IE2090
Professional Engineering Practice and Industrial Management

2nd Year, 2nd Semester

Final Report

Smart Home Switch and Outlet Automation

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

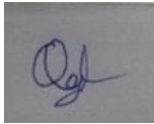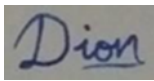Bachelor of Science Special Honors Degree in Information Technology

18/06/2024

Declaration

We declare that this project report or part of it was not a copy of a document done by any organization, university any other institute or a previous student project group at SLIIT and was not copied from the Internet or other sources.

Project Details

Project Title	Smart Home Switch and Outlet Automation
Project ID	PEP_04

Group Members

Reg. No	Name	Signature
IT22196088	Q JOYAL	
IT22169730	D M T N DISSANAYAKA	
IT22360496	D R WICKRAMA ARACHCHI	
IT22231628	SHARVAJEN S	

Abstract

Currently, the most crucial problems with existing smart home automation solutions are security, compatibility issues, energy efficiency, and user-friendliness. This product addresses these issues by providing energy monitoring and prediction capabilities to the user while ensuring the security and user-friendliness of the overall system. The final product consists of a hub for communication between internal systems and a private backend server, which users can access and manage through an app. The proposed solution enhances energy efficiency with monitoring capabilities using a bottom-up approach with the possibility of acting as a global solution for energy consumption and efficiency. Older and younger users will be able to easily familiarize themselves with system usage thanks to self-actuating switches with programmable capabilities.

Key Words: Self-actuating, Internet of Things, Module, Interface, WebSocket.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

- **IoT (Internet of Things):** A network of interconnected devices that communicate and exchange data with each other through the internet, enabling smart automation and control of home appliances.

- **UI (User Interface):** The interface through which users interact with the system, including the mobile and web applications.

- **ESP8266:** A low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability, used in the project for wireless communication between the smart devices and the central hub.

- **MQTT (Message Queuing Telemetry Transport):** A lightweight messaging protocol designed for small sensors and mobile devices, used in the project for efficient data transmission between devices.

- **TLS/SSL (Transport Layer Security/Secure Sockets Layer):** Cryptographic protocols designed to provide secure communication over a computer network, used to encrypt data transmission between the web application, mobile app, and smart devices.

- **AP (Access Point):** A device that allows wireless devices to connect to a wired network using Wi-Fi, used in the project to enable communication between the smart devices and the central hub.

- **RBAC (Role-Based Access Control):** A method of regulating access to a system based on the roles of individual users within an enterprise, implemented in the project to ensure secure access to system functionalities.

- **DBMS (Database Management System):** Software for creating and managing databases, specifically MySQL in this project, to store user data, device configurations, and usage logs.

- **JSON (JavaScript Object Notation):** A lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate, used for data exchange between the server and client applications.

- **ER Diagram (Entity-Relationship Diagram):** A diagram that shows the relationships of entity sets stored in a database, used in the project to design the database schema.

- **C/C++:** Programming languages used for developing the firmware for the Arduino and ESP8266 microcontrollers.

- **RESTful API (Representational State Transfer Application Programming Interface):** An architectural style for designing networked applications, used in the backend server to handle requests from the mobile and web applications.

- **IEC (International Electrotechnical Commission):** An international standards organization that prepares and publishes standards for electrical, electronic, and related technologies, ensuring electrical safety standards in the project components.

- **UL (Underwriters Laboratories):** A global safety certification company that establishes safety standards for products, ensuring compliance with electrical safety regulations in the project components.

- **APIs (Application Programming Interfaces):** Sets of functions and protocols that allow different software applications to communicate with each other, used extensively in the project to enable interaction between various components.

1. Introduction

1.1 Problem Statement

Currently, available smart home solutions contain a multitude of problems. At first the product was designed to solve the current global energy consumption and sustainability related issues, the integration incompatibility between available system components and security issues that persists in the smart home automation solutions. Upon further investigation into user interactions and market trends it was understood that the user-friendliness of the switches was also lacking. The following lead to the discovery of another insignificant problem, which is the lack of a proper solution for self-actuating programmable switches were identified. Self-actuation refers to the ability of the switch to work by itself without the need for signals or user interactions (Note: This is done using routines). Additionally, there is a lack of communication between the devices in Internet of Things (IOT) systems, which is one of the major underlining concepts of Internet of Things

1.2 Product Scope

1.2.1 Overall-Scope

In the following project the final product would include modules that can be installed within the household switches and a central hub which will be receiving control signals from the user and sensor data from the modules. Additionally, a mobile application can be downloaded by the user to control and monitor the switches. This application is connected to a private server or the backend which allows users to login and access the functionalities of the IOT system including controlling monitoring customization and prediction capabilities.

1.2.2 Individual Scope

Table 1.2.2-1: Individual Scope

Name	Work Allocation
Q Joyal	Mobile Application Development
	Implement predictions and consumption data
	Backend Server implementation
D R Wickrama Arachchi	Engineering module for mechanical toggling
	Setup current sensors
	Feasibility, safety, and compatibility testing
	Power Supply and component protection
Sharvajen S	Engineering module for mechanical toggling
	Setup current sensors
	Installation, hardware integration reliability and safety assessment.
	Power Supply and component protection
D M T N Dissanayaka	Central Hub implementation to receive data from modules and communicate with backend
	Communication protocols and system security
	Sensor and system calibrations for data accuracy

1.3 Project Report Structure

The rest of the report after the outlining of the introduction is divided into 5 sections, namely Methodology, Evaluation, Conclusion, References and Appendices. Methodology focusses on the Requirement analysis, design of the project, project implementation, and testing data. Evaluation underlines the overall impact of the research and future capabilities and adjustments. The conclusion section is an overall analysis and evaluation of the project.

2. Methodology

2.1 Requirements and Analysis

2.1.1 Functional Requirements

Table 2.1.1-1: Switch/Outlet Control Functionality

F1	Switch/Outlet Control Functionality
Input	User command from the mobile app or manual switch toggle
Process	Micro-controller interprets the command and controls the switch accordingly
Output	On/Off state change of the connected device
Description	The system must allow users to control the switches both manually and through the mobile app.

Table 2.1.1-2: Power Monitoring

F2	Power Monitoring
Input	Continuous power readings from the current sensor
Process	Microcontroller interprets current readings
Output	Real-time power data
Description	The system must monitor and provide real-time power information for user awareness.

Table 2.1.1-3: Wireless Communication

F3	Wireless Communication
Input	User commands from the mobile app
Process	Encode and transmit commands through the wireless module
Output	Successful reception of commands by switches or the central hub
Description	The system must support reliable wireless communication for seamless control.

Table 2.1.1-4: Central Hub Functionality

F4	Central Hub Functionality
Input	Wireless signals from switches and outlets
Process	Decode and interpret signals
Output	Execute corresponding actions, update status
Description	The central hub must efficiently manage communication with all connected devices and perform required actions.

Table 2.1.1-5: Routine and Automation

F5	Routine and Automation
Input	User-defined routines, schedules, or triggers
Process	System executes predefined actions based on configured routines
Output	Automated control of devices according to specified routines

Description	Users shall be able to create and schedule routines for automating tasks, such as turning lights on/off at specific times or adjusting thermostat settings based on occupancy patterns.

Table 2.1.1-6: User-Based Authentication

F6	User-Based Authentication
Input	User-provided credentials (username and password)
Process	System validates user credentials against stored records
Output	Authentication token or session identifier
Description	The system must authenticate users before granting access to system features, ensuring that only authorized individuals can interact with the system.

Table 2.1.1-7: Module Interface

F7	Module Interface
Input	Commands or data from the backend server
Process	Microcontroller interprets commands and interfaces with connected modules
Output	Control signals or data transmitted to connected modules
Description	The system shall provide a standardized interface for communication between the backend server and connected modules, facilitating seamless integration and control of devices within the smart home environment.

Table 2.1.1-8: WebSocket Communication

F8	WebSocket Communication
Input	Messages or commands from the backend server
Process	Backend server establishes WebSocket connections with client devices
Output	Real-time data updates or commands sent to client devices
Description	The system shall utilize WebSocket communication to enable real-time, bidirectional communication between the backend server and client devices, such as the ESP8266 hub, facilitating timely control and monitoring of devices within the smart home system.

Table 2.1.1-9: Data Visualization in UI

F9	Data Visualization in UI
Input	Data retrieved from backend server or external sources
Process	System processes and aggregates data for visualization
Output	Graphs, charts, or visual representations of data
Description	The system shall provide data visualization features in the user interface, allowing users to view and interpret data trends, patterns, and insights more easily. Visualization elements such as graphs, charts, and diagrams shall be used to present data in a visually appealing and intuitive manner, enhancing user understanding and decision-making.

Table 2.1.1-10: Controls in UI

F10	Controls in UI
Input	User interactions with the user interface
Process	System interprets user inputs and triggers corresponding actions
Output	Changes in device states or system behavior
Description	The user interface shall include interactive controls, such as buttons, sliders, and toggles, allowing users to perform actions such as turning devices on/off, adjusting settings, or initiating routines. Users shall be able to interact with controls intuitively and effectively, enabling seamless control and management of devices within the smart home system.

2.1.2 Nonfunctional Requirements

2.1.2.1 Performance Requirements

2.1.2.1.1 Response Time

- The web application should respond to user commands within 1-2 seconds under normal operating conditions.
- Real-time feedback should be provided for actions such as device control and energy consumption monitoring.

2.1.2.1.2 Scalability

- The system should support at least 100 simultaneous user connections without significant performance degradation.
- As the number of connected devices increases, the system should scale horizontally to maintain responsiveness and reliability.

2.1.2.1.3 Data Processing Speed

- Data from current sensors and other monitoring devices should be processed and updated in the system in real-time or near-real-time.
- The system should handle large volumes of data efficiently to provide accurate insights into energy consumption and device status.

2.1.2.2 Safety Requirements

2.1.2.2.1 Electrical Safety

- All components and modules of the system must comply with relevant electrical safety standards and regulations (e.g., IEC 60950, UL 60950).
- Safeguards should be implemented to prevent electrical hazards such as short circuits, overloads, and electrical fires.

2.1.2.2.2 Protection against Overheating

- Components, particularly the servomotors and ESP modules, should be designed to prevent overheating during operation.
- Temperature sensors and thermal protection mechanisms should be integrated to detect and mitigate overheating risks.

2.1.2.2.3 Prevention of Electrical Shock

- Ensure that all exposed electrical connections and components are properly insulated to prevent electric shock hazards.
- Users should be cautioned against tampering with internal components without proper training or authorization.

2.1.2.2.4 Fire Safety

- Use fire-resistant materials for the construction of system components, especially those near electrical circuits.
- Implement overcurrent protection devices (e.g., circuit breakers) to prevent overheating and potential fire hazards.

2.1.2.3 *Security Requirements*

2.1.2.3.1 Authentication Mechanisms

- Implement robust user authentication mechanisms, such as username/password authentication or multi-factor authentication, to ensure secure access to the system.
- Users should be required to authenticate themselves before accessing the web application or controlling devices.

2.1.2.3.2 Authorization Controls

- Define role-based access control (RBAC) to restrict access to system functionalities based on user roles and privileges.
- Only authorized users should be allowed to perform administrative tasks or access sensitive information.

