TITLE OF THE PROJECT

SMART HOME AUTOMATION SYSTEM USING IOT

NAME OF CANDIDATE
WILLIAM CHAGOMA NGWIRA

REGISTRATION NUMBER

18321055006

GUIDE

MS. FANNY CHATOLA

Project Report

Submitted

In partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING IN COMPUTER SCIENCE

June, 2025

DMI ST JOHN THE BAPTIST UNIVERSITY

LILONGWE, MALAWI

CERTIFICATE OF THE GUIDE

This is to certify that the project work entitled: **SMART HOME AUTOMATION SYSTEM USING IOT** is a Bonafide work of **MR. WILLIAM CHAGOMA NGWIRA** Registration No: **18321055006** in partial fulfillment for the award of the Degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE, DMI - ST JOHN THE BAPTIST UNIVERSITY** under my guidance. This Thesis work is original and not submitted earlier for the award of any degree elsewhere.

Signature of the Student **Signature of the Guide**

DECLARATION BY THE CANDIDATE

I **WILLIAM CHAGOMA NGWIRA** hereby declare that this project report **SMART HOME AUTOMATION SYSTEM USING WOT** submitted to **DMI ST JOHN THE BAPTIST UNIVERSITY** in the partial fulfillment of requirements for the award of the degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE** is a record of the original work done by me under the supervision of **MS.FANNY CHATOLA**.

Enrolment No. :

Register No : 18321055006

Date : 19 JUNE 2025

Signature :

PROFORMA FOR APPROVAL OF PROJECT PROPOSAL

Proposed Project Team: William Chagoma Ngwira

S.No.	Reg. No.	Name of the students	Semester	Branch
1	18321055006	WILLIAM CHAGOMANGWIRA	X	B.E

Title of the Project:	SMART HOME AUTOMATION SYSTEM USING IOT
Subject Area:	WEB OF THINGS/INTERNET OF THINGS
Name of the Guide:	MS. FANNY CHATOLA
Designation:	Lecturer –II
Address with Phone No.:	DMI SJBU, +265 99 10 15 730
Office:	DMI – SJBU
Residence:	LILONGWE, AREA 22
No. of projects & students currently working under the Guide:	45

Signature of the Student **Signature of the Guide**

Date..... with seal

N.B.: Please do not forget to enclose the synopsis of the project and the Bio-data of the Guide. In case the complete and signed Bio-Data of the Guide is not enclosed, the proposal will not be entertained

For Office Use only:

SYNOPSIS	APPROVED		NOT APPROVED	
GUIDE	APPROVED		NOT APPROVED	

Comments / Suggestions for reformulation of the Project.

Date..... **SIGNATURE OF THE HOD**

BIO-DATA OF THE PROPOSED GUIDE FOR PROJECT WORK

1. PERSONAL INFORMATION

Name	: MS FANNY CHATOLA
Date of Birth & Age	: 13.12.1997:
Sex	: FEMALE
Academic Qualification	: MSC, BSC
Official Address	: P.O.BOX 2398.
Phone No. and Fax.	: +265 991 01 57 30
Residential Address	: AREA 22
e-mail id	: fionachatola@gmail.com

2. DETAILS OF EMPLOYMENT

Designation : LECTURER II

Field of Specialization : COMPUTER SCIENCE

Teaching Experience (in years) : 4

Industrial Experience (in years) : 4 Particulars of contribution / experience in the field of specialization:

No. of Projects guided : 45

I MS FANNY CHATOLA do hereby accept to guide **WILLIAM CHAGOMA NGWIRA** student of the **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE** program of the **DMI - ST JOHN THE BAPTIST UNIVERSITY.**

Signature of the Students **Signature of the Guide with Seal**

DMI-ST. JOHN THE BAPTIST UNIVERSITY

LILONGWE, MALAWI

BONAFIDE CERTIFICATE

Register No: 18321055006

Certified that this is Bonafide record of work done in **SMART HOME AUTOMATION SYSTEM USING WOT by Mr. WILLIAM CHAGOMA NGWIRA** of the **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE at DMI ST JOHN THE BAPTIST UNIVERSITY**

During the 2024/2025 academic year

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

From the depths of my heart and every fiber of my being, I offer my deepest and sincerest gratitude to the ALMIGHTY for His divine presence, infinite grace, and countless blessings, which have sustained and guided me throughout every step of this journey. Without His favor, the successful completion of this project would not have been possible. I remain profoundly indebted to our beloved Founder and Chancellor, **Rev. Fr. Dr. J.E. Arul Raj**, for his visionary leadership and for providing the enabling environment and all necessary facilities through our esteemed institution. My heartfelt thanks also extend to **Dr. T. X. A. Ananthi**, President of the University Council, and **Dr. Ignatius A. Herman**, Director of Education, DMI Group of Institutions Malawi, for offering me this golden opportunity to undertake and complete this project with excellence. I am equally grateful to Professor Ambrose, our respected Vice Chancellor, **Rev. Sr. Pradeeba**, Management Representative, and **Ms. Agnes Msonda**, Vice Principal Academic, for their continuous encouragement, wise counsel, and support, which motivated me to stay committed and focused. I extend special appreciation to my project guide, Ms. Fanny Chatola, for her patience, mentorship, and insightful feedback that shaped the quality and direction of this work. I also thank all the dedicated faculty and staff members of my department for their assistance and moral support during this academic endeavor. Finally, I express my heartfelt appreciation to my entire family, whose unwavering love, encouragement, sacrifices, and prayers have been my foundation of strength, inspiration, and determination throughout this journey. This achievement is not mine alone—it belongs to all who stood by me and believed in me.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACRONYMS	**DEFINITION**
IOT	Internet Of Things
WOT	Web of Things
MQTT	Mosquitto
STT	Speech to Text

TABLE OF CONTENTS

ABSTRACT

The proposed Smart Home Automation System integrates IoT (Internet of Things or Web of Things) to bring intelligence and convenience into the home environment. Leveraging IoT technology, the system integrates various smart devices and sensors to automate household tasks such as lighting control, fan control, and voice speech recogniton. The system allows users to remotely control and monitor their home appliances and security systems via a smartphone offering real-time feedback and control. At the heart of the system is a central microcontroller (Raspberry Pi), which connects to the cloud and communicates with different smart devices using protocols like Wi-Fi or Bluetooth. The system gathers environmental data using sensors (e.g., temperature, light, motion) and triggers specific actions based on predefined user settings or real-time conditions. For instance, lights can be turned off when no one is in the room, or the HVAC system can adjust temperature settings based on current weather conditions. In addition to automating routine tasks, the system can enhance home security through integrated motion detectors, smart locks, and cameras. It can notify homeowners of potential security breaches and allow for remote monitoring of live video feeds. Energy efficiency is also a critical feature, as the system optimizes electricity usage by regulating lights, heating, and cooling based on occupancy or schedule, which helps reduce electricity costs and environmental impact. This project demonstrates the feasibility of creating a smart home using IoT technologies and highlights the potential for scalability as more devices are integrated into the system. Through the application of IoT, this system not only improves comfort and convenience but also addresses growing concerns about energy efficiency and home security.

CHAPTER I
INTRODUCTION

1.1 BACKGROUND OF STUDY

The increasing advancement in digital technology, particularly the emergence of the Internet of Things (IoT), has drastically transformed how we interact with our environment. IoT refers to the network of physical devices embedded with sensors, software, and connectivity capabilities, enabling these devices to collect and exchange data. One of the major applications of IoT is in smart home automation, where various household appliances and systems are automated for better control, energy efficiency, and user convenience. This project focuses on implementing a Smart Lighting System using a Raspberry Pi as the control hub, a Flutter-built mobile application as the user interface, and MQTT as the communication protocol. This seamless integration of hardware and software allows users to remotely switch on or off a light bulb through a mobile app, providing a practical and real-world example of IoT. The aim of this documentation is to provide a detailed explanation of the development process, system architecture, and functionalities of this IoT-based Smart Lighting System. The Raspberry Pi, acting as the main controller, interfaces with a relay switch to control the bulb. The Flutter mobile app, in turn, sends MQTT messages to the Pi for execution. With its simple yet scalable design, this system is an ideal foundation for further enhancements in smart home automation.

1.2 OBJECTIVES

The objective is to develop an IOT or WOT based system that will be able to perform the following tasks in the home industry:

1.2.1 Implement an IoT solution that can be used for real-world smart home automation tasks.

The Core objective is to practically implement an Internet of Things (IoT) solution that addresses real-world smart home automation needs. This involves designing and deploying a functional system that allows users to remotely monitor and control home appliances such as lights, fans, and other electrical devices through connected technology. The aim is to demonstrate how IoT can enhance convenience,

energy efficiency, and security within a household by integrating sensors, microcontrollers, and communication protocols into a seamless, user-friendly automation system.

1.2.2 Develop a simple yet powerful mobile app for remote control of lighting system.

The project aims to develop a simple yet powerful mobile application that enables remote control of the lighting system within a smart home environment. The app will serve as an intuitive user interface, allowing users to switch lights on or off, monitor their status, and potentially schedule operations from any location. The focus is on creating a lightweight, responsive, and user-friendly application that communicates effectively with the IoT hardware, offering both functionality and convenience to enhance everyday living.

1.2.3 Utilize the MQTT protocol for lightweight, real-time communication between the app and the Raspberry Pi.

This objective aims to implement the MQTT (Message Queuing Telemetry Transport) protocol to establish efficient, lightweight, and real-time communication between the mobile application and the Raspberry Pi. MQTT is a highly reliable protocol widely used in IoT systems due to its minimal overhead and publish-subscribe architecture, which supports fast and scalable data transmission. By leveraging MQTT, the project ensures seamless interaction between the user interface and the hardware components, enabling prompt control commands and status updates essential for a responsive and dependable smart home automation experience.

1.2.4 To create a modular system that can be expanded to include additional sensors, appliances, or AI-based decision-making

To design a modular and scalable system architecture that allows for future expansion and integration of additional components such as sensors, appliances, or AI-based decision-making modules. By adopting a flexible design approach, the system can easily accommodate evolving smart home needs, supporting features like environmental monitoring, energy optimization, or intelligent automation. This ensures the solution remains adaptable, future-proof, and capable of supporting advanced functionalities as technology and user requirements evolve.

1.2.5 To explore open-source technologies for cost-effective, scalable, and secure automation projects.

The System aims to explore and leverage open-source technologies to develop a cost-effective, scalable, and secure smart home automation system. By utilizing widely supported open-source tools and platforms, the project not only reduces development costs but also promotes transparency, flexibility, and community-driven innovation. This approach enables the creation of a robust solution that can be easily maintained, enhanced, and scaled, while ensuring security and reliability in real world applications.

1.3 SYSTEM DESCRIPTION

The proposed smart home automation system is composed of three primary components: a Raspberry Pi microcontroller, a mobile application developed using Flutter, and a lighting bulb connected via a relay module. The system enables users to control the lighting either through the mobile application or via integrated voice commands, offering a hands-free, user-friendly experience. The mobile app serves as the central user interface, allowing users to send control commands that are transmitted to an MQTT broker. The Raspberry Pi, which is subscribed to specific topics on the broker, receives these commands and responds by toggling its GPIO pins to activate or deactivate the relay, thereby switching the lighting bulb on or off as instructed. In addition to manual control through the app, the system is equipped with voice integration functionality, enabling users to issue voice commands to perform the same actions seamlessly. This enhances accessibility and convenience, particularly in situations where physical interaction with the app may be impractical. Furthermore, the system supports two-way communication by sending real-time feedback messages from the Raspberry Pi back to the mobile application, updating the user on the current status of the light (e.g., ON or OFF). This architecture is designed for flexibility, responsiveness, and scalability. The use of the MQTT protocol ensures lightweight and efficient message delivery, even in low-bandwidth environments, while the intuitive Flutter-based app interface provides a smooth and engaging user experience. Overall, the system exemplifies a practical and expandable IoT-based solution for modern smart home automation

1.4 LITERATURE REVIEW

This section presents a review of selected studies relevant to the development of smart home automation systems, highlighting key contributions in the field of IoT-based home control. In 2013, Baraka et al. introduced a low-cost Arduino and Android-based energy-efficient home automation system featuring smart task scheduling. Their work offered a comprehensive examination of the capabilities of affordable microcontrollers and mobile platforms, emphasizing energy conservation and ease of implementation.

In 2015, Bhide and Wagh proposed an intelligent, self-learning system for home automation using IoT technologies. Their framework focused on adaptability, presenting an objective methodology for selecting the most appropriate IoT platforms based on varying use cases, which underscored the importance of system flexibility and user-specific customization.

The study conducted in 2016 by Huang and Tseng developed a predictive smart home system that integrated heterogeneous networks with cloud computing. Their system employed a deep neural network-based prediction algorithm designed to enhance responsiveness and efficiency, showcasing the potential of artificial intelligence in real-time smart home environments.

