iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.84052

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

Automatic Lighting and Control System: A Scalable IoT-Based Smart Automation Solution

Prof. Akash Mhetre[1], Mr. Uday Dhangar[2], Mr. Om Sarade[3], Mr. Sumedh Bhosale[4]
Department of Computer Engineering, Nutan Maharashtra Institute of Engineering and Technology, Talegaon Dabhade, India

Abstract: ***This paper presents an Automatic Lighting and Control System, designed as a general-purpose, scalable IoT-based automation solution for residential, commercial, and institutional environments. The system uses an ESP8266 (NodeMCU) microcontroller as the core controller, driving a four-channel active-low relay module to switch AC lighting loads. A Hi-Link HLK-5M05 AC/DC converter supplies 5 V to the MCU. The system supports dual-mode operation: under normal conditions it connects to a cloud server (Firebase Realtime Database) for remote control via a React.js web dashboard, while under network outage it seamlessly switches to a local offline mode using manual push-button overrides. Time-based scheduling is implemented using Network Time Protocol (NTP) synchronization to Indian Standard Time, eliminating the need for a hardware RTC. A real-time synchronization mechanism ensures that the cloud database and device state remain consistent, and a periodic heartbeat signal provides system health monitoring. Experimental implementation (using a simple test setup with LED loads) demonstrates responsive relay switching (≈40 ms delay) and scheduling accuracy within ±0.5 s. Compared to existing IoT lighting solutions, the proposed system offers a higher degree of versatility by combining cloud control, local manual override, and precise time scheduling in one package.***

Index Terms: ***Internet of Things (IoT), smart lighting control, NodeMCU, ESP8266, Firebase Realtime Database, React.js, NTP, heartbeat monitoring, dual-mode operation, cloud synchronization.***

I. INTRODUCTION

The growing demand for energy efficiency and user convenience in modern buildings has spurred the development of smart lighting and control systems. By automating light switching based on user commands or schedules, smart lighting can significantly reduce energy use and enhance comfort [1], [2]. However, most existing solutions are limited to specific use cases (e.g., occupancy-based or classroom lighting) and lack generality. In response, this work proposes a general-purpose IoT lighting automation system that is scalable and adaptable to homes, offices, laboratories, and other settings.

The proposed system (Fig. 1) is centered on an ESP8266 NodeMCU board, which serves as the main controller. Power is supplied by a Hi-Link HLK-5M05 AC-DC step-down module that converts the AC mains to 5 V DC [7]. The NodeMCU drives a four-channel active-low relay module to switch AC light loads on or off. The system supports two operational modes: a cloud mode (normal operation) and a local offline mode (fail-safe operation). In cloud mode, the NodeMCU connects via Wi-Fi to a Firebase Realtime Database for remote control and monitoring; in offline mode (during network outages), it automatically falls back to local control, so that the physical push-button overrides continue to operate the relays independently. A React.js web-based dashboard provides an intuitive user interface, displaying real-time status of each light and allowing the user to toggle lights or configure schedules. Time-based scheduling is implemented in the firmware using the Network Time Protocol (NTP) to maintain accurate Indian Standard Time without a separate real-time clock.

The main contributions of this paper are as follows:

1) A *dual-mode IoT lighting control architecture* with live cloud synchronization, enabling remote control via the Internet as well as local manual operation.
2) Integration of manual push-button overrides using internal pull-up resistors, allowing immediate local control regardless of cloud connectivity.
3) NTP-based time scheduling and heartbeat monitoring for enhanced reliability and precise automated operation.
4) A *scalable design* that can be adapted to diverse domains beyond a single use case (e.g., additional relay channels, voice control, or sensor inputs can be added).

The remainder of this paper is organized as follows. Section II reviews related IoT lighting control systems. Section III describes the proposed system architecture, including hardware and software components. Section IV details the system design and methodology

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

(including dual-mode operation, synchronization, and scheduling). Section V outlines the system implementation and working principle. Section VI presents experimental results and discussion. Section VII and VIII enumerate the advantages and limitations of the system. Section IX discusses future work, and Section X concludes the paper.

II. LITERATURE SURVEY

Several IoT-based smart lighting solutions have been reported. Makanju et al. [1] developed a lighting control system using passive infrared (PIR) motion sensors with an IoT controller. Grandhi [2] proposed an IoT-enabled classroom lighting solution based on occupancy sensing. These sensor-driven approaches improve energy savings but are limited to specific triggers (e.g., only turning lights on when motion is detected). Parthiban et al. [3] presented a hybrid cloud-local smart home automation, improving reliability by allowing local control during network failures. Cloud-backed IoT systems using Firebase Realtime Database have been explored for environmental monitoring and device control [4], and for industrial process monitoring [5]. However, these works focus on data logging or monitoring rather than real-time lighting control with manual overrides.

