IoT-Based Smart Home Automation

ABSTRACT- **Amid technological advancements and heightened internet and mobile communication use, data collection and transmission via personal cellular devices has become commonplace. These advancements, notably in computing power and telecommunications, have not only streamlined daily activities but also revolutionized business operations, promoting efficiency and waste reduction. As a result, companies are now investing heavily in products that seamlessly blend these two realms, fostering connectivity among networked technologies through the Internet of Things (IoT). Leveraging IoT, companies can enhance adaptive functionality, automation, and remote operation capabilities. With the surging demand for smart technology in various markets, competition has skyrocketed, with companies focusing on niche areas to distinguish themselves. However, the resultant array of 'smart' products, ranging from lighting to kitchen appliances, often operate independently or only with similar products from the same company, diminishing their overall utility. Consequently, companies with durable, simple, and versatile network designs gain most market attention. This paper elucidates the network architecture of our IoT-Based Smart Home Automated system, including the role of sensors and mechanical components in data collection and system alteration. The discussion spans MQTT brokers, Cloud-based storage, sensor connectivity to NodeMCU, and the Cloud's real-time database's role in storing data points and relaying information to end users via Google's Firebase on Android applications.**

Keywords: *MQTT broker, NodeMCU, Cloud-based storage, Google's Firebase*

1. INTRODUCTION

As technology evolves, homes are undergoing a similar transformation as cities, integrating more smart features to accommodate growing needs for convenience, security, and energy efficiency. One such adaptation involves the implementation of an IoT-based home automation system, effectively transforming homes into "smart homes." This system is crucial for daily living, aiming to deliver maximized convenience, enhanced security, and energy conservation. In line with this vision, our goal is to utilize IoT's potential by integrating smart lighting systems, temperature control, air quality sensing, a smart door lock, and a motion detection security system all into one comprehensive package.

Our system is designed to offer a host of advantages. The smart lighting system provides convenient and energy-efficient control of brightness and color through a mobile application. Temperature control allows for energy-saving comfort regulation. An air quality sensor monitors the indoor air purity, tracking gas leakage and harmful gases such as smoke, propane, and methane. The smart door lock enables remote door management for added security, and a motion detection system alerts users to unexpected movements within their homes.

At the heart of this project is the NodeMCU microcontroller, known for its versatility and user-friendliness, making it ideal for home automation projects. We utilize Wi-Fi for intra-house data transfer and cloud storage communication. Simultaneously, users' cellular signals through their phone carrier allow data display and secure command transfer via a 4G/LTE network connection. This project, therefore, aspires to deliver a comprehensive and user-friendly home automation solution. It harnesses IoT's power to enhance convenience, security, and energy efficiency. We will delve into the design and implementation of each feature in the following sections.