More recently, Mehra (2022) demonstrated a Raspberry Pi–Flask web-based interface for smart home control, emphasizing affordability, scalability, and sensor integration . Meanwhile, MDPI researchers (2023) designed a secure smart plug capable of real-time energy monitoring and relay control via voice commands and MQTT over Wi-Fi mesh, underscoring security and efficient communication in voice-enhanced systems. In the AI domain, a June 2025 Xiv paper introduced an innovative offline speech recognition model for low-latency, decentralized voice control, addressing energy consumption and privacy concerns.

These studies collectively demonstrate the evolution of smart home automation from basic control systems to intelligent, predictive models, laying the foundation upon which the present project builds by integrating real-time control, mobile and voice interfaces, and scalable IoT architecture.

1.4.1 Summary Review

The reviewed systems each contribute valuable features—ranging from affordability and smart scheduling to predictive modeling and voice integration—but often fall short in scalability, user-friendliness, or reliance on stable internet connections. In contrast, this project presents a more practical and well-rounded solution by combining a lightweight Flutter mobile app, MQTT-based real-time

communication, and voice control through a modular Raspberry Pi setup. Unlike earlier works, it offers flexibility, responsiveness, and ease of use, making it more adaptable and user-focused for real-world smart home automation.

CHAPTER II
SYSTEM ANALYSIS

2.1 INTRODUCTION

In recent years, the concept of smart homes has gained significant traction, driven by the rapid evolution of Internet of Things (IoT) technologies and increasing demand for convenience, energy efficiency, and enhanced security. A smart home integrates various electronic devices, appliances, and sensors into a unified system that can be monitored and controlled remotely or automatically. This integration not only improves user comfort but also optimizes energy usage and enhances the overall safety of the home environment. As IoT devices become more affordable and accessible, the opportunity to transform traditional homes into intelligent, connected spaces is more achievable than ever. With mobile applications, cloud computing, and lightweight communication protocols like MQTT, users can now interact with their home appliances in real time, whether they are at home or away. This project proposes the development of a smart home automation system that utilizes a Raspberry Pi, relay-controlled appliances, a Flutter-based mobile app, and voice command integration to deliver a flexible, scalable, and efficient solution for modern home management. The system leverages the power of IoT to streamline the interaction between users and their home devices, making day-to-day living more convenient, responsive, and intelligent.

2.2 PROBLEM DEFINITION

As homes become increasingly populated with electronic devices, managing these appliances manually has proven to be inefficient, time-consuming, and prone to human error. Traditional systems often operate independently—lighting, heating, cooling, and security systems lack coordination, resulting in increased energy consumption and operational inefficiencies. Homeowners commonly encounter issues such as leaving lights or appliances on unintentionally, forgetting to adjust systems according to environmental changes, or facing security concerns when away from home. Existing automation solutions are often fragmented and costly, requiring different applications or platforms for each device, which complicates control and reduces user convenience. Furthermore, many systems lack real-time responsiveness and integration with intuitive technologies such as voice commands, limiting accessibility for various users. These limitations highlight the need for a comprehensive solution that

not only centralizes control but also offers remote accessibility, real-time feedback, energy optimization, and scalability. The proposed IoT-based Smart Home Automation System addresses these challenges by providing a unified, intelligent platform that allows users to control lighting via a mobile app and voice commands, with real-time updates and communication facilitated through MQTT. This solution not only reduces energy wastage and improves security but also enhances the quality of life by simplifying the way people interact with their home environments.

2.3 EXISTING SYSTEM

Many commercial home automation systems are available in the market, such as Philips Hue compatible lighting systems. However, these systems are often expensive, rely heavily on cloud-based infrastructure, and may require professional installation. Moreover, many users are concerned about privacy and the long-term costs associated with these platforms. On the other hand,we have alexa which is offered by amazon. While these alternatives provide flexibility, they can be technically challenging for beginners and lack robust app integration. DIY enthusiasts and developers often resort to using Arduino or ESP8266-based solutions for automation. This project seeks to fill this gap by using a Raspberry Pi and Flutter app, offering both ease of use and technical flexibility

2.4 FEASIBILITY STUDY

The Internet of Things (IoT) provides a highly feasible solution for implementing smart home automation due to its ability to connect devices and enable real-time communication between them and the cloud. In this project, IoT is used to facilitate seamless interaction between a mobile app, a Raspberry Pi, and a relay-controlled lighting system. Its lightweight protocols, like MQTT, allow for efficient data transfer, automation, and remote control. The system is cost-effective, scalable, and easy to expand with additional features such as voice commands and environmental sensors. Overall, IoT ensures the project is both technically practical and economically viable for modern home automation.

2.4.1 Executive Summary

The feasibility of this project was evaluated based on financial, operational, and technical perspectives. Financially, the components required are low-cost and readily available. Operationally,

the system is easy to use and maintain. Technically, the integration of MQTT, Flutter, and Raspberry Pi ensures a reliable and scalable system.

2.4.2 Finding and Recommendations

The findings from this project demonstrate that the proposed smart home automation system is not only functional but also highly applicable in real-world environments. The use of open-source technologies such as the Raspberry Pi, MQTT protocol, and Flutter framework contributes to a low-cost and flexible solution that is ideal for both educational settings and prototyping new ideas. The system proved to be reliable in controlling lighting through both mobile and voice interfaces, with real-time responsiveness and ease of use. Its modular design and scalability also make it adaptable for broader smart home applications. Based on these results, it is recommended that future projects explore the integration of additional devices such as ceiling fans, smart door locks, motion sensors, and temperature or humidity monitors. Incorporating these elements would enhance automation, improve energy efficiency, and expand the overall functionality of the system, making it even more robust and aligned with the evolving demands of modern smart homes.

2.5 PROPOSED SYSTEM

The proposed smart home automation system is a robust, modular, and scalable solution designed to bring real-time control and convenience into everyday household management. At its core, the system integrates a Flutter-based mobile application, an MQTT broker, and a Raspberry Pi microcontroller connected to a relay module. The mobile application provides an intuitive and user-friendly interface that allows users to remotely switch the lighting system ON or OFF from any location. When a command is issued through the app, it is published to the MQTT broker—a lightweight, efficient protocol ideal for IoT environments. The broker then forwards the message to the Raspberry Pi, which is subscribed to specific control topics. Upon receiving the message, the Pi interprets it and toggles the corresponding GPIO pin to activate or deactivate the connected lighting bulb via the relay module. This architecture not only ensures low-latency communication and efficient operation but also lays the foundation for a highly extensible system. The modular design allows for the seamless addition of more devices, such as fans, door locks, or environmental sensors, and supports the integration of more advanced control logic including automation schedules or AI-based decision-making. The use of MQTT further enhances the system’s performance in low-bandwidth environments, ensuring smooth

message delivery without significant overhead. Additionally, the system is designed with future improvements in mind, such as implementing user authentication and role-based access control for enhanced security. The combination of reliable hardware, efficient communication protocols, and a clean user interface makes the proposed system not only technically sound but also highly usable and practical for real-world deployment in both residential and educational settings.

2.6 SYSTEM OBJECTIVE

The primary objective of the proposed smart home automation system is to provide an energy-efficient, user-friendly solution for controlling household devices remotely. By enabling users to switch appliances such as lighting ON or OFF through a mobile application, the system helps reduce unnecessary energy consumption and promotes responsible usage. Remote access allows users to monitor and manage their devices from anywhere, enhancing convenience, especially when away from home. Furthermore, the system is designed with a clean and intuitive user interface, ensuring ease of use for individuals with varying levels of technical expertise. These objectives collectively aim to improve daily living by making home management smarter, more efficient, and accessible.

2.6.1 Energy-efficiency

Energy efficiency is a fundamental objective of the proposed smart home automation system. By enabling precise control over household devices such as lighting, the system helps minimize unnecessary energy consumption. Traditional home setups often lead to lights or appliances being left on unintentionally, resulting in wasted electricity and higher utility bills. Through automation and real-time control, users can ensure that devices operate only when needed, thus conserving energy. The system's capability to provide timely feedback and support scheduling further enhances energy savings by allowing devices to turn off automatically after a set period or based on environmental conditions. Ultimately, promoting energy-efficient behavior not only reduces operational costs but also contributes to broader environmental sustainability goals by lowering the household's carbon footprint.

2.6.2 Remotely Access

Remote access is another critical objective that significantly enhances the convenience and functionality of the smart home automation system. By integrating IoT technology and a mobile application, users gain the ability to control their home devices from virtually anywhere with an internet

connection. Whether at work, traveling, or simply away from a particular room, users can switch lights on or off, check device status, or activate preset routines without being physically present. This capability not only improves user convenience but also strengthens home security by allowing users to simulate occupancy or respond to unexpected situations immediately. The system's use of the MQTT protocol ensures that commands and status updates are transmitted quickly and reliably, making remote control seamless and responsive.

2.6.3 Easy user interface

An easy-to-use interface is essential to ensure that smart home technology is accessible to a wide range of users, regardless of their technical proficiency. The proposed system's mobile application is designed with simplicity and clarity in mind, featuring intuitive controls and clear visual feedback. This user-centered design approach reduces the learning curve, enabling users to effortlessly operate and customize their home automation settings. By offering straightforward navigation and responsive interaction, the system encourages consistent use and maximizes the benefits of automation. Additionally, the integration of voice command capabilities further simplifies control, providing a hands-free option that enhances accessibility for users with mobility challenges or those who prefer voice interaction. Overall, the focus on usability ensures that the system is practical and appealing for everyday use.

2.7 SYSTEM SPECIFICATION

The system specification outlines the essential hardware and software components required for the effective development and implementation of the proposed smart home automation system. It defines the technical foundation of the project, detailing the devices, platforms, communication protocols, and development tools used to ensure seamless integration and functionality. This specification serves as a blueprint for building a reliable, scalable, and user-friendly system capable of real-time control and automation within a smart home environment.

2.7.1 Software Requirements

Hardware components on their own cannot build the app and execute successfully as needed, hence we also need the software requirements to help us build the mobile application and establish the required configurations. Here are the essential software requirements needed for this application:

2.7.1.1 Flutter SDK

Flutter SDK is an open-source UI software development kit created by Google, and it is used in this project for developing the mobile application that serves as the user interface for the smart home automation system. Flutter allows for the creation of natively compiled applications for both Android and iOS from a single codebase, which significantly reduces development time and ensures consistent performance across platforms. The framework offers a rich set of pre-built widgets and tools that enable the creation of highly responsive and visually appealing interfaces. In this system, the Flutter app allows users to interact with the lighting system by sending commands and receiving feedback in real time. Its support for asynchronous operations, network communication, and integration with MQTT libraries makes it well-suited for IoT-based applications.

2.7.1.2 Eclipse Mosquitto

Eclipse Mosquitto is a lightweight, open-source message broker that implements the MQTT (Message Queuing Telemetry Transport) protocol. It is used in this project to facilitate efficient and reliable communication between the mobile application and the Raspberry Pi. Mosquitto acts as the central hub where messages are published by the mobile app and subscribed to by the Raspberry Pi. The MQTT protocol's publish-subscribe model ensures low-latency, low-bandwidth communication, which is ideal for IoT environments. By using Mosquitto, the system can maintain real-time updates, minimize network overhead, and ensure consistent delivery of control commands and status messages. Its compatibility with various platforms and ease of configuration make it a robust choice for message brokering in home automation systems.

2.7.1.3 Python

Python is a powerful and versatile programming language that is used in this project to control the GPIO (General Purpose Input/Output) pins of the Raspberry Pi. These pins are connected to the relay module, which in turn controls the lighting system. Python scripts run on the Raspberry Pi, continuously listening to MQTT messages and executing the appropriate logic to turn the lights ON or OFF. Python's readability and extensive library support make it ideal for rapid development and integration of automation logic. Additionally, it allows for easy implementation of additional features such as feedback messages, error handling, logging, and potential future extensions like sensor data processing or AI-based decision-making.

2.7.1.4 Raspbian - OS

Raspbian, now officially known as Raspberry Pi OS, is the operating system used to run the Raspberry Pi in this project. It is a Debian-based OS optimized specifically for the Raspberry Pi hardware, offering a stable, secure, and resource-efficient environment. Raspbian provides all the necessary tools and drivers to interact with hardware components such as the GPIO pins, and it supports a wide range of programming languages, including Python. It also offers compatibility with MQTT clients, network services, and development tools. The lightweight nature of Raspbian ensures that the system remains responsive and power-efficient, even while running multiple processes such as MQTT communication, Python scripts, and system monitoring tasks.

2.7.2 Hardware Requirements

The successful implementation of the smart home automation system relies on a carefully selected set of hardware components that facilitate control, communication, and prototyping. Each component plays a crucial role in building a functional, scalable, and reliable system. The following are the primary hardware requirements:

2.7.2.1 Raspberry Pi 4

The Raspberry Pi 4 serves as the core processing unit of the system. It is a powerful single-board computer capable of running full operating systems like Raspbian and executing scripts to control external devices. It provides GPIO (General Purpose Input/Output) pins used to interface with hardware such as relay modules and sensors. The Raspberry Pi handles communication with the MQTT broker, processes commands received from the mobile application, and controls the AC bulb accordingly. The Pi 4 offers more processing power and RAM, while the 3B remains cost-effective and suitable for simpler tasks. Either model supports wireless networking, making them ideal for IoT-based smart home applications.