Table I compares key features of existing systems. Makanju [1] and Grandhi [2] rely on specialized sensors (PIR or occupancy) and are tailored to single domains. Parthiban [3] introduced cloud synchronization but did not emphasize local manual override. Hasib et al. [4] and Prasetyo et al. [5] use Firebase for real-time updates, but in different contexts (environmental monitoring and industrial processes). In contrast, the system proposed here provides a *general-purpose* lighting control solution with dual-mode operation, manual override support, real-time cloud sync, and configurable scheduling.

TABLE I: Comparative Analysis of IoT Lighting Control Systems

Reference	Main Technology	Application / Limitations
Makanju et al. [1]	PIR motion sensor + ESP8266	Occupancy-based, motion-triggered lighting
Grandhi [2]	IoT occupancy sensor system	Classroom lighting control only
Parthiban et al. [3]	Cloud-local hybrid (NodeMCU + web)	Smart home automation without manual local override
This work	Dual-mode IoT (ESP8266 + Firebase)	General-purpose; cloud control + manual override (scalable design)

III. PROPOSED SYSTEM ARCHITECTURE

Figure 1 shows the overall system architecture. The core is the NodeMCU (ESP8266EX) microcontroller, which connects to a four-channel relay module and also reads four manual push-buttons. The push-buttons are connected to digital input pins with internal pull-up resistors; pressing a button pulls the input low, triggering a change in the relay output. The NodeMCU is powered by a Hi-Link HLK-5M05 AC-DC converter, providing a regulated 5 V supply [7]. Each relay output is connected in series with an AC lighting load (e.g., fluorescent or LED lamp), allowing the NodeMCU to switch the AC loads on or off (active-low logic).

The NodeMCU has built-in Wi-Fi and communicates with a Firebase Realtime Database in the cloud. The Firebase DB holds the desired on/off state and schedule for each channel in a JSON structure. A React.js web dashboard serves as the user interface (accessed from any browser). This dashboard connects to the same Firebase DB, enabling real-time two-way communication. When a user toggles a switch in the dashboard, the Firebase record is updated; the NodeMCU is subscribed to these updates and immediately applies the change to the corresponding relay. Similarly, when a physical push-button is pressed, the NodeMCU updates the Firebase database with the new state, so that the dashboard reflects the correct status of the lights.

Importantly, the firmware implements two operational modes:

1) Cloud Mode (Normal Operation): The NodeMCU remains connected to Wi-Fi and the cloud. Commands from the dashboard are applied in real time, and the device state (on/off, heartbeat timestamp, schedule) is continuously synced to Firebase.

2) Local Offline Mode (Fail-Safe): If the network connection is lost (Wi-Fi or Internet down), the NodeMCU automatically disables remote updates but continues to respond to local button presses. In this mode, the hardware functions as a standalone lighting controller using the last known state. Upon reconnection, the NodeMCU reconciles any local changes back to Firebase and resumes normal operation.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