2.1.2.3.3 Data Encryption

- Encrypt all data transmission between the web application, mobile app, and smart devices using industry-standard encryption protocols (e.g., TLS/SSL).
- Data stored in the system's database should be encrypted to protect against unauthorized access in case of a data breach.

2.1.2.3.4 Secure Device Communication

- Ensure that communication between the central hub and smart devices (e.g., switches, outlets, sensors) is encrypted and authenticated to prevent unauthorized access or tampering.
- Use secure communication protocols (e.g., MQTT over TLS) to transmit data between devices and the central hub securely.

2.1.2.4 *Software Quality Attributes*

2.1.2.4.1 Reliability

- The system should operate consistently without unexpected crashes or failures.
- Smart switches and outlets should reliably respond to user commands and automation schedules.

2.1.2.4.2 Maintainability

- The codebase should be well-structured and documented to facilitate future updates and modifications.
- Modular design principles should be followed to allow for easy addition of new features or enhancements.

2.1.2.4.3 Usability

- The mobile app and web interface should have an intuitive user interface, with clear navigation and controls.
- User interactions should be straightforward and easy to understand, catering to users of all technical levels.

2.1.2.4.4 Compatibility

- The system should be compatible with a wide range of mobile devices (iOS and Android) and web browsers (Chrome, Firefox, Safari, etc.).
- Compatibility with existing smart home ecosystems and protocols (e.g., Zigbee, Z-Wave) should be ensured for seamless integration.

2.1.2.4.5 Scalability

- The system architecture should be designed to scale horizontally and vertically to accommodate a growing number of users and devices.
- Performance should not degrade significantly as the user base and device count increase.

2.1.2.4.6 Performance

- The system should respond promptly to user commands, with minimal latency in device control and data retrieval.
- Data processing tasks, such as power consumption monitoring and predictive analytics, should be executed efficiently to minimize processing time.

2.1.2.4.7 Security

- Data transmission between the mobile app, web interface, and smart devices should be encrypted to prevent unauthorized access or interception.

- Role-based access control mechanisms should be implemented to ensure that users only have access to the devices and functionalities they are authorized to use.

2.1.2.4.8 Testability

- The system should be designed with testability in mind, allowing for thorough testing of individual components, modules, and system integrations.
- Automated testing frameworks and tools should be utilized to streamline the testing process and ensure comprehensive test coverage.

2.1.2.5 Business Rules

2.1.2.5.1 User Roles and Permissions

- Administrators have full access to system settings, user management, and device configurations.
- Standard users can control switches and outlets assigned to them but cannot modify system settings or add new users.
- Guests may have limited access to certain devices or functionalities based on predefined permissions set by administrators.

2.1.2.5.2 Device Ownership

- Each device (switch or outlet) can only be owned by one user at a time.
- Device ownership can be transferred from one user to another by administrators or device owners themselves.

2.1.2.5.3 Routine Execution

- Routines can only be created, edited, or deleted by users with appropriate permissions, typically administrators or device owners.
- Routines should adhere to a set schedule and should not interfere with other routines or manual control actions.

2.1.2.5.4 Energy Monitoring and Reporting

- Users should have access to their own energy consumption data and reports, but access to aggregated data across multiple users may be restricted to administrators.
- Energy consumption data should be updated in real-time or with minimal delay to provide accurate insights for users.

2.1.2.5.5 Security and Privacy

- User authentication is required for accessing the system, and strong password policies should be enforced.
- Personal data collected for user accounts or device configurations should be handled in accordance with relevant privacy regulations and industry standards.

2.1.2.5.6 Support and Maintenance

- Technical support requests should be directed to designated support channels, and response times should be defined based on the severity of the issue.
- Regular system maintenance, including updates and patches, should be scheduled during off-peak hours to minimize disruptions for users.

2.1.2.5.7 Compliance and Regulations

- The system should comply with applicable laws, regulations, and industry standards related to data security, electrical safety, and consumer privacy.
- Any changes to regulations or standards should be promptly reviewed, and system updates should be implemented to ensure continued compliance.

2.1.3 Use Case Diagram

The use case diagram below illustrates the interactions between users and the system, outlining various scenarios and user roles.
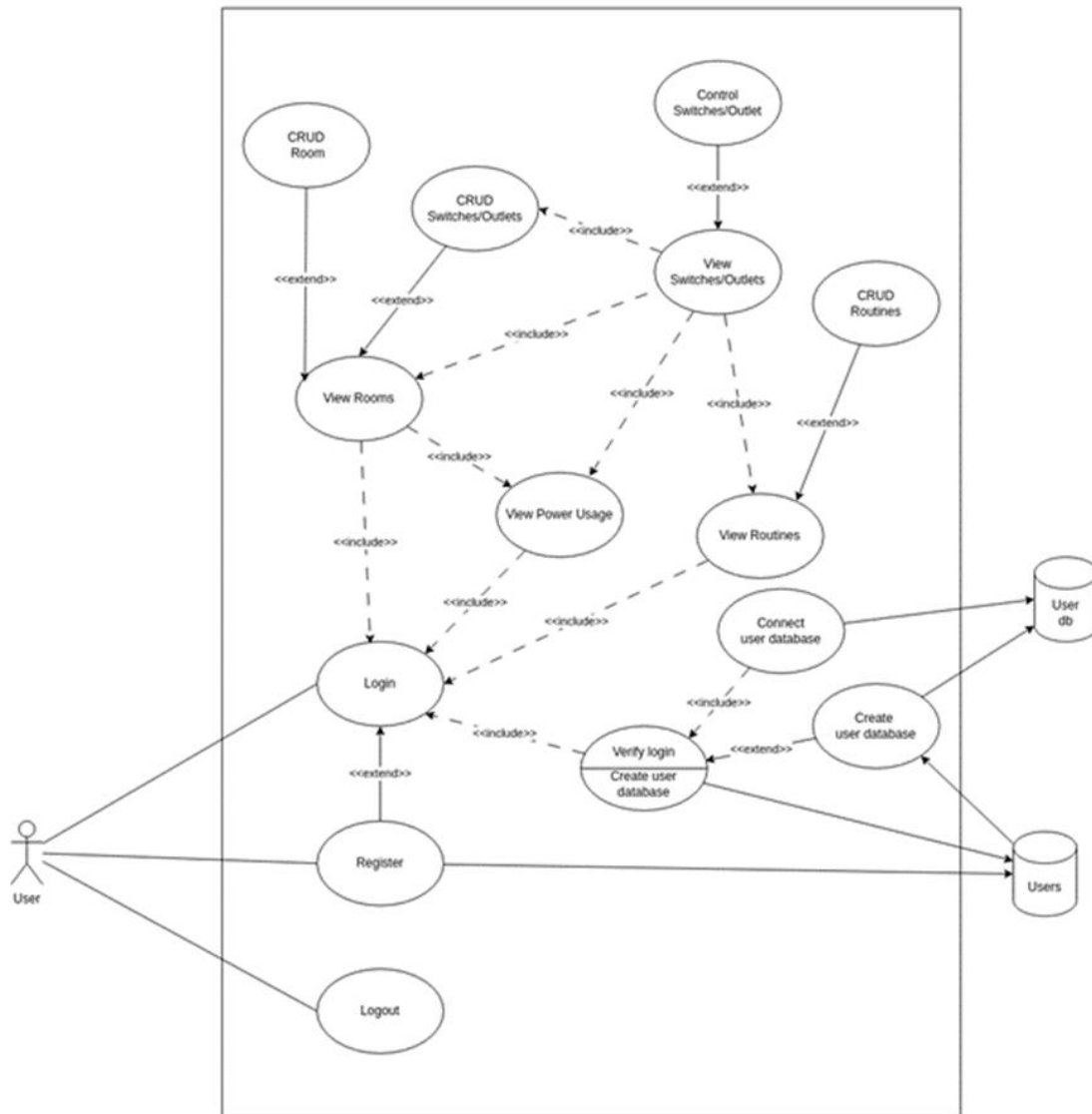
Figure 2.1.3.1: Use Case Diagram

2.1.4 Activity Diagram

The activity diagram below provides a step-by-step workflow for how a user interacts with the system to control the switch and outlet.
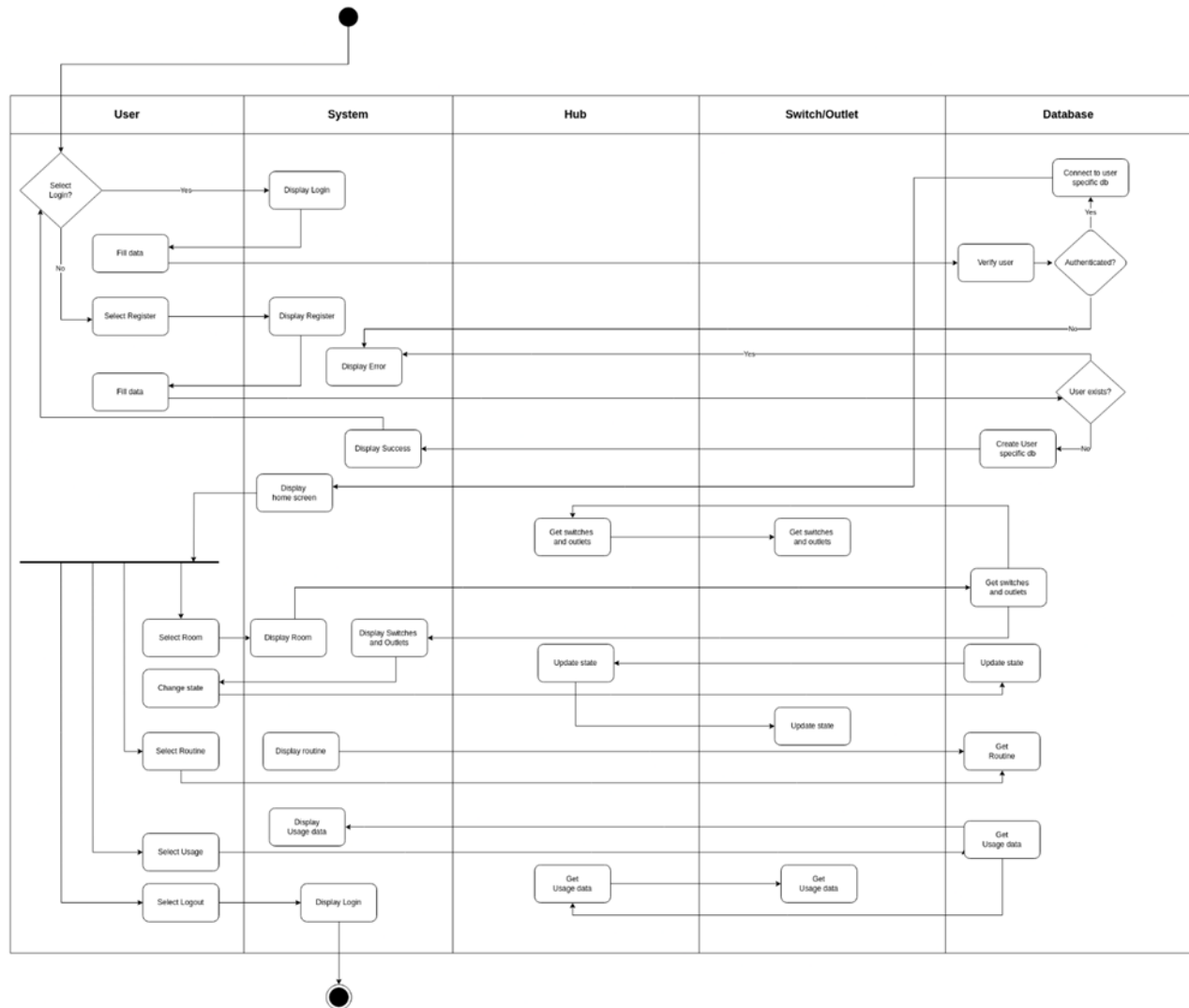
Figure 2.1.4.1: Activity Diagram

