, an HTTP request node is integrated to periodically fetch external data such as temperature, weather conditions, and location-specific information. These external values are combined with the MQTT device data and stored together in InfluxDB. By enriching the local sensor dataset with contextual weather and location information, the system ensures that AI models have access to more comprehensive inputs, improving prediction accuracy and enabling more effective adaptive automation.

5.2.3 InfluxDB Implementation Details

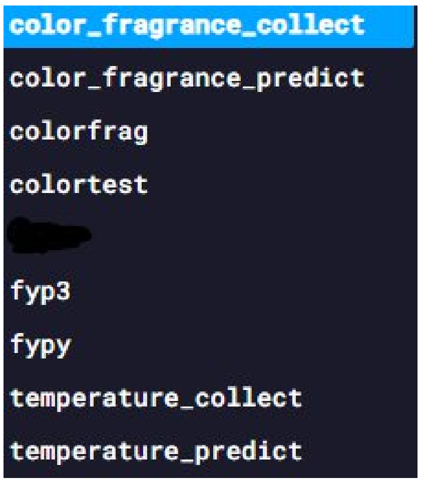
InfluxDB serves as the time-series database of the smart home system. It provides efficient storage and retrieval of real-time and historical data, enabling both long-term monitoring and AI-driven predictions. The database acts as the central repository for sensor readings, device states, and AI-generated recommendations.

5.2.3.1 InfluxDB Roles:

- **Historical Storage:** Collects device telemetry and contextual data from Node-RED, supporting long-term analysis and AI learning.
- **Predicted Data Storage:** Stores AI-generated recommendations for retrieval by the Python API, enabling adaptive automation and mobile app initialization.

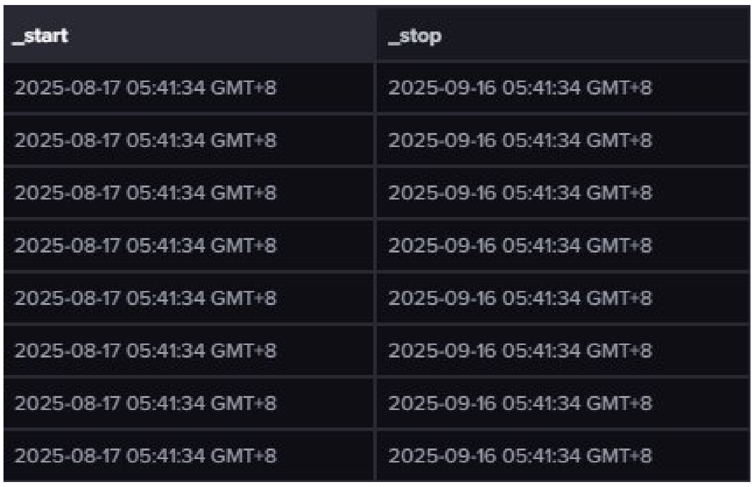
5.2.3.2 Historical Data Storage from Node-RED

Node-RED forwards selected device data to InfluxDB into different specific bucket for persistent storage. This includes readings from thermostats, lighting, fragrance dispensers. Additionally, an HTTP request node is used to periodically fetch contextual data, such as temperature, weather conditions, and location-specific information. These external data points are combined with the MQTT device data and stored together in InfluxDB. This integration ensures long-term monitoring of device usage, environmental conditions, and energy efficiency. Besides, a richer dataset for AI models, which improves the accuracy of predictions and adaptive automation. Bucket colorfrag, colortest are used for development testing of lightning and fragrance, and the color_fragrance_collect bucket are used for store lightning and fragrance data from user input, and color_fragrance_predict bucket are used for store predicted lightning ad fragrance data from python. Bucket fyp3, fypy are used for development testing of temperature, and the temperature_collect bucket are used for store lightning and fragrance data from user input, and temperature_predict bucket are used for store predicted lightning ad fragrance data from python.



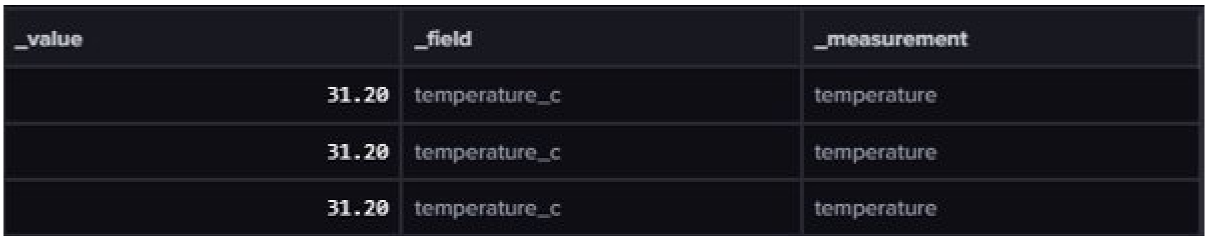
```
color_fragrance_collect
color_fragrance_predict
colorfrag
colortest
[redacted]
fyp3
fypy
temperature_collect
temperature_predict
```

Figure 5.34 Bucket's name



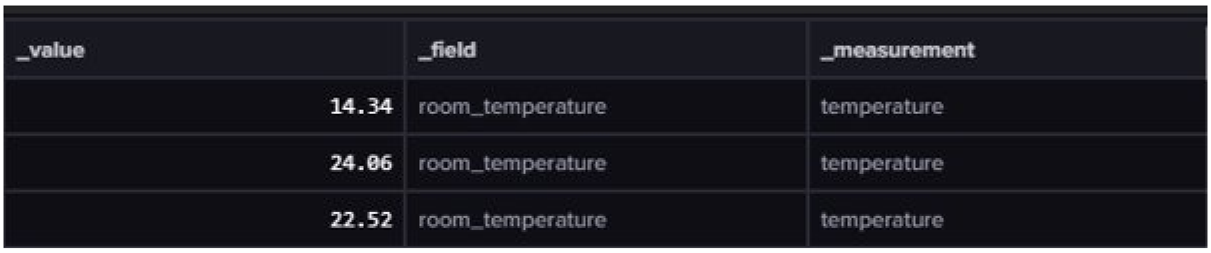
_start	_stop
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8
2025-08-17 05:41:34 GMT+8	2025-09-16 05:41:34 GMT+8

Figure 5.35 _start column and _stop column in Influx DB

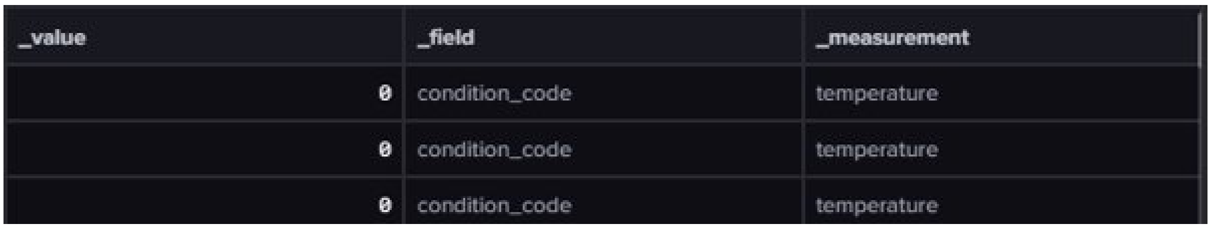


_value	_field	_measurement
31.20	temperature_c	temperature
31.20	temperature_c	temperature
31.20	temperature_c	temperature

Figure 5.36 Historical Data Storage from Node-RED of outdoor temperature



_value	_field	_measurement
14.34	room_temperature	temperature
24.06	room_temperature	temperature
22.52	room_temperature	temperature



_value	_field	_measurement
0	condition_code	temperature
0	condition_code	temperature
0	condition_code	temperature

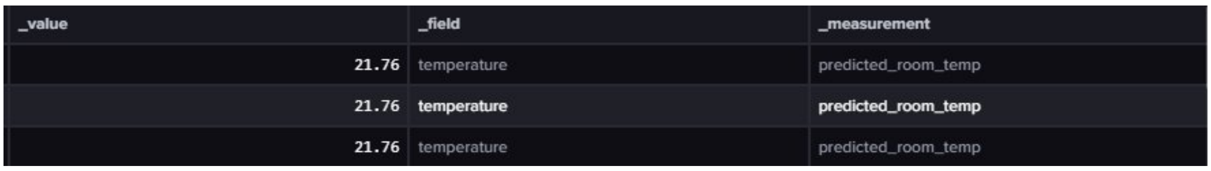
Figure 5.37 Historical Data Storage from Node-RED of room temperature and weather condition

From Figure5.35 it has a _start column showing the starting time of creating this bucket, _stop column for the time of storing the data, _value column can be outdoor temperature value(which retrieve from the http request from weatherapi) as the _field is state temperature_c in Figure5.36, can also be room temperature as _field is room_temperature in Figure5.37, can also be weather condition as _field column show condition_code in Figure5.37, the _value of 0 is indicating clear weather. Column of _measurement are temperature it is where the influx DB node in Node-RED already state.

Noted: The table are originally connected together, only separate it into two figures for better showing.

5.2.3.3 Predicted Data Storage from AI Module

The AI module generates predicted device settings based on historical trends and real-time data. These predictions, such as recommended thermostat targets or lighting adjustments, are also written to InfluxDB. A Python API retrieves this predicted data to initialize the mobile app and provide users with real-time AI recommendations. This dual storage mechanism enables Centralized access to both raw sensor data and AI predictions, and a seamless interface for the mobile app to display accurate initial values. And these are the roles of color_fragrance_predict and temperature_predict bucket. But for testing bucket we have fypy for temperature predicted data, and colorfrag for lightning and fragrance predicted data.



_value	_field	_measurement
21.76	temperature	predicted_room_temp
21.76	temperature	predicted_room_temp
21.76	temperature	predicted_room_temp

Figure 5.38 Predicted Data Storage from AI Module

Figure 5.38 shows as the _measurement columns is different as predicted_room_temp. As this is a fypy bucket storing of predicted temperature value.

5.2.4 Python AI & Machine Learning Module

The Python AI/ML module predicts room conditions to support adaptive automation in the smart home system. It retrieves historical and real-time data from InfluxDB, preprocesses and encodes it for machine learning, generates predictions, stores them in a dedicated bucket, and exposes the results via a Flask API to the Flutter mobile app. This allows the interface to display AI-driven suggestions alongside live measurements, enhancing adaptive control of smart home devices.

5.2.4.1 Data Retrieval and Preprocessing

This section is to retrieve relevant data from InfluxDB, clean it, and convert it into a format suitable for machine learning. This involves aligning time-series data, handling missing values, encoding categorical features, and extracting temporal features from timestamps. The processed data is then used to train predictive models.