2.7.2.2 5v Relay Module

The 5V relay module acts as a bridge between the low-voltage control signals from the Raspberry Pi and the high-voltage AC power used by household appliances like bulbs. In this project, the relay module is used to safely control the switching of an AC light bulb. When triggered by the GPIO pin of

the Raspberry Pi, the relay closes or opens the circuit, allowing or cutting off current to the bulb. The module provides electrical isolation and safety features that protect the control circuitry from high-voltage surges, making it a critical component for reliable and secure operation.

2.7.2.3 AC Bulb and Holder

An AC bulb along with its holder serves as the controlled appliance in this project, representing typical household devices that would be automated in a smart home setup. The bulb provides visual feedback and serves as the proof of concept for the system's ability to turn devices ON or OFF through commands sent via the mobile app. It connects directly to the relay module, which switches its power supply based on the signals processed by the Raspberry Pi.

2.7.2.4 Jumper Wires and Breadboard

Jumper wires and a breadboard are used for setting up and testing circuit connections during the development phase. They allow for flexible, tool-free assembly of electronic components, making it easy to prototype and troubleshoot the system without soldering. These tools are essential for connecting the Raspberry Pi's GPIO pins to the relay module and other components, enabling quick adjustments and safe experimentation as the system is being built and tested.

2.7.2.5 Android Phone

An Android smartphone is used to run the Flutter-based mobile application developed for this project. The phone acts as the primary user interface, allowing users to send control commands to the MQTT broker and receive feedback on the status of the lighting system. Since Flutter supports both Android and iOS, the system remains flexible; however, an Android device is preferred during development and testing due to easier debugging and deployment capabilities.

2.7.2.6 Computer

A personal computer or laptop is required for developing the mobile application, writing Python scripts, configuring the Raspberry Pi, and managing the overall project. Development tools such as the Flutter SDK, Python IDEs, and terminal utilities for Raspbian are installed on the computer. It serves as the

base for coding, compiling, testing, and deploying all software components that make the system function.

2.7.2.7 Wi-Fi Module

Reliable internet connectivity is essential for MQTT-based communication between the mobile application and the Raspberry Pi. Most Raspberry Pi 3B and 4 models come with built-in Wi-Fi modules; however, an external USB Wi-Fi adapter can be used if stronger or more stable connectivity is required. The Wi-Fi module ensures that the Raspberry Pi stays connected to the local network, allowing it to publish and subscribe to MQTT topics in real time and maintain seamless operation of the home automation system.

CHAPTER III
SYSTEM DESIGN

3.1 INTRODUCTION

The system design forms the critical backbone of the smart home automation project, providing a detailed architectural framework that guides the development, integration, and implementation of all system components. It involves the strategic planning of how data will flow through the system, how each module will interact, and how these interactions will contribute to achieving seamless, real-time automation. A well-thought-out design ensures that the system is not only functional, but also efficient, scalable, secure, and maintainable over time. In this project, the design incorporates key layers, including the user interface, the communication protocol layer, and the hardware control logic—each playing a distinct and interdependent role. The mobile application developed using Flutter serves as the front-end interface, offering an intuitive and user-friendly means for users to control and monitor their home appliances. It communicates with the back-end system via the MQTT protocol, which facilitates lightweight, fast, and reliable message delivery between the app and the Raspberry Pi. The Raspberry Pi, running on Raspbian OS, interprets incoming commands and controls a relay module that switches AC devices such as lighting on or off. This structured flow of information—from user input to device actuation—ensures precise execution and real-time responsiveness. Furthermore, the modular nature of the design allows for future expansion of the system to support additional appliances such as fans, door locks, and sensors, as well as more advanced features like automation scheduling and AI-based decision-making. Security, flexibility, and user accessibility are also factored into the design, making it robust enough for practical home use while remaining adaptable to educational, prototyping, and commercial applications. Overall, the system design is a deliberate and comprehensive effort to ensure that each component—software and hardware—functions in harmony to deliver a reliable, energy-efficient, and user-friendly smart home experience.

3.2 SYSTEM ARCHITECTURE

The system architecture of this smart home automation project serves as a comprehensive blueprint that defines the overall structure, design principles, and interaction logic of the entire system. It lays out a

high-level view of how various components are organized and how they communicate to perform specific tasks in a coordinated and efficient manner. A well-defined architecture is essential for maintaining clarity, consistency, and scalability throughout the system's development lifecycle. It establishes clear boundaries between different functional layers, supports modularity, and ensures that each component interacts seamlessly without creating unnecessary dependencies. In this project, the architecture is divided into three core layers: **Application Layer**, **Communication Layer**, and the **Device Layer**. Each of these layers is responsible for specific tasks and communicates with the others to ensure smooth data flow and command execution. The Application Layer is represented by a Flutter-based mobile application that acts as the primary user interface, allowing users to send control commands with ease and convenience. The Communication Layer consists of the MQTT broker, which acts as a reliable, lightweight messaging system that handles the publication and subscription of messages between the app and the hardware. This layer ensures real-time, low-latency communication, making it ideal for IoT-based systems. Finally, the Device Layer includes the Raspberry Pi, the relay module, and the AC lighting bulb—components responsible for receiving instructions and physically executing the required actions, such as turning the light ON or OFF. Together, these layers form a robust and cohesive architectural model that promotes efficiency, simplifies development, and supports future scalability. The separation of concerns within the architecture not only enhances maintainability and performance but also provides a clear path for integrating additional smart home features such as motion sensors, temperature monitoring, or voice assistants. This architectural design ensures that the system is both practical for current use and flexible enough to evolve with emerging technologies and user demands.

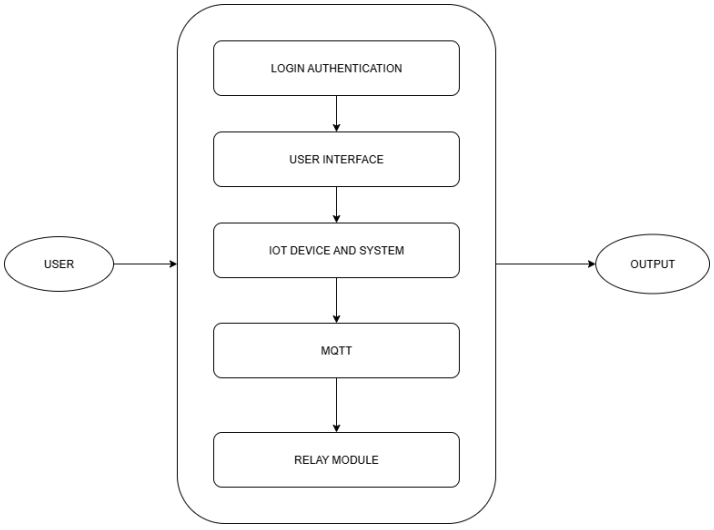


Figure 3.2.1 System Architecture

3.3 USE CASE DIAGRAM

The use case diagram serves as a visual representation of the functional interactions between the user and the smart home automation system, following the Unified Modeling Language (UML) standard. It provides a high-level overview of how the user communicates with the system to perform various tasks and how the system responds to those interactions. In this project, the primary actor is the user, who operates the system through a mobile application developed using Flutter. The core use cases include turning the light ON, turning the light OFF, receiving real-time feedback on the light's current status, and issuing voice commands to perform the same actions. When the user interacts with the app—either by tapping the interface or speaking a command—the system processes the input and sends a control message via the MQTT protocol to a broker. This broker acts as a bridge, relaying the command to a Raspberry Pi, which then activates or deactivates the connected GPIO pin controlling a relay module. The relay, in turn, switches the AC-powered light bulb ON or OFF depending on the user's request. In addition to executing commands, the Raspberry Pi is also configured to send back status messages through the same MQTT broker, enabling the mobile app to display the current state of the light, thereby providing a complete feedback loop. This two-way communication enhances reliability and ensures users are always informed of the system’s status. Moreover, the app features built-in voice recognition, allowing users to control the lighting using natural language voice commands such as “Turn on the light” or “Switch off the light.” This voice control feature increases accessibility, making the system more user-friendly for people with limited mobility or those seeking hands-free operation. Overall, the use case diagram encapsulates a responsive and intelligent interaction framework, emphasizing the convenience, real-time control, and flexibility that the proposed system brings to modern smart home environments.

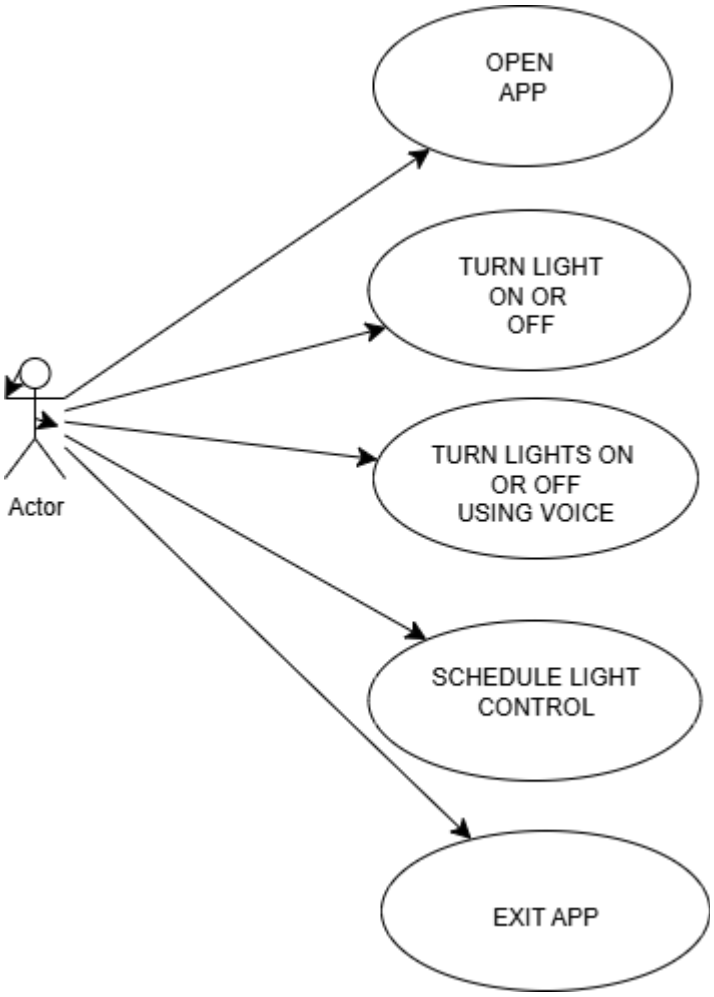


Figure 3.3.1 Use Case Diagram

3.4 DATA FLOW DIAGRAM

The Data Flow Diagram (DFD) is a vital tool in understanding how data moves through the proposed smart home automation system, from user interaction to the physical operation of connected devices. It provides a clear, structured representation of the processes, data stores, and external entities involved in the system. In this context, the DFD outlines the flow of data beginning with the user's interaction with the mobile application—either through touch or voice command. This input is processed by the app and translated into a control message, which is then transmitted to an MQTT broker. The broker acts as the central communication hub, forwarding the message to the Raspberry Pi, which serves as the system's local controller. Once the Raspberry Pi receives the command, it interprets the instruction and activates the appropriate GPIO pin to either switch the relay ON or OFF, effectively controlling

the AC bulb. Additionally, the Raspberry Pi generates a status response that flows back through the broker to the mobile application, ensuring the user receives real-time feedback about the system's current state. This cyclic, bidirectional flow of data between the user, communication layers, and device layer is critical for maintaining system accuracy, responsiveness, and user trust. The DFD not only illustrates the individual roles of the broker, Raspberry Pi, and GPIO control logic but also emphasizes how these components interact seamlessly to enable efficient, real-time automation in a smart home environment.

