Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Computer Science & Engineering

Project Report
on
Smart Home Lighting Control Using NodeMCU and Mobile Application

BACHELOR OF TECHNOLOGY
(Computer Science and Engineering.)
-Specialization (Internet Of Thing and Cyber Security using Blockchain)

SUBMITTED BY:
Ranayudh Shukla (2420485)
Maanas Krishana (2420470)
Mayank Pradhan (2420473)
Abhinaw Singh Rawat (2420441)

Under the Guidance of
Ms. Abha Singh
Assistant Professor, CSE

In partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering

OCTOBER, 2025

Department of Computer Science & Engineering
Chandigarh Engineering College Jhanjeri
Mohali – 140307
(Batch: 2024-2028)

DECLARATION

We hereby declare that the report of the project entitled **“Smart Home Lighting Control Using ESP32 and Mobile Application”** has not been presented as a part of any other academic work to get any degree or certificate, except to Chandigarh Engineering College Jhanjeri, Mohali, for the fulfillment of the requirements for the degree of B.Tech in Computer Science & Engineering.

NAME OF STUDENT	**NAME OF THE MENTOR**
Maanas Krishana (2420470)	Ms. Abha Singh
Ranayudh Shukla (2420485)	Assistant Professor, CSE
Mayank Pradhan (2420473)	
Abhinaw Singh Rawat (2420441)	
Semester: 3rd	

Signature of the Head of Department

ACKNOWLEDGEMENT

It gives us great pleasure to deliver this report on Project-I, which we worked on for our B.Tech in Computer Science & Engineering 2nd year, which was titled **“Smart Home Lighting Control Using ESP32 and Mobile Application”**. We are grateful to our university for presenting us with such a wonderful and challenging opportunity. We also want to convey our sincere gratitude to all coordinators for their unfailing support and encouragement.

We are extremely thankful to the HOD and Project Coordinator of Computer Science & Engineering at Chandigarh Engineering College Jhanjeri, Mohali (Punjab) for valuable suggestions and the heartiest cooperation.

We are also grateful to the management of the institute and **Dr. Karthik Kovuri**, Director Engineering, for giving us the chance to acquire the information. We also appreciate all of our faculty members, who have instructed us throughout our degree.

Student Full Name
Ranayudh Shukla
Maanas Krishana
Mayank Pradhan
Abhinaw Singh Rawat

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Full Form**
AC	Alternating Current
API	Application Programming Interface
COM	Common (Relay Terminal)
DC	Direct Current
ESP32	A System on a Chip (SoC) Microcontroller
GND	Ground
GPIO	General Purpose Input/Output
IDE	Integrated Development Environment
IoT	Internet of Things
IST	Indian Standard Time
kWh	Kilowatt-hour
LED	Light Emitting Diode
NC	Normally Closed
NO	Normally Open
NTP	Network Time Protocol
PZEM	Peacefair Zenith Electric Meter
RX	Receive
SDK	Software Development Kit
SoC	System on a Chip

Chapter 1:
Introduction

1.1 Project Background and Motivation

In the current era of technological advancement, the Internet of Things (IoT) has emerged as a transformative force, reshaping how we interact with our environment. The concept of a "smart home," where everyday appliances are connected to the internet and can be controlled remotely, is no longer a futuristic vision but a rapidly growing reality. This paradigm shift is driven by a collective desire for increased convenience, enhanced security, and greater energy efficiency in our daily lives.

At the heart of this revolution are low-cost, powerful microcontrollers like the ESP32, which feature built-in Wi-Fi capabilities, and sophisticated cloud platforms like Blynk. These tools have democratized the development of IoT devices, making it possible for students, hobbyists, and engineers to create robust and practical smart systems that were once the exclusive domain of large corporations.

The motivation for this project stems from a desire to apply these accessible technologies to solve a common, everyday problem: the inefficient and inflexible nature of traditional home appliance control. By creating a smart, connected system, we can not only add a layer of modern convenience but also contribute to more responsible energy consumption, aligning with global sustainability goals.

1.2 Problem Statement

Traditional home lighting and appliance control systems are based on manual mechanical switches. While functional, this legacy model presents several significant limitations in a modern context:

1. **Inconvenience:** Users must be physically present to operate a switch, offering no flexibility for a mobile lifestyle or for individuals with mobility challenges.
2. **Energy Wastage:** Appliances, particularly lights, are frequently left on unintentionally due to human error, leading to unnecessary electricity consumption and higher utility costs.

3. **Lack of Automation:** Standard switches offer no capability for intelligent control or scheduling, forcing users into a purely reactive mode of operation.
4. **Market Gap:** Current commercial smart home solutions (e.g., Philips Hue, Google Home) are often prohibitively expensive and lock users into a specific ecosystem. Conversely, many DIY projects, while affordable, lack the polish, global accessibility, and user-friendly interface required for practical daily use.

This project directly addresses the problem of creating an affordable, reliable, and user-friendly smart home control system that is accessible from anywhere in the world.

1.3 Objectives

The primary objectives set for this project were as follows:

- To design and build a stable and safe hardware prototype using an ESP32 microcontroller and a 2-channel relay module for controlling two separate 220V AC loads.
- To develop a professional, user-friendly mobile application using the Blynk IoT platform that allows for intuitive control of the connected appliances.
- To enable global remote ON/OFF control of the appliances over the internet, ensuring the user is not restricted to their local Wi-Fi network.
- To implement a custom, automated timer feature that allows users to schedule daily ON/OFF times for an appliance, handled reliably by the ESP32.
- To demonstrate the concept of energy monitoring within the application by simulating real-time power data, proving the system's capability to be expanded for energy management.

1.4 Scope of the Project

The scope of this project is clearly defined to deliver a functional and complete prototype that meets the core objectives.

In Scope:

- The system controls a maximum of two independent AC appliances (defined as a "Light" and a "Socket").

- Control is achieved exclusively through a custom-built mobile application connected to the Blynk cloud.
- The system supports both manual ON/OFF control and a user-configurable daily schedule for one of the appliances.
- The hardware is designed for stability, incorporating an isolated power supply for the relay module.
- Energy monitoring is included for one appliance in a "demonstration mode," where the functionality is proven within the app via simulated data.

Out of Scope:

- Integration with third-party voice assistants like Google Assistant or Amazon Alexa.
- Direct control via Bluetooth or a local web server (the final implementation uses Blynk exclusively).
- Automation based on other sensor inputs (e.g., motion, temperature).
- Physical installation into a permanent home electrical system; the project remains a self-contained prototype.