Table 5.8 Data Retrieval and Preprocessing libraries and method

Library / Method	Function / Purpose	Usage / Example
influxdb-client	Query device and environmental data from InfluxDB	InfluxDBClient.query_api().query()
pandas	Convert query results into DataFrames, align datasets, handle missing values	pd.DataFrame()
datetime	Process timestamps and round them to the nearest minute	pd.to_datetime().dt.round('1min')
LabelEncoder (from sklearn.preprocessing)	Encode categorical features (e.g., weather) into numeric values	.fit() and .transform()
Temporal feature extraction	Derive contextual inputs from timestamps, e.g., hour, minute, day of week	Used as model input features

Process:

The module fetches room temperatures, weather conditions, and device states. Data is aligned by timestamp, and categorical and temporal features are encoded numerically. These processed features are combined into the feature matrix X for training the machine learning model, while the target variable y represents the room temperature. Example: model.fit(X, y) trains the model, and predicted = model.predict([[25]]) demonstrates generating predictions.

5.2.4.2 Machine Learning Prediction

This section is showing that a machine learning model is trained to predict room temperature based on environmental and temporal features. The Random Forest Regressor is used to provide robust, adaptive predictions suitable for real-time automation.

Library: sklearn.ensemble.RandomForestRegressor

Table 5.9 Function and Workflow for Machine Learning Prediction

Function / Method	Purpose / Workflow	Usage / Example
train_test_split() (sklearn.model_selection)	Split dataset into training and testing sets	X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model.fit(X_train, y_train)	Train the Random Forest model on training data	model.fit(X_train, y_train)
model.predict(X_test)	Generate predictions for evaluation	y_pred = model.predict(X_test)
mean_squared_error()	Calculate error metrics between predicted and actual values	mse = mean_squared_error(y_test, y_pred)
matplotlib.pyplot.plot()	Visualize predicted vs actual values to assess model performance	plt.plot(y_test, label='Actual') plt.plot(y_pred, label='Predicted')

The trained model predicts future room temperatures based on environmental and temporal features, enabling adaptive automation.

5.2.4.3 Storing Predicted Data

This section is showing that the data is stored in InfluxDB for use in analytics and by the mobile application. Predicted values are stored in a separate bucket to maintain clear organization between raw and predicted data.

Library: influxdb-client

Table 5.10 Function and Workflow of Storing Predicted Data

Function / Method	Purpose / Workflow	Usage / Example
Point().field().time()	Constructs a data point containing predicted values and timestamps for InfluxDB	Point("predicted_room_temp").field("temperature", float(predicted[0])).time(datetime.utcnow())
write_api.write(bucket, org, record)	Writes the constructed data point to a dedicated predicted data bucket in InfluxDB	write_api.write("predicted_data_bucket", "org_name", point)

By separating predicted and raw input data, the system maintains clear organization for historical analysis and visualization.

5.2.5 API Provision for Mobile App

The AI module provides an API using Flask, allowing the mobile application to access the latest predicted data in real-time. This ensures that the user interface can display accurate and up-to-date predictions.

Library: Flask with flask_cors.CORS

Table 5.11 of Code of Functions of API Provision for Mobile App

Function / Method	Purpose / Workflow	Usage / Example
@app.route()	Defines API endpoints for Flask	@app.route('/latest_prediction')
InfluxDBClient.query_api().query()	Queries InfluxDB to fetch recent predictions	`query_api.query('from(bucket:"predicted_data")')
jsonify()	Returns JSON responses to the client (Flutter app)	return jsonify(prediction=result)

5.3 System Operation

5.3.1 Flutter Mobile App

Function

The Flutter mobile app serves as the primary interface for users to monitor and control smart home devices in real-time. Its core functions include capturing user inputs (such as adjusting temperature, turning lights on/off, or selecting fragrance settings), updating the local UI using state management (setState()), and transmitting commands via MQTT (publish() or publishToTopic()). Additionally, the app fetches AI-generated predictions from the Flask API to display recommended environmental settings. It also implements mechanisms like debounced publishing to ensure stable communication with the MQTT broker without flooding messages during rapid user interactions.

Roles and Integration with Other Components

The mobile app acts as the **front-end controller** in the smart home ecosystem. It integrates with:

- MQTT Broker: Publishes user commands and subscribes to device updates, enabling real-time communication with Node-RED.
- Node-RED: Reflects user commands and device states in the SVG smart home emulator, allowing visual feedback for every action.
- Python AI Module (Flask API): Retrieves predicted room conditions, displaying AI-assisted suggestions alongside real-time measurements.
- InfluxDB: Stores historical user actions and AI predictions, providing a data source for analysis and machine learning.

Through these integrations, Flutter ensures **bi-directional communication**: user actions update the system, and system responses and predictions are displayed back to the user, creating an interactive and adaptive smart home experience.

5.3.2 Node-RED

Function

Node-RED acts as the orchestration layer of the smart home system, handling message flows and device emulation. It receives MQTT messages from the mobile app, parses and processes commands, and updates the SVG-based emulator in real time to reflect device states (e.g., lights, AC, purifier). Node-RED also logs incoming data to InfluxDB for historical storage and forwards necessary information to the AI module for further analysis. Additionally, it manages control loops where AI-generated recommendations are automatically applied to devices.

Roles and Integration with Other Components

Node-RED serves as the central middleware of the system, integrating with:

- MQTT Broker: Facilitates inbound and outbound communication between the mobile app and backend devices.
- Flutter Mobile App: Displays device feedback visually through the emulator and acknowledges user commands.
- InfluxDB: Stores real-time device data and user interactions for monitoring and long-term analysis.
- Python AI Module: Provides AI predictions for adaptive automation, which Node-RED applies to the SVG emulator and devices.

5.3.3 Python AI Module

Function

The Python AI module implements machine learning for predictive and adaptive automation. It retrieves historical user data and environmental conditions from InfluxDB, preprocesses and encodes the data, and uses a trained model (e.g., Random Forest Regression) to predict future device states such as optimal temperature or lighting. Predictions are stored back into a dedicated InfluxDB bucket and are exposed to external systems via a Flask API. This enables both Node-RED and the Flutter mobile app to consume AI-driven insights in real time.

Roles and Integration with Other Components

The AI module functions as the intelligence engine, integrating with:

- InfluxDB: Retrieves historical sensor and user data, and stores prediction results for later use.
- Node-RED: Sends predictions to be applied to the smart home emulator and devices.
- Flutter Mobile App: Exposes predictions through the Flask API so users can view AI-suggested settings.
- MQTT Broker: Works indirectly by enabling feedback loops through Node-RED for applying AI recommendations to devices.

5.3.4 InfluxDB

Function

InfluxDB acts as the time-series database for the system, storing both real-time and historical data. Device commands, user interactions, and sensor readings are written into dedicated buckets for organized storage. The AI module retrieves these datasets for training and prediction, while prediction outputs are written back into separate buckets. This dual role ensures that InfluxDB provides both historical context and live prediction data to support adaptive automation.

Roles and Integration with Other Components

InfluxDB functions as the system's data backbone, integrating with:

- Node-RED: Receives incoming data streams and writes them into appropriate buckets.

- Python AI Module: Supplies historical data for analysis and accepts AI-generated predictions for storage.
- Flutter Mobile App: Provides a source of recorded and predicted data that can be retrieved via APIs for display.
- MQTT Broker: Indirect integration, since MQTT messages routed through Node-RED are logged into InfluxDB for persistence.

5.3.5 Observable Outputs

This section show generated of several observable outputs that demonstrate its functionality and integration with the smart home system via the four functional components(Flutter mobile app, Node-RED emulator, Inkscape DB, and Python AI machine learning script)

5.3.5.1 AI-driven Initialization of Device States

Upon launching the app, AI-predicted values are retrieved from the backend API, including room temperature and recommended device settings. For example, the living room air conditioner may be automatically turned on. This ensures that the environment starts in an optimized state, providing users with an AI-assisted baseline before interaction.