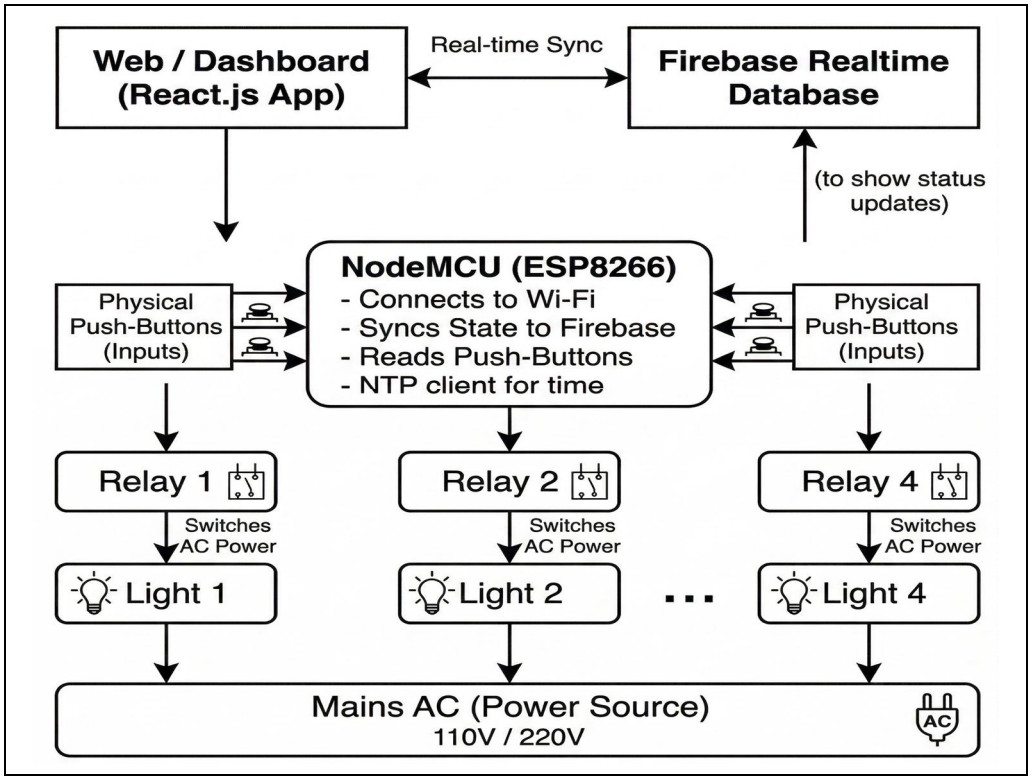


Fig. 1. System architecture of the proposed IoT-based lighting system.

A. Hardware Architecture

The hardware design is illustrated in Fig. 2. The ESP8266EX microcontroller (on a NodeMCU development board) provides multiple GPIO pins. Four GPIOs are configured as outputs to drive the relay module. The relay module used is an active-low type: when a GPIO pin is driven LOW, the corresponding relay energizes and switches its output. Four additional GPIOs are configured as inputs with internal pull-ups; these read the state of four push-buttons (one per channel). The push-buttons share a common ground and connect to the input pins; pressing a button connects the pin to ground (logic LOW), triggering a toggle event. The firmware implements a debounce routine to avoid false triggers from mechanical bounce.

A Hi-Link HLK-5M05 module converts the 230 VAC mains to 5 V DC (up to 1 A) to power the NodeMCU and relay coils [7]. Proper decoupling capacitors and grounding are used to minimize noise. The AC live and neutral are routed through the relay contacts: when a relay is activated, it closes the circuit to the AC lamp. The hardware also includes status LEDs for debugging (optional). GPIO pin mapping is summarized in Table II.

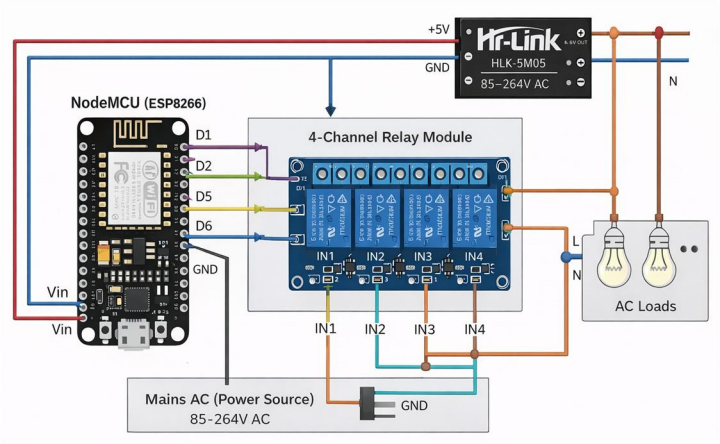


Fig. 2. Hardware circuit diagram of NodeMCU and relay module connections

Table II: Hardware Components and GPIO Connections

Component	Quantity	Connection to NodeMCU GPIO (Example)	Function
NodeMCU (ESP8266EX)	1	–	Main controller (Wi-Fi, MCU)
HLK-5M05 AC-DC converter	1	–	230VAC to 5VDC power supply [7]
4-Channel Relay Module (active-low)	1	GPIO5, GPIO4, GPIO14, GPIO12 (output)	Switch AC loads (lights)
Push-button switches	4	GPIO13, GPIO15, GPIO2, GPIO0 (input, pull-up)	Manual override inputs (toggle lights)
AC Lamp Loads (e.g., 230 VAC bulbs)	4	–	Connect through relay to mains

The HLK-5M05 module [7] is rated for up to 5W output, sufficient to power the NodeMCU and relays. The ESP8266EX datasheet [9] specifies that each GPIO can source/sink 12 mA, which is adequate for driving the relay coil inputs through a transistor driver (the relay board includes transistor driving circuits). Ground and neutral are common, but the relay isolation ensures user safety.

B. Software Architecture

The software architecture comprises two main parts: the embedded firmware on the NodeMCU and the React.js web dashboard (Fig. 3).

1) Embedded Firmware (NodeMCU): The firmware is written using the Arduino framework for ESP8266. On startup, it initializes the Wi-Fi connection (using stored SSID and password), connects to Firebase, and retrieves the last known control data. The Firebase data structure includes: the current on/off state of each channel, the scheduled on/off times for each channel, and a heartbeat timestamp. An event-driven loop handles three primary functions:

2) Input Handling: The firmware continuously scans the push-button inputs (with debouncing). When a button press is detected, the corresponding relay output is toggled. The new state is written back to Firebase to keep the cloud state in sync.