2.2 Design

2.2.1 High Level Architecture Diagram
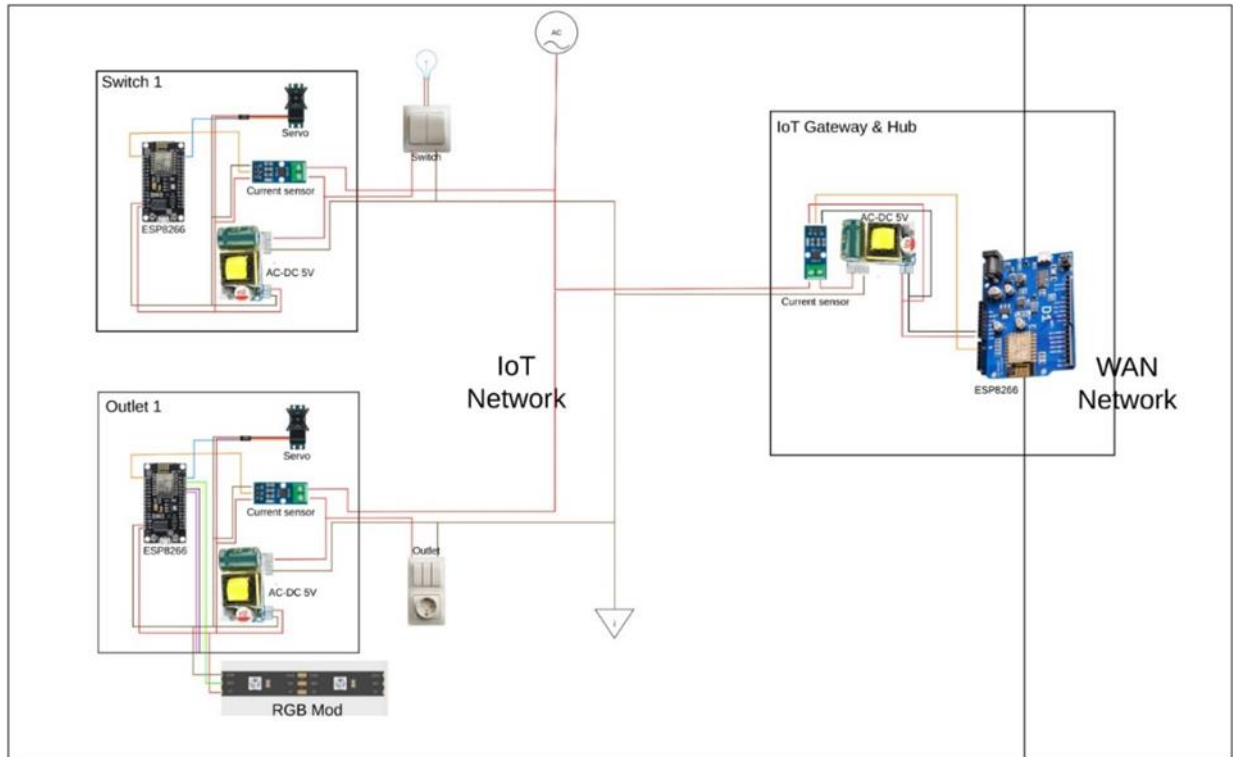
Figure 2.2.1.1: High Level Architecture Diagram

2.2.2 ER Diagram

The ER diagram represents the entities in the database and the relationships between them.
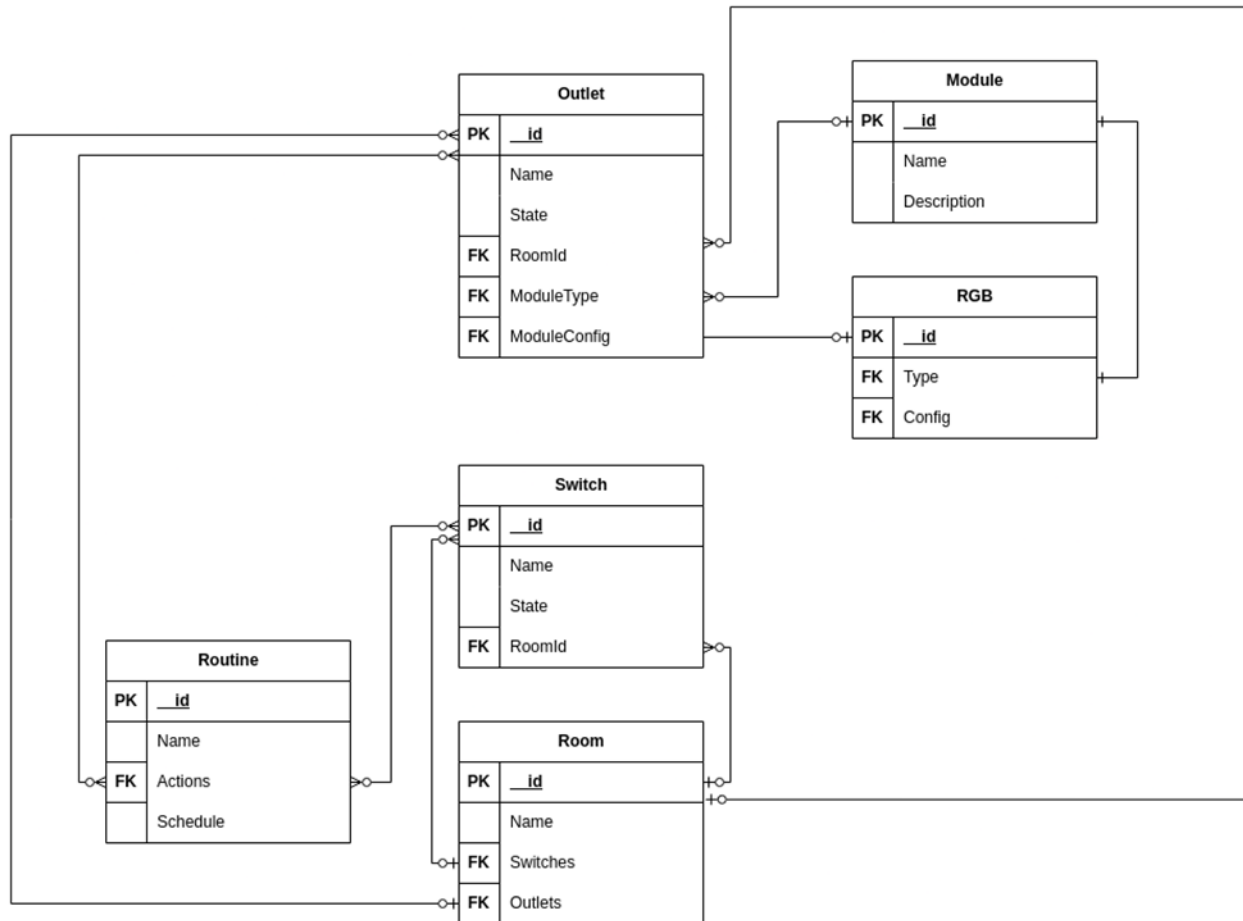
Figure 2.2.2.1: ER Diagram