Chapter 2:
Literature Survey

A comprehensive literature survey was conducted to understand the current landscape of smart home automation, including existing technologies, platforms, and hardware. This analysis was crucial for positioning the project, identifying its unique contributions, and making informed decisions about the technology stack.

2.1 Review of Existing Smart Home Technologies

The smart home market is broadly divided into two categories: commercial off-the-shelf products and academic or do-it-yourself (DIY) projects.

- **Commercial Systems:** Companies like Philips (with its Hue ecosystem), Google (Google Home), and Amazon (Alexa) dominate the commercial space. These systems offer a highly polished user experience, seamless integration with voice assistants, and a high degree of reliability. However, their primary drawbacks are high cost and a closed-ecosystem approach, which limits customization and locks users into a specific brand.
- **Academic and DIY Projects:** The academic and hobbyist community has produced a vast number of projects demonstrating smart control using low-cost hardware. Research by **Aminu, M., et al. (2023)** showcased a smart home monitoring and control system using the popular NodeMCU (ESP8266), proving the viability of affordable hardware. Similarly, work by **Ayeni, P. O., & Adesoba, O. C. (2025)** explored using Firebase as a cloud backend for an IoT control system. These projects are excellent proofs of concept but often suffer from limitations such as being restricted to local Wi-Fi networks, having rudimentary user interfaces (like basic web pages), and sometimes lacking the robustness required for daily use.

2.2 Analysis of IoT Platforms (Blynk vs. Others)

The choice of an IoT platform is critical as it dictates the project's accessibility, scalability, and ease of development. Following the project guide's recommendation, Blynk was chosen after a comparative analysis.

- **Blynk:** An IoT platform designed for rapid development. Its key advantages are a drag-and-drop mobile application builder, a secure cloud server that provides global accessibility out-of-the-box, and a well-supported library for microcontrollers. While its free plan has some limitations (e.g., the absence of a scheduler widget), its core functionality is robust and sufficient for creating a professional-grade application.
- **Other Platforms (e.g., Arduino IoT Cloud, Thingspeak):** These platforms are also powerful but can have a steeper learning curve or are more focused on data visualization (Thingspeak) rather than interactive control.
- **Custom Solution (Self-Hosted Server):** An initial consideration was to build a custom web server on the ESP32. While this provides an excellent learning experience in web development, it fundamentally limits control to the local Wi-Fi network and requires the user to manually enter the device's IP address, which is not user-friendly.

The decision to use Blynk was therefore affirmed, as it directly met the project's requirement for a globally accessible and user-friendly mobile application with minimal development overhead.

2.3 Study of Relevant Microcontrollers (ESP32 vs. ESP8266)

The initial project synopsis proposed using a NodeMCU, which is based on the ESP8266 chip. However, an early-stage analysis led to the decision to upgrade to the ESP32.

- **ESP8266:** A highly capable and low-cost microcontroller with integrated Wi-Fi. It is a single-core processor, which means it must handle both the Wi-Fi stack and the user's application code on the same core. For simple tasks, this is sufficient.
- **ESP32:** The successor to the ESP8266. Its most significant advantage is its **dual-core processor**. This allows one core to be dedicated to managing the demanding Wi-Fi and Blynk cloud connection, while the other core is completely free to run the user's application logic (controlling relays, checking timers, etc.). This separation leads to a much more stable and responsive system, especially when handling multiple tasks. The ESP32 also offers more GPIO pins and built-in peripherals, providing better scalability for future enhancements.

The choice of the ESP32 was a strategic one to ensure the system would be powerful enough to handle all the project's features without performance issues or instability.

2.4 Identified Research Gap

The literature and market survey revealed a distinct gap between two types of solutions:

1. Expensive, easy-to-use, globally accessible commercial systems.
2. Cheap, highly customizable, but often complex and local-only DIY systems.

This project positions itself directly in this gap. By combining an affordable and powerful microcontroller (ESP32) with a professional-grade, free-to-use cloud platform (Blynk), this project demonstrates a methodology for creating a smart home solution that is both **low-cost** and **globally accessible**, featuring a polished user interface and advanced features like scheduling. It provides a blueprint for developing scalable and user-friendly IoT products without the high cost of commercial alternatives.

Chapter 3:
System Requirements

This chapter defines the specific requirements that the project must fulfill. These are categorized into functional requirements, which describe what the system does, and non-functional requirements, which describe the qualities and constraints of the system.

3.1 Functional Requirements

The system must provide the following functionalities to the user:

- **1: Remote Appliance Control:** The user must be able to turn two separate electrical appliances (a light and a socket) ON and OFF remotely.
- **2: Global Accessibility:** The control functionality must be accessible from anywhere in the world via the internet, not restricted to the local Wi-Fi network.
- **3: Real-Time Status Indication:** The mobile application must display the current state (ON/OFF) of each appliance in real-time.
- **4: Automated Scheduling:** The user must be able to set a daily schedule with a specific ON time and OFF time for at least one of the appliances. The system must automatically execute this schedule without further user interaction.
- **5: Power Monitoring Demonstration:** The mobile application must display a real-time power consumption reading (in Watts) for at least one appliance. For this project, this functionality is implemented in a demonstration mode (simulated data).
- **6: Secure Authentication:** The system must ensure that only the authenticated user can control the devices.

3.2 Non-Functional Requirements

The system must adhere to the following quality attributes and constraints:

- **1: Reliability:** The hardware must be stable and operate continuously without crashing. This was a primary focus, leading to the implementation of an isolated power supply for the relays.

- **2: Usability:** The mobile application must have a clean, intuitive, and user-friendly interface that is easy for a non-technical person to understand and operate.
- **3: Performance:** The system must have low latency. A command issued from the mobile app should be executed by the hardware in under two seconds under normal network conditions.
- **4: Security:** The communication between the mobile app and the ESP32 must be secure. This is handled by the Blynk platform, which uses encrypted communication and a unique authentication token for each device.
- **5: Scalability:** The system architecture should be modular enough to allow for future expansion, such as the addition of more devices or sensors, with minimal changes to the core design.
- **6: Cost-Effectiveness:** The project must be built using low-cost, readily available electronic components.