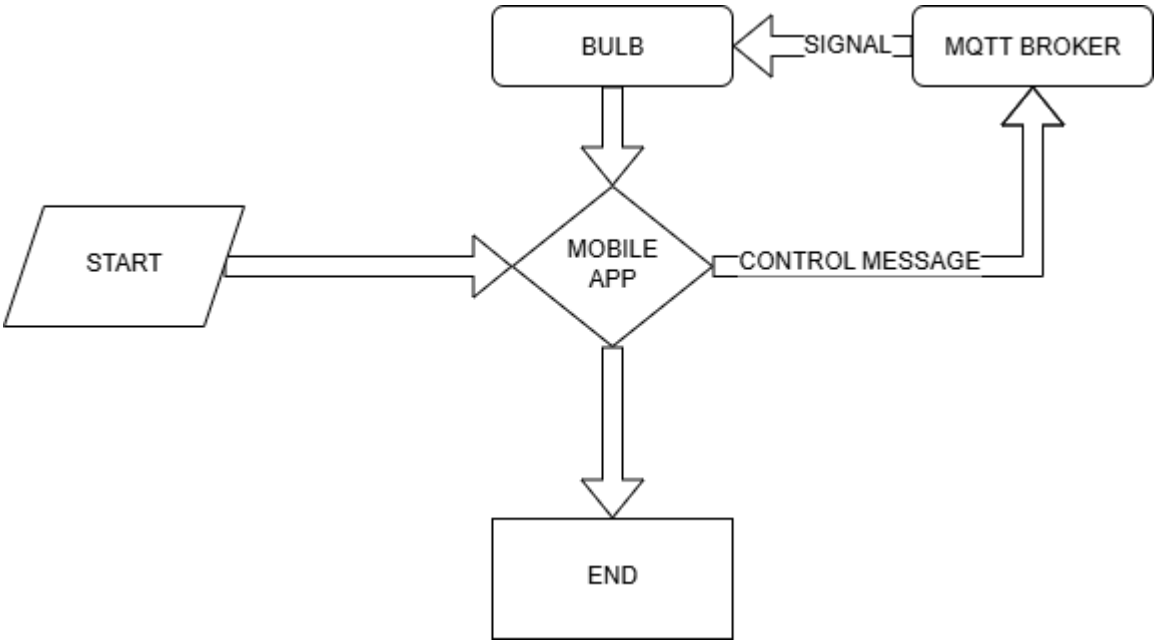


Figure 3.4.1 Data Flow Diagram

3.5 CLASS DIAGRAM

The class diagram plays a critical role in defining the static structure of the smart home automation system by detailing the various classes, their attributes, behaviors (methods), and the associations or relationships among them. As an essential part of object-oriented system design, the class diagram allows for a deeper understanding of how the system is logically organized and how different components communicate internally. It provides a blueprint that helps guide the development process, ensures clarity during implementation, and lays the groundwork for maintainability and scalability. In this particular project, the class diagram models core classes such as `MobileApp`, `VoiceRecognition`, `MQTTClient`, `MessageHandler`, `RaspberryPiController`, and `RelayModule`, each of which has a well-defined role in ensuring the system functions as intended. The `MobileApp` class is responsible for handling user interaction—both through touch and voice—and initiating control requests. The `VoiceRecognition` class processes spoken commands and translates them into executable app actions. The `MQTTClient` class manages the messaging logic, allowing the app to publish commands and subscribe to feedback topics. The `MessageHandler` class acts as an intermediary that processes incoming and outgoing data to and from the MQTT broker. On the hardware side, the `RaspberryPiController` class interprets the received MQTT messages and manages the appropriate GPIO pins to control physical devices such as lights. The `RelayModule` class represents the actual interface to the hardware, switching electrical current based on the signals received. These classes are linked through associations and method calls, making the overall architecture more modular and easier to debug or extend. For instance, adding additional features like fan or lock control would require minimal changes—mainly adding new classes that follow the same structural pattern. Furthermore, the class diagram facilitates clearer documentation and better communication among team members, particularly in collaborative or educational environments. It also supports the design of unit tests and simulation scenarios before actual hardware deployment. Ultimately, the class diagram provides not only a technical reference but also a conceptual framework that aligns with the principles of clean architecture, ensuring the smart home automation system is robust, extendable, and professionally structured.

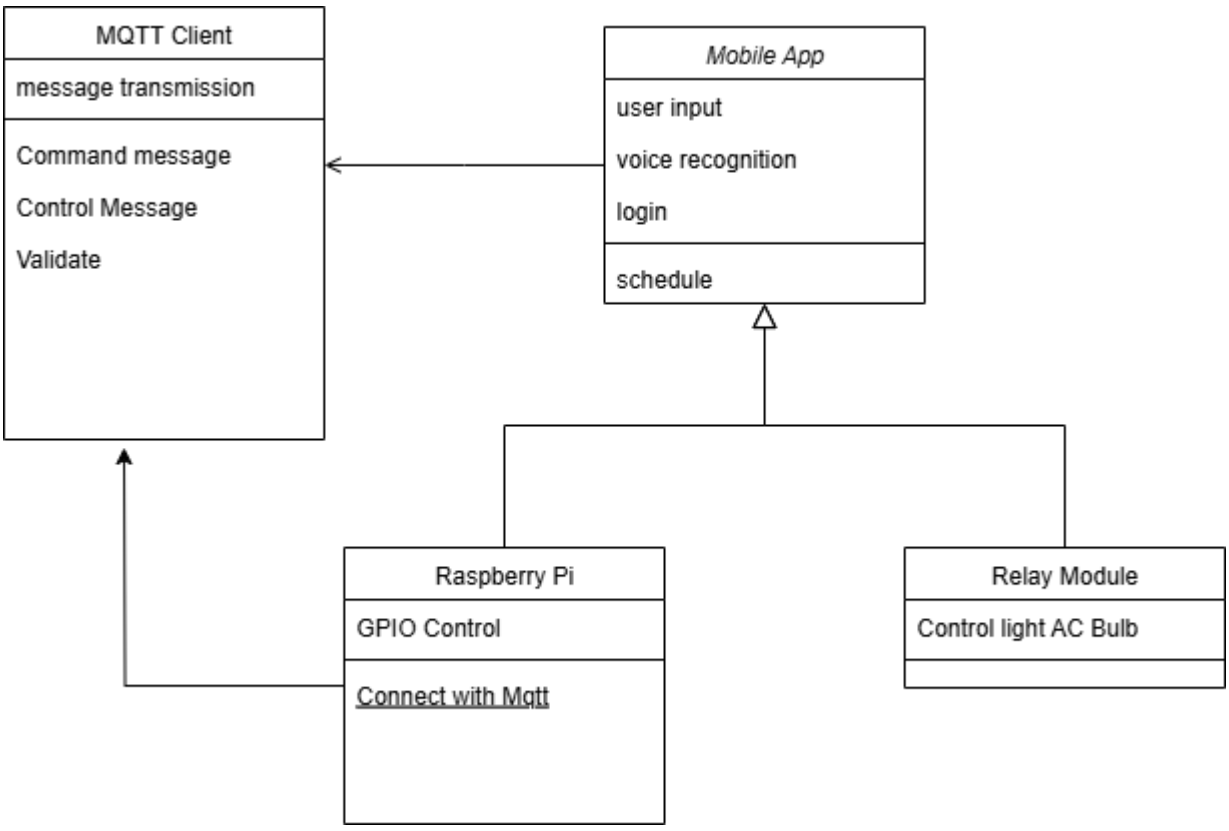


Figure 3.5.1 Class Diagram

3.6 INPUT DESIGN

The input design of the smart home automation system focuses on creating a user-friendly, intuitive interface that enables users to interact with the system easily and efficiently. It is a critical component of the overall system design, as it determines how the user communicates their intent to the system—whether through button taps or voice commands. In this project, the input primarily consists of simple touch-based actions on the mobile application, where users tap clearly labeled and color-coded buttons to control the lighting system. The design ensures that ON and OFF states are visually distinguishable, allowing users to understand the current state of the light at a glance. Each button is strategically placed and sized to accommodate different screen sizes and user needs, enhancing accessibility and reducing the chance of errors. Additionally, the interface includes confirmation messages or subtle feedback—such as icon changes or toast notifications—to prevent accidental activations and to reassure the user that their input has been successfully received and processed. The input design also integrates voice

command functionality, where spoken instructions are interpreted by the app's built-in voice recognition module and mapped to corresponding system actions. This feature enhances the hands-free control of home devices, making the system more inclusive and adaptable to modern user expectations. Overall, the input design prioritizes clarity, responsiveness, and error prevention, ensuring that users can interact with the system confidently and comfortably.

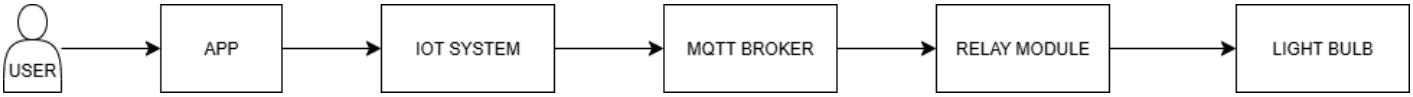


Figure 3.6.1 Input Design Diagram

3.7 OUTPUT DESIGN

The output design of the smart home automation system focuses on delivering clear, timely, and meaningful feedback to the user in response to their interactions with the system. It plays a crucial role in ensuring transparency and enhancing user confidence by confirming that their commands—whether touch-based or voice-activated—have been successfully executed. In this project, the primary output is the real-time status display within the mobile application, which informs the user whether the light is currently ON or OFF. This feedback is displayed through dynamic interface elements such as color changes, icons, or status text updates, all designed to be easily noticeable and understandable at a glance. The system also supports subtle feedback mechanisms such as toast messages or animations that confirm each successful command execution. These outputs are generated after the Raspberry Pi processes the MQTT message and activates the relay module, which physically switches the light. Once this action is completed, the system sends a confirmation message back to the app through the MQTT broker, ensuring the feedback loop is complete. This real-time two-way communication enhances the reliability and interactivity of the system. Additionally, the output design takes into account accessibility by maintaining a clean, minimalistic layout and using contrasting colors and readable fonts, ensuring it remains user-friendly across various devices and lighting conditions. Altogether, the

output design ensures that users are always informed about the current state of their home devices, reinforcing trust and delivering a seamless smart home experience.

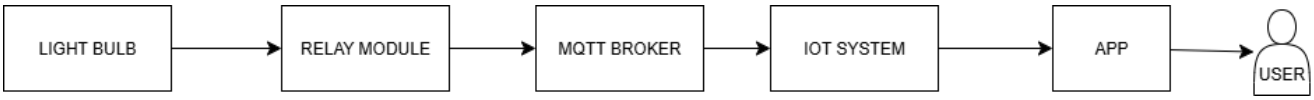


Figure 3.7.1 Output Design

CHAPTER IV
SYSTEM DEVELOPMENT

4.1 INTRODUCTION

The system development section provides a comprehensive overview of the processes, methodologies, and tools used in building the smart home automation system. This stage marks the transformation of theoretical concepts and design blueprints into a fully functional solution, integrating both hardware and software components to achieve the intended objectives. In this project, the development process involved a combination of mobile app development, embedded system programming, and network communication setup, all orchestrated to create a seamless user experience for home automation. At the heart of the development is the Flutter framework, which was used to build a responsive and intuitive cross-platform mobile application that allows users to control their home lighting through simple button taps and voice commands. Parallel to this, the Raspberry Pi was configured to act as the central controller for the lighting system, running a Python script that listens for incoming commands over the MQTT protocol and triggers GPIO actions to switch the relay module—and ultimately the connected light bulb—ON or OFF. The Eclipse Mosquitto MQTT broker served as the communication bridge between the mobile application and the Raspberry Pi, enabling lightweight, real-time messaging over a local Wi-Fi network. Special attention was given to designing a modular and scalable system architecture, which would allow future integration of additional smart devices like fans, locks, and sensors without overhauling the entire system. During development, various testing stages were employed to ensure stability, including unit testing of individual components and system-level testing to verify the communication flow and responsiveness. The system also underwent iterative improvements based on practical observations, particularly in optimizing user interaction, handling potential command conflicts, and refining feedback mechanisms. Overall, the system development process was rooted in modern IoT practices and agile principles, ensuring that the final implementation is not only functional and reliable but also extensible and well-aligned with the needs of contemporary smart living environments.

4.2 MODULE DESCRIPTION

The module description section outlines the core components of the smart home automation system and their respective roles in ensuring smooth operation. The system is divided into three main modules: Mobile Application Module, Communication Module, and Device Control Module. The Mobile Application Module, developed using Flutter, allows users to interact with the system through touch or voice commands, sending control requests and displaying real-time feedback. The Communication Module is powered by the MQTT protocol and handled through the Eclipse Mosquitto broker, which efficiently routes messages between the mobile app and the hardware. Finally, the Device Control Module includes the Raspberry Pi, a relay module, and an AC bulb, where the Pi interprets incoming messages and triggers the appropriate GPIO actions to switch the light ON or OFF. These modules work together to deliver a responsive, scalable, and user-friendly smart home solution.

4.2.1 MOBILE APP MODULE

The Mobile App Module serves as the primary interface between the user and the entire system. Developed using the Flutter SDK, this cross-platform mobile application allows users to interact with the lighting system using intuitive touch-based controls and integrated voice commands. The app presents a simple and user-friendly UI with clearly labeled ON and OFF buttons, color-coded to reflect the current status of the light. When a user performs an action—such as tapping a button or giving a voice command—the app sends an MQTT message to the broker, initiating communication with the hardware controller. The app also subscribes to feedback topics, enabling it to display real-time status updates and confirmations once the command has been executed. This module enhances user experience by offering responsive controls, visual feedback, and voice accessibility, making the system easy to operate for users of all technical backgrounds.