2.2.3 User Interface

The user interface for the mobile application includes the following screens:
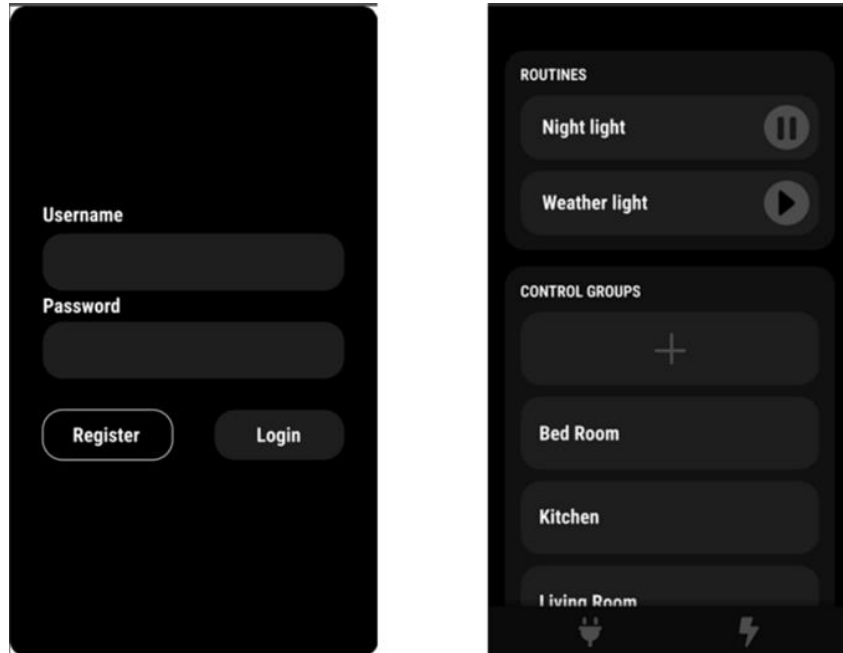
Figure 2.2.3.1: Login and Home Screen
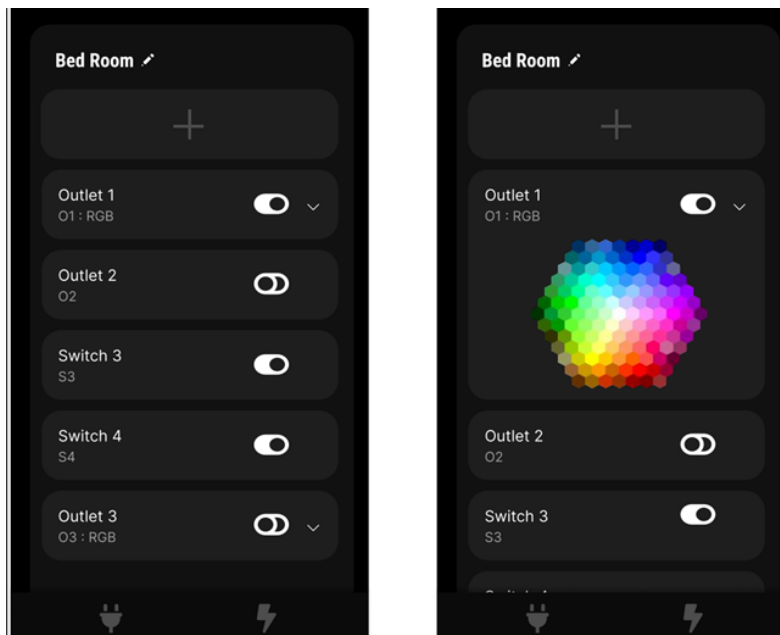
Figure 2.2.3.2: Control Group (Room) Screen
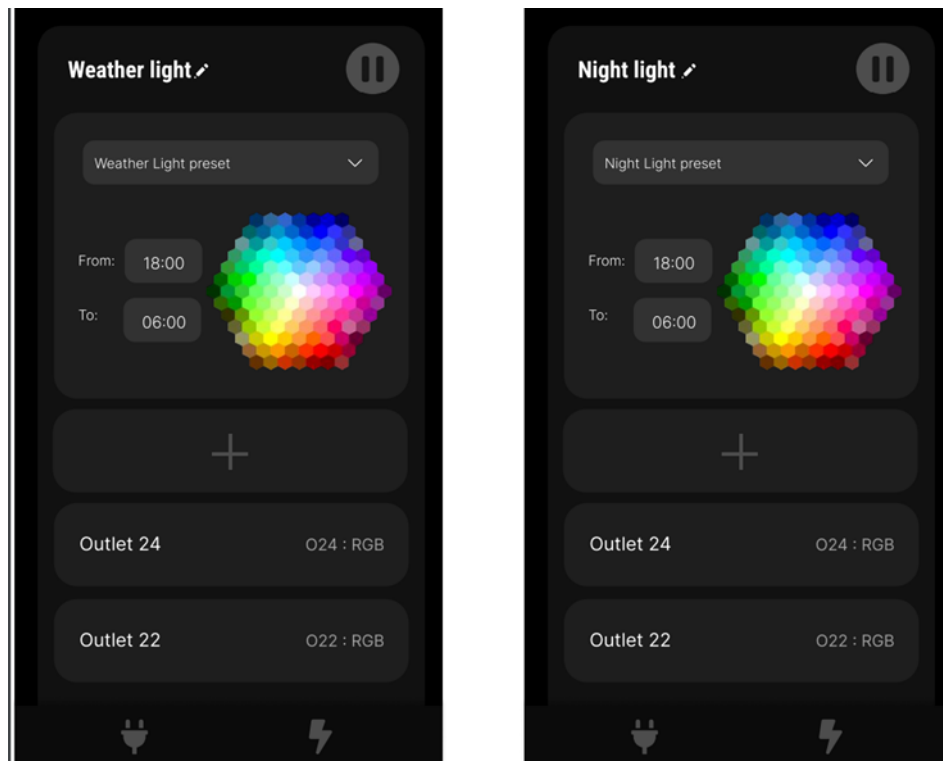
Figure 2.2.3.3: Preset/Routine Screen
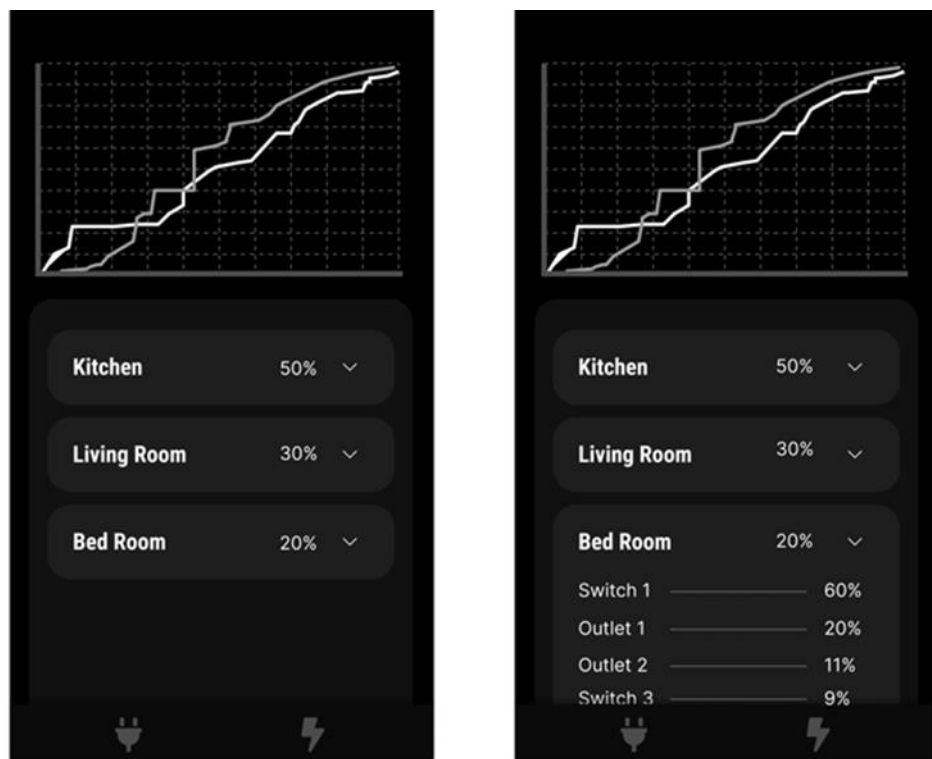
Figure 2.2.3.4: Statistic Screen

2.3 Implementation

2.3.1 Major Module Structures

2.3.1.1 User Interface (UI) Module

The mobile application, developed using React Native for cross-platform compatibility on iOS and Android, allows users to control and monitor their smart devices, while the web application, built with React.js, offers device control, energy consumption visualization, and scheduling routines.

2.3.1.2 Central Hub Module

The Arduino microcontroller manages communication between smart devices and the server, using MQTT for lightweight messaging and real-time updates.

2.3.1.3 Smart Device Module

The Smart Device Module comprises ESP8266 Wi-Fi modules, voltage sensors, relays, and servo motors, and a custom C/C++ firmware for device operations and Wi-Fi communication.

2.3.1.4 Backend Server Module

The backend server module utilizes Node.js for efficient I/O operations and a RESTful API for handling requests from mobile and web applications.

2.3.1.5 Database Module

MySQL is a reliable and efficient database module, managing user data, device configurations, and usage logs, while its schema optimizes query performance and ensures data integrity.

2.3.2 Development Tools and Reusable Code

2.3.2.1 Development Environment

Visual Studio Code is utilized for its robust features, Git for source code management and collaboration, and Jenkins for automated testing and deployment processes.

2.3.2.2 Reusable Code

The code utilizes utility libraries for common functionalities like logging and error handling, and third-party libraries like Express.js for server-side logic and Axios for front-end HTTP request handling.

2.3.3 Database Management System (DBMS)

MySQL was the chosen DBMS due to its reliability, performance, and scalability, making it ideal for real-time status updates and heavy-read operations.

2.3.4 Implementation Languages

The frontend for mobile applications utilizes React Native, while web applications utilize React.js. The backend is composed of Node.js and Express.js for server-side operations. Used C/C++ for firmware development on Arduino and ESP8266 microcontrollers.

2.3.5 Special Algorithms

The implementation includes several special algorithms for tasks such as predictive analytics for energy consumption. Detailed code for these algorithms is included in the appendices.

2.3.5.1 Example: Predictive Energy Consumption Algorithm

```
BEGIN
    // Step 1: Load Historical Data
    historical_data <- LOAD historical energy consumption data
    // Step 2: Preprocess Data
```

```
    CONVERT timestamps in historical data to datetime format
    EXTRACT day_of_week, month, hour from timestamps
    CREATE dataset with features (day_of_week, month, hour) and target (energy consumption)

    // Step 3: Split Data into Training and Testing Sets
    SPLIT dataset into training_set (80%) and testing_set (20%)

    // Step 4: Train the Predictive Model
    model <- INITIALIZE linear regression model
    TRAIN model using training_set (features: day_of_week, month, hour; target: energy
consumption)
    EVALUATE model using testing set to ensure accuracy

    // Step 5: Prepare Future Data for Prediction
    future_timestamps <- GENERATE future timestamps for prediction
    EXTRACT day_of_week, month, hour from future_timestamps
    future_data <- CREATE dataset with future features (day_of_week, month, hour)

    // Step 6: Predict Future Energy Consumption
    future_predictions <- PREDICT energy consumption using model and future_data

    // Step 7: Present Predictions
    OUTPUT future_predictions
    VISUALIZE predictions (e.g., plot predictions against future timestamps)
END
```

2.3.6 User Interface Designs

The UI design focuses on simplicity and ease of use, ensuring a seamless user experience.

- **Login Screen:** Users enter their credentials to access the system.
- **Dashboard:** Displays an overview of all connected devices, energy consumption trends, and quick actions for device control.
- **Device Control Screen:** Detailed controls for each device, including on/off toggles, scheduling, and status monitoring.
- **Settings Screen:** Allows users to manage their profiles, configure system settings, and access help resources.

2.4 Testing

2.4.1 Login Functionality

Table 2.4.1-1: Login-1A

Test Scenario ID	Login-1
Test Case ID	Login-1A
Description	User logs in with valid credentials
Pre-requisite	User account exists with valid credentials
Steps	1. Launch the application 2. Enter valid email and password 3. Click on login button
Expected Output	User is redirected to the home page
Actual Output	User is redirected to the home page
Test Result	Pass

Table 2.4.1-2: Login-1B

Test Scenario ID	Login-1
Test Case ID	Login-1B
Description	User tries to log in with invalid credentials
Pre-requisite	None
Steps	1. Launch the application 2. Enter invalid email or password 3. Click on login button
Expected Output	Error message is displayed indicating invalid credentials
Actual Output	Error message is displayed indicating invalid credentials
Test Result	Pass

2.4.2 Device Control (Switches)

Table 2.4.2-1: Switch-1A

Test Scenario ID	Switch-1
Test Case ID	Switch-1A
Description	User turns on a smart switch via the app
Pre-requisite	Smart switch is installed and configured
Steps	1. Open the mobile app 2. Navigate to the switch control screen 3. Turn on the switch
Expected Output	The switch is turned on
Actual Output	The switch is turned on
Test Result	Pass

Table 2.4.2-2: Switch-1B

Test Scenario ID	Switch-1
Test Case ID	Switch-1B
Description	User turns off a smart switch via the app
Pre-requisite	Smart switch is installed and configured
Steps	1. Open the mobile app 2. Navigate to the switch control screen 3. Turn off the switch
Expected Output	The switch is turned off
Actual Output	The switch is turned off
Test Result	Pass

2.4.3 Device Control (Outlets)

Table 2.4.3-1: Outlet -1A

Test Scenario ID	Outlet-1
Test Case ID	Outlet-1A
Description	User turns on a smart outlet via the app
Pre-requisite	Smart outlet is installed and configured
Steps	1. Open the mobile app 2. Navigate to the outlet control screen 3. Turn on the outlet
Expected Output	The outlet is turned on
Actual Output	The outlet is turned on
Test Result	Pass

Table 2.4.3-2: Outlet-1B

Test Scenario ID	Outlet-1
Test Case ID	Outlet-1B
Description	User turns off a smart outlet via the app
Pre-requisite	Smart outlet is installed and configured
Steps	1. Open the mobile app 2. Navigate to the outlet control screen 3. Turn off the outlet
Expected Output	The outlet is turned off
Actual Output	The outlet is turned off
Test Result	Pass

2.4.4 Energy Monitoring

Table 2.4.4-1: Energy-1A

Test Scenario ID	Energy-1
Test Case ID	Energy-1A
Description	Accurate display of energy consumption data
Pre-requisite	Energy monitoring system is installed
Steps	1. Open the mobile app 2. Navigate to the energy monitoring screen
Expected Output	Energy consumption data is displayed accurately
Actual Output	Energy consumption data is displayed accurately
Test Result	Pass