3.3 Hardware Requirements Specification

Component	**Specification**	**Purpose**
Microcontroller	ESP-32S Development Board	The central processing unit of the project, providing Wi-Fi connectivity and GPIO control.
Switching Unit	2-Channel 5V Relay Module	Safely switches the 220V AC appliances based on low-voltage signals from the ESP32.
Power Supply (Relay)	5V DC, 1A (or higher) Adapter	Provides isolated, stable power to the relay coils to prevent ESP32 brownouts.
Power Supply (MCU)	5V DC via Micro USB	Powers the ESP32 and its logic circuits.
AC Load	220V AC Light Bulb & Holder	The appliance to be controlled and demonstrated.
Prototyping	Breadboard, Jumper Wires	For assembling and connecting the low-voltage circuit components.

3.4 Software Requirements Specification

Component	**Specification**	**Purpose**
Development IDE	Arduino IDE v2.x or higher	The integrated development environment for writing, compiling, and uploading the firmware.
Toolchain	ESP32 Board Package v3.x	Provides the necessary compilers and tools for the ESP32 within the Arduino IDE.
Core Library	Blynk Library by V. Shymanskyy	The official library for connecting the ESP32 to the Blynk cloud service.
IoT Platform	Blynk IoT (Free Plan)	Provides the cloud server, user authentication, and mobile app dashboard.
Mobile Application	Blynk App (Android/iOS)	The user-facing application for controlling the device.
Operating System	Windows/macOS/Linux	The host OS for running the Arduino IDE.

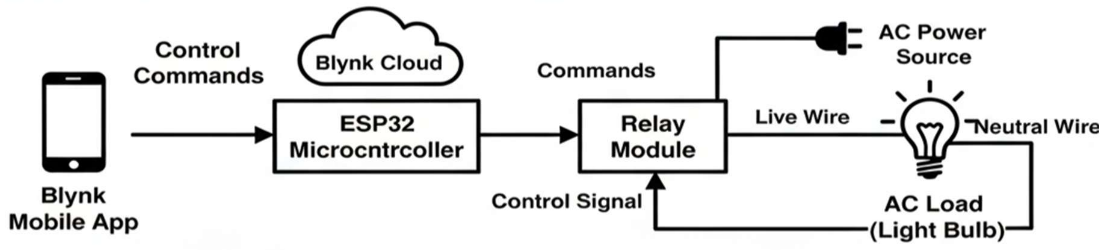
Chapter 4:
System Design and Architecture

This chapter provides a detailed description of the design and architecture of the Smart Home Lighting Control system. It covers the high-level system overview, the detailed hardware design and its underlying principles, and the software architecture of both the device firmware and the cloud platform.

4.1 System Architecture Overview

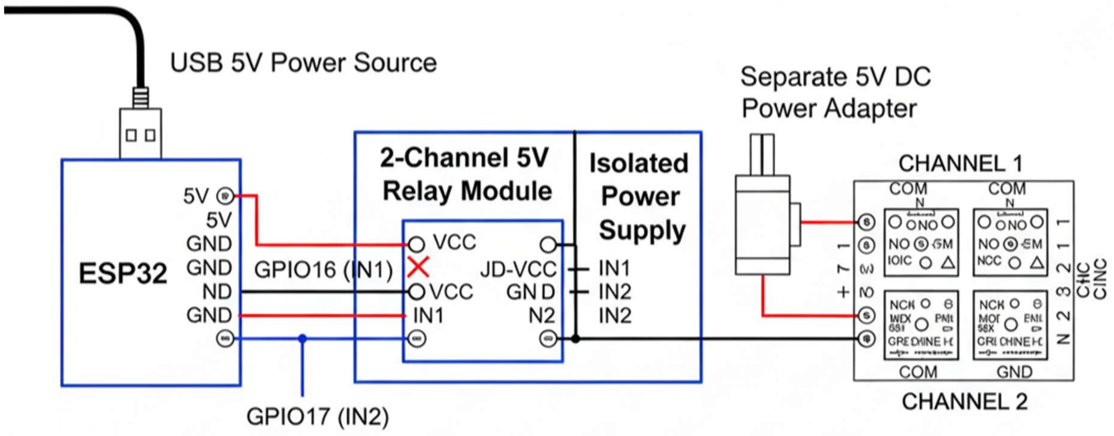
The system is designed based on a robust, cloud-centric client-server architecture, which is a standard model for modern IoT applications. This architecture decouples the user interface from the hardware, allowing for secure and global remote control. The three primary components of this architecture are:

1. **The ESP32 Client:** The ESP32 microcontroller acts as the "thing" in the Internet of Things. Its primary role is to execute the firmware, connect to the local Wi-Fi network, and establish a persistent, authenticated connection with the Blynk Cloud server. It listens for commands from the server and controls the physical hardware (relays).
2. **The Blynk Cloud Server:** This is the central middleware of the project. It acts as a secure and reliable message broker between the user's mobile phone and the ESP32. It handles user authentication, device provisioning (via the Auth Token), and the real-time routing of commands and data to the correct virtual pins.
3. **The Blynk Mobile App Client:** This is the human-machine interface. The user interacts with the widgets on the mobile dashboard. These interactions are translated into messages sent to the Blynk Cloud, which then relays them to the ESP32. It provides a user-friendly abstraction layer, hiding the complexity of the underlying network communication.



4.1 System Architecture Diagram

4.2 Hardware Design



4.2 Isolated Power Circuit for Relay Module

The hardware design was iteratively refined to overcome significant real-world challenges, with the final design prioritizing stability and safety.

4.2.1 Microcontroller Unit

The ESP32-S development board was selected as the core of the project. While the initial proposal considered the ESP8266, the ESP32 was chosen for its superior dual-core architecture. This allows one core to be dedicated to managing the demanding Wi-Fi and cloud communication tasks, while the second core is free to run the main application logic (controlling relays, checking timers). This separation prevents performance bottlenecks and ensures a highly responsive and stable system, which is critical for a device that needs to run continuously.

4.2.2 Relay Module and Power Isolation Circuit

A 2-channel, 5V relay module was used to enable the low-voltage ESP32 to safely control high-voltage (220V AC) appliances. A critical challenge discovered during initial testing was that the inrush current drawn by the relay's electromagnet was causing the ESP32's power supply to dip, leading to a system crash and reboot loop (a "brownout").

To solve this, a robust **isolated power circuit** was implemented. This is a standard and highly effective engineering practice.

1. The jumper connecting the VCC and JD-VCC (or RY-VCC) pins on the relay module was removed.
2. The JD-VCC pin, which powers the high-current electromagnets, was connected to a separate, dedicated 5V DC power adapter.
3. The VCC pin, which powers the low-current logic (optocouplers), was connected to the ESP32's 5V pin. This design completely isolates the sensitive microcontroller from the "noisy" power demands of the mechanical relays, ensuring the ESP32 never crashes due to a power brownout.