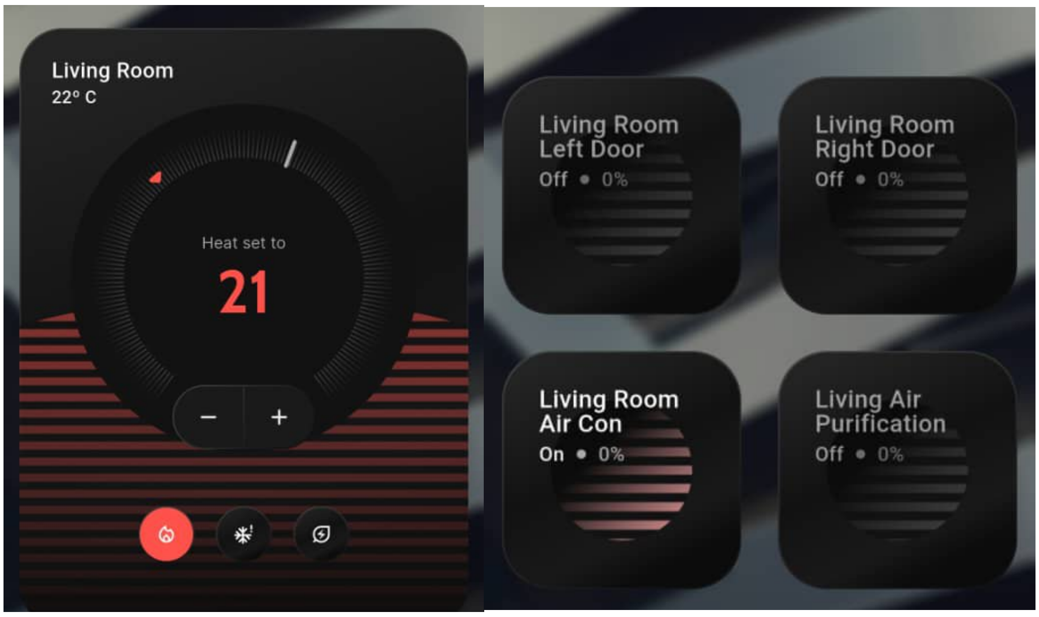


Figure 5.39 Flutter App's UI when initialized

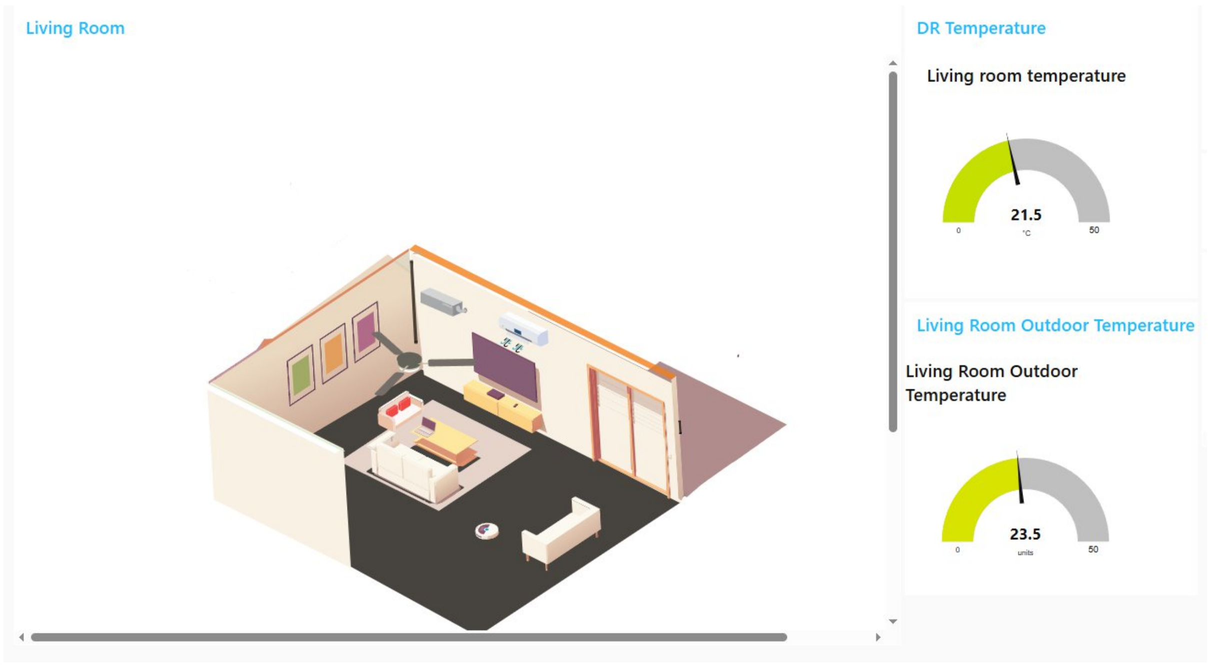


Figure 5.40 Node-RED smart home emulator when initialized

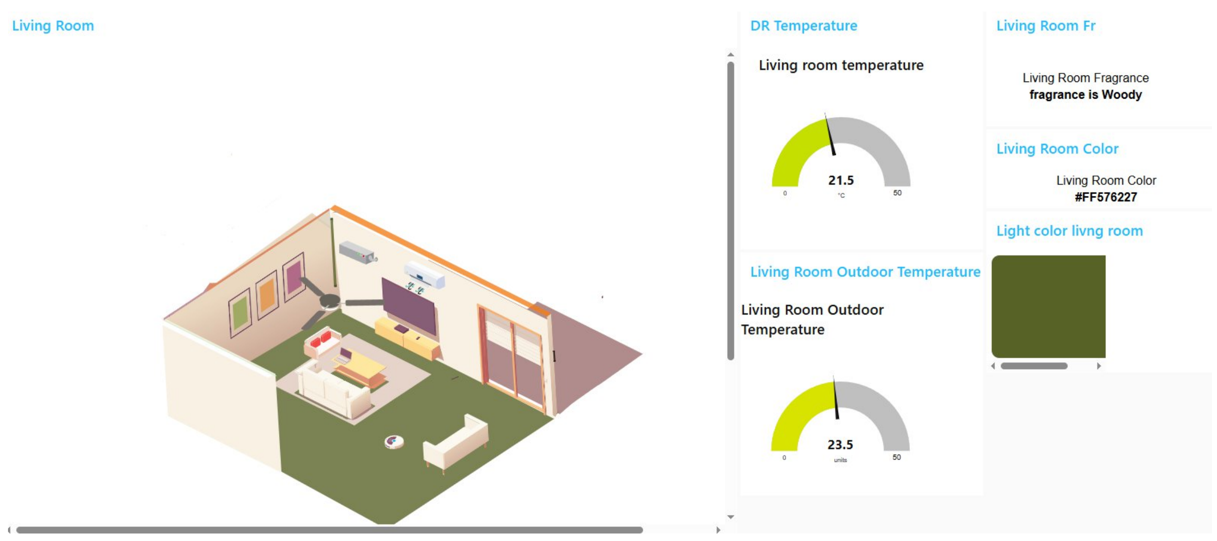
```
Predicted Room Temp: 21.44°C in 2025-09-16 07:01:14.183039
```

Figure 5.41 Predicted value from AI module

Scenario: When the predicted value is 21.44°C, the mobile app control device rounds this to 21°C, and the living room air conditioner switch is automatically turned on. On the Node-RED dashboard, the temperature gauge is updated to 21.5°C (as it retains one decimal point). Additionally, the outdoor temperature is set based on the http-request node in Node-RED, which transmits the value alongside the message. In the SVG floor plan, the living room air conditioner is represented with a wind icon, visually indicating that it is active.

5.3.5.2 User Interaction for Device Monitoring and Control

The UI provides a visual representation of all device states, such as lighting, doors, air purifiers, and fragrance dispensers. User interactions—like toggling a switch, adjusting a slider, or changing a light color—are immediately reflected in the interface, enabling seamless monitoring and control of the smart home environment.



Living Room Fragrance
fragrance is Woody

Living Room Color
#FF576227

Figure 5.42 Node-RED smart home emulator in Living Room after modifying via app

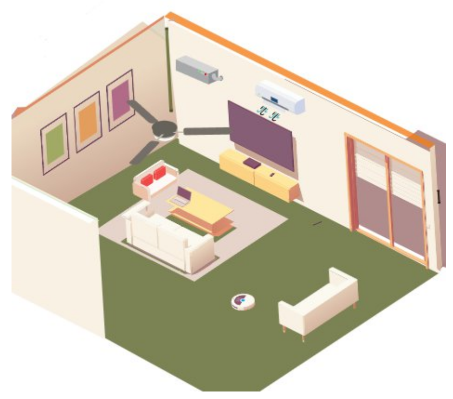
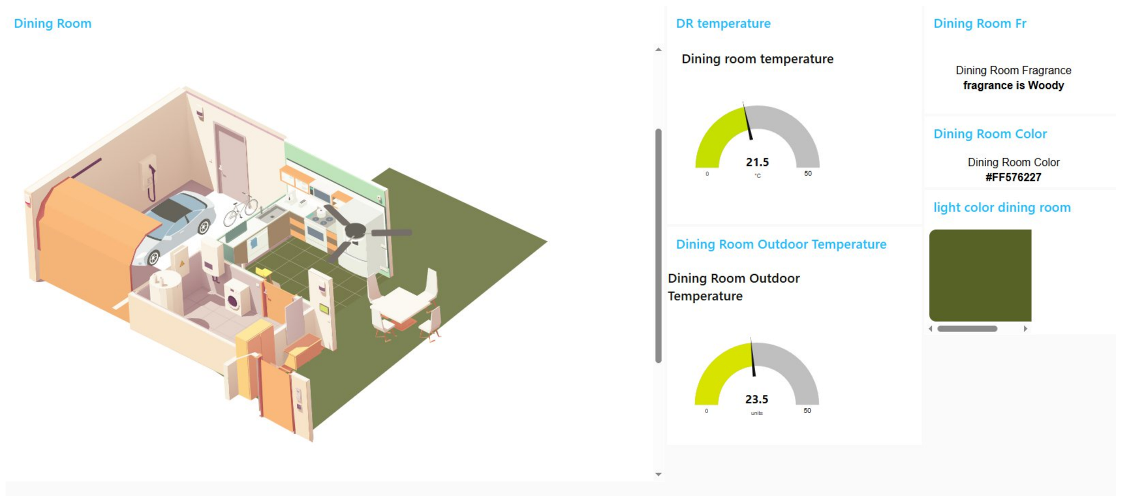


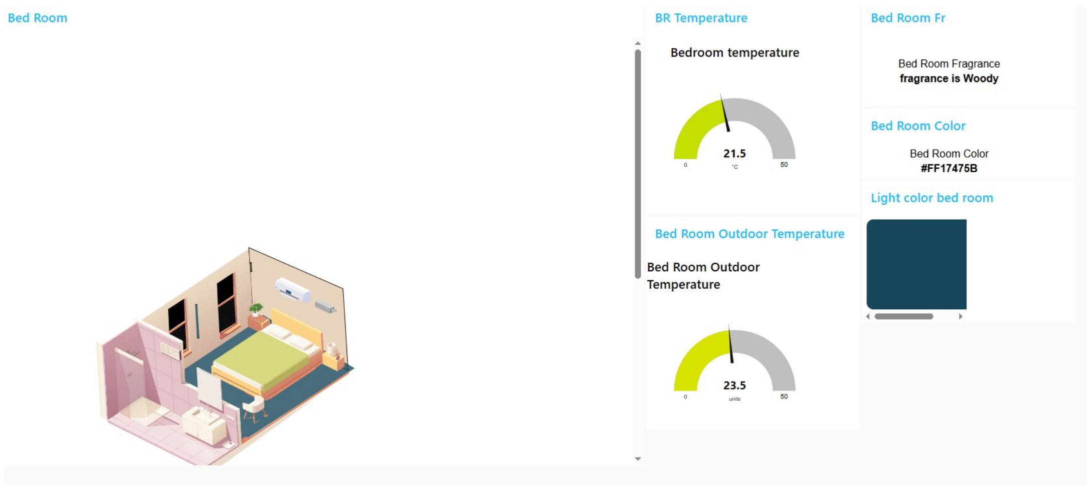
Figure 5.43 zoom in for the essential part of living room



Dining Room Fragrance
fragrance is Woody

Dining Room Color
#FF576227

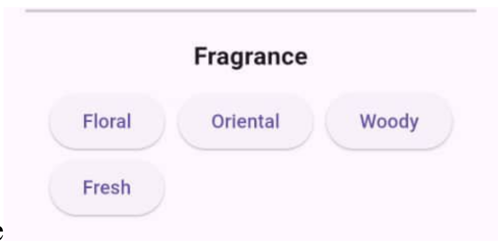
Figure 5.44 Node-RED smart home emulator in Dining Room after modifying via app