Table 2.4.4-2: Energy-1B

Test Scenario ID	Energy-1
Test Case ID	Energy-1B
Description	Accurate forecasts of energy consumption
Pre-requisite	Predictive analytics feature is enabled
Steps	1. Open the mobile app 2. Navigate to the energy consumption forecast screen
Expected Output	Accurate energy consumption forecasts are displayed
Actual Output	Accurate energy consumption forecasts are displayed
Test Result	Pass

2.4.5 Routine Management

Table 2.4.5-1: Routine-1A

Test Scenario ID	Routine-1
Test Case ID	Routine-1A
Description	User creates a new routine
Pre-requisite	None
Steps	1. Open the mobile app 2. Navigate to the routine management screen 3. Create a new routine
Expected Output	New routine is created and saved
Actual Output	New routine is created and saved
Test Result	Pass

Table 2.4.5-2: Routine-1B

Test Scenario ID	Routine-1
Test Case ID	Routine-1B
Description	User edits an existing routine
Pre-requisite	Existing routine available
Steps	1. Open the mobile app 2. Navigate to the routine management screen 3. Edit an existing routine
Expected Output	Routine is edited and saved
Actual Output	Routine is edited and saved
Test Result	Pass

2.4.6 User Management

Table 2.4.6-1: User-1A

Test Scenario ID	User-1
Test Case ID	User-1A
Description	Admin adds a new user
Pre-requisite	Admin login credentials
Steps	1. Login as admin 2. Navigate to user management screen 3. Add a new user
Expected Output	New user is added
Actual Output	New user is added
Test Result	Pass

Table 2.4.6-2: User-1B

Test Scenario ID	User-1
Test Case ID	User-1B
Description	Admin deletes an existing user
Pre-requisite	Admin login credentials and existing user
Steps	1. Login as admin 2. Navigate to user management screen 3. Delete an existing user
Expected Output	User is deleted
Actual Output	User is deleted
Test Result	Pass

2.4.7 Device Status Update

Table 2.4.7-1: Status-1A

Test Scenario ID	Status-1
Test Case ID	Status-1A
Description	Real-time device status update
Pre-requisite	Devices are connected and configured
Steps	1. Open the mobile app 2. Control a device 3. Observe the status update
Expected Output	Status updates in real-time
Actual Output	Status updates in real-time
Test Result	Pass

Table 2.4.7-2: Status-1B

Test Scenario ID	Status-1
Test Case ID	Status-1B
Description	Consistent status between mobile and web app
Pre-requisite	Devices are connected and configured
Steps	1. Open the mobile app 2. Control a device 3. Check status on the web app
Expected Output	Status remains consistent
Actual Output	Status remains consistent
Test Result	Pass

2.4.8 Security

Table 2.4.8-1: Security-1A

Test Scenario ID	Security-1
Test Case ID	Security-1A
Description	Encryption of data transmission
Pre-requisite	Secure connection configured
Steps	1. Launch the mobile app 2. Perform a network capture 3. Analyze the data packet
Expected Output	Data is encrypted
Actual Output	Data is encrypted
Test Result	Pass

Table 2.4.8-2: Security-1B

Test Scenario ID	Security-1
Test Case ID	Security-1B
Description	Enforcement of role-based access controls
Pre-requisite	User roles defined
Steps	1. Login with different user roles 2. Attempt various operations
Expected Output	Access is restricted based on roles
Actual Output	Access is restricted based on roles
Test Result	Pass

3. Evaluation

3.1 Assessment of Project Results

3.1.1 Electricity Cost Reduction

Simulations and internal testing indicated a significant potential for electricity cost savings. The smart switches and outlets enabled users to avoid high-cost electricity usage times, leading to lower overall electricity bills. Detailed analysis of pre- and post-simulation costs confirmed the projected savings.

3.1.2 User Adoption and Engagement

High adoption and engagement levels were observed among internal users. Usage metrics indicated frequent interactions with the mobile application, with users actively monitoring and controlling their energy consumption. Daily usage logs and feature engagement statistics showed regular utilization of predictive analytics and control functionalities.

3.1.3 Energy Usage Optimization

The system effectively optimized energy usage patterns in the simulations. Users were able to adjust their consumption habits to avoid peak rates, leading to overall energy savings. Graphical data from the simulations illustrated a clear shift in energy usage behavior, particularly during peak hours.

3.1.4 User Satisfaction

Feedback from the internal team was overwhelmingly positive. Users found the mobile application intuitive and appreciated the insights into their energy consumption. Surveys and feedback sessions revealed high levels of satisfaction, with users highlighting the ease of use and the value of cost prediction and energy monitoring features.

3.1.5 System Reliability and Performance

The system demonstrated high reliability and performance during simulations and internal testing. Performance logs recorded minimal downtime and quick resolution of any encountered issues, ensuring consistent operation. Metrics such as system uptime, response times, and error rates were all within acceptable ranges, demonstrating robust performance.

3.2 Lessons Learned

3.2.1 User-Centric Design

Ensuring a user-friendly design is essential for adoption and satisfaction. The positive feedback from internal testing highlighted the importance of an intuitive interface. Future projects should prioritize user feedback in the design process to ensure the system meets user needs and expectations effectively. Regular usability testing and incorporation of user suggestions are recommended.

3.2.2 Effective Communication

Clear and concise communication about the system's benefits and functionalities enhances user engagement and adoption. Many users became more engaged after receiving detailed explanations of the system's features. Improving communication strategies with comprehensive tutorials, guides, and regular updates is essential. Utilizing multiple channels, such as emails, in-app notifications, and video tutorials, can help reach users effectively.

3.2.3 Robust Testing

Extensive testing, even within a controlled environment, is crucial to identify and resolve potential issues before broader deployment. Internal testing revealed minor bugs that were quickly addressed. Implementing comprehensive testing phases and gathering diverse user feedback are necessary to ensure the system is robust and meets various needs.

3.2.4 Data Privacy Concerns

Addressing data privacy concerns is vital for gaining user trust. Internal feedback underscored the importance of transparent data handling practices. Strengthening data privacy measures and clearly communicating these practices to users can build and maintain trust. Implementing robust security protocols and ensuring compliance with relevant data protection regulations are recommended.

3.2.5 Scalability Challenges

Planning for scalability from the beginning is essential to manage resources effectively and accommodate a larger user base. Internal simulations showed potential bottlenecks in scaling. Developing a scalable infrastructure and planning for phased rollouts can help manage resources efficiently and ensure smooth scaling. Creating a detailed scalability plan with incremental upgrades is recommended.

3.3 Future Work

3.3.1 Custom Module Development

Future work will involve developing custom modules to enhance the versatility and functionality of the smart home system. These modules could include advanced lighting controls, security features, and integrations with other smart home devices. Developing these modules will require detailed planning and collaboration with technology partners to ensure seamless integration and functionality.

3.3.2 Improved Communication Protocols

Current reliance on web sockets for communication can be improved. Future iterations will explore more robust and efficient communication protocols, such as MQTT or CoAP, which are better suited for IoT applications due to their lightweight nature and reliability. Implementing these

protocols will enhance the system's performance, particularly in handling a large number of connected devices.

3.3.3 Blockchain Authentication

To enhance security and data integrity, the project will explore the implementation of blockchain-based authentication methods. This approach will ensure secure, decentralized, and tamper-proof authentication, providing users with a higher level of trust and security in managing their smart home devices. Collaboration with blockchain experts and integration with existing blockchain platforms will be necessary for this development.

3.3.4 System Enhancements

Enhance mobile app features and analytics to provide more detailed energy usage insights and personalized tips for energy savings. Conduct user workshops and surveys to gather detailed requirements and prioritize enhancements based on user feedback. Develop a roadmap for implementation, including features like detailed usage reports, personalized saving tips, and enhanced user interface elements.

3.3.5 Expansion Strategy

Prepare for broader testing with external users to validate the system's performance and benefits in real-world conditions. Partner with local utilities, community organizations, and pilot program participants to recruit a diverse group of external users. Secure funding for extended testing phases and develop a structured plan for scaling the project. Monitor performance closely and gather comprehensive feedback from these external users.

3.3.6 Continuous Monitoring

Establish a continuous monitoring framework to track system performance and user satisfaction over time. Implement automated reporting tools to collect data on system usage, performance

metrics, and user feedback. Schedule regular check-ins and surveys to gather ongoing input from users. Use this data to make continuous improvements and quickly address any emerging issues.

3.3.7 Educational Campaigns

Launch educational campaigns to raise awareness about energy efficiency and the benefits of the system. Develop and distribute educational materials, including brochures, online articles, and social media content. Host community workshops and webinars to educate users on energy-saving practices and how to maximize the benefits of the smart home system. Collaborate with schools and local organizations to spread awareness.

3.3.8 Research and Development

Invest in R&D to explore new technologies and features that could further enhance energy savings and user experience. Collaborate with academic institutions, technology partners, and industry experts to stay at the forefront of smart home innovations. Explore advancements in areas such as machine learning for better predictive analytics, integration with renewable energy sources, and advanced user interfaces. Allocate budget and resources for ongoing research projects.

3.3.9 Long-Term Sustainability

Develop strategies to ensure the long-term sustainability of the project, including exploring alternative funding models and maintaining high user engagement. Create a comprehensive sustainability plan that includes regular updates to the system, user incentives for continued engagement, and potential subscription models for premium features. Engage with stakeholders and potential investors to secure long-term funding. Monitor market trends and adapt the project strategy to remain relevant and effective.

4. Conclusion

4.1 Realization of Objectives/Goals

1. The project successfully developed a system that allows users to reduce electricity costs during peak rate periods by controlling and monitoring their power usage through a mobile application. The predictive analytics feature provided valuable insights into consumption patterns and cost estimates.

2. The mobile application developed during the project provided an intuitive interface for users to control switches and outlets, enhancing convenience and user control.

3. The modular approach allowed for the integration of additional functionalities, such as the RGB lighting module, ensuring the system's adaptability and scalability.

4. The system was designed to seamlessly integrate with existing home automation platforms, providing a versatile and scalable smart home solution.

4.2 Weaknesses/Limitations and Suggested Solutions

1. The project operated within a limited budget, which constrained the choice of components and potential future expansions. To address this, securing additional funding or partnerships with hardware manufacturers could help overcome these limitations.

2. The system's reliance on a stable Wi-Fi network could impact on its reliability. Implementing alternative communication protocols, such as Zigbee or Z-Wave, could mitigate this issue.

3. Ensuring that end-users have a basic understanding of smart home technology is crucial. Providing comprehensive user manuals, video tutorials, and customer support can enhance user adoption and satisfaction.

4.3 Benefits to the Client Organization

1. The project provides households with a practical solution to reduce their electricity expenses, aligning with the client's goal of addressing high unit rates.

2. By offering an innovative and user-friendly smart home solution, the client organization can position itself as a leader in the smart home technology market.

3. The modular and scalable nature of the system allows for future expansions and integrations, ensuring the client can meet evolving market demands and user needs.

4. The intuitive mobile application and seamless integration with existing home automation platforms enhance the overall user experience, promoting long-term customer satisfaction and loyalty.