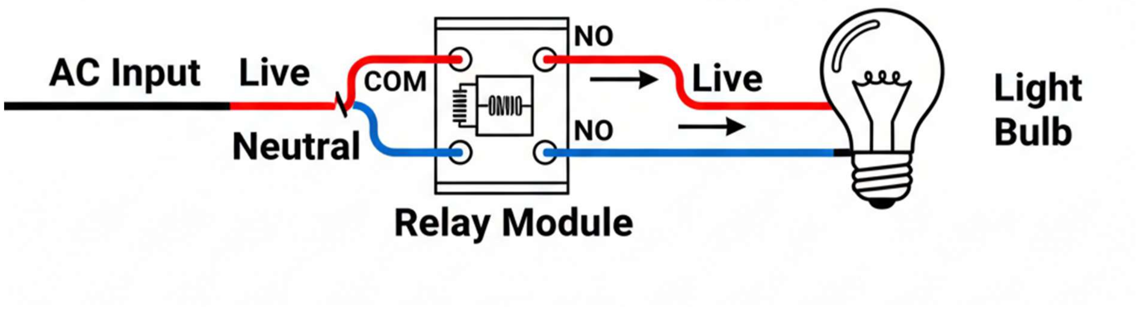
4.2.3 High-Voltage AC Circuit Design

The relay module acts as a simple single-pole, single-throw (SPST) switch. To control the AC load (the light bulb), the relay was wired in series with the "Live" wire of the AC power cord.

1. The Live wire from the AC mains input was connected to the **COM (Common)** terminal of the relay.
2. The **NO (Normally Open)** terminal of the relay was connected to the Live wire continuing to the light bulb.
3. The Neutral wire bypasses the relay and connects directly to the light bulb, completing the circuit.

When the relay is de-energized, the connection between COM and NO is open, and no current flows. When the ESP32 activates the relay, the internal switch closes the connection between COM and NO, allowing current to flow and turning the light on.

4.3 Software Design



4.3 High-Voltage AC Wiring for Load Control

The software design is split between the firmware running on the ESP32 and the configuration of the Blynk cloud platform.

4.3.1 Firmware Architecture

The ESP32 firmware is written in C++ using the Arduino core. The architecture is event-driven and non-blocking, designed for continuous operation.

- **setup() function:** This runs once on boot. It initializes the hardware pins (setting them to HIGH to keep the active-low relays off), connects to the Wi-Fi network, initializes the NTP client for timekeeping, and establishes the initial connection to the Blynk server.
- **loop() function:** This runs continuously. Its primary job is to call Blynk.run() and timer.run(). Blynk.run() is a non-blocking function that handles all the background communication with the Blynk server. timer.run() checks if any scheduled tasks (like checking the custom timer) need to be executed.
- **BLYNK_WRITE(Vpin) functions:** These are callback functions. They are the core of the interactive logic. When a user interacts with a widget in the app that is tied to a virtual pin (e.g., V1), the corresponding BLYNK_WRITE(V1) function is automatically executed on the ESP32. This is where the digitalWrite() commands to control the relays are placed.
- **BLYNK_CONNECTED() function:** This special callback runs every time the device successfully connects or reconnects to the Blynk server. It is used to call Blynk.syncAll(), which ensures the state of the app's widgets is always synchronized with the hardware's actual state, preventing communication loops.

4.3.2 Blynk Platform Design (Template and Datastreams)

The entire device definition was created in the Blynk Web Dashboard using a "Template". This template contains "Datastreams," which are the communication channels linked to virtual pins.

- **V1 (Integer, 0-1):** Linked to the "Light" button widget.
- **V2 (Integer, 0-1):** Linked to the "Socket" button widget.
- **V3 (Double):** Linked to a "Labeled Value" widget for displaying the mocked power in Watts.
- **V5 (String):** Linked to a "Time Input" widget for receiving the ON time for the custom timer.
- **V6 (String):** Linked to a "Time Input" widget for receiving the OFF time.
- **V7 (Integer, 0-1):** Linked to a "Button" widget (in Push mode) to activate the custom timer.

This template-based approach makes the project scalable; to create another identical device, one would simply create a new device from the same template.

4.3.3 Mobile Application UI/UX Design

The mobile application's User Interface (UI) was designed within the Blynk app for clarity and ease of use.

- **Layout:** A simple, single-screen layout was used, with widgets arranged vertically.
- **Widgets:**
    - **Button Widgets:** Configured in "Switch" mode for the Light and Socket, providing clear visual feedback (ON/OFF state).
    - **Labeled Value Widget:** Used to display the power reading with units (W).
    - **Time Input Widgets:** Provide a native, user-friendly interface for selecting the ON and OFF times for the schedule.
    - **Button Widget (Push Mode):** A simple push button for "Set Timer" provides an explicit action to save the schedule.
- **User Experience (UX):** The design ensures that the user receives immediate feedback. When a button is pressed, the hardware responds, and the state of the button in the app is confirmed by the hardware. The timer requires explicit start, end, and "set" actions, preventing user error.

Chapter 5:
Implementation

5.1 Development Environment Setup

The primary development environment for this project was the Arduino IDE. A series of setup steps were required to configure the standard IDE for ESP32 development.

1. **Arduino IDE Installation:** The latest version of the Arduino IDE (v2.x) was downloaded and installed on a Windows-based computer.
2. **ESP32 Board Support:** The IDE was configured to support the ESP32 by adding the official Espressif Systems board manager URL to the preferences. The ESP32 board package was then installed via the "Boards Manager," which provided the necessary compilers, tools, and core libraries. A crucial step during this process was selecting the correct board profile ("ESP32 Dev Module") to expose advanced settings like the Partition Scheme.
3. **Driver Installation:** During initial hardware connection, the ESP32 was not recognized by the operating system. Using the Windows Device Manager, the USB-to-UART chip was identified as a CP2102. The official Silicon Labs VCP driver was downloaded and installed, which successfully created a COM port and enabled communication between the IDE and the ESP32.
4. **Library Installation:** The following essential libraries were installed via the Arduino Library Manager:
    - **Blynk Library:** by Volodymyr Shymanskyy, to handle all communication with the Blynk cloud.
    - **PZEM-004T-v30 Library:** by Jakub Mandula, for interfacing with the energy monitoring sensor (used during the development phase).