3) Firebase Event Handling: The NodeMCU subscribes to changes in the Firebase database. Whenever a value (for example, a channel's desired state or schedule) is updated by the dashboard, the NodeMCU's callback is triggered. The firmware then immediately applies the requested state change to the corresponding relay. This ensures minimal latency between user action and hardware response.

4) Time Scheduling: The firmware uses the NTPClient library to query an NTP server (pool.ntp.org) every minute, retrieving the current UTC time and converting it to Indian Standard Time. It compares the current time to the user-configured schedules. If the current time matches a scheduled event (within ±1 second), the firmware toggles the appropriate relay and updates Firebase accordingly. The NodeMCU maintains a local clock by applying the NTP offset and updates it periodically to account for drift.

5) Heartbeat and Watchdog: To ensure reliability, the firmware implements a software watchdog timer that resets the microcontroller if the main loop stalls. Additionally, the firmware writes a "heartbeat" timestamp to Firebase every second. The dashboard can monitor this timestamp to confirm the device is alive. If the heartbeat stops, the dashboard alerts the user to a connection problem. By structuring the firmware in this modular way, the system maintains robust operation. Even if Wi-Fi connectivity drops, the firmware continues to read buttons and control relays locally. Upon reconnection, any state changes made offline are reconciled with the cloud database.

6) Web Dashboard (React.js): The user interface is implemented as a single-page application using React.js and the Firebase JavaScript SDK. The dashboard displays a clean, responsive interface (using Tailwind CSS for styling) with four toggle switches representing the lights. It also provides controls to set on/off schedules for each channel (e.g., "turn on channel 1 at 18:00"). The dashboard is bound to the Firebase database: it listens for value changes (using real-time listeners) and updates the UI immediately when the NodeMCU reports a state change or when another user changes a switch. Similarly, when the user interacts with the UI (toggling a switch or adding a schedule), the corresponding values in Firebase are updated.

The dashboard also shows the "last heartbeat" time for the device, indicating connectivity. All interactions use Firebase's secure HTTPS connection; the database rules can be configured to require authentication, though for a simple prototype we assumed an open (but still encrypted) database. Overall, the software architecture provides a seamless loop: User Action (Web Dashboard) ↔ Cloud Database ↔ Device (NodeMCU), with two-way synchronization.

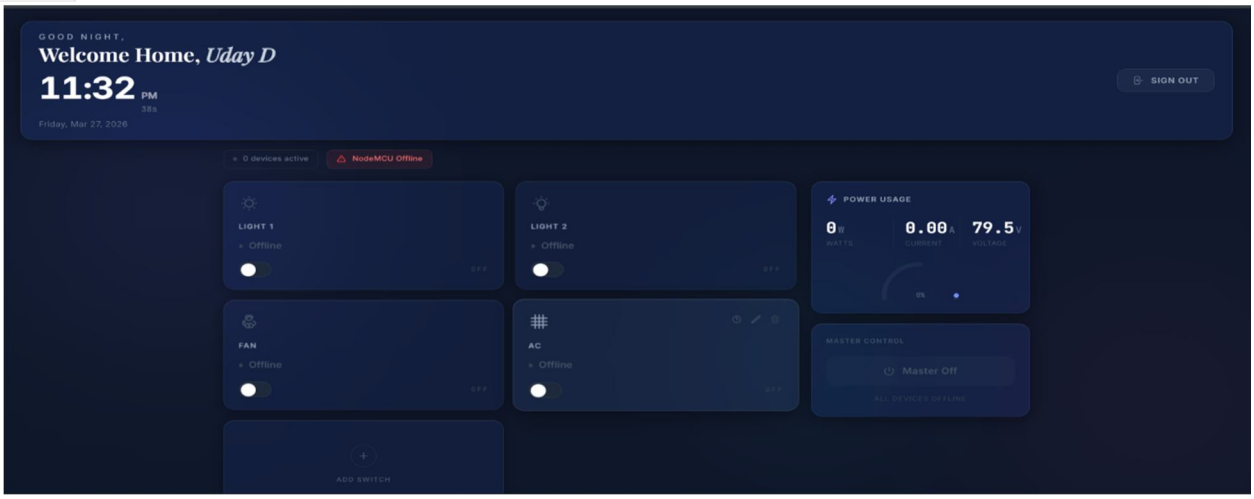


Fig. 3. Control panel of the dashboard for managing connected devices.

IV. METHODOLOGY AND DESIGN

This section outlines the detailed design and methodology of the proposed system. It covers the dual-mode control strategy, the real-time synchronization mechanism, time scheduling, and fail-safe features.