4.2.2 BROKER MODULE

The Broker Module is the core communication layer of the system, responsible for managing the exchange of messages between the mobile application and the Raspberry Pi. This is handled through Eclipse Mosquitto, a lightweight and open-source MQTT broker ideal for IoT applications. The broker listens for messages published by the app, such as "Turn ON" or "Turn OFF", and forwards them to the appropriate subscriber—in this case, the Raspberry Pi. It also handles messages sent back from the Pi

to the mobile app, including feedback on the current status of the light. This publish-subscribe model ensures decoupled and efficient communication, allowing multiple devices to communicate reliably in real-time with minimal bandwidth usage. The use of MQTT makes the system scalable and flexible, enabling future enhancements like the addition of new smart devices without significant changes to the communication structure.

4.2.3 CONTROL MODULE

The Control Module encompasses the hardware responsible for executing the user's commands. It includes the Raspberry Pi, a 5V relay module, and an AC bulb. When the Raspberry Pi receives a command message from the broker, it parses the message and uses a Python script to toggle a designated GPIO pin. This pin is connected to the relay, which then switches the AC bulb ON or OFF based on the user's instruction. In addition to performing the requested action, the Raspberry Pi also sends a status message back to the broker, which is then delivered to the mobile app for real-time feedback. The control module is designed to be modular and extensible, allowing for the integration of additional appliances such as fans, door locks, or sensors. It acts as the physical layer of the automation system, bridging the digital instructions from the app with tangible, real-world electrical actions.

4.3 METHODOLOGY

The methodology used for the development of this smart home automation system was based on the Agile framework, which promotes iterative development, continuous feedback, and adaptability throughout the project lifecycle. Given the project's integration of both hardware and software components, Agile provided the flexibility needed to manage complexity and respond to evolving requirements. The entire development process was broken down into focused sprints, with each sprint dedicated to specific tasks such as hardware setup and GPIO configuration, MQTT protocol integration for real-time messaging, and mobile application development using Flutter, including both touch-based and voice control features. At the end of each sprint, rigorous testing and evaluation were carried out to ensure the functionality of each module, followed by documentation and improvements informed by the results. This iterative approach allowed for early detection of issues, seamless integration between modules, and progressive refinement of the system. Agile also supported continuous integration and parallel development, making it easier to manage interactions between the mobile app, broker, and Raspberry Pi. By maintaining a feedback-driven workflow and flexible planning, the Agile

methodology significantly contributed to the successful and efficient development of a responsive, scalable, and user-centric smart home automation solution.

4.3.1 AGILE METHODOLOGY

The Agile methodology is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement.

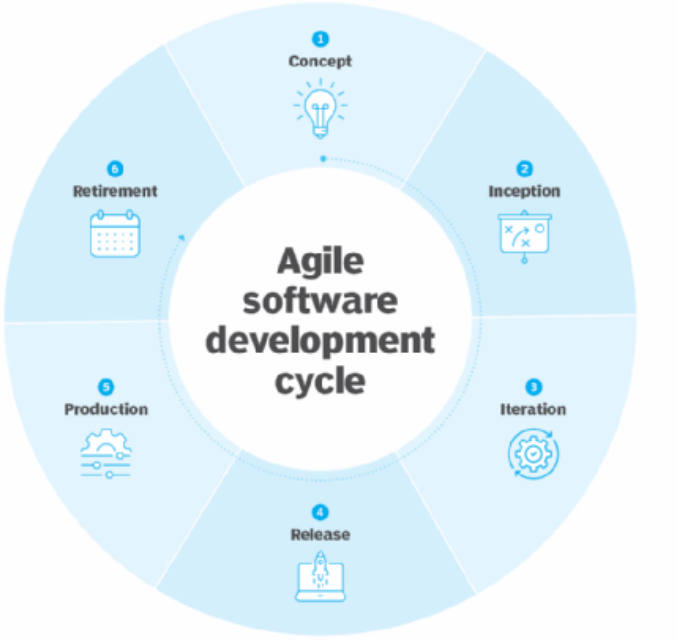


Figure 4.3.1 Agile Methodology life Cycle

4.3.1.1 Planning

The planning phase marked the beginning of the project and focused on defining clear goals, identifying system requirements, and breaking down the project into manageable tasks. During this stage, the core idea of automating home lighting using IoT was shaped, with specific features such as mobile control, voice commands, and MQTT-based communication being prioritized. The entire system was divided into modules—namely, the mobile app, broker communication, and Raspberry Pi control. Each module was assigned to individual sprints based on dependencies and estimated time for implementation. A project backlog was created, and milestones were established to guide the sprint cycles.

4.3.1.2 Design

In the design phase, both system and component-level designs were developed to guide implementation. This included creating use case diagrams, class diagrams, and architectural blueprints that described how different modules—like the mobile app, MQTT broker, and hardware controller—would interact. System design also covered the input and output mechanisms, including the user interface layout, voice command functionality, and GPIO configurations for hardware control. These design artifacts helped ensure that each part of the system was planned with scalability, modularity, and ease of integration in mind.

4.3.1.3 Development

The development phase followed an iterative sprint structure, where each sprint targeted the completion of a specific module or feature. The mobile application was developed using Flutter, providing cross-platform compatibility and integrating voice control features. In parallel, the Raspberry Pi was programmed using Python to control the GPIO pins based on MQTT messages. The Eclipse Mosquitto broker was installed and configured to facilitate lightweight, real-time messaging between the app and the hardware. With Agile, code was written, reviewed, and improved incrementally, allowing continuous progress without waiting for the entire system to be built.

4.3.1.4 Testing

Testing was a continuous process embedded into each sprint. After each development cycle, unit tests were conducted on individual components—such as the relay control script, MQTT message parsing, and app interface interactions—to validate their behavior. Integration testing ensured that communication between the mobile app, broker, and Raspberry Pi was consistent and reliable. Special emphasis was placed on testing voice recognition accuracy, system responsiveness, and real-time feedback from the light status. Any bugs or inconsistencies discovered were addressed in the same or following sprint, ensuring a stable and evolving system.

4.3.1.5 Deployment

Deployment involved bringing the developed components together in a live environment. The mobile application was tested on an Android device to ensure full functionality outside the development

emulator. The Raspberry Pi was configured with all necessary software (Python scripts, Mosquitto client) and connected to the relay and AC light bulb. The system was deployed over a local Wi-Fi network, and MQTT topics were verified for real-time communication. This phase marked the transition from development to real-world application, simulating how users would actually interact with the system.

4.3.1.6 Operations and Maintenance

Operations consisted of running the system in its intended environment and monitoring performance over time. This included observing how the system handled multiple commands, whether the feedback loop was consistent, and ensuring that the voice feature worked reliably in various ambient conditions. Real-time logging and MQTT message tracing were used to identify operational issues. User feedback and personal usage insights during this stage helped refine the system further, especially in improving UI responsiveness and handling edge cases. Maintenance is an ongoing phase that ensures the system continues to function optimally after deployment. Any bugs or performance bottlenecks identified during operation were logged and resolved. Updates were applied to improve system performance, such as refining the voice recognition model, optimizing MQTT communication for better latency, and making small UI adjustments for better usability. This phase also opens the door for future enhancements, such as integrating additional smart devices or implementing cloud-based remote access, keeping the system scalable and future-ready. Maintenance is an ongoing phase that ensures the system continues to function optimally after deployment. Any bugs or performance bottlenecks identified during operation were logged and resolved. Updates were applied to improve system performance, such as refining the voice recognition model, optimizing MQTT communication for better latency, and making small UI adjustments for better usability. This phase also opens the door for future enhancements, such as integrating additional smart devices or implementing cloud-based remote access, keeping the system scalable and future-ready.

4.4 ALGORITHMS

The algorithms used in this smart home automation project form the logical core of the system, enabling reliable control, communication, and feedback mechanisms between the user interface and the physical hardware. These algorithms are designed to process user input—whether through button taps or voice commands—translate them into control messages via the MQTT protocol, and execute corresponding

actions on the Raspberry Pi. They also manage feedback by continuously monitoring the status of the light and sending real-time updates back to the mobile application. Simple yet efficient logic structures and control flow techniques were applied to ensure quick responsiveness, accurate device toggling, and seamless two-way communication. These algorithms not only enhance the performance of the system but also ensure energy efficiency, reliability, and scalability, aligning with the core objectives of the smart home automation solution.

4.4.1 MQTT Parsing Algorithm

The MQTT Message Handling Algorithm is fundamental to enabling reliable communication between the mobile application and the Raspberry Pi using the MQTT protocol. This algorithm operates by having the Raspberry Pi run a Python script that connects to the MQTT broker and subscribes to a specific topic, such as home/light/control. When the user interacts with the app or issues a voice command, the app publishes a control message—like "TURN_ON" or "TURN_OFF"—to this topic. Upon receiving the message, the MQTT client on the Raspberry Pi triggers a callback function that parses and interprets the instruction. If the command is "TURN_ON", the Raspberry Pi sets the designated GPIO pin connected to the relay to a high state, activating the relay and turning the light on. Conversely, if the command is "TURN_OFF", the GPIO pin is set low, switching the light off. After executing the requested action, the Raspberry Pi publishes a status update message such as "Light is ON" back to another MQTT topic like home/light/status to provide real-time feedback to the mobile application. This algorithm ensures lightweight, real-time communication that is efficient and scalable, making the system responsive and reliable even in low-bandwidth environments.

4.4.2 Speech-to-text(STT) Algorithm

The Voice Command Interpretation Algorithm empowers users to control the lighting system through natural spoken commands, enhancing accessibility and convenience. Within the Flutter mobile app, a voice recognition module—is integrated to capture and convert spoken words into text. When the user activates the voice input and says a command like “Turn on the light,” the app processes the audio, transcribes it into text, and applies a keyword-matching algorithm to interpret the intent. Keywords such as “on” or “turn on” are mapped to the internal command "TURN_ON", while phrases like “off” or “turn off” correspond to "TURN_OFF". Once the appropriate command is determined, it is published to the MQTT broker following the same messaging protocol used for manual input, which triggers the

Raspberry Pi to execute the action accordingly. This algorithm significantly enhances the user experience by enabling hands-free control, making the system more user-friendly and aligned with modern smart home expectations.

CHAPTER V
SYSTEM TESTING

5.1 INTRODUCTION

System testing is a critical phase in the development lifecycle of the smart home automation project, aimed at verifying that the integrated system meets the specified requirements and functions correctly in a real-world environment. Unlike unit testing, which focuses on individual components, system testing evaluates the entire solution as a cohesive whole, ensuring that hardware, software, communication protocols, and user interfaces work seamlessly together. Given that this project integrates multiple modules—such as the Flutter-based mobile application, MQTT communication infrastructure, and Raspberry Pi hardware controller—comprehensive system testing was essential to validate the interaction between these components and to detect potential issues that could affect usability, reliability, and performance. The testing process was designed to be iterative and methodical, reflecting the Agile development methodology used throughout the project. Each sprint concluded with testing phases to verify the newly developed features or improvements. Initial testing focused on individual module functionality, such as validating MQTT message transmission and GPIO pin control on the Raspberry Pi. Following successful unit tests, integration tests were conducted to ensure that messages published by the mobile app correctly triggered hardware responses and that feedback messages were accurately received and displayed by the app. Emphasis was also placed on testing the voice command functionality to verify the accuracy of speech recognition and the correctness of command interpretation. System testing involved a variety of scenarios designed to assess the system's robustness under different conditions, including network latency, command concurrency, and user input errors. Real-time responsiveness was a key metric, as the system's primary goal is to provide instantaneous control and feedback for home automation tasks. Furthermore, the security and reliability of the MQTT communication were scrutinized to prevent unauthorized access or message loss. The feedback loop between the hardware and mobile app was monitored to ensure users were consistently informed of the system's current state, minimizing confusion and enhancing trust. In addition to functional testing, usability tests were performed to evaluate the intuitiveness of the user interface and the effectiveness of voice commands in real-life settings. These tests helped identify areas where the interface could be refined for clarity and ease of use. The system was also tested across different

Android devices to guarantee compatibility and performance consistency. Overall, the system testing phase was comprehensive, encompassing functional, integration, usability, and reliability testing to deliver a robust and user-friendly smart home automation system. The insights gained from testing informed iterative improvements, ensuring that the final deployed solution meets user expectations and industry best practices for IoT-based automation.

5.2 SYSTEM TESTING METRICS

System testing metrics provide essential measures to evaluate the quality, security, and performance of the mobile application within the smart home automation system. Each metric targets key aspects critical to the application's reliability and user satisfaction, offering a comprehensive assessment that guides improvements and ensures the system meets its functional and experiential goals.

5.2.1 Performance Evaluation

The first focus of our testing endeavors is on assessing the application's performance. This encompasses scrutinizing its speed, responsiveness, and resource utilization under varying 27 conditions. By subjecting the application to different scenarios and workloads, we aim to guarantee that it operates seamlessly, providing users with swift and efficient interactions.

5.2.2 Functional Validation

Functional validation metrics focus on verifying that the system’s features and operations perform as intended according to the specified requirements. These metrics assess whether each function—such as turning the light on or off, processing voice commands, and providing accurate feedback—works correctly, ensuring the system delivers a reliable and seamless user experience.