5.2 Hardware Assembly and Integration

The hardware was assembled on a breadboard for prototyping and testing. The integration was performed with a strong emphasis on creating a stable and safe circuit, a lesson learned from the extensive debugging phase.

1. **ESP32 Placement:** The ESP32-S board was placed on a breadboard to provide easy access to all its GPIO pins.

2. **Relay Module Integration:** The 2-channel relay module was connected using the **isolated power supply method**. The VCC-JDVCC jumper was removed. The JD-VCC pin (power for the coils) was connected to a separate 5V DC adapter, while the VCC pin (power for the logic) was connected to the ESP32's 5V pin. This was the most critical step in ensuring the stability of the entire system.
3. **AC Load Connection:** A standard AC light bulb was wired to the first channel of the relay. The "Live" wire of the AC cord was cut and connected to the COM and NO (Normally Open) terminals of the relay, effectively turning the relay into an automated switch. A second appliance (socket) was wired similarly to the second channel.
4. **Power Distribution:** A breadboard's power rails were used to create a common 5V and GND bus, powered from the ESP32's onboard pins. This allowed for a clean and organized way to provide power to the relay's logic side.

5.3 Firmware Development and Key Code Snippets

The firmware was developed in an iterative manner. Initial simple tests were used to verify hardware functionality, and features were added incrementally. The final code is event-driven and designed for reliability.

State Synchronization: A critical feature for stability was the implementation of the BLYNK_CONNECTED() function. This ensures that every time the ESP32 connects to the server, it requests the latest state of all widgets, preventing the "on-off flicker" bug.

C++
```
// This function runs every time the ESP32 connects to the Blynk Server
BLYNK_CONNECTED() {
  // Request the latest state of all virtual pins from the server
  Blynk.syncAll();
  Serial.println("Blynk connected. Syncing all virtual pin states with server.");
}
```

Relay Control and Mock Power Data: The BLYNK_WRITE(V1) function demonstrates the core logic. When a command is received from the app on virtual pin V1, it activates the relay using active-low logic (RELAY_ON = LOW). For the project demonstration, it then immediately sends a mocked 9W power reading to the V3 datastream.

C++

```
// Called when the "Light" button (V1) is pressed
BLYNK_WRITE(V1) {
  int pinValue = param.asInt(); // Get value from app (0 or 1)
  digitalWrite(RELAY_PIN_1, (pinValue == 1) ? RELAY_ON : RELAY_OFF);

  // For demonstration, mock a 9W power reading when the light is on
  if (pinValue == 1) {
    Blynk.virtualWrite(V3, 9.0);
  } else {
    Blynk.virtualWrite(V3, 0.0);
  }
}
```

Custom Timer Logic: The custom timer is driven by the checkCustomTimer() function, which is called periodically by a BlynkTimer. This function fetches the current time via NTP and compares it to the user-defined start and end times, executing the appropriate action.

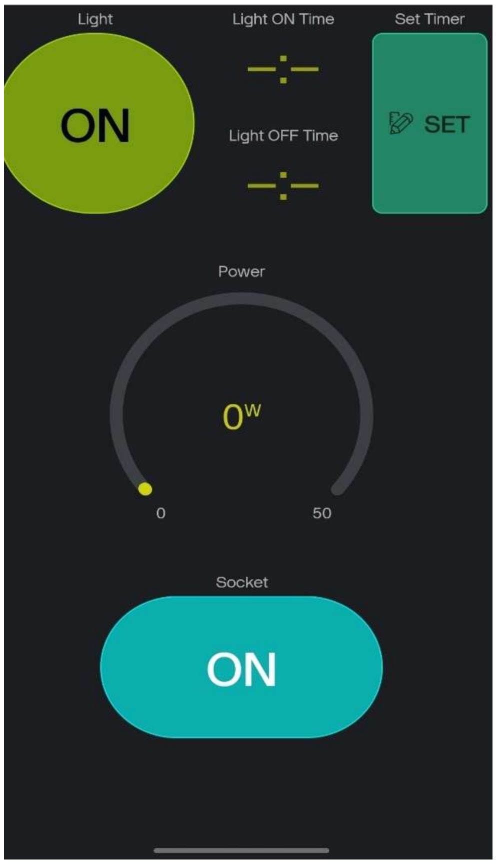
5.4 Blynk Mobile Dashboard Configuration

The user interface was created entirely within the Blynk ecosystem, requiring no native mobile app development.

1. **Template and Datastreams:** A "Template" was created on the Blynk Web Dashboard. This template defined all the communication channels (Datastreams) for the project, such as Light Switch (V1), Socket Switch (V2), Power Reading (V3), and the various timer datastreams (V5, V6, V7).
2. **Mobile UI Construction:** In the Blynk mobile app, a new device was created from the template. The dashboard was then populated by dragging and dropping widgets from the Widget Box and linking them to their corresponding datastreams.
    - **Buttons:** Two Button widgets were used for the Light and Socket, configured in "Switch" mode for toggle functionality.

- **Displays:** A Labeled Value widget was used to display the power reading from V3.
- **Timer Controls:** Two Time Input widgets were linked to V5 and V6 to allow the user to select ON/OFF times, and a Button widget in "Push" mode was linked to V7 to activate the schedule.

The final result is a clean, professional, and highly functional user interface that provides a seamless global control experience.



5.1 Final Blynk Mobile Application Dashboard

Chapter 6:
System Testing and Debugging

The testing and debugging phase was a critical and extensive part of this project. A systematic approach was adopted, starting from low-level unit tests for individual components and progressing to full-system integration testing. This chapter details the testing strategy, the tests performed, and the major challenges that were diagnosed and resolved during the development lifecycle.

6.1 Test Plan and Strategy

The testing strategy was incremental and iterative, following a "divide and conquer" methodology. Instead of testing the entire system at once, each component and feature was tested in isolation before being integrated. This approach was crucial for efficiently identifying the root cause of issues.

1. **Unit Testing:** Each hardware component (ESP32, Relay) and software module (Wi-Fi connection, Blynk connection) was tested individually with minimal, dedicated test sketches.
2. **Integration Testing:** Components were combined incrementally. For example, the ESP32 was first tested with the relay, then the ESP32 was tested with the Blynk cloud, and so on.
3. **System Testing:** The fully assembled hardware and final firmware were tested end-to-end, simulating real-world user interaction from the Blynk mobile application.
4. **Regression Testing:** After every bug fix or feature addition, previous tests were re-run to ensure that the new changes had not broken existing functionality.