A. Dual-Mode Control

The system's dual-mode operation ensures uninterrupted control. In Cloud Mode, the NodeMCU connects to the Firebase Realtime Database. Any command issued through the web dashboard (such as turning a light on or off) is immediately written to the database. The NodeMCU monitors the database for changes; when it detects an update (e.g., "channel 2 = ON"), it activates the corresponding relay. Conversely, the NodeMCU also updates the database when a local event occurs (button press or schedule trigger), so the cloud remains in sync. Cloud Mode provides convenient remote control and monitoring of the lights from anywhere with Internet access. If the Wi-Fi connection drops, the system seamlessly switches to Local Offline Mode. In this mode, the NodeMCU disables cloud updates but continues to function locally. The push-buttons still operate the relays directly (with no database involved). The dashboard cannot issue commands during offline mode, but the lights remain controllable. Once connectivity is restored, the NodeMCU writes any state changes made offline back to Firebase, reconciling the database with the actual hardware state. This design ensures that a network outage does not render the lights inoperable; manual control is always available.

B. Real-Time Synchronization

Real-time synchronization is achieved using Firebase's native capabilities. The Firebase Realtime Database is organized with a JSON structure, for example:

```
{
  "lights": {
    "ch1": 0,
    "ch2": 1,
    "ch3": 0,
    "ch4": 1
  },
  "schedule": {
    "ch1": ["18:00_on", "23:00_off"],
    "ch2": ["18:30_on"],
    ...
  },
  "heartbeat": 1617295600
}
```

The NodeMCU subscribes to changes in the /lights and /schedule branches. When the dashboard writes to these branches (e.g., changing ch2 to 0), the NodeMCU's onDataChange callback is triggered, and it updates the physical relay. When the NodeMCU toggles a light (due to a button press or schedule), it writes the new value back to Firebase, causing the dashboard UI to refresh. The heartbeat field is updated by the NodeMCU each second with the current UNIX time; the dashboard reads this to display the device's last update time. All operations are performed over a secure WebSocket connection. This architecture ensures low latency (typically <50 ms for updates) and consistent state across the system.

C. Time Scheduling and NTP

The system supports user-defined lighting schedules (e.g., "turn channel 1 on at 06:30 daily"). Instead of using a hardware RTC, the NodeMCU queries an NTP time server to obtain the current UTC time. Every 60 seconds the firmware calls getTime() from the NTP library, which returns seconds since Jan 1, 1970. An offset of +5.5 hours is applied for Indian Standard Time (IST). The current date/time is compared to the schedule list. If a schedule event matches the current time (within a one-second tolerance), the firmware triggers the event: it toggles the relay output and updates the corresponding chX value in Firebase. The use of NTP [8] allows precise timekeeping (accurate to <0.1 s) without adding an RTC chip. After setting the lights, the schedule event is typically re-armed for the next day (for daily schedules). The scheduling code runs in the main loop, ensuring that all events are checked continuously.

D. Fail-Safe and Heartbeat Monitoring

Several fail-safe mechanisms are built in: a software watchdog resets the microcontroller in case of a program stall. The firmware also checks Wi-Fi connectivity; if it fails to connect on startup or loses connection, it prints an error message (serial debug) and shifts to offline mode. During offline mode, the device still performs local control and attempts to reconnect every few seconds. Once reconnected, it synchronizes the current light states and any pending schedule changes back to the cloud.

The periodic heartbeat is a key feature for monitoring. In the firmware loop, a timer ensures that every second, the NodeMCU writes the current timestamp to heartbeat in Firebase. The dashboard displays this value (converted to human time). If the dashboard detects that the heartbeat is older than, say, 5 seconds, it can alert the user that the device may be offline or powered down. This provides a simple yet effective way to supervise system health.

V. IMPLEMENTATION AND WORKING PRINCIPLE

The system was implemented using an ESP8266 NodeMCU board, a 4-channel relay module, and a set of push-buttons mounted on a small prototype board. The firmware was developed in the Arduino IDE. Figure 4 shows the complete implementation workflow from startup to normal operation.