5. References

[1] M. Jyotsna, P. Gabhane, M. Shradha Thakare, and M. Craig, "Smart Homes System Using Internet-of-Things: Issues, Solutions and Recent Research Directions." Available: https://www.irjet.net/archives/V4/i5/IRJET-V4I5531.pdf.

[2] "(PDF) Internet of Things (IoT): Definitions, Challenges, and Recent Research Directions," *ResearchGate*. https://www.researchgate.net/publication/320532203_Internet_of_Things_IoT_Definitions_Challenges_and_Recent_Research_Directions.

[3] S. Kumar, P. Tiwari, and M. Zymbler, "Internet of Things is a revolutionary approach for future technology enhancement: a review," *Journal of Big Data*, vol. 6, no. 1, Dec. 2019, doi: https://doi.org/10.1186/s40537-019-0268-2.

[4] Sharad, Shwetav & Kumar, Krishna, "IOT Based Home Automation Using Smart Switch," ResearchGate, Jan. 2020. [Online]. Available: https://www.researchgate.net/publication/338680972_IOT_Based_Home_Automation_Using_Smart_Switch.

[5] Ajibola, Olawale & Balogun, Abdulwahab, "Development of Smart Switch for Household Appliances Using Web-Based Technology," ResearchGate, Feb. 2019. [Online]. Available: https://www.researchgate.net/publication/331177086_Development_of_Smart_Switch_for_Household_Appliances_Using_Web-Based_TechnologyDevelopment_of_Smart_Switch_for_Household_Appliances_Using_Web-Based_Technology.

[6] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K.-C. Wang, "Review of Internet of Things (IoT) in Electric Power and Energy Systems," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 847–870, Apr. 2018, doi: https://doi.org/10.1109/jiot.2018.2802704.

[7] S. Wu et al., "Survey on Prediction Algorithms in Smart Homes," in IEEE Internet of Things Journal, vol. 4, no. 3, pp. 636-644, June 2017, doi: 10.1109/JIOT.2017.2668061. [Online]. Available: https://ieeexplore.ieee.org/document/7851022.

[8] J. Singh, T. Pasquier, J. Bacon, H. Ko and D. Eyers, "Twenty Security Considerations for Cloud-Supported Internet of Things," in IEEE Internet of Things Journal, vol. 3, no. 3, pp. 269-284, June 2016, doi: 10.1109/JIOT.2015.2460333. [Online]. Available: https://ieeexplore.ieee.org/document/7165580.

[9] M. Nitti, L. Atzori and I. P. Cvijikj, "Friendship Selection in the Social Internet of Things: Challenges and Possible Strategies," in IEEE Internet of Things Journal, vol. 2, no. 3, pp. 240-247, June 2015, doi: 10.1109/JIOT.2014.2384734. [Online]. Available: https://ieeexplore.ieee.org/document/6994231.

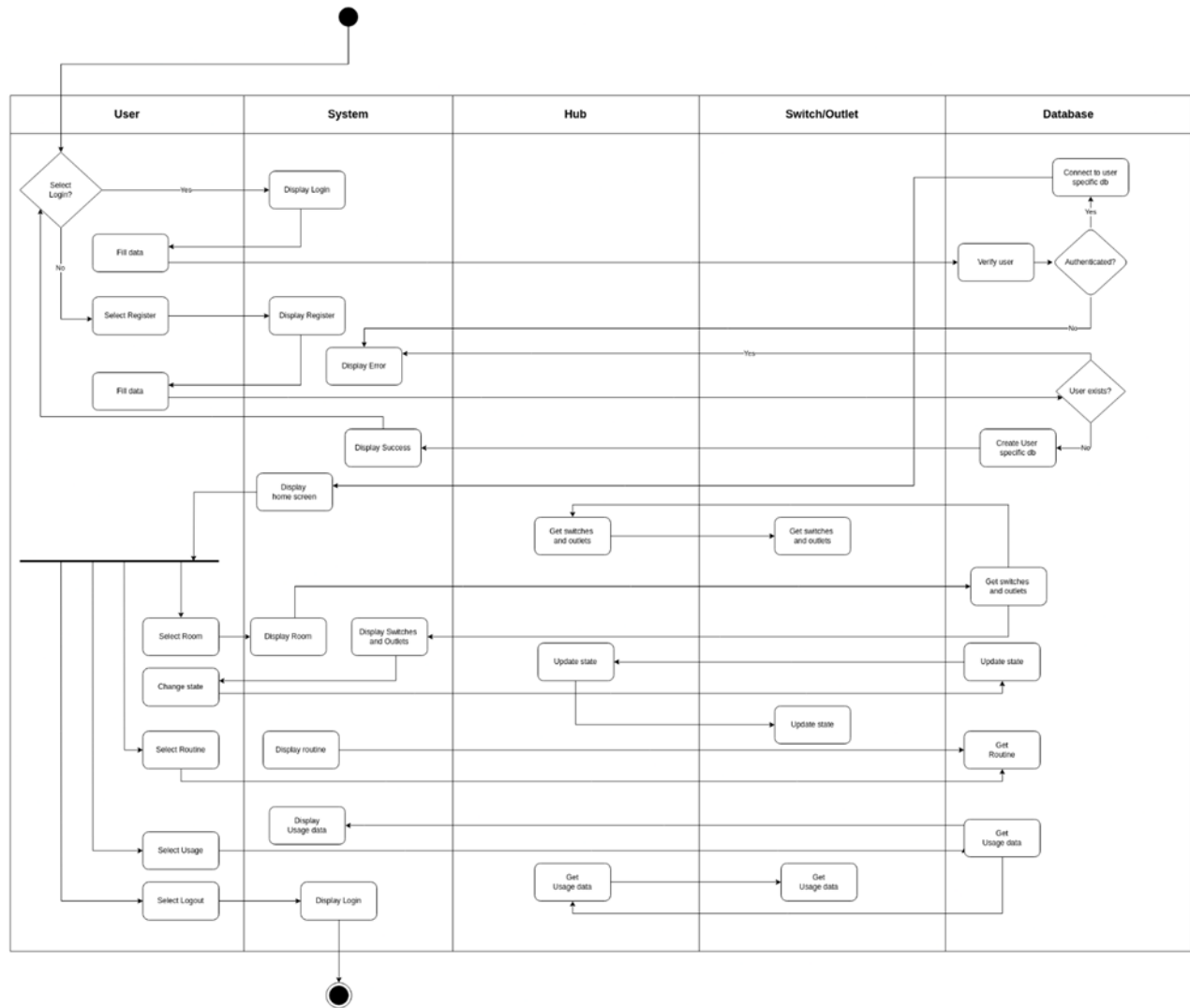# 6. Appendix A: Design Diagrams
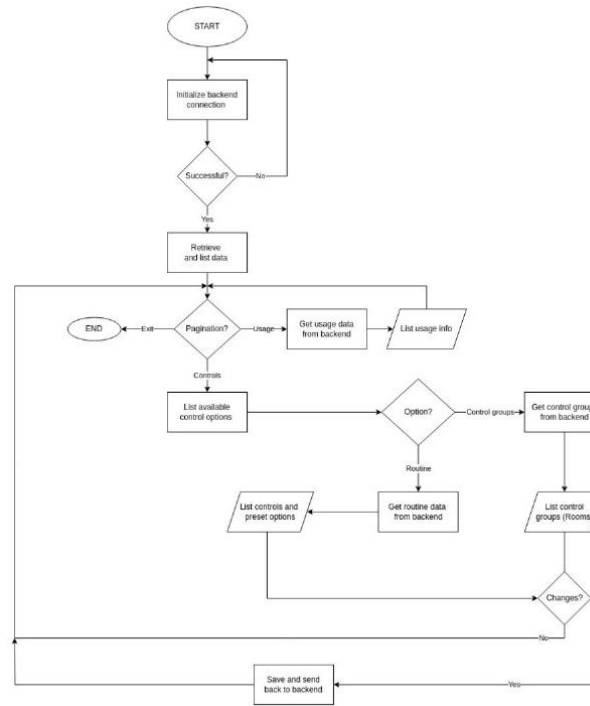

Figure 3.3.9.1: Activity Diagram

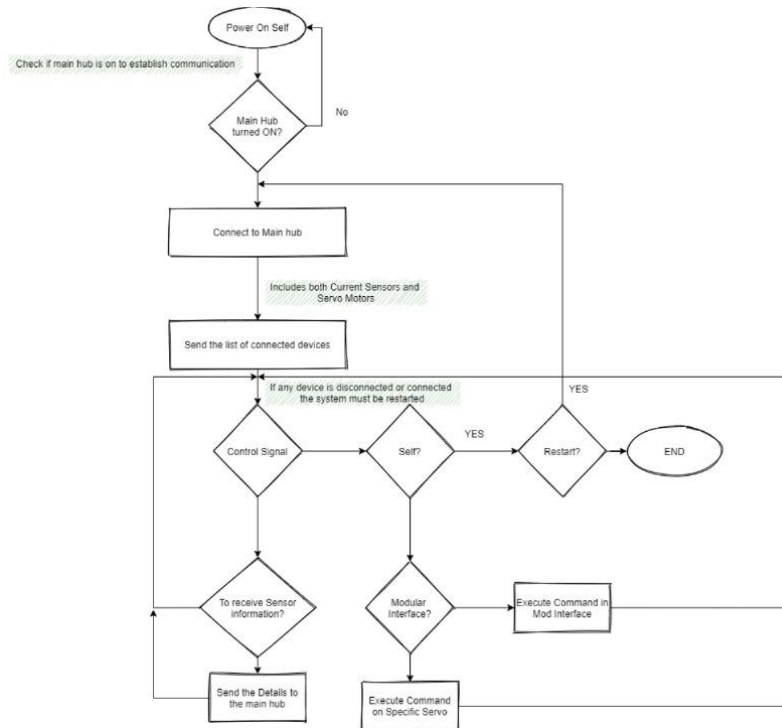*Figure 3.3.9.2: Mobile App Functionality*

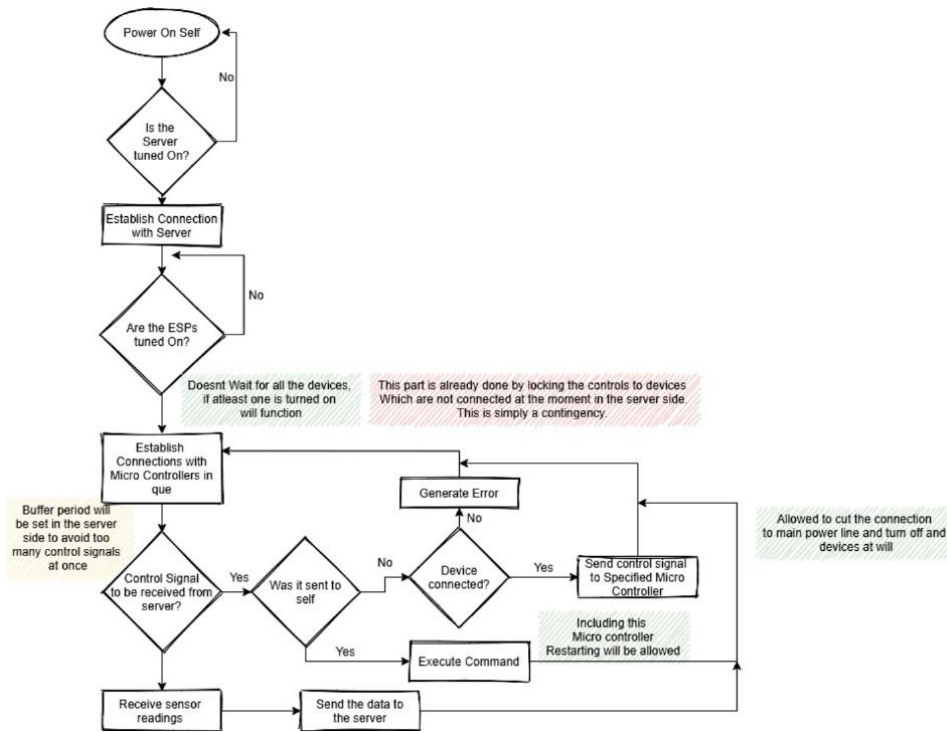*Figure 3.3.9.3: Outlet Integration Functionality*