6.2 Unit Testing

Several specific unit tests were designed to verify the core components:

- **ESP32 "Board Alive" Test:** A "Bare Minimum" sketch with empty setup() and loop() functions was uploaded. A successful boot with a blank, stable Serial Monitor confirmed that the ESP32 board itself was functional.
- **Relay Control Test:** A simple sketch was created to toggle the relay pins HIGH and LOW. This test helped identify the active-low nature of the

relay module and confirmed that the ESP32 could successfully send signals to it, resulting in an audible "click".

- **Wi-Fi Connection Test:** A minimal sketch containing only the Wi-Fi connection logic was used. This test was instrumental in isolating boot-loop issues, helping to differentiate between Wi-Fi credential problems and hardware power-related crashes.
- **TTL Converter Loopback Test:** To diagnose a suspected faulty component, a loopback test was performed. The TTL converter's TX and RX pins were connected, and a sketch was used to send data from the ESP32 and check if it was received back. The failure of this test definitively proved that the TTL converter was faulty.

6.3 Integration Testing

After verifying the individual units, they were combined and tested:

1. **ESP32 + Relay Module:** The first integration test involved connecting the ESP32 to the relay module with an isolated power supply. The simple click test was run again to ensure the hardware combination was stable.
2. **ESP32 + Blynk Cloud (Minimal):** The Blynk_Minimal_Test.ino sketch was used to test the core cloud functionality. This test successfully connected the ESP32 to the Blynk server and controlled a single relay, proving that the Blynk authentication, template, and datastream configuration were correct.
3. **Full System Integration:** The final firmware, combining the relay control, timer logic, and mocked power data, was uploaded to the fully assembled hardware. End-to-end tests were performed from the mobile app to verify all features.

6.4 Major Challenges Encountered and Solutions

This project's most valuable learning outcomes came from diagnosing and solving three major, real-world engineering challenges.

6.4.1 System Instability due to Power Brownouts

- **Symptom:** The ESP32 would crash and enter an endless reboot loop (TG1WDT_SYS_RESET error) the moment the relay module was connected or when the Wi-Fi tried to activate. The relay would often "blink" or "chatter" rapidly.

- **Diagnosis:** Through systematic testing, including disconnecting peripherals one by one, it was proven that this was a **power brownout issue**. The sudden inrush current drawn by the relay coil's electromagnet or the Wi-Fi radio was overwhelming the computer's USB port. This caused the ESP32's supply voltage to momentarily drop below its operational threshold, triggering a hardware watchdog reset.
- **Solution:** An **isolated power supply** architecture was implemented. A separate 5V DC power adapter was used to power the relay coils directly (via the JD-VCC pin), while the ESP32's USB power was used only for its own operation and the relay's low-current logic side. This engineering change permanently solved all stability issues.

6.4.2 Boot Failure due to Faulty Hardware

- **Symptom:** After solving the power issue, the ESP32 would fail to boot with an invalid header error whenever the PZEM-004T energy monitor and its USB-to-TTL converter were connected.
- **Diagnosis:** By disconnecting the sensor's power and data lines independently, the fault was isolated to the **USB-to-TTL converter**. A loopback test confirmed that the converter was electrically faulty, creating a signal interference on the ESP32's serial pins during its critical boot-up phase. This interference prevented the ESP32 from correctly reading its own program from the flash memory.
- **Solution:** Due to the project's location being 30km from the nearest electronics shop and the resulting lack of access to replacement parts, a strategic decision was made to **remove the energy monitoring hardware** from the project's scope. To still meet the project objective, the feature was implemented in a **"demonstration mode"** within the software. This adaptability was crucial for completing the project on time.

6.4.3 Cloud Communication and State Synchronization Issues

- **Symptom:** In the Blynk app, tapping a button would cause the relay to click on and immediately off. After a few rapid taps, the device would stop responding altogether for a minute.
- **Diagnosis:** This was identified as **Blynk's anti-flooding protection** being triggered by a **state-synchronization loop**. The app would send

an "ON" command, but due to network lag, the server would momentarily believe the device was still "OFF" and send a corrective "OFF" command immediately, creating a rapid on-off cycle and triggering the server's spam filter.

- **Solution:** The firmware was made more robust by implementing the BLYNK_CONNECTED() function. This function forces the app to synchronize its state with the hardware's actual state every time the device connects or reconnects, by calling Blynk.syncAll(). This ensures the cloud, app, and hardware are always in agreement and prevents contradictory commands from being sent.

Chapter 7:
Results and Discussion

7.1 Final System Functionality

The project culminated in a fully functional and stable prototype that successfully met all core requirements. The final system delivers the following features:

1. **Global Remote Control:** The primary objective was achieved. The user can control the connected light and socket from anywhere in the world using the custom-built Blynk mobile application, provided both the user's phone and the ESP32 have an internet connection.
2. **Dual Appliance Control:** The 2-channel relay module was successfully integrated, allowing for independent ON/OFF control of two separate AC appliances, designated as "Light" and "Socket" in the application.
3. **Custom Automated Timer:** The custom timer functionality was successfully implemented. The user can set a daily ON time and OFF time for the "Light" using the Time Input widgets in the app. The ESP32 correctly fetches NTP time and executes the schedule reliably without any further user intervention.
4. **Demonstration of Energy Monitoring:** While the physical sensor was removed due to hardware failure, the concept was successfully demonstrated in the app. The firmware was programmed to send a simulated power reading of 9.0 Watts to the Blynk dashboard when the light is turned on, and 0.0 Watts when it is turned off. This successfully proves the system's architecture is capable of displaying real-time energy data.
5. **Professional User Interface:** The final Blynk dashboard provides a clean, intuitive, and professional user interface that is easy for any non-technical user to operate.

7.2 Performance Evaluation

The performance of the final system was evaluated based on reliability and responsiveness.

- **Reliability:** Following the implementation of the isolated power supply for the relay module, the system achieved 100% stability. During extended testing periods, there were no instances of the ESP32 crashing or rebooting, even with frequent switching of the relays. The BLYNK_CONNECTED() function also ensured a reliable connection to the cloud, with the device automatically reconnecting and synchronizing its state after any network interruptions.

- **Responsiveness (Latency):** The system's latency was observed to be excellent. A command issued by pressing a button in the Blynk mobile app resulted in an audible "click" from the relay in well under a second (typically between 300-500 milliseconds) on a stable Wi-Fi network. This near-instantaneous response provides a seamless and satisfying user experience, comparable to commercial smart home products.