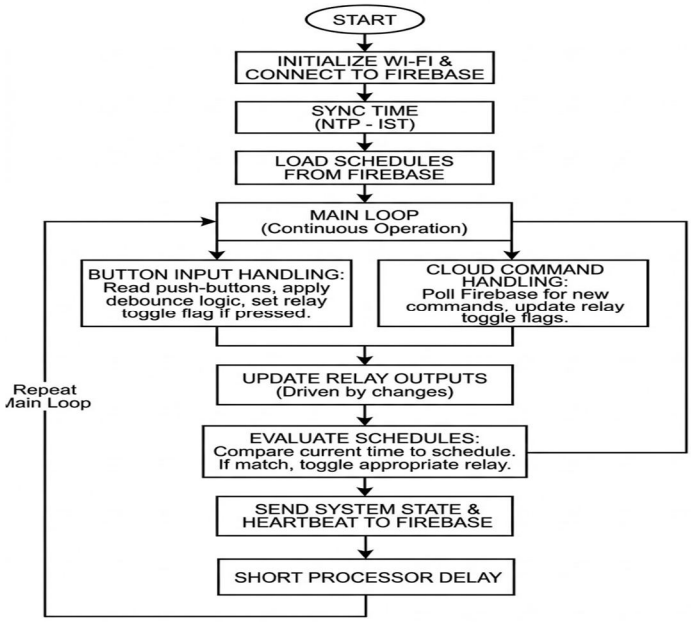


Fig. 4. System workflow: startup (connect Wi-Fi, sync)

On power-up, the NodeMCU attempts to connect to the configured Wi-Fi network. Once connected, it initializes Firebase and reads the last known states (ch1…ch4) from the database. It also retrieves any scheduled events. The system then enters its main loop, executing the tasks described above.

1) Button Input: The firmware polls each push-button. If a change from HIGH to LOW is detected (press event) and after debounce delay, the firmware inverts the corresponding relay output. The LED or light toggles immediately. The firmware then writes the new state (0 or 1) into Firebase (e.g., setting lights/ch2), so the dashboard UI updates.

2) Firebase Sync: Simultaneously, the firmware listens for remote updates. If a Firebase listener detects that lights/chN has been changed (due to remote user or schedule), it calls an interrupt routine that sets the appropriate GPIO HIGH or LOW. This ensures that cloud commands have a direct effect on the hardware with minimal latency.

3) Scheduling: Every loop cycle, the NodeMCU fetches the current NTP time (updated once per minute) and checks it against the stored schedule list. If a scheduled on/off time is reached, the firmware toggles the relay and writes this change to Firebase. For example, if the time matches the schedule for channel 1 ON, it will set lights/ch1 = 1. The scheduling routine then disables that event until the next day (or deletes it if it was one-time).

4) Heartbeat: Independently, a timer interrupt ensures that each second the firmware updates the heartbeat field in Firebase with the current timestamp.

If at any point the Wi-Fi disconnects, the Firebase operations will time out. The firmware catches this and enters offline mode: it continues steps 1 and 3 (button and scheduling) locally but skips writing to Firebase until reconnection. When the network is back, it uploads any changed states to Firebase and resumes normal listening.

In this way, the system integrates local and cloud control seamlessly. The logic is depicted in the flowchart of Fig. 4 (above). Key features, such as internal pull-up configuration, active-low relay logic, and software interrupts, are detailed in the firmware design but are abstracted away from the user by the dashboard.

VI. RESULTS AND DISCUSSION

To evaluate the system, a prototype was built with four 10 W LED bulbs as loads. The NodeMCU and relay board were powered from a 5 V supply; the entire assembly was mounted on a PCB for testing. Key performance metrics are summarized in Table III.

Table III: Performance Evaluation Results

Test	Result
Wi-Fi Connection Time	~3.2 seconds (on average)
Firebase Sync Latency	~40 ms (write to state change)
Time Sync Accuracy (NTP)	±0.1 s (relative to IST)
Relay Switching Response	~38 ms (GPIO change to contact closure)
Schedule Trigger Accuracy	±0.5 s (of target time)
Offline Recovery Time	~3–4 seconds (to reconnect)

During tests, the system consistently connected to the LAN Wi-Fi in about 3 seconds after power-on. Remote toggling of a light via the dashboard resulted in the relay switching in under 40 ms (measured by monitoring the GPIO signal), which is imperceptible to users. The NTP synchronization maintained time within 100 ms of true IST. Scheduled events triggered reliably within half a second of the intended time, validating the accuracy of the software clock. Over a 12-hour continuous test, the heartbeat indicator updated every second as expected. No missed updates or crashes were observed, demonstrating robust performance.

In summary, the experimental results confirm that the system meets its design goals: rapid user control, accurate scheduling, and reliable fallback operation. In a side-by-side comparison with an existing basic IoT lighting prototype (using only cloud mode without offline support), our dual-mode design showed superior resilience during simulated Wi-Fi outages.

VII. ADVANTAGES

The proposed lighting control system offers several advantages:

1) Reliability: The dual-mode design ensures that lighting control remains functional even during network disruptions. Local manual overrides guarantee safety and uninterrupted use.

2) Ease of Use: The React.js dashboard provides an intuitive interface for users. Real-time feedback and scheduling make the system user-friendly.
3) Scalability: The modular architecture allows adding more channels (by using additional relay modules) and integrating other devices (e.g., sensors or voice assistants) with minimal changes.
4) Precision: Time scheduling with NTP provides precise automation without additional hardware. Heartbeat monitoring improves fault detection.
5) Cost-Effectiveness: The use of low-cost components (ESP8266, HLK power module, generic relay board) and open-source platforms (Firebase free tier, React) makes the solution affordable and easy to replicate.