Bed Room Fragrance
fragrance is Woody

Bed Room Color
Bed Room Color
#FF17475B

Figure 5.45 Node-RED smart home emulator in Bed Room after modifying via app



e

Figure 5.46 Fragrance UI button

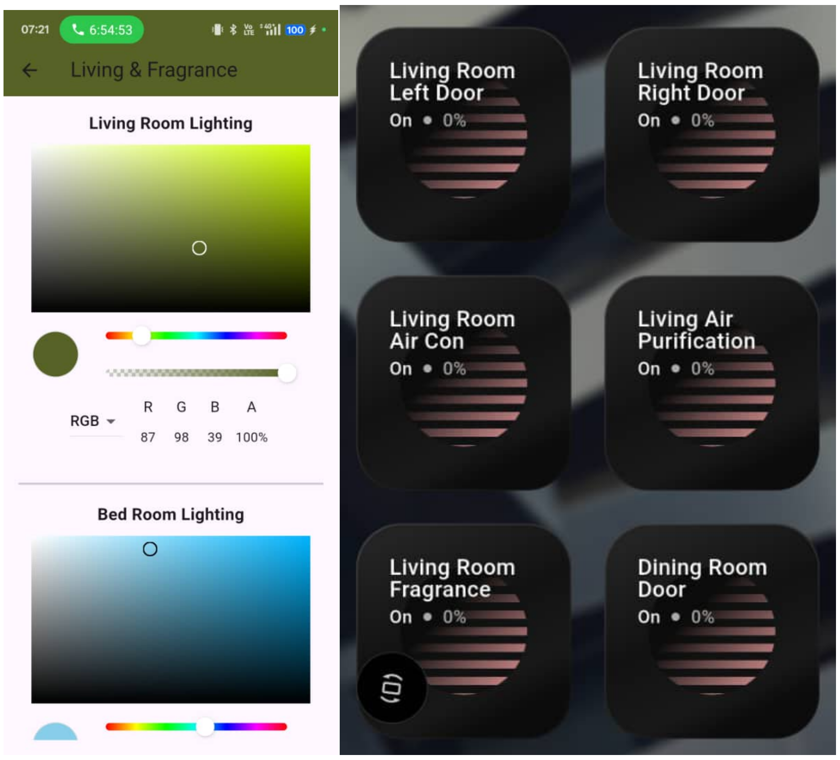
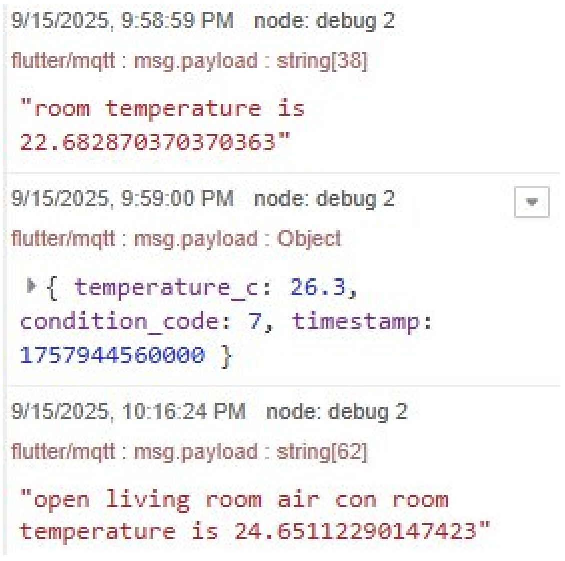


Figure 5.47 switch and lightning UI pages

Scenario: All device switches (air conditioner, door, air purifier, fragrance dispenser) are toggled on. A fragrance is selected for three rooms, and custom lighting colors are applied to the **living room** and **bedroom**. The floor plan and wall-mounted bulb icons in the living room and dining room change to match the selected colors. In addition, the fragrance and lighting text labels update to display the chosen names. At the bottom of the UI, a template section displays blank color squares to match the selected room themes. Note: The dining room and living room are interconnected, so they share devices.

5.3.5.3 MQTT-Based Synchronization with Backend Systems

Every user action in the mobile app is converted into an MQTT message and sent to Node-RED. This ensures synchronization across the system: the SVG emulator updates device states, InfluxDB records the actions for historical storage, and the AI module uses these logs to refine predictions.



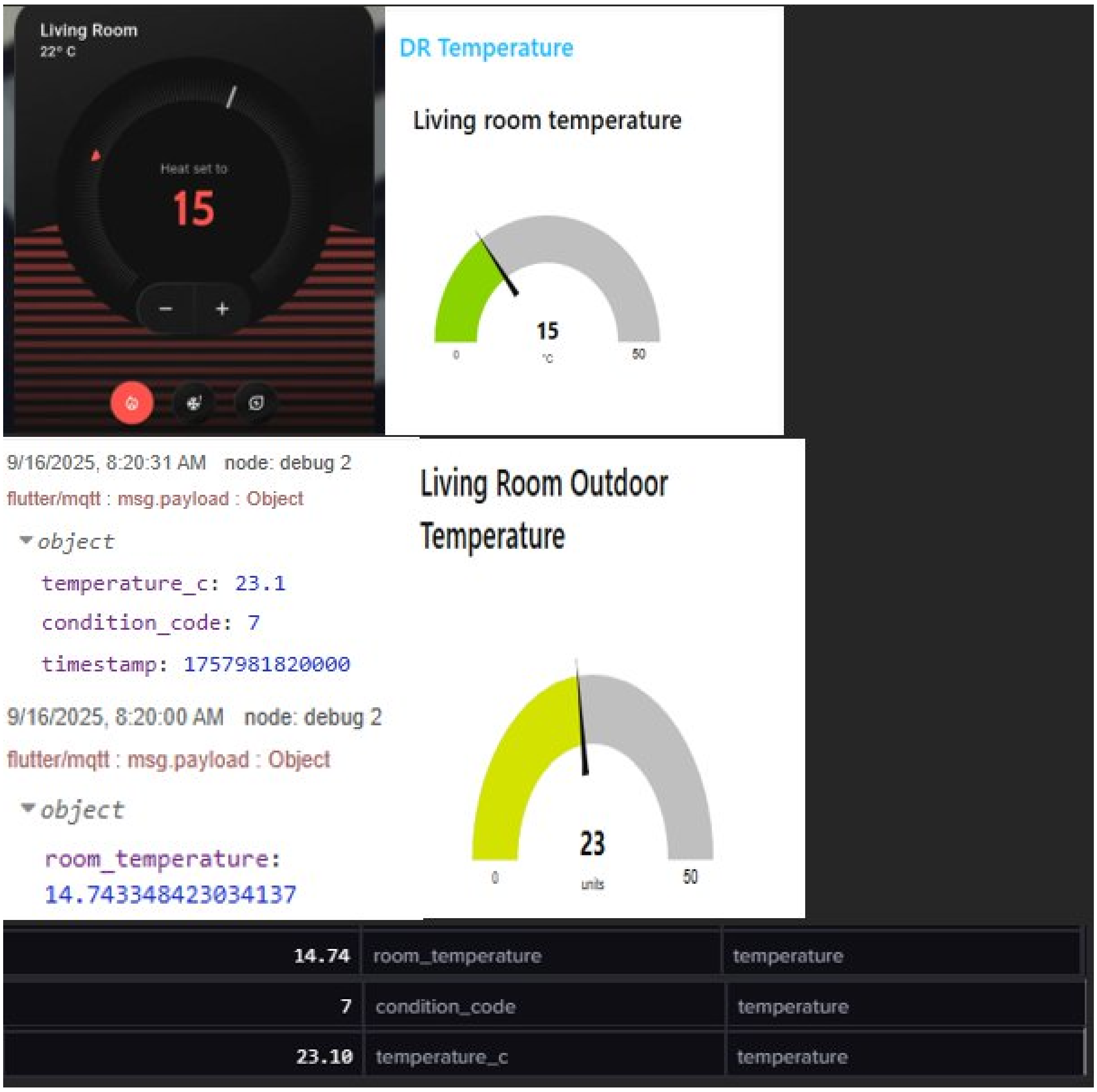
```
9/15/2025, 9:58:59 PM   node: debug 2
flutter/mqtt : msg.payload : string[38]
"room temperature is
22.682870370370363"

9/15/2025, 9:59:00 PM   node: debug 2
flutter/mqtt : msg.payload : Object
▸{ temperature_c: 26.3,
condition_code: 7, timestamp:
1757944560000 }

9/15/2025, 10:16:24 PM   node: debug 2
flutter/mqtt : msg.payload : string[62]
"open living room air con room
temperature is 24.65112290147423"
```

Figure 5.48 MQTT message sent to Node-RED from mobile app

Scenario: At initialization, the mobile app publishes messages containing the predicted room temperature and the living room air conditioner's ON state. These MQTT messages are received by Node-RED, reflected in the SVG dashboard, logged in InfluxDB, and form part of the data stream available for AI-driven automation.

5.3.5.4 InfluxDB – Receiving User Messages

InfluxDB serves as the time-series database that receives messages from the mobile app through Node-RED and MQTT. Each user action—such as switching on the living room air conditioner, adjusting the bedroom light color, or setting fragrance types—is recorded with a timestamp. This enables chronological tracking of device states and ensures all user inputs are available for further processing by the AI module or for historical analysis.



```
9/16/2025, 8:20:31 AM   node: debug 2
flutter/mqtt : msg.payload : Object
▾object
  temperature_c: 23.1
  condition_code: 7
  timestamp: 1757981820000
9/16/2025, 8:20:00 AM   node: debug 2
flutter/mqtt : msg.payload : Object
▾object
  room_temperature:
  14.743348423034137
```

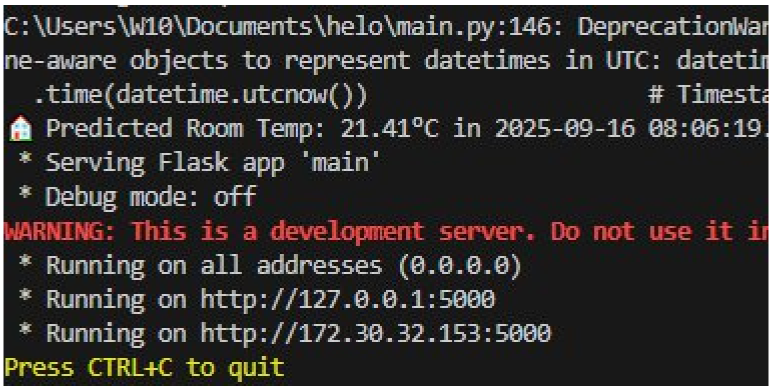
14.74	room_temperature	temperature
7	condition_code	temperature
23.10	temperature_c	temperature

Figure 5.49 Modifying of changing the temperature into 15 degrees

Scenario: When change the temperature in the mobile app to 15 degree, the living room temperature has been changed at the same time. This publish a MQTT object message with the outdoor temperature of 23.1, condition code is 7 which maps to patchy rain nearby, and the timestamp. Then publish another MQTT object message with the room temperature of 14.74. The three lines are data in Influx DB, showing exactly the same as MQTT published message.

5.3.5.5 Python AI Module – Processing and Prediction

The Python AI module retrieves the raw user messages from InfluxDB and analyzes historical and contextual data using machine learning algorithms (e.g., Random Forest Regression). It predicts optimal environmental conditions such as room temperature, lighting color, and fragrance preferences. Once predictions are generated, they are sent back to a dedicated bucket in InfluxDB. Simultaneously, a Flask API is started to make these predictions accessible to the mobile app for whenever the mobile app is started by the user, allowing real-time AI-assisted control.



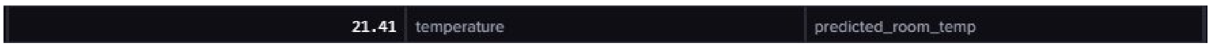
```
C:\Users\W10\Documents\helo\main.py:146: DeprecationWar
ne-aware objects to represent datetimes in UTC: datetin
  .time(datetime.utcnow())                    # Timesta
🏠 Predicted Room Temp: 21.41°C in 2025-09-16 08:06:19.
 * Serving Flask app 'main'
 * Debug mode: off
WARNING: This is a development server. Do not use it in
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.30.32.153:5000
Press CTRL+C to quit
```

Figure 5.50 Part of output of the AI module for temperature prediction

Scenario: As initial of the mobile app, the Python script generated a prediction of 21.41°C. Running on http://127.0.0.1:5000 allows access via localhost on port 5000, and it is also reachable at the local network address 172.30.32.153..