2. PROJECT SCOPE

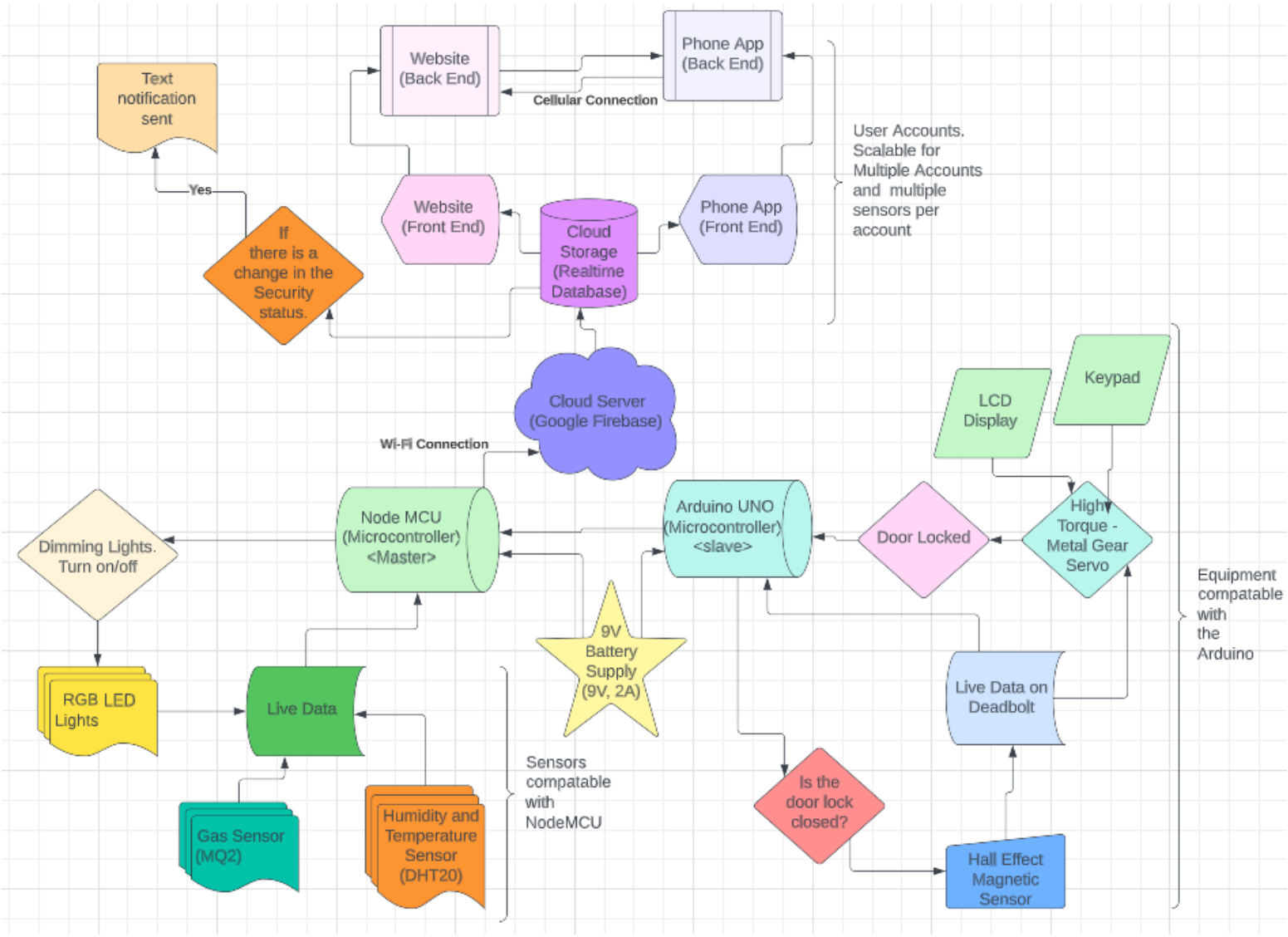


Figure 1) Depicts the system diagram layout of our project.

Figure 1 presents the design for our IoT-based Smart Home Automation system, with the NodeMCU ESP8266 microcontroller serving as the central processing unit. This NodeMCU manages both existing and newly added sensors, including temperature, gas, and magnetic sensors, along with lighting systems. The NodeMCU connects to the Firebase cloud server via a Wi-Fi connection, facilitating real-time data sharing and remote control. These devices are strategically placed throughout the home environment to accurately monitor and control various household parameters. Data generated by the sensors are transmitted in real-time to the cloud server, which then communicates with the Laravel/Vue.js based web application through a cellular connection. Users can view and analyze the reported data and control their home automation system via the web application, completing a comprehensive IoT-based home automation setup.

I. PARTS

1. ESP8266 NodeMCU

The NodeMCU ESP8266 will serve as the beating heart of our home automation system. As a versatile and robust microcontroller, it boasts built-in Wi-Fi capabilities, making it an optimal choice for IoT applications. This highly capable device acts as a nerve center, interfacing seamlessly with our array of sensors, diligently collecting their data, and expertly relaying this information to our dedicated cloud storage and user-friendly mobile application. What sets NodeMCU apart is its capacity to simultaneously manage multiple tasks, making it a dependable overseer of the many elements in our home automation system.