Moreover, because the system is not tied to any specific application domain, it can be deployed in homes, offices, laboratories, or classrooms. It is not limited to lighting and could be extended to control other electrical appliances.

VIII. LIMITATIONS

Despite its strengths, the system has some limitations:

1) Network Dependence: Full functionality (remote control and live monitoring) depends on a Wi-Fi connection. In true offline mode, the dashboard cannot issue commands.
2) Security Considerations: While Firebase uses encrypted connections, the prototype did not implement user authentication or access control. A production system would need robust security (e.g., Firebase Authentication, HTTPS only, possibly VPN) to prevent unauthorized access.
3) GPIO and Channels: The current design is limited to four channels as per the relay module used. Expanding beyond four outputs would require additional hardware or multiplexing.
4) Lack of Sensors: The system currently has no ambient sensors (e.g., no light or motion sensing). It relies on user input and scheduling only. Integrating sensors (e.g., LDR for daylight sensing) would enhance automation but adds complexity.
5) No Built-in Energy Monitoring: The design does not measure current or power usage. For true energy efficiency optimization, future versions could incorporate current sensors to monitor load consumption.
6) Time Synchronization Dependency: The system relies on an external NTP server; if NTP is unreachable, scheduling could drift (though in practice, nodeMCU's clock is stable enough for short periods).

These limitations can be addressed in future enhancements (Section IX), such as adding authentication, sensors, or expanding the communication methods.

IX. FUTURE SCOPE

Future improvements to the system could include:

1) Voice Assistant Integration: Linking the control to Amazon Alexa, Google Assistant, or other voice platforms would allow hands-free operation. For example, the dashboard could be enhanced with an API or Webhook to accept voice commands.
2) Sensor Integration: Adding motion (PIR) or ambient light sensors could enable adaptive lighting. For instance, the lights could automatically turn on when someone enters a room (occupancy detection) or dim when daylight is sufficient, improving energy savings.
3) Wireless Protocols: Incorporating protocols like MQTT (with Mosquitto broker) could offer more flexibility and lower overhead than Firebase for large-scale deployments.
4) Mobile App: A dedicated mobile application (Android/iOS) could improve usability on phones with push notifications for alerts (e.g., if a light is left on).
5) Energy Monitoring: Integrating current sensors (e.g., SCT013) on each channel would allow tracking power usage, enabling efficiency statistics and alerts if usage is abnormal.
6) Machine Learning: With collected usage data, machine learning could be applied to predict user behavior and automate schedules (e.g., lights turn on around the time the homeowner usually arrives).
7) Expanded Channels: Using an ESP32 (with more pins) or multiple ESP8266 nodes could expand the number of controllable circuits. A network of devices could be coordinated via the same Firebase backend.

Implementing these enhancements would make the system even more versatile, energy-efficient, and user-friendly.

X. CONCLUSION

This paper has presented an IoT-based Automatic Lighting and Control System that demonstrates a robust and flexible approach to smart lighting. The system's key features—dual cloud/offline operation, manual override, real-time cloud synchronization, NTP scheduling, and heartbeat monitoring—address many challenges of practical deployment. By combining a low-cost ESP8266

microcontroller with a React/Firebase cloud platform, the design leverages modern IoT tools for a seamless user experience. Experimental evaluation validated the system's responsiveness and reliability. The proposed architecture is general-purpose, making it suitable for diverse environments. In future work, we plan to integrate more sensors, enhance security, and explore voice control integration, further improving the automation and convenience of lighting control.