5.3.5.6 InfluxDB – Receiving AI Predictions

InfluxDB receives the AI-generated predictions and stores them in a separate bucket to keep raw user inputs distinct from predicted values. This allows the mobile app to retrieve recommended settings for display and Node-RED to apply automated control. By separating these data streams, the system ensures clarity between what the user has manually set and what the AI suggests for optimal comfort.



```
21.41 temperature predicted_room_temp
```

Figure 5.51 Receive AI prediction in Influx DB

Scenario: After the prediction has been made by the Python script, it will also sent into Influx DB fypy bucket. To let the mobile app retrieve it.

5.4 Concluding Remark

This chapter presented the complete setup, configuration, and implementation of the smart home control and monitoring system. Each component—Flutter for the mobile app, Node-RED for logic flows and visualization, MQTT for message communication, InfluxDB for data storage, and the AI module for predictive analysis—was systematically configured to ensure interoperability within the system.

The environment preparation detailed the installation and configuration of essential tools, programming languages, and platforms. The implementation section explained how the individual modules were integrated and made operational, while the system operation section highlighted the flow of data from user interactions to device control, logging, and AI analysis.

Through this structured process, the system achieved a functional prototype capable of real-time device control, monitoring, and adaptive automation. The work carried out in this chapter establishes a strong foundation for the performance evaluation and challenges discussion that follow, ensuring that the subsequent analysis is based on a fully implemented and operational system.

Chapter 6

System Evaluation and Discussion

6.1 System Performance Evaluation

The performance evaluation of the smart home system aims to assess its responsiveness, efficiency, and reliability under different operational conditions. Observations focus on device control, thermostat behavior, AI prediction handling, MQTT network stability, and database performance.

Testing Environment

The smart home system testing environment is designed to replicate realistic operational conditions for Load Testing, Response Time Analysis, Network Performance, and Database Performance.

Hardware:

- Personal Computer (PC/Laptop): Hosts Node-RED, InfluxDB, and Python ML/Flask API; central server for device management and AI predictions.
- More than one Mobile Devices (Smartphone): Runs the Flutter app for real-time control and monitoring; connected via the same network for MQTT communication.
- Wi-Fi Router / Network: Provides stable local network connectivity and supports internet access for external API calls.

Software:

- Node-RED: Manages automation flows, device emulation (SVG graphics), and MQTT messaging.
- Flutter & Dart: Provides the mobile interface for controlling and monitoring devices.
- Python with Flask API: Executes AI predictions and provides endpoints for mobile app integration.
- InfluxDB: Stores time-series data for devices, environmental metrics, and predictions.

- MQTT Broker (HiveMQ / Mosquitto): Facilitates publish–subscribe messaging between the mobile app, Node-RED, and database.

Configuration Notes:

- All devices are connected on the same local network to ensure reliable MQTT communication.
- Node-RED is configured with emulated devices for real-time updates.
- Mobile app subscribes/publishes MQTT messages and fetches AI predictions from Flask API.
- InfluxDB contains separate buckets for raw and predicted data.
- Debugging and logging tools are enabled to monitor system performance during tests.

6.1.1 Load Testing

Load testing evaluates the system's ability to handle multiple device commands simultaneously without degradation of performance.

Load Testing Tools Environment

The load testing environment for the smart home system uses specific tools to simulate multiple device commands and measure system performance.

Exclusive Tools (Other testing environment follows section above):

- Node-RED Debug Nodes – Monitors message flow, especially to monitor message flow from different mobile phone, also monitor the error of message sending.
- Windows Task Manager / Resource Monitor – Observes CPU and memory usage that may affect response time.
- Visual Studio Code Debug Console – Displays real-time debugging messages from the Python script, allowing verification of API functionality and connectivity between the Flask API, when it comes to multiple devices.

Observations:

- Multiple devices can be operated at the same time.
- Device switches (lights, color, fragrance, air purifier) respond very quickly, with a delay of approximately 50–100 milliseconds, which is too fast to be perceptible to the user.
- Thermostat adjustments have a slight delay of approximately 1–3 seconds before the change is reflected.

Conclusion:

The system handles simultaneous device operations efficiently, with only minor delays observed for thermostat control, which does not significantly affect user experience.

6.1.2 Response Time Analysis

Response time analysis measures the time taken for the system to reflect user interactions in the mobile app.

Response Time Analysis Tools Environment

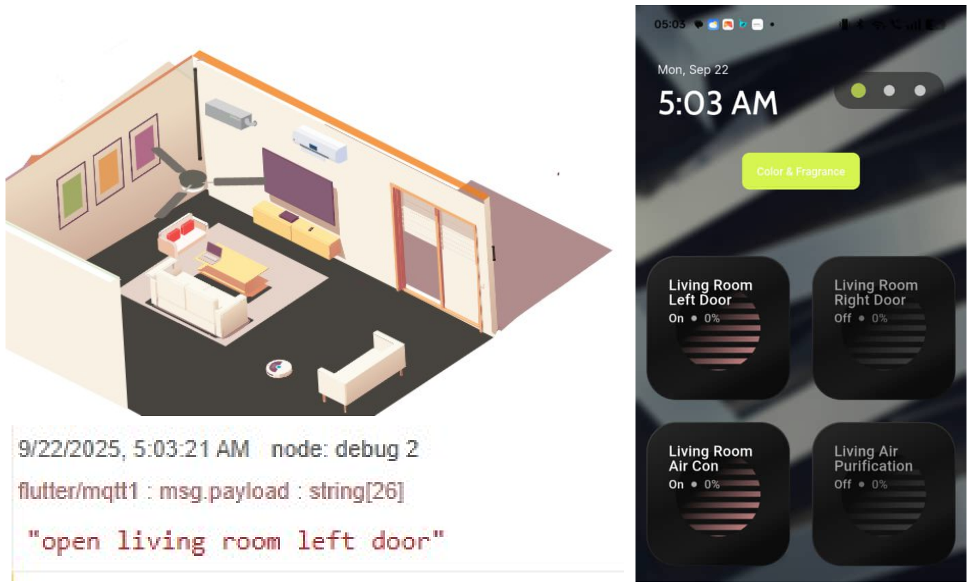
This environment uses specific tools to measure how quickly the smart home system reflects user interactions on the mobile app.

Exclusive Tools (Other testing environment follows section above):

- Node-RED Debug Nodes – Monitors the exact timing when messages are received and executed by emulated devices.
- Stopwatch– Measures the precise delay between sending a command and observing the response in the app.
- Windows Task Manager / Resource Monitor – Observes CPU and memory usage that may affect response time.
- InfluxDB Timestamps Logging – Records the exact timestamps of device commands, sensor readings, and AI prediction updates in InfluxDB for response timing analysis.

Observations:

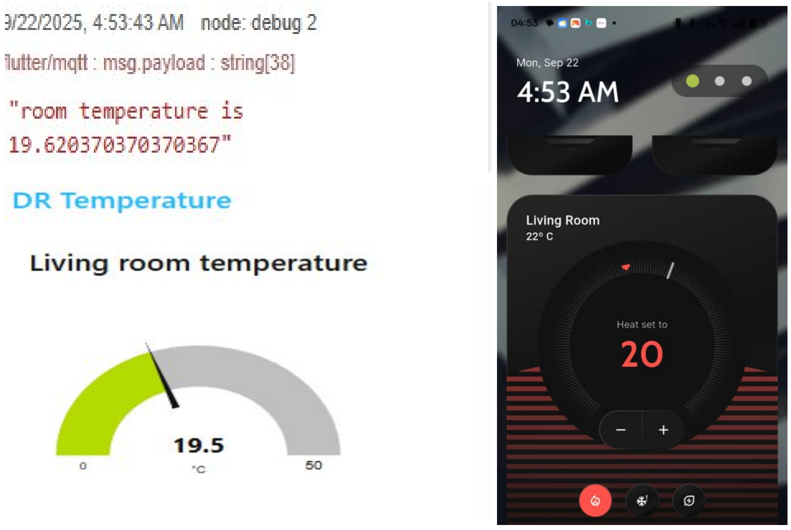
- Device toggles (devices) show immediate response. For Figure 6.1 is showing an device of living room left door. By opening it at 5.03 am on 22/09/2025, the mobile phone, and the MQTT debug message in Node-RED are showing the same time.



```
9/22/2025, 5:03:21 AM   node: debug 2
flutter/mqtt1 : msg.payload : string[26]
"open living room left door"
```

Figure 6.1 Observations of opening living room left door

- Thermostat and initial AI-predicted temperature updates have a delay of 5 seconds to 30 seconds before values appear in the mobile app. Figure 6.2 showing that the by changing the temperature, the mobile phone, and the MQTT debug message in Node-RED are showing the same time at 4.53am on 22/09/2025.



```
9/22/2025, 4:53:43 AM   node: debug 2
flutter/mqtt : msg.payload : string[38]
"room temperature is
19.620370370370367"
```

Figure 6.2 Observations of changing temperature

Conclusion:

The response time is generally acceptable for manual device control, while AI-driven temperature predictions may require optimization to improve real-time feedback.

6.1.3 Network Performance

Network performance testing assesses the stability and latency of the MQTT-based communication between the mobile app and Node-RED.

Network Performance Testing Tools Environment

This environment uses specific tools to evaluate the stability, reliability, and latency of MQTT-based communication between the mobile app and Node-RED.

Exclusive Tools (Other testing environment follows section above):

- Node-RED Debug Nodes – Tracks message reception timing and execution within the flows, and monitors MQTT connectivity to detect if the connection is lost or active.
- Ping– Measures network connection between mobile phone and pc/laptop.
- Resource Monitor – Tracks CPU, memory, and network usage that may impact MQTT performance.
- InfluxDB UI – Allows visualization of data writes and reads in real-time, helping to monitor the connection and data flow between Node-RED, Python scripts, and the database.
- Visual Studio Code Debug Console – Displays real-time debugging messages from the Python script, allowing verification of API functionality and connectivity between the Flask API, Node-RED, and InfluxDB.

Observations:

- MQTT network connection is stable and exhibits negligible latency during active use. This is proven in the 6.1.2 Response Time Analysis
- When the app runs in the background, MQTT requires the app to restart to re-establish the connection.

Conclusion:

The MQTT network is highly reliable under normal operation, but background reconnection handling could be improved to maintain persistent connectivity.

6.1.4 Database Performance

Database performance evaluation ensures that data logging, storage, and retrieval are accurate and efficient.

Database Performance Testing Tools Environment

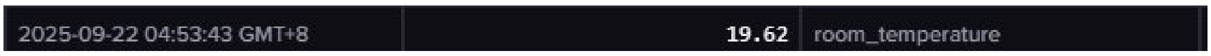
This environment uses specific tools to evaluate the efficiency, accuracy, and reliability of InfluxDB for logging, storing, and retrieving smart home data.