5.2.3 Security Assurance

Security assurance metrics evaluate the system’s ability to protect user data and prevent unauthorized access or malicious attacks. These metrics ensure that communication channels, such as MQTT messaging, are secure and that only authorized users can control home devices, thereby safeguarding the system’s integrity and user privacy.

5.2.4 User Experience Validation

User experience validation metrics measure how intuitive, responsive, and satisfying the system is from the user's perspective. These metrics focus on interface usability, ease of navigation, responsiveness of controls, and the effectiveness of voice command features to ensure a smooth and enjoyable interaction with the smart home system.

5.2.5 Reliability and Seamless Operation

Reliability and seamless operation metrics assess the system's consistency in performing tasks without failures or interruptions. These metrics track uptime, error rates, response times, and the ability to maintain continuous communication between the app, broker, and hardware, ensuring the system operates smoothly in real-world conditions.

5.3 TEST PLAN

The Test Plan outlines the comprehensive strategy for evaluating the application's various aspects. This table outlines the different types of tests, their descriptions, the aims of each test, and the expected results.

Test Case Type.	**Test Aim**	**Expected results**	**Test result**
Functional Testing	Validate the correctness of each functional component.	All features perform as intended without errors or unexpected behavior	Successful
Usability Testing	Assess the user interface for ease of use and overall user experience	Users can navigate the application effortlessly, and the interface is intuitive.	Successful

Performance Testing	Measure the efficiency and reliability of the application under varied scenarios.	Application responds quickly, maintains responsiveness, and utilizes resources efficiently.	Successful
Security Testing	Ensure the application is resilient against security threats and vulnerabilities	Security measures effectively protect user data, and vulnerabilities are addressed.	Successful
Compatibility Testing	Confirm the application's compatibility with various devices and operating systems.	Application functions correctly on different devices and platforms.	Successful
Regression Testing	Validate that modifications or additions do not introduce issues to existing functionalities.	Existing features continue to operate correctly after updates or changes.	Successful
User Acceptance Testing	Obtain feedback from users to validate that the application meets their needs and expectations.	Users express satisfaction, and the application aligns with their expectations.	Successful
Load Testing	Evaluate the application's capacity to handle different user loads without performance degradation.	Application maintains performance and responsiveness under varying user loads.	Successful

Table 5.3.1 Test Plan

CHAPTER VI
SYSTEM IMPLEMENTATION

6.1 INTRODUCTION

System implementation represents the culmination of design and development efforts into a fully functioning solution, where both the visual front end and backend logic are brought together to execute the intended operations of the Smart Home Automation System. This phase goes beyond coding to encompass the strategic integration of software frameworks, communication protocols, and hardware components, ensuring that the entire system operates smoothly and reliably under real-world conditions. In this project, the implementation integrates a Flutter-based mobile application and Python-powered control logic on a Raspberry Pi, unified through the MQTT messaging protocol. The Flutter app acts as the user's control panel, providing a seamless interface where users can manage home devices through intuitive touch controls or voice commands. Screens such as login pages, control dashboards, and status displays are not only functional but designed with usability in mind, featuring real-time status indicators, color-coded toggles, and confirmation alerts to guide user interaction. Visual feedback mechanisms like spinners, alerts, and toast messages further enhance responsiveness and clarity. On the backend, the system's intelligence lies in Python scripts running on the Raspberry Pi. These scripts utilize libraries like paho-mqtt for broker communication and RPi.GPIO for direct hardware control, enabling precise and efficient switching of connected devices via GPIO pins. When the app sends a command—such as turning a light ON—the message is routed through the Eclipse Mosquitto broker, received by the Pi, and translated into an electrical signal that toggles the relay module controlling the appliance. This architecture enables real-time interaction with minimal latency, making the system not only responsive but highly energy-efficient. The hardware side of the implementation involves configuring the Raspberry Pi with essential peripherals including a 5V relay module, jumper wires, AC bulb and holder, a breadboard for prototyping, and a Wi-Fi-enabled smartphone for issuing commands. All these components are connected and tested to ensure synchronized operation. Moreover, screenshots of the mobile app interface are used as documentation to showcase the implementation outcomes, demonstrating user navigation paths, functional controls, and how feedback is delivered visually. Importantly, the system's modularity allows easy expansion—additional appliances such as fans, smart locks, or sensors can be integrated using the same core logic

and protocols. This practical execution of both software and hardware underlines the system's real-world applicability, validating its role as a scalable, intelligent, and user-centric smart home automation solution.

6.2 SCREENSHOTS

System screenshots are visual captures of the software in action, offering a precise snapshot of the user interface at specific stages of interaction. These images play a vital role in system documentation by showcasing how the application appears and functions in real-time scenarios. In the context of this smart home automation system, screenshots illustrate key components of the Flutter-based mobile app, including the login screen, dashboard interface, control buttons for toggling devices, feedback messages, and the integration of voice command functionality. These visual elements highlight user interaction patterns, system responsiveness, and the intuitive layout designed to facilitate seamless control of home appliances. Screenshots are particularly important because they provide visual confirmation of the system's features and behaviors, bridging the gap between technical documentation and user understanding. They simplify the explanation of dynamic processes such as system startup, MQTT command issuance, voice input handling, and real-time feedback display. By visually representing data input, processing logic, and output response, screenshots help communicate how the system operates in actual use—something that text alone cannot fully capture. Moreover, system screenshots serve as verification tools during evaluations, stakeholder presentations, and user training, demonstrating that each module has been implemented as intended. They also support debugging and future development by documenting the UI at various stages, providing a visual reference for improvements or modifications. Overall, the inclusion of system screenshots enhances the clarity, transparency, and credibility of the project, making them an indispensable part of the implementation report.

6.2.1 Authentication

The authentication module of the smart home automation system is built with robust biometric security to ensure that only authorized users can access the application. It integrates seamlessly with the mobile device's native biometric settings, supporting both fingerprint and facial recognition depending on the hardware capabilities of the phone. When a user attempts to open the app, they are immediately prompted to verify their identity using one of these biometric methods. This process adds a critical

layer of protection, preventing unauthorized access and securing control over home devices. If the authentication attempt fails—whether due to an unrecognized fingerprint or incorrect facial scan—the system blocks entry, maintaining the integrity and privacy of the smart home environment. By combining advanced security with user-friendly functionality, the biometric authentication module delivers a highly secure yet convenient login experience.

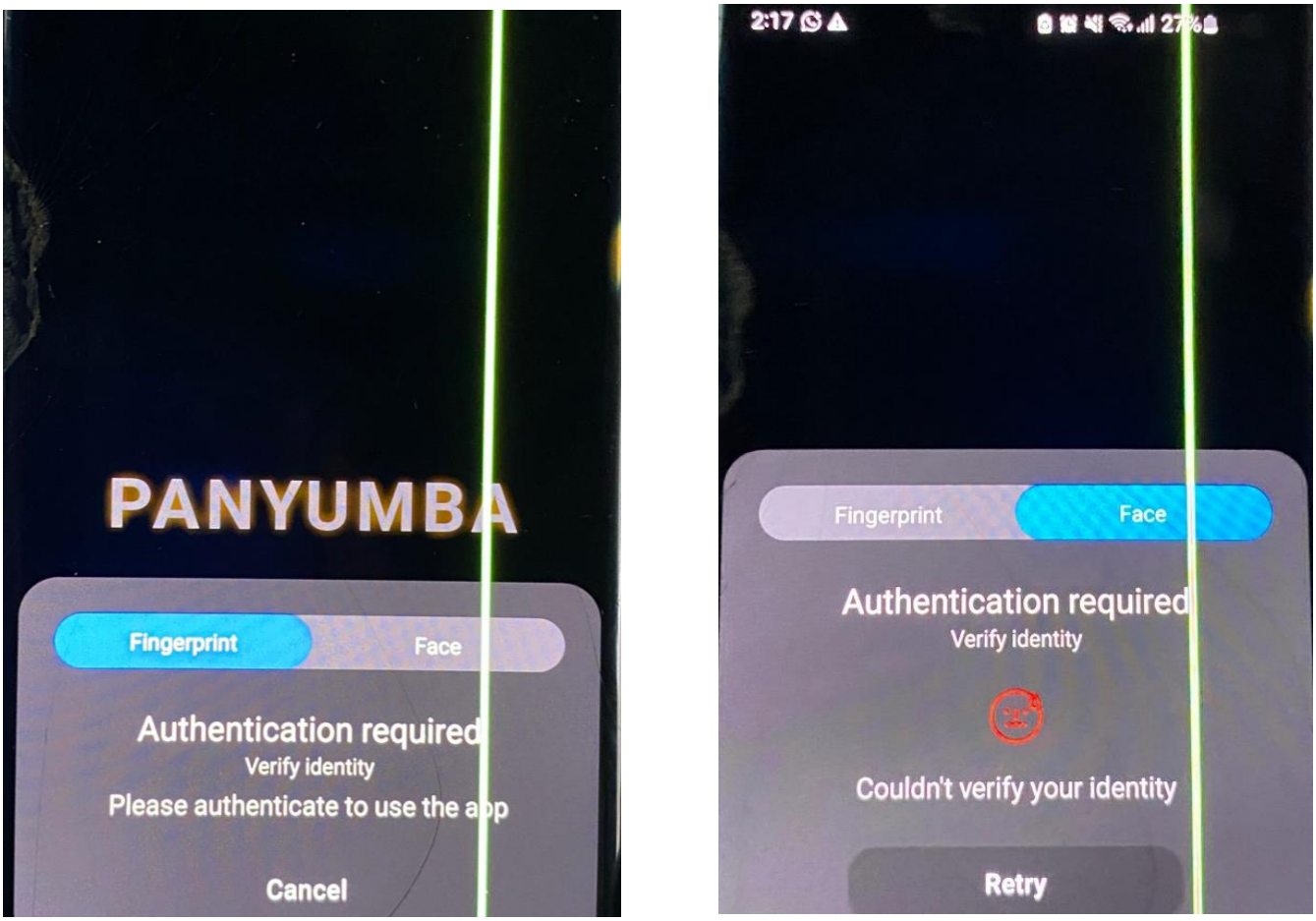


Figure 6.2.1.1 Login Authentication

6.2.2 Home Screen

The home screen of the smart home automation app serves as the central hub for user interaction and control. It is designed with simplicity and clarity in mind, featuring a clean layout anchored by a bottom navigation bar that provides quick access to three

main sections: Dashboard, Schedule, and Settings. The dashboard, which also functions as the home screen, includes essential control features such as a toggle switch and a voice command button, both used to turn the connected light on or off. Users can simply tap the switch for manual control or use the voice button to issue spoken commands for hands-free operation. At the center of the screen is a large animated bulb icon that visually reflects the current state of the light—glowing when the light is on and dimmed when it is off—offering intuitive, real-time feedback. This interactive design ensures users can easily monitor and control their home environment with minimal effort while enhancing the overall user experience through responsive visual elements.

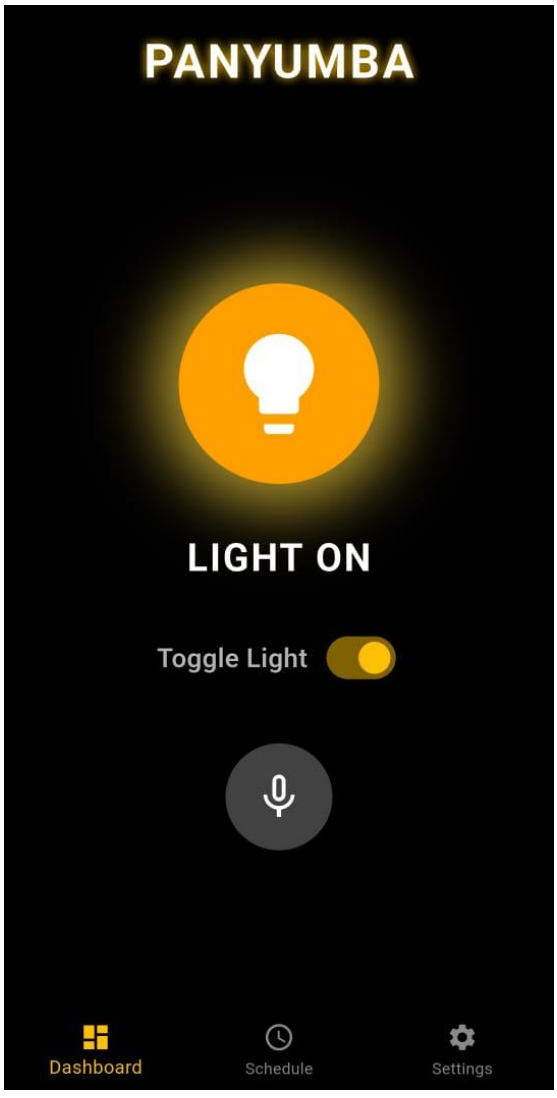
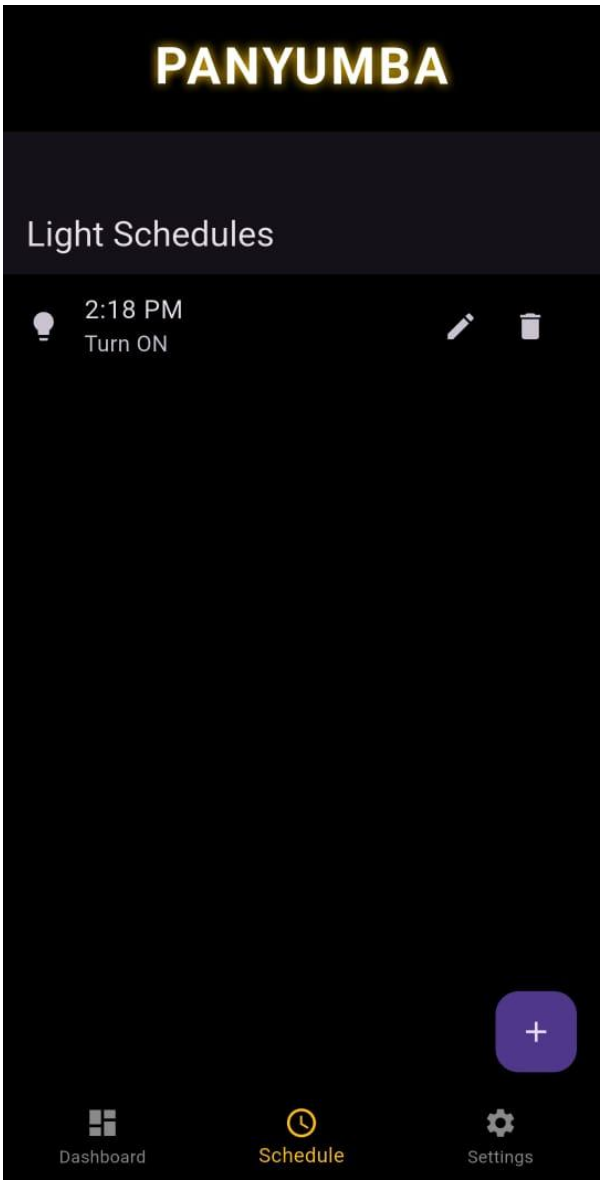