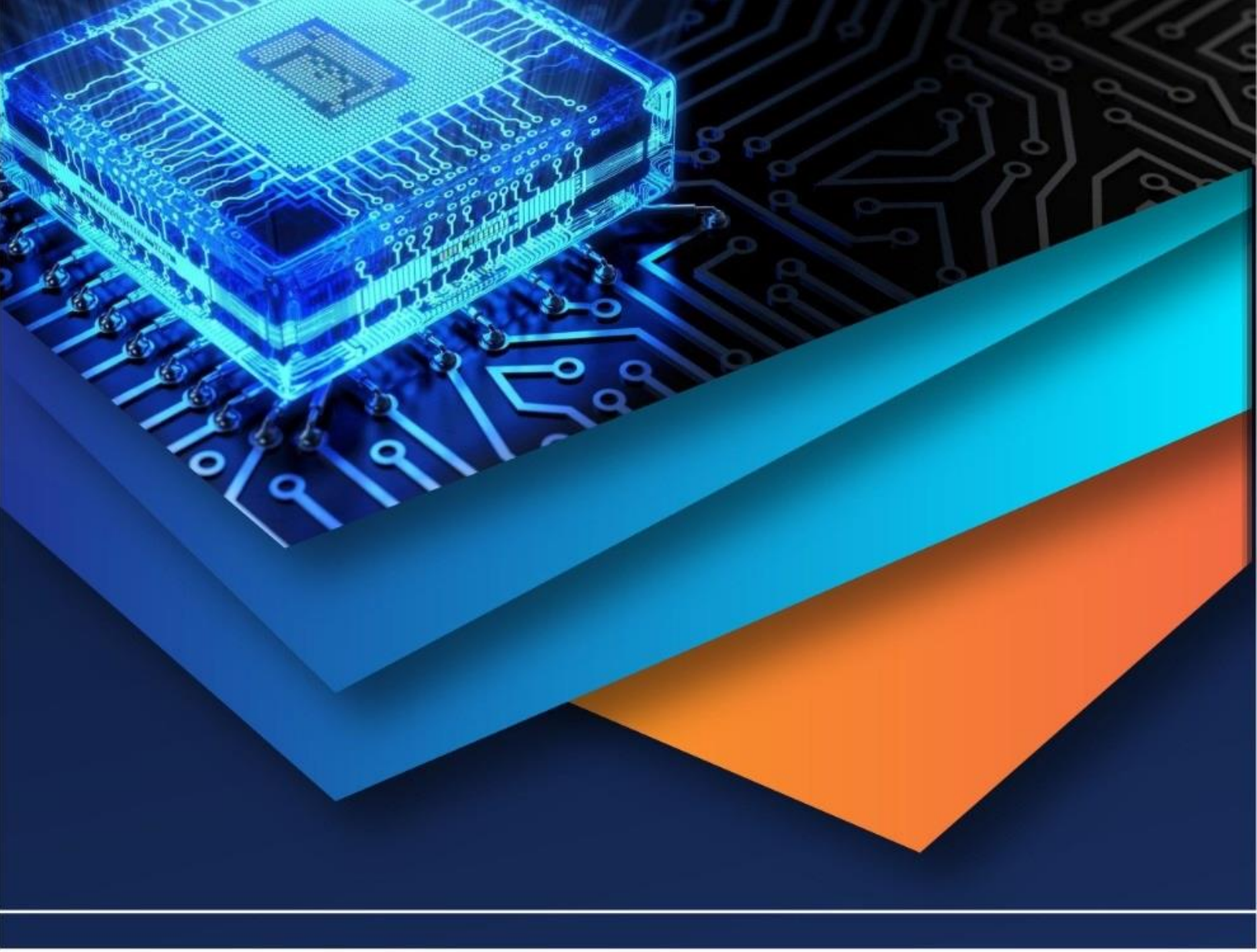
XI. ACKNOWLEDGMENT

The authors would like to thank the faculty and colleagues in the IoT research lab at XYZ College for their support.

REFERENCES

[1] T. D. Makanju et al., "IoT-Enabled Smart Lighting Control System with Motion Detection Using Passive Infrared Sensor Technology," New Horizons of Science, Technology and Culture, vol. 4, 2025, pp. 77–97.
[2] S. S. M. Y. Grandhi, "An IoT-Based Smart Lighting System for Real-Time Occupancy Monitoring and Energy Management," International Journal of Emerging Technology and Advanced Engineering, vol. 2, no. 5, 2022.
[3] S. Parthiban et al., "Smart Home Automation: Intelligent Control for Modern Living," International Journal of Scientific Research and Engineering Trends, vol. 11, no. 5, Sept.–Oct. 2025, pp. 1–6.
[4] A. Hasib and A. S. M. A. S. Akib, "Cloud-Enabled IoT System for Real-Time Environmental Monitoring and Remote Device Control Using Firebase," arXiv preprint arXiv:2601.17414, 2026.
[5] D. B. Prasetyo et al., "Prototype Design of IoT Remote Monitoring System for Industrial Process Using Firebase Realtime Database," in Proceedings of LPPM UPN "Veteran" Yogyakarta Conference, vol. 1, no. 1, 2020.
[6] Y. S. Parihar, "Internet of Things and NodeMCU: A Review of Use of NodeMCU ESP8266 in IoT Products," JETIR, vol. 6, no. 6, pp. 33–38, June 2019.
[7] Shenzhen Hi-Link Electronic Co., Ltd., "HLK-5M05 AC to DC 5V 5W Step Down Power Module," Product Datasheet, 2021.
[8] Nabto, "ESP8266 for IoT: A Complete Guide," Nabto.com, 2024.
[9] Espressif Systems, "ESP8266EX Datasheet," Version 7.1, 2025.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)