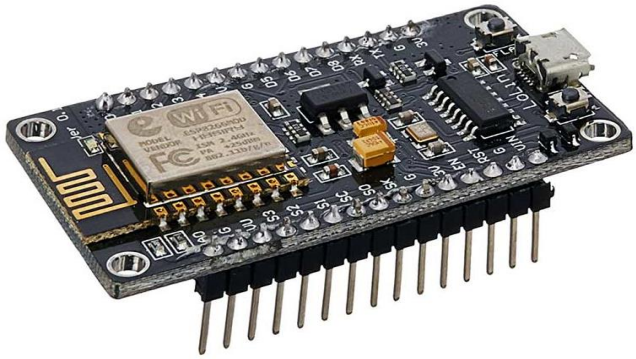


Figure 2) Depicts the NodeMCU ESP 826.[1]

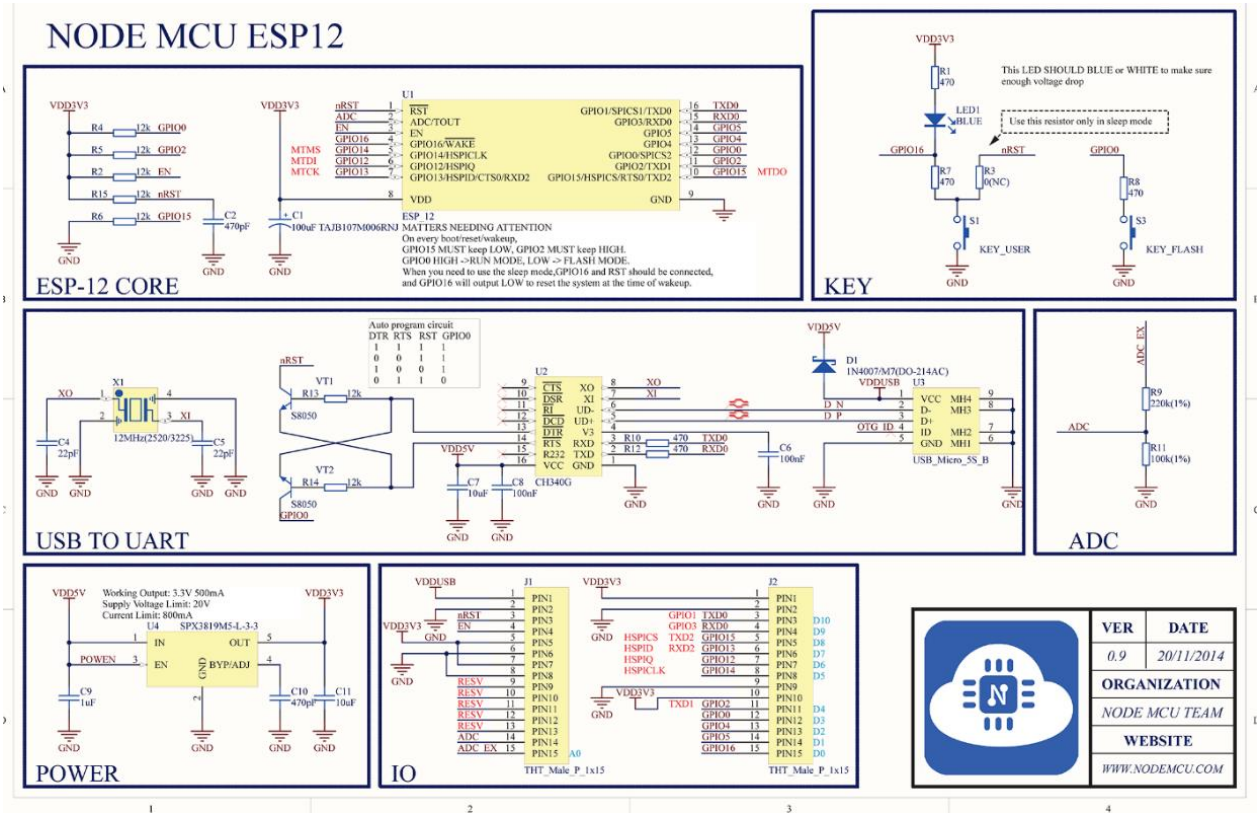


Figure 3) Depicts the general internal design structure of a NodeMCU. [25]