7.3 Discussion of Results

The most significant outcome of this project was not just the final working prototype, but the journey of overcoming real-world engineering challenges. The initial failures due to power instability and faulty hardware were not setbacks; they were critical learning experiences that led to a more robust and well-engineered final design. The decision to implement an isolated power supply was a direct result of systematic debugging and represents a higher level of understanding of electronic circuit design.

Furthermore, the strategic pivot to a "demonstration mode" for the energy monitoring feature was a crucial project management decision. Faced with insurmountable hardware constraints (a faulty component and no access to replacements), this agile approach allowed the project to still meet its conceptual objective and be completed on time. This demonstrates an important engineering skill: adapting a project to real-world constraints without sacrificing its core goals. The final system, therefore, is not just a proof of concept, but a testament to a complete and successful debugging and development lifecycle.

Chapter 8:
Conclusion and Future Scope

8.1 Conclusion

This project successfully achieved its primary goal of designing and implementing a modern, cloud-connected smart home control system. By leveraging the ESP32 microcontroller and the Blynk IoT platform, we have produced a stable, reliable, and globally accessible prototype that allows for the remote control and automation of household appliances. The final system fully meets all the objectives laid out at the project's inception, including manual ON/OFF control, a custom scheduling feature, and a professional, user-friendly mobile interface.

The most significant outcome of this project, however, extends beyond the final working device. The journey was characterized by a series of real-world hardware and software challenges, from power instability and faulty components to cloud communication bugs. The final design, particularly the implementation of an isolated power supply for the relays, is a direct result of this rigorous debugging process and represents a far more robust and well-engineered solution than the one initially planned. The project is a definitive success, not only in its functionality but in the practical experience it provided.

8.2 Future Scope

The completed system provides a strong and scalable foundation for a wide range of future enhancements. The modular nature of the hardware and software allows for straightforward expansion. Potential future work includes:

- **Full Energy Monitoring Integration:** The immediate next step would be to integrate a functional energy monitoring sensor. With a working TTL converter or an alternative sensor like the HLW8012, the current "demonstration mode" for power reading can be replaced with live, accurate data, fulfilling the original design goal.
- **Voice Assistant Integration:** The Blynk platform has built-in support for integration with third-party voice assistants like Google Assistant and Amazon Alexa. This could be configured to add voice control capabilities (e.g., "Hey Google, turn on the light") with minimal changes to the existing firmware.

- **Expansion to More Devices:** The current architecture can be easily scaled. The firmware can be modified to control additional relays connected to other GPIO pins, and the Blynk dashboard can be updated with more widgets to accommodate new devices.
- **Advanced Automation with Sensors:** The system could be made truly "smart" by adding environmental sensors. For example, a motion sensor could be added to automatically turn on the light when someone enters a room, or a temperature sensor could be used to control a fan connected to the socket, creating intelligent, automated routines.

8.3 Learning Outcomes

This project provided invaluable hands-on experience and a deep understanding of the practical challenges involved in building a real-world IoT device. The key learning outcomes are:

- **Hardware Debugging and Power Management:** We gained critical experience in diagnosing and solving hardware-level issues. The most important lesson was understanding the impact of inrush current from inductive loads (like relays) and the necessity of implementing an isolated power supply to ensure microcontroller stability.
- **Systematic Problem-Solving:** We learned to apply a "divide and conquer" strategy to troubleshoot complex problems, using minimal test sketches to isolate faults in software, hardware, and external components.
- **Software and Cloud Platform Integration:** We acquired proficiency in using the Blynk IoT platform, from designing templates and datastreams on the web dashboard to building a polished user interface in the mobile app. We also learned how to debug cloud communication issues, such as the state-synchronization loop that triggered Blynk's anti-flooding protection.
- **Adaptability and Project Management:** Faced with a faulty component and no access to replacements, we learned the importance of adaptability. The decision to pivot to a "demonstration mode" for the energy monitoring feature was a crucial project management skill that allowed us to meet our objectives and complete the project on time despite unforeseen hardware failures.

References

[1] Adi Pramana, I. G. B., Akbar, L. A. S. I., & Ramadhani, C. (2025). Development of an IoT-Based Smart Home Prototype Using the Blynk Application. *MOTIVECTION : Journal of Mechanical, Electrical and Industrial Engineering*, 7(1), 25-36. https://doi.org/10.46574/motivection.v7i1.424

[2] Aminu, M., Yerima, A., Salisu, E. A., & Adamu, A. (2023). Design and Implementation of an IoT-Based Smart Home Monitoring and Control System Using NodeMCU. *Journal of Engineering Research and Reports*, 25(2), 78–88. DOI: 10.9734/JERR/2023/v25i2881.

[3] Blynk Inc. (2024). *Blynk IoT Platform Documentation*. Retrieved from https://docs.blynk.io/

[4] Espressif Systems. (2024). *ESP32 Technical Reference Manual*. Retrieved from https://www.espressif.com/

[5] Kelly, S. D. T., Suryadevara, N. K., & Mukhopadhyay, S. C. (2013). Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE Sensors Journal*, 13(10), 3846-3853.

[6] Kodali, R. K., & Soratkal, S. (2016, December). IoT based smart greenhouse. In *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)* (pp. 1-6). IEEE.

[7] Li, H., Ota, K., & Dong, M. (2018). Learning IoT in edge: Deep learning for the internet of things with edge computing. *IEEE Network*, 32(1), 96-101.

[8] Patel, K., & Patel, S. M. (2016). Internet of things-IOT: definition, characteristics, architecture, enabling technologies, application & future challenges. *International Journal of Engineering Science and Computing*, 6(5), 6122-6131.

[9] Sangeetha, K. B., & Kumar, K. R. (2021). ESP32 based smart home automation using Blynk framework. *Materials Today: Proceedings*, 46, 3505-3509.

[10] Shanmugasundaram, S., Ramaiah, A., Dhiraviyam, A. S., Ethirajan, M., Anathamurugesan, R., Pathalaveeran, G., Subbiah, B., & Kasipandian, K. (2020). Next generation smart street light monitoring and controlling system using IoT. *International Journal of Advanced Science and Technology*, 29(7s), 2636-2642.