Exclusive Tools (Other testing environment follows section above):

- InfluxDB UI – Visualizes stored data using table, queries historical records, and checks real-time updates.
- Python Scripts Debug Output (VS Code Debug Console) – Shows real-time messages from Python scripts, including data insertion, retrieval, and verification of records in InfluxDB, allowing monitoring of successful execution and connectivity.
- Resource Monitors (PC) – Tracks CPU, memory, and disk usage during database operations.

Observations:

- All device and sensor data are accurately recorded in InfluxDB. From Figure 6.2 temperature are tested to be set on 19.62 degree at 4.53am on 22/09/205. From Figure 6.3, a screenshot of Influx DB respectively bucket showing the exact same timing and date.



```
2025-09-22 04:53:43 GMT+8                                19.62  room_temperature
```

Figure 6.3 Observation from Influx DB

- No data loss or missing records were observed during testing.

Conclusion:

The database system demonstrates high accuracy and reliability, effectively supporting the data storage and retrieval needs of the smart home system.

Table 6.1 Summary of latency

Device / Operation	**Observed Response Time**	**Notes**
Lights Switch	~1 sec	Immediate toggle
Fragrance Control	~1 sec	Immediate toggle
Color Control	~1 sec	Immediate toggle
Air Purifier Switch	~1 sec	Immediate toggle
Thermostat Adjustment	~2-3 sec	Delay due to MQTT connection in the flutter code.
Initial AI Temperature Prediction	30 sec – 1 min	Reflected on mobile after prediction update

6.2 System Testing Setup and Result

System functional testing is an essential phase in the development process, aimed at verifying that the implemented system meets the defined requirements and performs as intended. It focuses on evaluating the smart home system's core functionalities, including device control, data storage, adaptive automation, and user interaction through the mobile application, ensuring that input and output flows are accurate and that the system behaves consistently under various scenarios. By systematically executing test cases, potential issues can be identified and resolved early, allowing the prototype to be validated against its objectives and confirming that the integration of Flutter, Node-RED, InfluxDB, and AI modules delivers a seamless, efficient, and intelligent automation experience.

6.2.1 Test Case for System Functional Testing

Table 6.2 Test Case Table

TC ID	FR ID	Test Data	Expected Result	Actual Result	Status
TC01	FR-01, FR-02, FR-03	Action applied to device switches, thermostat, color picker, and fragrance button	Devices provide visual feedback: switches light up, thermostat value changes, color picker shows selected color, fragrance button activates	Devices respond as expected with visible feedback; thermostat updates value; color picker displays selected color	Pass, performed 15–30 times
TC02	FR-06, FR-07	User interacts with device switches, thermostat, color picker, and fragrance button in Flutter app	Flutter app publishes respective MQTT messages to Node-RED; device states update accordingly	MQTT messages sent successfully and devices updated in Node-RED	Pass, performed 20–30 times

TC03	FR-01, FR-02, FR-03	MQTT messages sent to Node-RED	Node-RED function nodes execute commands on the SVG smart home emulator, updating devices (AC, fragrance, doors, lights); real-time states reflected	Devices updated correctly in Node-RED as per received MQTT messages	Pass, performed 20–30 times
TC04	FR-10, FR-11	Changing thermostat, color, and fragrance values	Object-type data messages sent to InfluxDB buckets: temperature (with weather info) and updated color/fragrance	Data successfully stored in respective InfluxDB buckets	Pass, performed 30–40 times
TC05	FR-04, FR-05	InfluxDB data for temperature, color, and fragrance	Python script retrieves data, performs preprocessing, trains model, and generates predictions	AI module successfully retrieves data, trains model, and predicts	Pass, performed 15–25 times
TC06	FR-05, FR-09, FR-11, FR-12	Predicted data for temperature, color, and fragrance	Predicted data stored in InfluxDB; Flutter app retrieves temperature for initial setup	Predicted data stored in bucket and Flutter app successfully retrieves initial temperature	Pass, performed 10–20 times
TC07	FR-01, FR-	Dashboard access in Node-RED UI; user	Dashboard correctly displays real-time device states and logs	All devices and sensor states displayed correctly	Pass, performed 20–30

	02	views device states and data history		in Node-RED dashboard	times
TC08	FR-04, FR-05, FR-08	Adaptive control tested: system auto-adjusts thermostat based on AI prediction	System automatically applies control commands; changes logged in InfluxDB	Thermostat adjusted automatically; fragrance activated via AI predictions; logs confirmed in InfluxDB	Pass, performed 10–15 times

6.3 Project Challenges

One significant challenge encountered during the development of the system was related to the SVG-based graphical user interface in Node-RED. While the SVG graphics effectively represented rooms such as the living room, dining room, and bedroom, a critical limitation emerged when switching between tabs. Specifically, any state or modification applied to devices in the living room (for example, turning on lights or adjusting the air conditioner) would reset to its default state once the user navigated to another tab, such as the dining room or bedroom.

This behavior disrupted the intended functionality of persistent state management across different rooms. As a result, the system's emulation did not fully reflect real-world smart home operations, where device states should remain consistent regardless of user navigation between different interfaces**.**

6.4 Objectives Evaluation

The project set out three key objectives, each addressing the challenges of static automation, limited adaptability, and reduced efficiency in conventional smart home systems. All three objectives have been successfully achieved as outlined below:

First Objective: To emulate a home environment using Node-RED and SVG graphics for temperature regulation, lighting management, and air purification with relevant sensors. This objective is accomplished through the implementation of **FR-01, FR-02, and FR-03**, which provide the visual interface, real-time device state monitoring, and interactive device control. The functionality of these FRs has been validated by **TC01, TC02, and TC03**, confirming that the Node-RED emulator correctly visualizes devices, responds to user interaction, and executes MQTT commands to update device states in real time. A fully functional Node-RED environment was created with SVG graphics to represent different rooms, enabling visualization of devices such as lights, air conditioners, and air purifiers. Real-time environmental monitoring (e.g., temperature and air quality) was integrated with sensors, and the visualization dynamically reflected device states. This provided a controllable simulation platform where adaptive automation scenarios could be effectively tested.

Second Objective: To develop an adaptive and responsive control system for seamless interaction with the smart home system using Node-RED and Flutter. This objective is accomplished through **FR-06 and FR-07**, which provide user control of devices and enable the publishing of user-side data via MQTT for real-time updates. The successful implementation of these FRs has been validated by **TC01 and TC02**, which confirm that user interactions through the Flutter app are properly transmitted and reflected in the Node-RED emulator, ensuring a responsive and adaptive control system. A Flutter-based mobile app was developed to enable real-time control and monitoring of the simulated home environment. Through integration with Node-RED, device states and environmental conditions could be dynamically updated and controlled, ensuring responsiveness to user input. This eliminated the rigidity of static automation, allowing for more flexible and user-centered interactions with the system.

Third Objective: To implement real-time decision-making for device control using AI-based analysis of environmental and user behavior data within the smart home simulation. This objective is accomplished through **FR-04, FR-05, and FR-11**, which cover AI prediction, API integration for mobile access, and storage of predicted data in InfluxDB. The functionality of these FRs has been validated by **TC05 and TC06**, which confirm that the AI module can retrieve data, perform predictions, and store results for use by the mobile app, enabling predictive and context-aware device control. AI modules were integrated into the system to analyze environmental data and user interactions, enabling predictive and context-aware decision-making. Devices such as air purifiers and lighting could adjust automatically based on AI-driven analysis, enhancing efficiency and comfort. This marked a shift from reactive, rule-based control to proactive, intelligent automation.

Summary:
In conclusion, all three project objectives have been met. The successful implementation of a simulated environment, adaptive control through mobile and backend integration, and AI-driven decision-making demonstrates the effectiveness of the system in addressing the challenges of conventional smart home automation. The outcomes confirm that the project has advanced toward building a more intelligent, efficient, and user-centric smart home solution.

6.5 Concluding Remark

This chapter presented the evaluation of the smart home system in terms of performance, functionality, and objectives fulfillment. System performance testing demonstrated that device control, response time, MQTT communication, and database operations were reliable and efficient, with only minor delays observed in thermostat adjustments and AI-driven predictions. Functional testing through structured test cases validated the correctness of the system's behavior across multiple scenarios, confirming its ability to handle device interactions, data storage, and AI-based predictions consistently.

The challenges encountered, such as the limitation of SVG-based visualization resetting device states when switching between rooms, highlighted areas for further refinement. Despite these constraints, the project successfully achieved its intended objectives, demonstrating the effectiveness of adaptive automation in enhancing smart home environments.

In summary, the results affirm that the system provides a stable, responsive, and intelligent platform for smart home control and monitoring. This lays a strong foundation for future enhancements, such as improving interface persistence, optimizing AI response times, and extending the system for real-world deployment beyond the simulated environment.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

This study has successfully designed, implemented, and evaluated an adaptive smart home control and monitoring system that addresses the limitations of static automation in conventional platforms. Beginning with the identification of the research problem, the project emphasized the need for a user-centric, AI-driven approach to improve personalization, energy efficiency, and overall comfort in smart home environments.

Through the literature review, suitable technologies were identified—Flutter for mobile interaction, Node-RED for middleware integration, Python for predictive intelligence, and InfluxDB for efficient time-series data handling. The system design established a multi-layer architecture integrating communication, data management, and user interaction, while the implementation phase translated these designs into a fully functional prototype capable of real-time device control, monitoring, and predictive automation.

The evaluation confirmed the system's effectiveness in meeting its objectives. Functional and performance testing demonstrated reliable device communication, accurate data handling, and consistent AI-driven predictions. Despite challenges such as limitations in SVG-based visualization persistence, the project proved that adaptive automation can enhance smart home functionality beyond static, rule-based approaches.

In conclusion, this project contributes a working prototype that demonstrates how intelligent middleware, responsive mobile apps, and lightweight machine learning models can be combined to create a scalable and adaptive smart home system. The findings affirm that adaptive automation not only improves user comfort and system responsiveness but also establishes a strong foundation for future research in areas such as persistent state management, AI optimization, and real-world deployment.

7.2 Recommendations for Future Work

While the project has demonstrated the feasibility and effectiveness of adaptive automation in smart home systems, two possible areas remain open for refinement and extension:

1. **Development of a Smart Home Demonstrator**
    - Setting up a real house environment for system deployment is costly and impractical at the prototype stage. As a feasible alternative, future work could focus on building a dedicated smart home demonstrator or miniaturized model house equipped with sensors, actuators, and a visualization system. This demonstrator would allow stakeholders to interact with the system in a tangible way, providing a realistic experience of adaptive automation without the high expenses of a full-scale setup.
2. **Enhancement of AI Capabilities**
    - While this project successfully implemented predictive automation using a RandomForest regression model, future improvements should focus on enhancing AI capabilities. This could include adopting reinforcement learning, deep learning, or hybrid models to achieve more accurate predictions and context-aware decision-making. Additionally, integrating more diverse datasets—such as user routines, occupancy trends, or external data sources like energy tariffs and weather forecasts—would allow the system to deliver smarter, highly personalized, and energy-efficient automation.