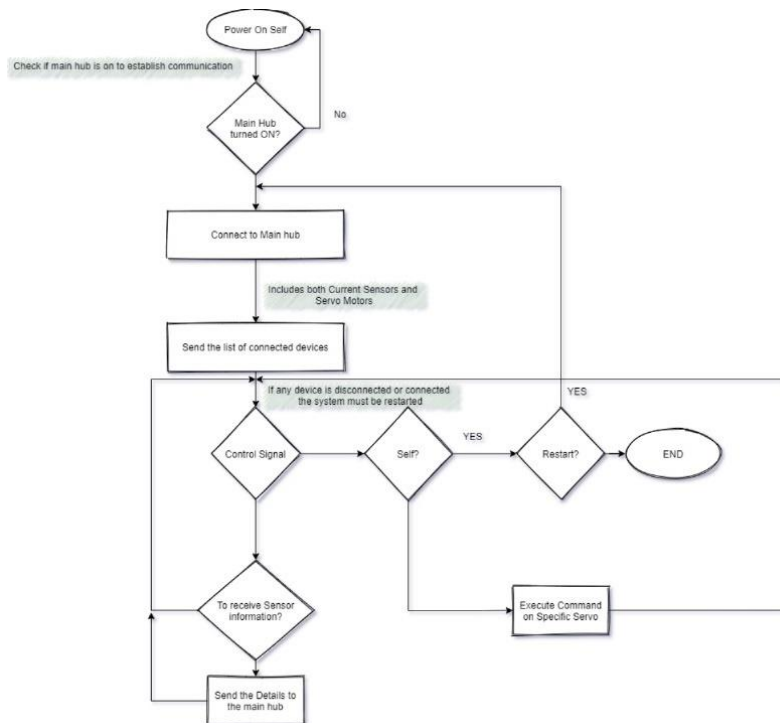*Figure 3.3.9.4: Central Hub Functionality*

Figure 3.3.9.5: Switch Integration Functionality

Figure 3.3.9.6: Gantt Chart
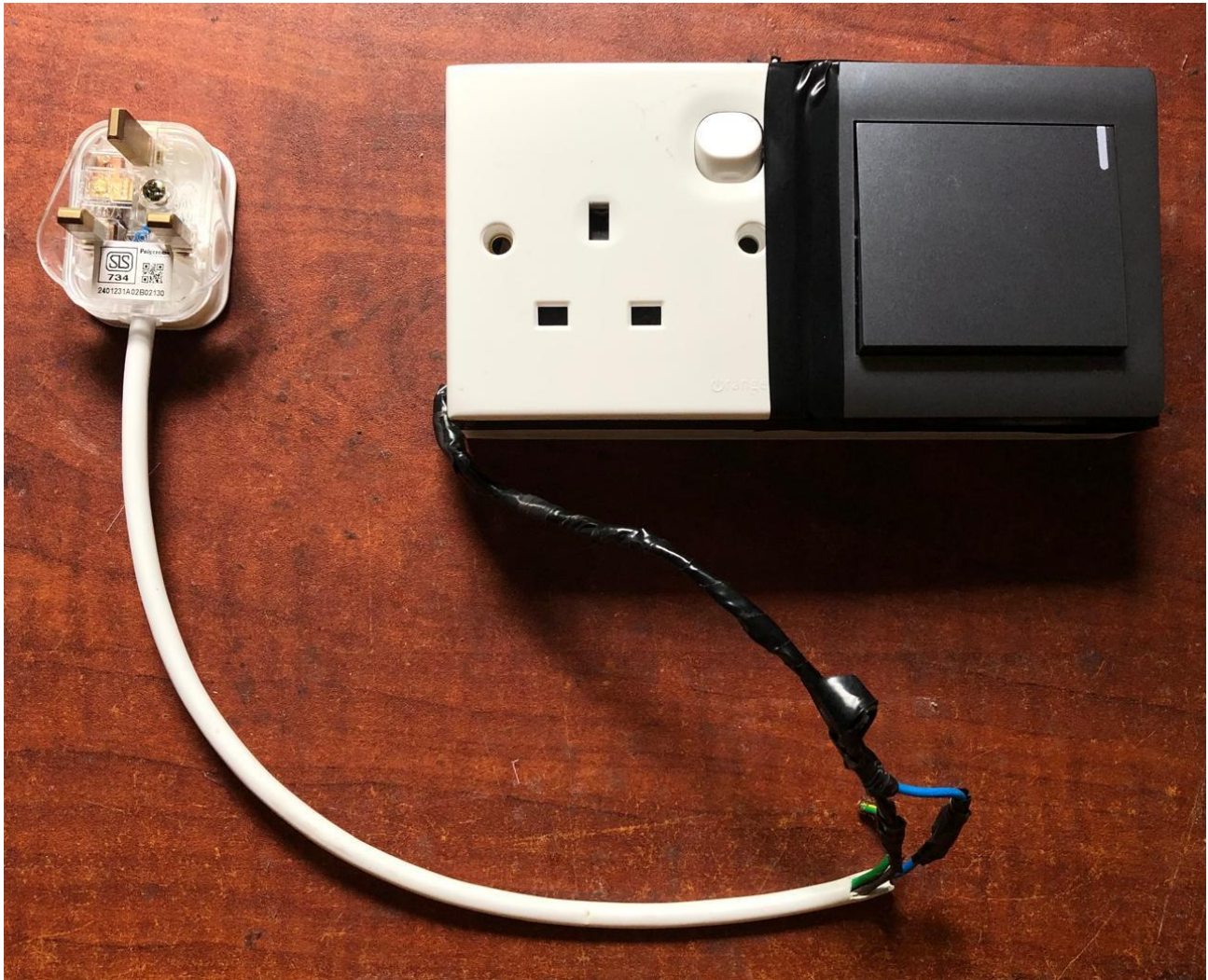
7. Appendix B: Test Results

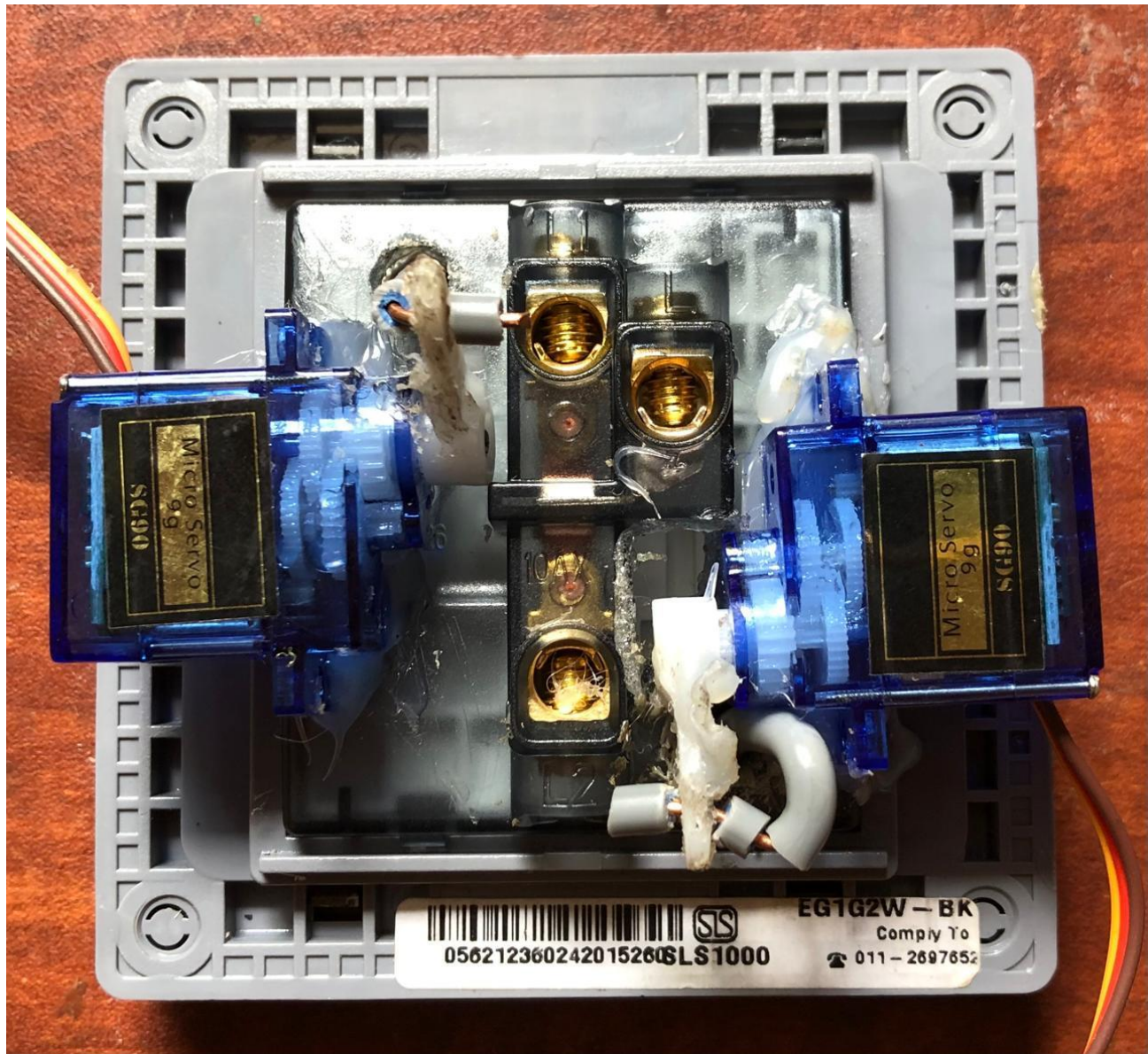*Figure 3.3.9.1: Combined Switch and Outlet*