Figure 6.2.2.1 Home Screen

6.2.3 Schedule Screen

The Schedule Screen in the smart home automation app is designed to give users a convenient way to automate lighting control within the same day using a straightforward and user-friendly time picker interface. Through this feature, users can select a specific time to either turn the light on or off, helping to automate routine tasks and promote energy efficiency. The schedule applies only to the current day and resets afterward, making it ideal for short-term use cases such as setting the light to switch off before sleep or turn on during the evening hours without requiring repeated daily inputs. This focused, same-day scheduling approach ensures simplicity while still offering meaningful automation capabilities for daily routines. What makes this feature even more powerful is its integration with

voice-based scheduling. Users can simply issue spoken commands like “Turn the light off at 9 PM,” and the app interprets and sets the schedule without requiring manual interaction. This hands-free functionality greatly improves accessibility and convenience, especially for users who may be multitasking or have limited mobility. Once a schedule is set—whether by touch or voice—the app stores the timing logic and ensures that the relay is triggered accordingly through MQTT communication with the Raspberry Pi. Overall, the Schedule Screen combines simplicity, flexibility, and voice interactivity to enhance the smart home experience while keeping the system intuitive and user-focused.



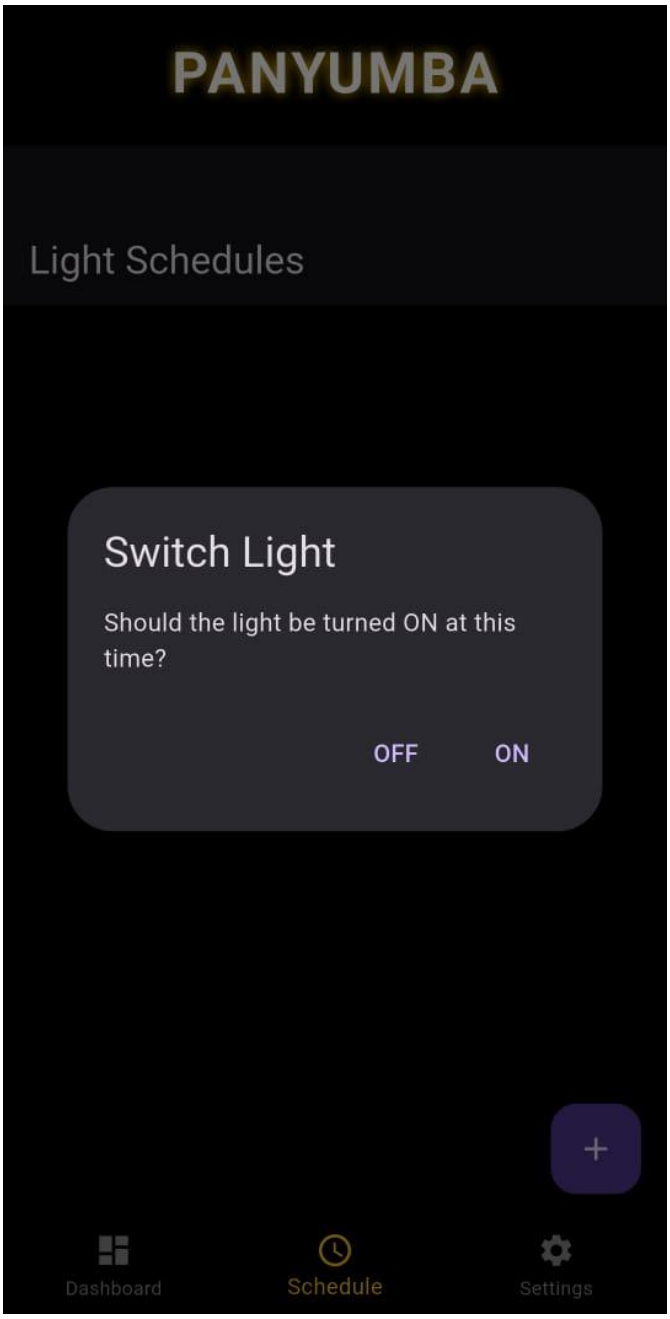
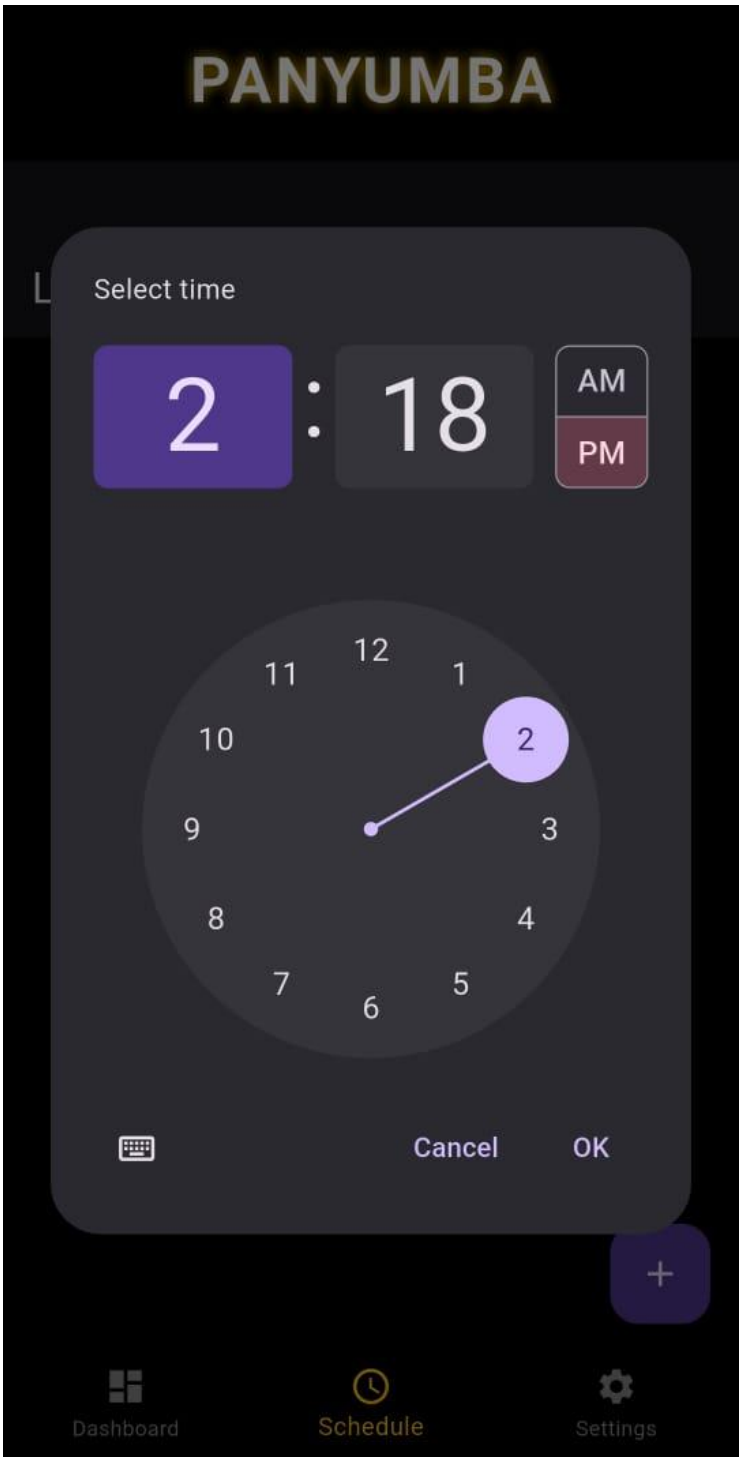


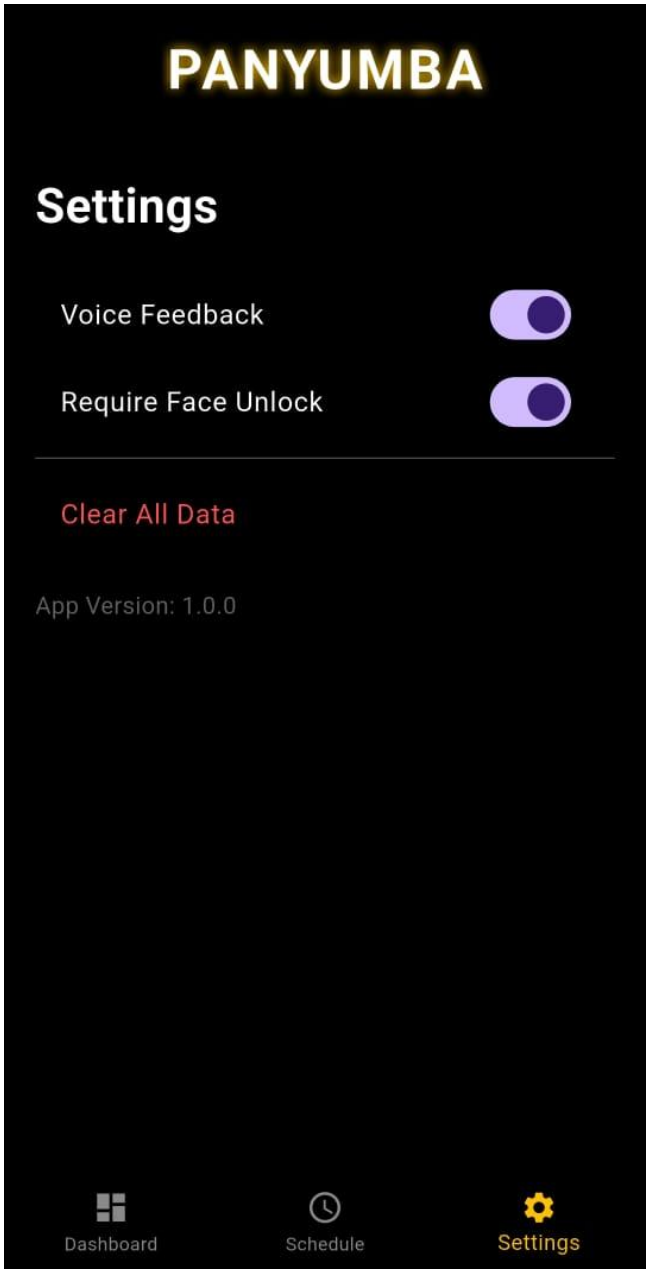
Figure 6.2.3.1 Schedule Screen



6.2.4 Settings Screen

The Settings Screen of the smart home automation app offers users basic yet essential customization options to personalize their experience with the system. The interface is clean and minimalistic, presenting toggle buttons that allow users to enable or disable specific features—namely, face authentication and voice feedback. These settings provide flexibility, giving users control over how they interact with the app. For instance, if a user prefers not to use biometric security, they can simply

turn off face authentication. Similarly, voice feedback, which provides audible confirmations of system actions, can be toggled off for a quieter user experience. This straightforward design ensures that even non-technical users can easily configure their preferences without navigating through complex menus. By allowing users to manage core features based on their comfort and needs, the Settings section enhances the system's adaptability while maintaining simplicity and ease of use.