Table 1) The Pin Descriptions of the NodeMCU

Pin Category	Name	Description
Power	*Micro-USB, 3.3V, GND, Vin*	Micro-USB: NodeMCU can be powered through the USB port 3.3V: Regulated 3.3V can be supplied to this pin to power the board GND: Ground pins Vin: External Power Supply
Control Pins	*EN, RST*	The pin and the button resets the microcontroller
Analog Pin	*A0*	Used to measure analog voltage in the range of 0-3.3V
GPIO Pins	*GPIO1 to GPIO16*	NodeMCU has 16 general purpose input-output pins on its board
SPI Pins	*SD1, CMD, SD0, CLK*	NodeMCU has four pins available for SPI communication.
UART Pins	*TXD0, RXD0, TXD2, RXD2*	NodeMCU has two UART interfaces, UART0 (RXD0 & TXD0) and UART1 (RXD1 & TXD1). UART1 is used to upload the firmware/program.
I2C Pins		NodeMCU has I2C functionality support but due to the internal functionality of these pins, you have to find which pin is I2C.

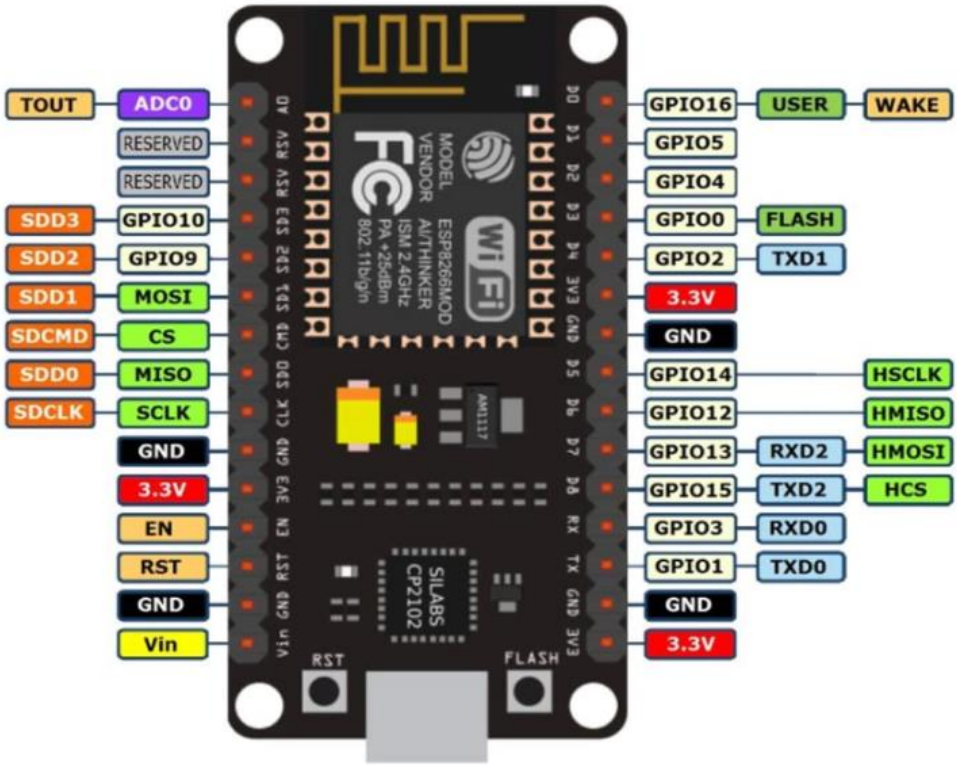


Figure 4) Pin diagram for the NodeMCU ESP8266. [1]

2. Arduino Uno Rev3 GMS Shield Sim900f

The Arduino Uno Rev3 GMS Shield Sim900f is an integral component of our system. Designed to seamlessly dovetail with the Arduino Uno, this shield equips our setup with GSM/GPRS functionality. This means that our system can tap into the power of the cellular network for internet access, as well as sending and receiving SMS messages, and even make voice calls. The compatibility and added features this shield provides form the perfect synergy with our system.