REFERENCES

[1] M. Chen, S. Mao, and Y. Liu, “Big Data: A Survey,” *Mobile Networks and Applications*, vol. 19, pp. 171–209, 2014.

[2] M. Alaa, A. A. Zaidan, B. B. Zaidan, M. Talal, and M. L. M. Kiah, "A review of smart home applications based on Internet of Things," *Journal of Network and Computer Applications*, vol. 97, pp. 48–65, 2017.

[3] S. Rashidi and D. Cook, “Keeping the Resident in the Loop: Adapting the Smart Home to the User,” *IEEE Transactions on Systems, Man, and Cybernetics*, Part A, vol. 39, no. 4, pp. 836–848, 2009.

[4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[5] T. A. Khoa, T. H. Phuc, and T. H. Dinh, "A survey on context-aware smart home systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 11, pp. 4539–4558, 2020.

[6] N. Karakus, B. Kantarci, and H. T. Mouftah, “Node-RED Based Middleware for IoT Applications,” *IEEE Access*, vol. 8, pp. 158009–158024, 2020.

[9] S. Kumar and S. K. Singh, "A review on smart home present state and challenges: linked to context-awareness Internet of Things (IoT)," *Wireless Personal Communications*, vol. 109, pp. 3353–3376, 2019.

[10] N. Balta-Ozkan, R. Davidson, M. Bicket, and L. Whitmarsh, "Social barriers to the adoption of smart homes," *Energy Policy*, vol. 63, pp. 363–374, 2013.

[11] D. Marikyan, S. Papagiannidis, and E. Alamanos, "A systematic review of the smart home literature: A user perspective," *Technological Forecasting and Social Change*, vol. 138, pp. 139–154, 2019.

[12] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, "Machine learning in IoT security: Current solutions and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1686–1721, 2019.

[13] Y. Zhang, L. T. Yang, J. Chen, and P. Li, "A survey of energy-efficient and secure techniques for emerging mobile cloud computing systems," *IEEE Systems Journal*, vol. 11, no. 2, pp. 1157–1169, 2017.

[17] Google, "Flutter Documentation," 2020. [Online]. Available: https://flutter.dev (retrieved April 30, 2025).

[18] I. Al-Mashaqbeh, et al., "Predictive Modeling for Smart Home Automation Using Random Forest," *Journal of IoT Research*, vol. 9, no. 2, pp. 45–60, 2021.

[19] L. Chen, et al., "Time-Series Databases for IoT Applications: A Review," *IEEE Access*, vol. 8, pp. 156–170, 2020.

[20] M. Karakus, et al., "Node-RED as a Middleware for IoT Applications," *Sensors*, vol. 20, no. 7, p. 1987, 2020.

[21] M. Moghadam, et al., "Mobile Frameworks and Real-Time IoT Applications: Flutter in Smart Homes," *Journal of Embedded Systems*, vol. 14, no. 1, pp. 12–25, 2022.

[22] K. O'Leary, et al., "Flow-Based Programming for IoT Systems," *Journal of Systems Architecture*, vol. 97, pp. 45–59, 2019.

[23] A. Saini and R. Kumar, "Cross-Platform Mobile Development for IoT Applications: Flutter vs React Native," *International Journal of Mobile Computing*, vol. 12, no. 3, pp. 12–24, 2021.

[24] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Python Software Foundation, 2009.

[25] J. Davis and S. McCreary, *Time-Series Databases: New Ways to Store and Access Data*. O'Reilly Media, 2018.

[26] U. Bajwa, et al., "Comparative Study of Smart Home Automation Platforms: Home Assistant vs OpenHAB," *International Journal of Smart Home Technologies*, vol. 15, no. 3, pp. 33–48, 2021.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

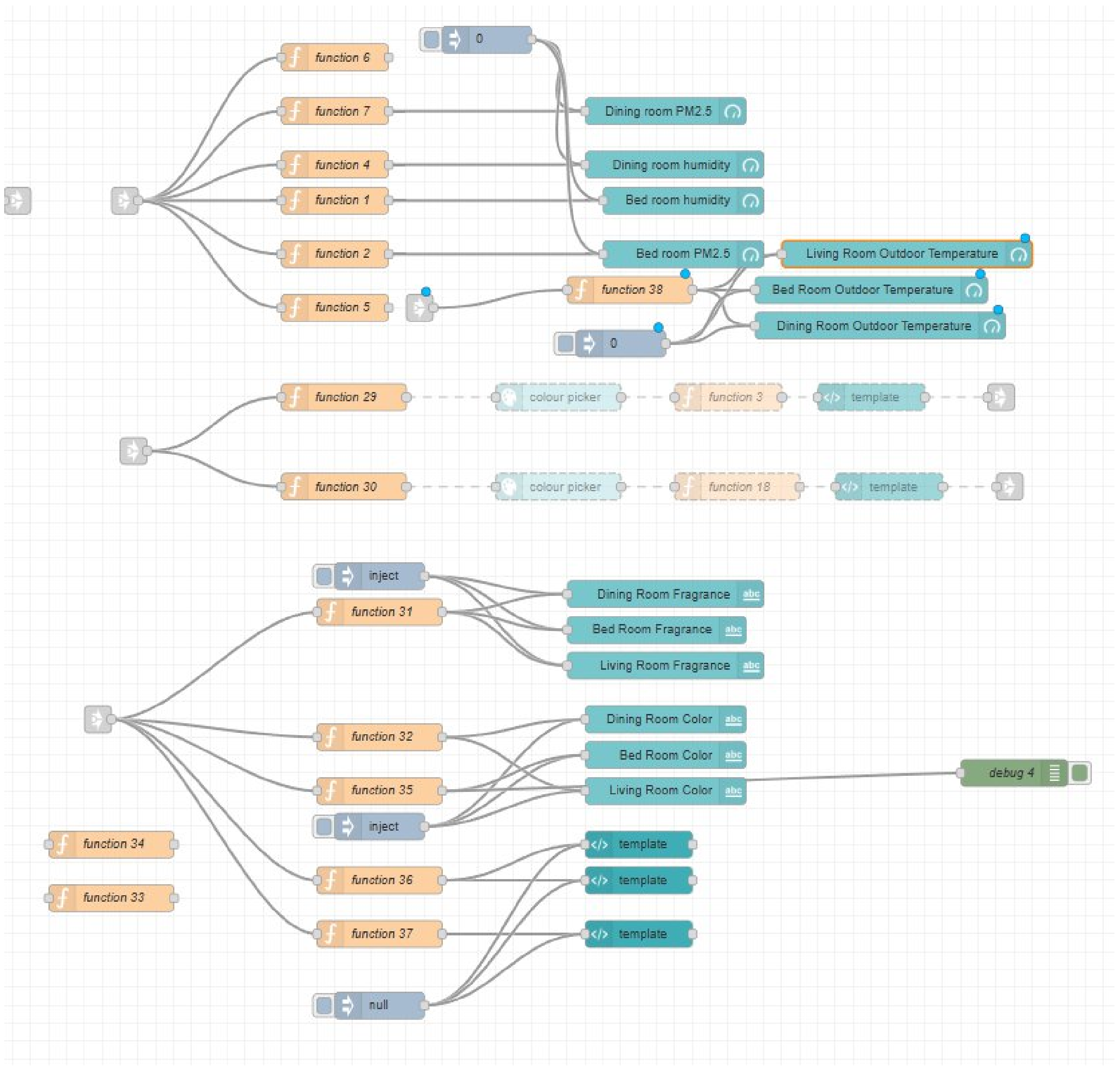
[28] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, San Francisco,

CA, USA, 2016, pp. 785–794.

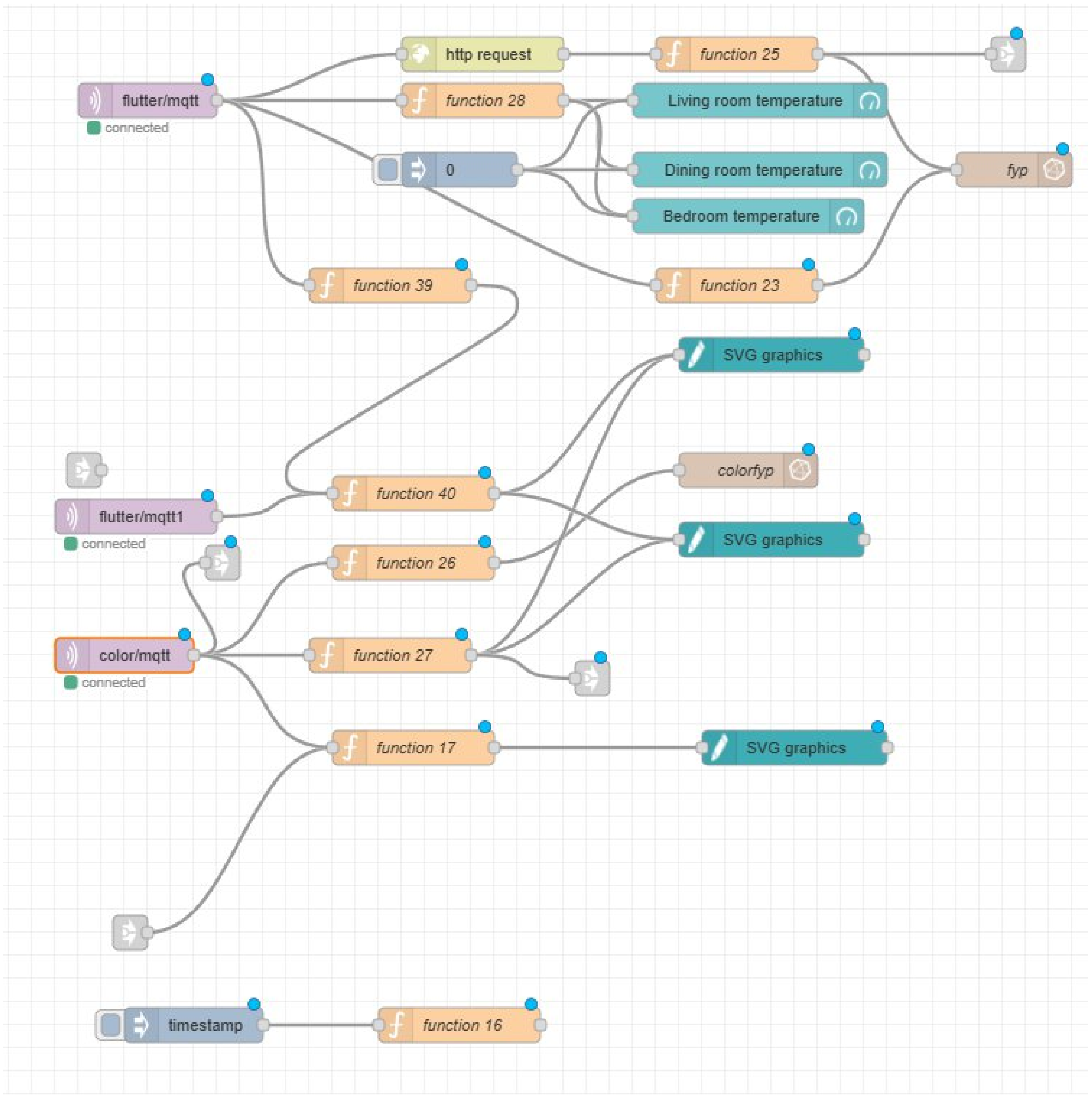
[29] R. Ihaka and R. Gentleman, "R: A language for data analysis and graphics," ***J. Comput. Graph. Stat.*****, vol. 5, no. 3, pp. 299–314, 1996.**