6.3 CODING

Coding is the backbone of any software development project, representing the translation of logic and system requirements into a language that computers can understand and execute. In the context of this smart home automation system, coding plays a central role in defining both the frontend user interface and the backend control logic. The application involves writing instructions in multiple programming languages-Dart for the Flutter mobile app and Python for the Raspberry Pi control scripts. Each piece of code consists of structured elements such as variables to store values, control structures like loops and conditionals to handle decision-making, and functions to modularize tasks and promote reusability. Comments are also included throughout the codebase to improve readability and ease future maintenance. The mobile app code defines the behavior of buttons, manages the flow of user input, and communicates with the MQTT broker using external Dart libraries like `mqtt_client`. It also incorporates native device features such as biometrics and voice recognition, which are accessed through platform-specific plugins and APIs. On the hardware side, Python scripts are used to handle GPIO pin interactions on the Raspberry Pi, where libraries such as `paho-mqtt` and `RPi.GPIO` manage incoming MQTT messages and toggle relay states to control electrical appliances. Throughout the coding process, developers utilize external packages and frameworks to simplify tasks that would otherwise require extensive development time. These libraries not only streamline MQTT integration and hardware communication but also ensure that the system is stable, responsive, and secure. After writing the code, it is thoroughly tested to confirm that each function behaves as intended under different conditions. Once verified, the code runs either natively on the device or via an interpreter, allowing the system to respond in real-time to user actions—whether they come from a button press, a voice command, or an automated schedule. Coding in this project is more than just technical implementation; it is the means through which user needs are transformed into a fully operational, interactive smart home system. The clear, modular, and scalable code structure ensures that the system is not only functional today but also easy to extend with additional features or devices in the future.

6.3.1 FRONT END

The Front-End Coding subsection highlights the client-facing layer of the smart home automation system, which is responsible for how users visually interact with and navigate the application. This section focuses on the code that builds and renders the user interface, manages dynamic elements, and

ensures smooth, responsive interactions. It details the technologies, frameworks, and design principles used to create a visually appealing and user-friendly experience. Developed using the Flutter SDK, the front-end leverages Dart programming to build cross-platform interfaces that are consistent, efficient, and intuitive across different Android devices. This part of the project plays a crucial role in bridging the user with the system's core functionalities, from controlling devices to receiving feedback, all within a clean and accessible interface.

For this project, here is the main.dart code.

```dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:get_storage/get_storage.dart';
import 'package:kunyumba_clean/controllers/auth_controller.dart';
import 'package:kunyumba_clean/controllers/schedule_controller.dart';
import 'package:kunyumba_clean/controllers/settings_controller.dart';
import 'package:kunyumba_clean/controllers/voice_controller.dart';
import 'package:kunyumba_clean/controllers/voice_feeback_controller.dart';
import 'package:kunyumba_clean/home/splash_screen.dart';
import 'controllers/home_controller.dart';
import 'controllers/mttq_controller.dart';

void main() async {
  await GetStorage.init();
  // ✅ Initialize controllers here
  WidgetsFlutterBinding.ensureInitialized();
  Get.put(VoiceFeedbackController());
  Get.put(HomeController());
  Get.put(MqttController());
  Get.put(VoiceController());
  Get.put(ScheduleController());
  Get.put(AuthController());
  Get.put(SettingsController()); // 👈 Add this next to other controllers

  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
```

```
      title: 'PANYUMBA',
      theme: ThemeData.dark().copyWith(
        scaffoldBackgroundColor: Colors.black,
        primaryColor: Colors.amber,
      ),
      home: const SplashScreen(),
      debugShowCheckedModeBanner: false,
    );
  }
}
```

The `main.dart` file serves as the **entry point** of the smart home automation mobile application. It initializes all essential controllers—such as for authentication, MQTT messaging, scheduling, voice control, and user settings—using the GetX dependency injection framework. It also sets up persistent local storage with `GetStorage`, ensures proper widget binding, and launches the app with a dark-themed interface. The app starts at the `SplashScreen`, and `GetMaterialApp` is used to provide reactive navigation and state management across the app, ensuring a smooth and structured user experience.

6.3.2 BACKEND

The backend of the Smart Home Automation System plays a pivotal role in orchestrating the core operational logic, ensuring seamless communication between the user-facing mobile application and the physical hardware components. Acting as the system's control center, the backend is built using **Python** and runs on a **Raspberry Pi**, a compact yet powerful computing device that serves as the hardware hub of the automation environment. It leverages the **MQTT protocol** to subscribe to specific topics and listen for commands issued from the Flutter mobile app—such as turning a light on or off, or executing a scheduled task. Upon receiving a message through the MQTT broker (e.g., Eclipse Mosquitto), the backend interprets the command and translates it into real-time actions by toggling **GPIO pins** connected to relay modules, which in turn control electrical appliances like lighting. Beyond basic device control, the backend also manages two-way communication by sending **status updates** back to the app, ensuring users receive immediate feedback regarding the current state of their devices. This real-time responsiveness is crucial for maintaining user confidence and system reliability. Additionally, the backend is equipped with logic for **connection monitoring**, **error

handling**, and **system recovery**, enabling it to detect failed operations or disconnections and respond accordingly. These features make the system robust against faults and ensure that the user experience remains consistent even under varying network conditions. By combining lightweight communication protocols with low-level hardware control, the backend provides a reliable, secure, and efficient platform for managing smart home operations.

```
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
# GPIO setup
RELAY_PIN = 26 # GPIO26, adjust based on your wiring
GPIO.setmode(GPIO.BCM)
GPIO.setup(RELAY_PIN, GPIO.OUT)
GPIO.output(RELAY_PIN, GPIO.LOW) # Default to OFF
# MQTT setup
BROKER = "192.168.1.100" # Replace with your Pi's IP address if acting as broker
TOPIC = "pi/topic"
def on_connect(client, userdata, flags, rc):
print("✅ Connected to MQTT Broker with result code " + str(rc))
client.subscribe(TOPIC)
def on_message(client, userdata, msg):
payload = msg.payload.decode()
print(f"📩 Received message: {payload}")
if payload == "LightOn":
GPIO.output(RELAY_PIN, GPIO.HIGH)
print("💡 Light turned ON")
elif payload == "LightOff":
GPIO.output(RELAY_PIN, GPIO.LOW)
print("💡 Light turned OFF")
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, 1883, 60)
try:
print("🚀 MQTT Client Running... Press CTRL+C to exit")
client.loop_forever()
except KeyboardInterrupt:
print("\n🛑 Exiting and cleaning up...")
finally:
GPIO.cleanup()
```

CHAPTER VII

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

In conclusion, this project stands as a comprehensive demonstration of how IoT technologies can be effectively harnessed to transform conventional living spaces into intelligent, automated environments that respond to user needs in real-time. Through the seamless integration of open-source tools such as Flutter, MQTT, and the Raspberry Pi, the Smart Home Automation System offers an efficient, affordable, and scalable solution for remote control of household lighting. The mobile application, developed using Flutter, provides a visually appealing, responsive, and easy-to-navigate interface, allowing users to interact with the system through traditional UI elements and voice commands. This approach ensures inclusivity and accessibility for users across varying levels of technical expertise. The system's core communication relies on the MQTT protocol—a lightweight, fast, and secure messaging standard that facilitates real-time interaction between the app and the backend running on the Raspberry Pi. The Python scripts operating on the Pi are responsible for interpreting commands, triggering the appropriate GPIO pins, and handling the actual switching of the relay module that controls the physical light bulb. One of the key strengths of the system lies in its modular and extensible architecture, which allows for future integration of additional devices such as fans, smart locks, motion sensors, or temperature monitoring units. The voice integration and biometric authentication further elevate the project's real-world applicability, enhancing both convenience and security. The inclusion of scheduling functionality—albeit within the scope of same-day operations—demonstrates how even simple automation can bring meaningful energy savings and improve daily routines. Moreover, the project employs robust software engineering practices, from agile development methodology and system testing metrics to clean UI/UX design and maintainable code architecture. The project also acknowledges the growing need for sustainable, energy-conscious living and offers a solution that reduces unnecessary energy consumption by ensuring appliances are only used when needed, whether controlled manually or via automation. The backend is built with fault tolerance and responsiveness in mind, capable of handling connectivity drops and user errors gracefully while providing consistent feedback to the user. The system not only addresses the key problems identified in traditional home setups—such as lack of remote access, inefficient manual control, and absence of feedback—but also

provides a blueprint for implementing similar systems in educational, prototyping, and even commercial settings. Its reliance on open-source tools significantly reduces the cost barrier, allowing students, hobbyists, and startups to replicate or build upon the foundation laid here. Overall, the successful completion of this project reflects a deep understanding of IoT systems, client-server communication, hardware integration, and user-centric application design. It validates how technology can be tailored to meet everyday needs without excessive complexity or cost. As IoT continues to grow and expand into more areas of daily life, systems like this one will pave the way for smarter, more efficient, and more connected homes. The Smart Home Automation System thus stands not just as a completed academic project, but as a real-world prototype with the potential for widespread application, adaptation, and impact in the evolving landscape of digital living.

7.2 FUTURE ENHANCEMENT

Looking ahead, the Smart Home Automation System offers a wide range of opportunities for future integration and enhancement that would significantly expand its functionality, intelligence, and real-world value. One of the most promising additions involves incorporating motion and ambient light sensors, which would allow the system to automatically turn lights on or off based on room occupancy or natural lighting conditions. This sensor-based automation would not only improve user convenience but also maximize energy efficiency by ensuring lights are used only when necessary. Another key area of development involves scaling the system to support multiple devices and appliances, such as fans, smart plugs, security cameras, and smart locks, all controlled through the same mobile application and MQTT infrastructure. This would transform the solution from a single-use lighting controller into a comprehensive smart home ecosystem. Additionally, integrating cloud-based services would open the door to remote access from anywhere in the world, along with real-time data logging, user analytics, and device usage reports. This would give users deeper insights into their energy habits while allowing for more secure, flexible control through cloud dashboards or companion web apps. Beyond hardware and connectivity, a powerful direction for future integration lies in the use of Artificial Intelligence (AI) and machine learning to introduce predictive automation. For example, the system could learn a user's daily routines and environmental preferences over time, then proactively adjust lighting or other devices without explicit input. It could also respond to changing conditions like weather forecasts, time of day, or even calendar events to intelligently manage home environments. Voice integration could also be enhanced with natural language processing to handle more complex, conversational commands.

These improvements would elevate the system from being a reactive tool to becoming a proactive, context-aware assistant. By continuing to build on its modular architecture and open-source foundation, this project is well-positioned for scalable evolution into a fully intelligent smart home framework—capable of delivering advanced automation, greater personalization, and deeper integration with the evolving Internet of Things ecosystem.

REFERENCES

1. D.Koutsouris Member “A new method for profile generation in an Internet of Things environment: An application in ambient assisted living” IEEE Internet of Things Journal,

2. Gomes, T. ; Centro Algoritmi - University of Minho, Portugal ; Pinto, S. ; Gomes, T. ; Tavares, A.” Towards an FPGA-based edge device for the Internet of Things”Emerging Technologies & Factory Automation (ETFA).IEEE Transactions on Industrial Electronics 10.1109/TIE.

3. Jeya Padmini, J.; Kashwan, K.R.” Effective power utilization and conservation in smart homes using IoT," in Computation of Power, Energy Information and Commuincation (ICCPEIC), 2015 International Conference on , vol., no., pp.0195-0199, 22-23 April 2015.

4. Jinsoo Han; Chang-sic Choi; Wan-Ki Park; Ilwoo Lee; Sang-Ha Kim,” Smart home energy management system including renewable energy based on ZigBee and PLC in 2014 .

5. José G. de Matos, Member, IEEE, Felipe S. F. e Silva, Student Member, IEEE, and Luiz A. de S. Ribeiro, Member, IEEE “Power Control in AC Isolated Microgrids with Renewable Energy Sources and Energy Storage Systems”

6. Mohanty, S.; Panda, B.N.; Pattnaik B.S., "Implementation of a Web of Things based Smart Grid to remotely monitor and control Renewable Energy Sources," in Electrical, Electronics and Computer Science (SCEECS),2014 IEEE Student Conference on vol no pp,1-5,1-2

7. Shiu Kumar “Ubiquitou Smart Home System Using Android Application” International Journal of Computer Networks & Communications (IJCNC) Vol.6, No.1, January

8. W.Huiyong, W. Jingyang, and H. Min, “Building a smart home system with WSN and service robot,” in Proc. 5th Int. Conf. Measuring Technol. Mechatronics Autom., Hong Kong, China.

9. "A Review on IoT Enabled Smart Systems for Energy Consumption Monitoring" by A. Sharma, A. Kumar, and A. Kumar - Journal of Electrical Engineering Education, 2020. 16.

10. "The Role of IoT in Modernizing Electric Utilities" by L. B. Lopes - IEEE Power and Energy Magazine, 2017. 17.

11. "Security Challenges in IoT-Based Smart Metering Systems" by S. Sultana, M. A. Al-Ali, and M. Othman - IEEE Access, 2020.