Figure 3.3.9.2: Inside View of Switch

8. Appendix C: Selected Code Listings

8.1 Statistic Calculations

```
1.  const dailyTask = async (tenant) => {
2.    const now = new Date();
3.    const day = now.getDate();
4.    const month = now.getMonth() + 1;
5.    const year = now.getFullYear();
6.
7.    const Statistic = await getStatisticModel(tenant);
8.    const latestStat = await Statistic.findOne().sort({ createdAt: -1 });
9.
10.   if (!latestStat || latestStat.createdAt.getMonth() + 1 !== month ||
    latestStat.createdAt.getFullYear() !== year) {
11.     await createNewMonthDoc(tenant);
12.   }
13.
14.   const Room = await getRoomModel(tenant);
15.   const Switch = await getSwitchModel(tenant);
16.   const Outlet = await getOutletModel(tenant);
17.
18.   const rooms = await Room.aggregate([
19.     {
20.       $lookup: {
21.         from: 'switches',
22.         localField: 'switches',
23.         foreignField: '_id',
24.         as: 'switches'
25.       }
26.     },
27.     {
28.       $lookup: {
29.         from: 'outlets',
30.         localField: 'outlets',
31.         foreignField: '_id',
32.         as: 'outlets'
33.       }
34.     },
35.     {
```

```
36.        $addFields: {
37.          totalSwitchUsage: { $sum: '$switches.usage' },
38.          totalOutletUsage: { $sum: '$outlets.usage' }
39.        }
40.      },
41.      {
42.        $project: {
43.          totalUsage: { $add: ['$totalSwitchUsage', '$totalOutletUsage'] }
44.        }
45.      }
46.   ]);
47.
48.   let usageValue = rooms.reduce((sum, room) => sum + room.totalUsage, 0);
49.
50.   console.log(usageValue);
51.
52.   await setDailyUsage(tenant, day, usageValue);
53. };
54.
55. // Function to schedule daily tasks for each user
56. export const scheduleTasksForAllUsers = async () => {
57.   const User = await getUserModel();
58.   const users = await User.find();
59.
60.   users.forEach(user => {
61.     cron.schedule('0 0 * * *', () => dailyTask(user._id), {
62.       scheduled: true,
63.       timezone: "Asia/Colombo"
64.     });
65.   });
66. };
67.
68. const getMonthlyUsage = async (tenant, month, year) => {
69.   const Statistic = await getStatisticModel(tenant);
70.   const startDate = new Date(year, month - 1, 1);
71.   const endDate = new Date(year, month, 0);
72.
73.   const result = await Statistic.findOne({
74.     createdAt: { $gte: startDate, $lt: endDate }
75.   });
76.
```

```
77.   if (!result) {
78.     // If no result, return an array of zeros for the month
79.     const daysInMonth = new Date(year, month, 0).getDate();
80.     return Array(daysInMonth).fill(0);
81.   }
82.
83.   return result.usage.map(value => parseFloat(value.toString()));
84. };
85.
86. export const getStatisticData = async (tenant) => {
87.   const Room = await getRoomModel(tenant);
88.   const now = new Date();
89.   const currentMonth = now.getMonth() + 1;
90.   const currentYear = now.getFullYear();
91.   const previousMonth = currentMonth === 1 ? 12 : currentMonth - 1;
92.   const previousYear = currentMonth === 1 ? currentYear - 1 : currentYear;
93.
94.   const currentMonthUsage = await getMonthlyUsage(tenant, currentMonth, currentYear);
95.   const previousMonthUsage = await getMonthlyUsage(tenant, previousMonth, previousYear);
96.
97.   const currentMonthTotalUsage = currentMonthUsage.reduce((sum, usage) => sum + usage, 0);
98.
99.   const rooms = await Room.aggregate([
100.         {
101.             $lookup: {
102.               from: 'switches',
103.               localField: 'switches',
104.               foreignField: '_id',
105.               as: 'switches'
106.             }
107.         },
108.         {
109.             $lookup: {
110.               from: 'outlets',
111.               localField: 'outlets',
112.               foreignField: '_id',
113.               as: 'outlets'
114.             }
115.         },
116.         {
117.             $addFields: {
```

```
118.                totalSwitchUsage: { $sum: '$switches.usage' },
119.                totalOutletUsage: { $sum: '$outlets.usage' }
120.            }
121.          },
122.          {
123.            $project: {
124.              name: 1,
125.              switches: {
126.                $map: {
127.                  input: '$switches',
128.                  as: 'switch',
129.                  in: {
130.                    name: '$$switch.name',
131.                    usage: '$$switch.usage',
132.                    percentage: {
133.                      $cond: {
134.                        if: { $eq: [currentMonthTotalUsage, 0] },
135.                        then: 0,
136.                        else: { $multiply: [{ $divide: ['$$switch.usage',
    currentMonthTotalUsage] }, 100] }
137.                      }
138.                    }
139.                  }
140.                }
141.              },
142.              outlets: {
143.                $map: {
144.                  input: '$outlets',
145.                  as: 'outlet',
146.                  in: {
147.                    name: '$$outlet.name',
148.                    usage: '$$outlet.usage',
149.                    percentage: {
150.                      $cond: {
151.                        if: { $eq: [currentMonthTotalUsage, 0] },
152.                        then: 0,
153.                        else: { $multiply: [{ $divide: ['$$outlet.usage',
    currentMonthTotalUsage] }, 100] }
154.                      }
155.                    }
156.                  }
```

```
157.                    }
158.                  },
159.                  totalUsage: { $add: ['$totalSwitchUsage', '$totalOutletUsage'] }
160.                }
161.              }
162.          ]);
163.
164.          const roomData = rooms.map(room => ({
165.            roomName: room.name,
166.            switches: room.switches.map(switchData => ({
167.              name: switchData.name,
168.              usage: parseFloat(switchData.usage.toString()).toFixed(2),
169.              percentage: switchData.percentage.toFixed(2)
170.            })),
171.            outlets: room.outlets.map(outletData => ({
172.              name: outletData.name,
173.              usage: parseFloat(outletData.usage.toString()).toFixed(2),
174.              percentage: outletData.percentage.toFixed(2)
175.            }))
176.          }));
177.
178.          return { roomData, currentMonthUsage, previousMonthUsage };
179.        };
```

8.2 App Pages

```
1.      <WebSocketProvider>
2.        <NavigationContainer>
3.          <View style={{
4.              position: 'absolute',
5.              top: 0,
6.              bottom: 0,
7.              right: 0,
8.              left: 0,
9.              backgroundColor: '#000'}}
10.         />
11.         <Stack.Navigator>
12.           <Stack.Screen name='Login' component={Login} options={{ headerShown: false }} />
13.           <Stack.Screen name='Register' component={Register} options={{ headerShown: false
      }} />
```

```
14.          <Stack.Screen name='Navbar' options={{ headerShown: false }}>
15.            {props => (
16.              <Navbar />
17.            )}
18.          </Stack.Screen>
19.          <Stack.Screen name='Home' component={Home} options={{ headerShown: false }} />
20.          <Stack.Screen name='Routine' options={{ headerShown: false }}>
21.            {props => (
22.              <Routine id={props.route.params.routine} />
23.            )}
24.          </Stack.Screen>
25.          <Stack.Screen name='Room' options={{ headerShown: false }}>
26.            {props => (
27.              <Room id={props.route.params.room} />
28.            )}
29.          </Stack.Screen>
30.          <Stack.Screen name='Usage' component={Usage} options={{ headerShown: false }} />
31.        </Stack.Navigator>
32.      </NavigationContainer>
33.    </WebSocketProvider>
```

8.3 Hub Data Communication

```
1.  // WiFi Connect
2.  String ssid = "IOT_HUB_AP";
3.  String password = "AP_Password";
4.  String webSocketServerAddress = "192.168.4.1";
5.  const int webSocketServerPort = 81;
6.
7.  void setupWiFi()
8.  {
9.    WiFi.mode(WIFI_STA);
10.   WiFi.begin(ssid, password);
11.   while (WiFi.status() != WL_CONNECTED)
12.   {
13.     delay(500);
14.     Serial.print(".");
15.   }
16.   Serial.println("");
17.   Serial.println("WiFi connected");
```

```
18.    Serial.println("IP address: ");
19.    Serial.println(WiFi.localIP());
20. }
21.
22. // Access Point Start
23. String apSSID = "IOT_HUB_AP";
24. String apPassword = "RaspBerry";
25. WebSocketsServer webSocketData = WebSocketsServer(81);
26.
27. void setupAP() {
28.    WiFi.softAP(apSSID, apPassword);
29.
30.    Serial.println("Access Point started");
31.    Serial.print("IP Address: ");
32.    Serial.println(WiFi.softAPIP());
33. }
34.
35. // WebSockets Events
36. WebSocketsServer webSocketData = WebSocketsServer(81);
37. WebSocketsServer webSocketCommand = WebSocketsServer(82);
38.
39. void webSocketDataEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {
40.    switch(type) {
41.      case WStype_DISCONNECTED:
42.        Serial.printf("[%u] WebSocket disconnected", num);
43.        Serial.println();
44.        break;
45.      case WStype_CONNECTED:
46.        {
47.          IPAddress ip = webSocketData.remoteIP(num);
48.          Serial.printf("[%u] WebSocket connected from %d.%d.%d.%d", num, ip[0], ip[1],
   ip[2], ip[3]);
49.          Serial.println();
50.        }
51.        break;
52.      case WStype_TEXT:
53.        {
54.          IPAddress ip = webSocketData.remoteIP(num);
55.          Serial.printf("[%u] Received data from %d.%d.%d.%d: %s", num, ip[0], ip[1], ip[2],
   ip[3], payload);
56.          Serial.println();
```

```
57.        webSocketData.sendTXT(num, payload, length);
58.      }
59.      break;
60.   }
61. }
62.
63. void webSocketCommandEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {
64.   switch(type) {
65.     case WStype_DISCONNECTED:
66.       Serial.printf("[%u] WebSocket disconnected", num);
67.       Serial.println();
68.       break;
69.     case WStype_CONNECTED:
70.       {
71.         IPAddress ip = webSocketCommand.remoteIP(num);
72.         Serial.printf("[%u] WebSocket connected from %d.%d.%d.%d", num, ip[0], ip[1],
   ip[2], ip[3]);
73.         Serial.println();
74.       }
75.       break;
76.     case WStype_TEXT:
77.       {
78.         IPAddress ip = webSocketCommand.remoteIP(num);
79.         Serial.printf("[%u] Received command from %d.%d.%d.%d: %s", num, ip[0], ip[1],
   ip[2], ip[3], payload);
80.         Serial.println();
81.         webSocketCommand.broadcastTXT(payload);
82.       }
83.       break;
84.   }
85. }
```

8.4 Servo Logic

```
#include <Servo.h>
Servo myservo1, myservo2; // Create 2 Servo objects


// Define the servo pins
const int servoPin1 = D4;
const int servoPin2 = D1;
```

```
void setup() {
  Serial.begin(115200); // Initialize serial communication

  // Attach the servos to the servo pins
  myservo1.attach(servoPin1);
  myservo2.attach(servoPin2);

  // Move servos to "off" position initially
  myservo1.write(90);
  myservo2.write(0);
}

void loop() {
  // Move the servos to "off" position
  myservo1.write(90);
  myservo2.write(0);
  delay(3000);

  // Move the servos to "on" position
  myservo2.write(90);
  myservo1.write(0);
  delay(3000);
}
```

8.5 Outlet

```
1.       // Check Git for functional Code
2.       void currentSenseLogic(){
3.       float sensitivity = 0.66; // check sensitivity chart
4.       float mCurrent = 30; // 30A
5.       int noise = 3; // Normally 1, always calibrate
6.       int state[1]={0};
7.       float vOffset = 2.5; // some 5v
8.       float Reading;
9.       currentReading=((analogRead(A0)*5.0/1024.0)-vOffset)/sensitivity)+noise);
10.      if((currentReading>1) && (state[0]==0)){
11.      sendState();
12.      }
```