APPENDICES

APPENDIX A: Overall System Node-RED Flow 1



APPENDIX B: Overall System Node-RED Flow 2



APPENDIX C: Code Snippet of MQTT service for topic 'flutter/mqtt1

```
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';
class MQTTService {
 final String broker = 'broker.hivemq.com';
 final int port = 1883;
 final String clientId =
     'flutter_publisher_${DateTime.now().millisecondsSinceEpoch}';
 final String topic = 'flutter/mqtt1';
 late MqttServerClient client;
 Future<void> connect() async {
   client = MqttServerClient(broker, clientId);
   client.port = port;
   client.logging(on: false);
   client.keepAlivePeriod = 20;
   client.onDisconnected = onDisconnected;
   client.onConnected = onConnected;
   final connMessage = MqttConnectMessage()
       .withClientIdentifier(clientId)
       .startClean()
       .withWillQos(MqttQos.atMostOnce);
   client.connectionMessage = connMessage;
   try {
     await client.connect();
   } catch (e) {
     print('MQTT: Connection failed - $e');
     disconnect();
     return;
   }
   if (client.connectionStatus!.state == MqttConnectionState.connected) {
     print('MQTT: Connected');
   } else {
      print(
        'MQTT: Failed to connect with status ${client.connectionStatus!.state}',
      );
      disconnect();
    }
  }
  void publish(String message) {
    if (client.connectionStatus?.state != MqttConnectionState.connected) {
      print('MQTT: Client not connected. Call connect() first.');
      return;
    }
    final builder = MqttClientPayloadBuilder();
    builder.addString(message);
    client.publishMessage(topic, MqttQos.atMostOnce, builder.payload!);
    print('MQTT: Published message: $message');
  }
  void disconnect() {
    client.disconnect();
    print('MQTT: Disconnected');
  }
  void onConnected() {
    print('MQTT: Connected');
  }
  void onDisconnected() {
    print('MQTT: Disconnected');
  }
}
```

APPENDIX D: Code Snippet for Thermostat Slider with MQTT Integration

```
CircularTempSliderWidget(
  value: currentTemp,
  minValue: 16,
  maxValue: 30,
  isCelsius: true,
  onChanged: (newValue) async {
    setState(() => currentTemp = newValue);
    mqttService.publish("thermostat:$newValue");
  },
)
```

APPENDIX E: Code Snippet for API Synchronization

```
Future<void> fetchThermostatFromApi() async {
  final response = await http.get(Uri.parse("http://<server-ip>:5000/latest"));
  if (response.statusCode == 200) {
    final List<dynamic> jsonResponse = json.decode(response.body);
    final latestRecord = jsonResponse.first;
    final apiValue = double.tryParse(latestRecord['value'].toString());
    if (apiValue != null) {
      setState(() {
        FFAppState().updateThermostatStruct(
          (e) => e..temperatureFahrenheit = (apiValue * 9 / 5) + 32,
        );
      });
    }
  }
}
```

APPENDIX F: Code Snippet for Debounced MQTT Publishing

```
void publishTemperature(double value) {
  if (_mqttService.client.connectionStatus?.state != MqttConnectionState.connected) return;

  final celsiusValue = (value - 32) * 5 / 9;
  _debounceTimer?.cancel();
  _debounceTimer = Timer(Duration(milliseconds: 1500), () {
    final publishMessage = 'room temperature is $celsiusValue';
    _mqttService.publish(publishMessage);
  });
          }
```

APPENDIX G: Code Snippet for Thermostat Slider Device Control

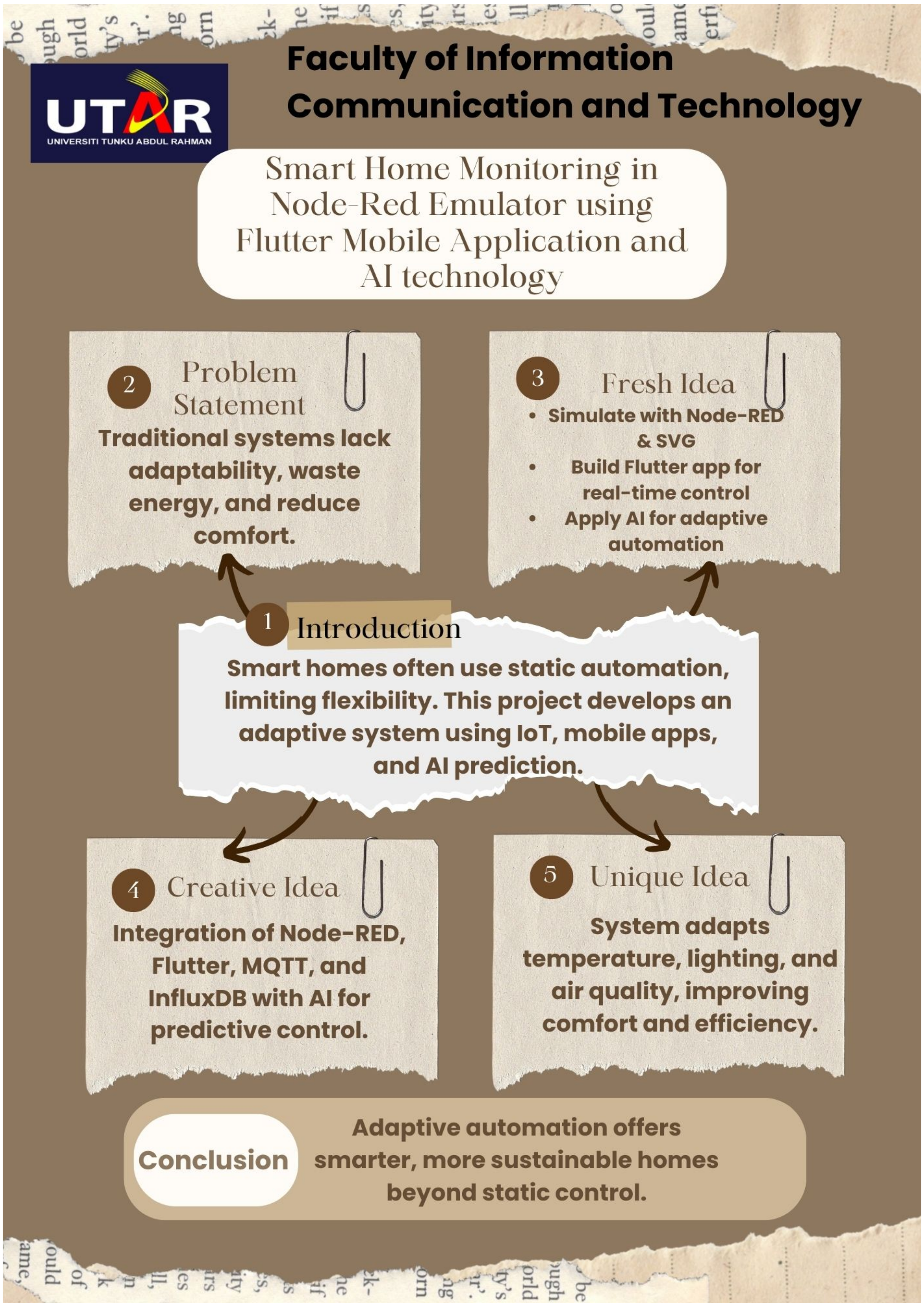
```
Slider(
  min: 60,
  max: 85,
  divisions: 25,
  value: _currentTemp,
  label: '${_currentTemp.round()}°F',
  onChanged: (newValue) {
    setState(() => _currentTemp = newValue);
    mqttService.publish('home/thermostat/set', newValue.toString());
  },
)
```

APPENDIX H: Code Snippet for Python Machine Learning Script

```
from influxdb_client import InfluxDBClient, Point, WriteOptions
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
from flask import Flask, jsonify
from datetime import datetime

# --- InfluxDB Setup ---
INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "<your-token>"
ORG = "<your-org>"
USER_BUCKET = "user_input"
PREDICT_BUCKET = "predicted_data"

# --- Fetch data ---
with InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=ORG) as client:
    query = f'from(bucket:"{USER_BUCKET}") |> range(start: -7d)'
    tables = client.query_api().query(query, org=ORG)
    data = []
    for table in tables:
        for record in table.records:
            data.append({
                "time": record.get_time(),
                "temp": record.get_value(),
                "field": record.get_field()
            })
df = pd.DataFrame(data)

# --- Train / Apply Model ---
model = RandomForestRegressor()
X = df[['temp']]  # Example feature
y = df['temp']    # Target
```

```
model.fit(X, y)
predicted = model.predict([[25]])  # Example prediction

# --- Write Prediction to InfluxDB ---
with InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=ORG) as client:
    write_api = client.write_api(write_options=WriteOptions(batch_size=1))
    point = Point("predicted_room_temp") \
        .field("temperature", float(predicted[0])) \
        .time(datetime.utcnow())
    write_api.write(bucket=PREDICT_BUCKET, org=ORG, record=point)

# --- Flask API to provide prediction ---
app = Flask(__name__)

@app.route("/latest_prediction")
def latest_prediction():
    with InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=ORG) as client:
        query = f'from(bucket:"{PREDICT_BUCKET}") |> range(start: -1h)'
        tables = client.query_api().query(query, org=ORG)
        result = [{"time": r.get_time().isoformat(), "value": r.get_value()}
                  for t in tables for r in t.records]
    return jsonify(result)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Poster



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Faculty of Information Communication and Technology

Smart Home Monitoring in Node-Red Emulator using Flutter Mobile Application and AI technology

2 Problem Statement

Traditional systems lack adaptability, waste energy, and reduce comfort.

3 Fresh Idea

- **Simulate with Node-RED & SVG**
- **Build Flutter app for real-time control**
- **Apply AI for adaptive automation**

1 Introduction

Smart homes often use static automation, limiting flexibility. This project develops an adaptive system using IoT, mobile apps, and AI prediction.

4 Creative Idea

Integration of Node-RED, Flutter, MQTT, and InfluxDB with AI for predictive control.

5 Unique Idea

System adapts temperature, lighting, and air quality, improving comfort and efficiency.

Conclusion

Adaptive automation offers smarter, more sustainable homes beyond static control.