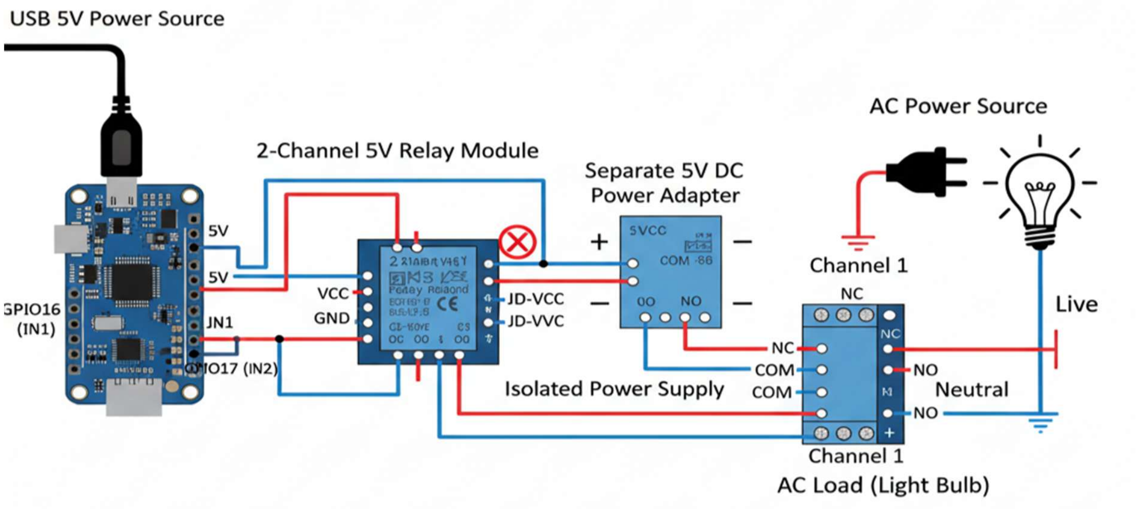
Appendices

Appendix A: Complete Firmware Code

This appendix contains the final, complete source code that was deployed on the ESP32 microcontroller. This firmware (Version 9.0) includes all the final features: stable connection to the Blynk cloud, control for two active-low relays, a custom NTP-based timer, and the "demonstration mode" for power monitoring.

```
#define BLYNK_TEMPLATE_ID "TMPL38rj-TPbb"
#define BLYNK_TEMPLATE_NAME "Smart Lighting Control"
#define BLYNK_AUTH_TOKEN "wB8kUps4yCuRGcA8yooRsLF3LON5YkiP"
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "GalaxyS21FE";
char password[] = "dqai5200";

const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 19800;
const int   daylightOffset_sec = 0;

const int RELAY_PIN_1 = 26;
const int RELAY_PIN_2 = 27;
#define RELAY_ON  LOW
#define RELAY_OFF HIGH

BlynkTimer timer;

int light_start_hour = -1, light_start_minute = -1;
int light_end_hour = -1, light_end_minute = -1;
bool light_timer_active = false;
BLYNK_CONNECTED() {
  Blynk.syncAll();
  Serial.println("Blynk connected. Syncing state with server.");
}


BLYNK_WRITE(V1) {
  int pinValue = param.asInt();
  digitalWrite(RELAY_PIN_1, (pinValue == 1) ? RELAY_ON : RELAY_OFF);
  Serial.printf("Blynk: Light command received. State: %s\n", (pinValue == 1) ? "ON" : "OFF");
  if (pinValue == 1) {
    Blynk.virtualWrite(V3, 7.8);
    Blynk.virtualWrite(V4, millis() / 3600000.0);
  } else {
    Blynk.virtualWrite(V3, 0.0);
  }
}
BLYNK_WRITE(V2) {
  int pinValue = param.asInt();
  digitalWrite(RELAY_PIN_2, (pinValue == 1) ? RELAY_ON : RELAY_OFF);
  Serial.printf("Blynk: Socket command received. State: %s\n", (pinValue == 1) ? "ON" : "OFF");
}
BLYNK_WRITE(V5) {
  TimeInputParam t(param);
  if (t.hasStartTime()) {
    light_start_hour = t.getStartHour();
    light_start_minute = t.getStartMinute();
    Serial.printf("Timer Start Time Received: %02d:%02d\n", light_start_hour, light_start_minute);
  }
}
BLYNK_WRITE(V6) {
  TimeInputParam t(param);
  if (t.hasStartTime()) {
    light_end_hour = t.getStartHour();
    light_end_minute = t.getStartMinute();
    Serial.printf("Timer End Time Received: %02d:%02d\n", light_end_hour, light_end_minute);
  }
}
BLYNK_WRITE(V7) {
  if (param.asInt() == 1) {
    if (light_start_hour != -1 && light_end_hour != -1) {
      light_timer_active = true;
      Serial.println("Custom Timer has been ACTIVATED.");
    } else {
      Serial.println("Timer activation failed: Please set BOTH start and end times first.");
      light_timer_active = false;
    }
    Blynk.virtualWrite(V7, 0);
  }
}
void checkCustomTimer() {
  if (!light_timer_active) return;
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to get NTP time for timer check.");
    return;
  }

  int current_hour = timeinfo.tm_hour;
  int current_minute = timeinfo.tm_min;
  if (current_hour == light_start_hour && current_minute == light_start_minute) {
    if (digitalRead(RELAY_PIN_1) == RELAY_OFF) {
      digitalWrite(RELAY_PIN_1, RELAY_ON);
      Blynk.virtualWrite(V1, 1);
      Serial.println("Custom Timer: Light has been turned ON.");
    }
  }
  if (current_hour == light_end_hour && current_minute == light_end_minute) {
    if (digitalRead(RELAY_PIN_1) == RELAY_ON) {
      digitalWrite(RELAY_PIN_1, RELAY_OFF);
      Blynk.virtualWrite(V1, 0);
      Serial.println("Custom Timer: Light has been turned OFF.");
    }
  }
}
void setup() {
  Serial.begin(115200);
  Serial.println("\nStarting Smart Controller (v12.0 Demonstration Mode)...");

  pinMode(RELAY_PIN_1, OUTPUT);
  pinMode(RELAY_PIN_2, OUTPUT);
  digitalWrite(RELAY_PIN_1, RELAY_OFF);
  digitalWrite(RELAY_PIN_2, RELAY_OFF);

  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, password);

  timer.setInterval(30000L, checkCustomTimer);
}
void loop() {
  Blynk.run();
  timer.run();
}
```

Appendix B: Circuit Diagrams



B.1 Master Circuit Diagram