Figure 5) Depicts the Arduino Uno Rev3 GMS Shield Sim900f. **[10]**

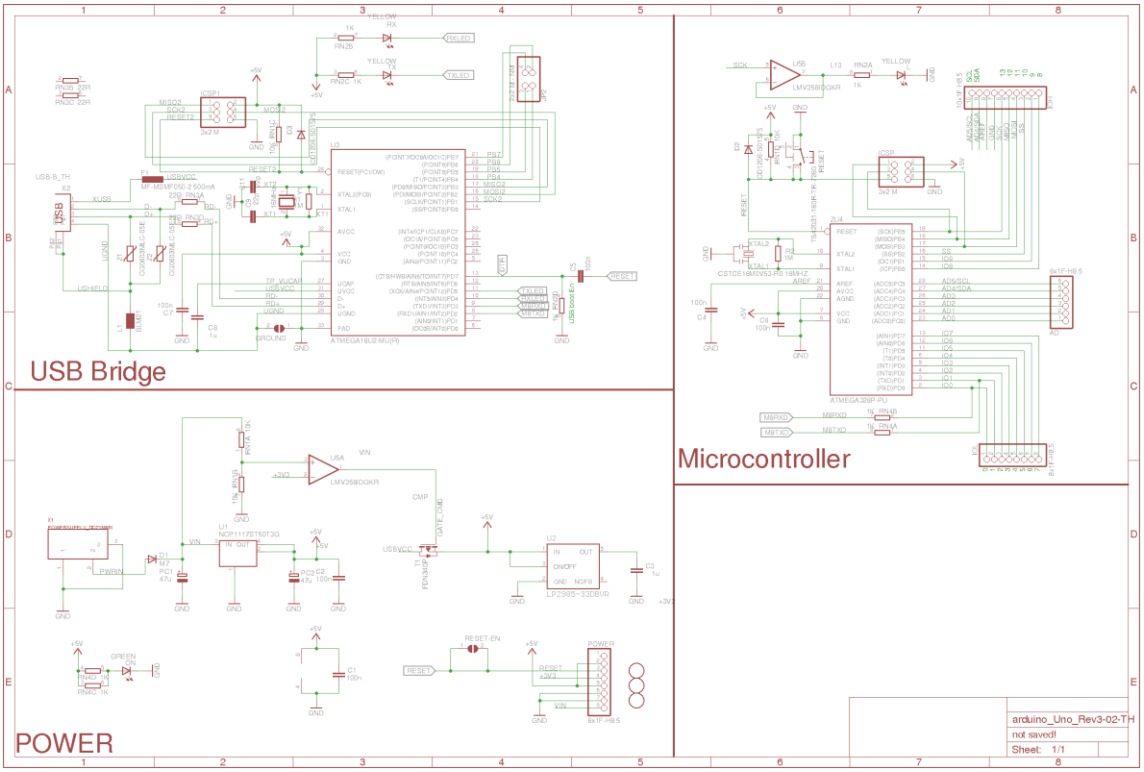


Figure 6) Depicts the general internal structure of an Arduino UNO. **[23]**

Table 2) The Pin Descriptions of the Arduino UNO

Pin Category	Name	Description
Power	*5V, 3.3V, GND, Vin*	5V & 3.3V: Regulated supply to power the board or components. GND: Ground pins. Vin: External Power Supply (7-12V).
Reset Pin	*RESET*	Resets the microcontroller.
Analog Pins	*A0 to A5*	Used to measure analog voltage in the range of 0-5V.
Digital I/O Pins	*D0 to D13*	Digital pins that can be used for both input and output.
PWM Pins	*D3, D5, D6, D9, D10, D11*	These digital I/O pins support PWM output (Pulse Width Modulation).
SPI Pins	*D10 (SS), D11 (MOSI), D12 (MISO), D13 (SCK)*	These pins are used for SPI communication.
UART Pins	*D0 (RX), D1 (TX)*	These pins are used for UART communication (Serial communication).
External Interrupts	*D2 (INT0), D3 (INT1)*	These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
LED	*D13*	There is a built-in LED driven by digital pin 13.
I2C Pins	*A4 (SDA), A5 (SCL)*	These pins support I2C (TWI) communication using the Wire library.

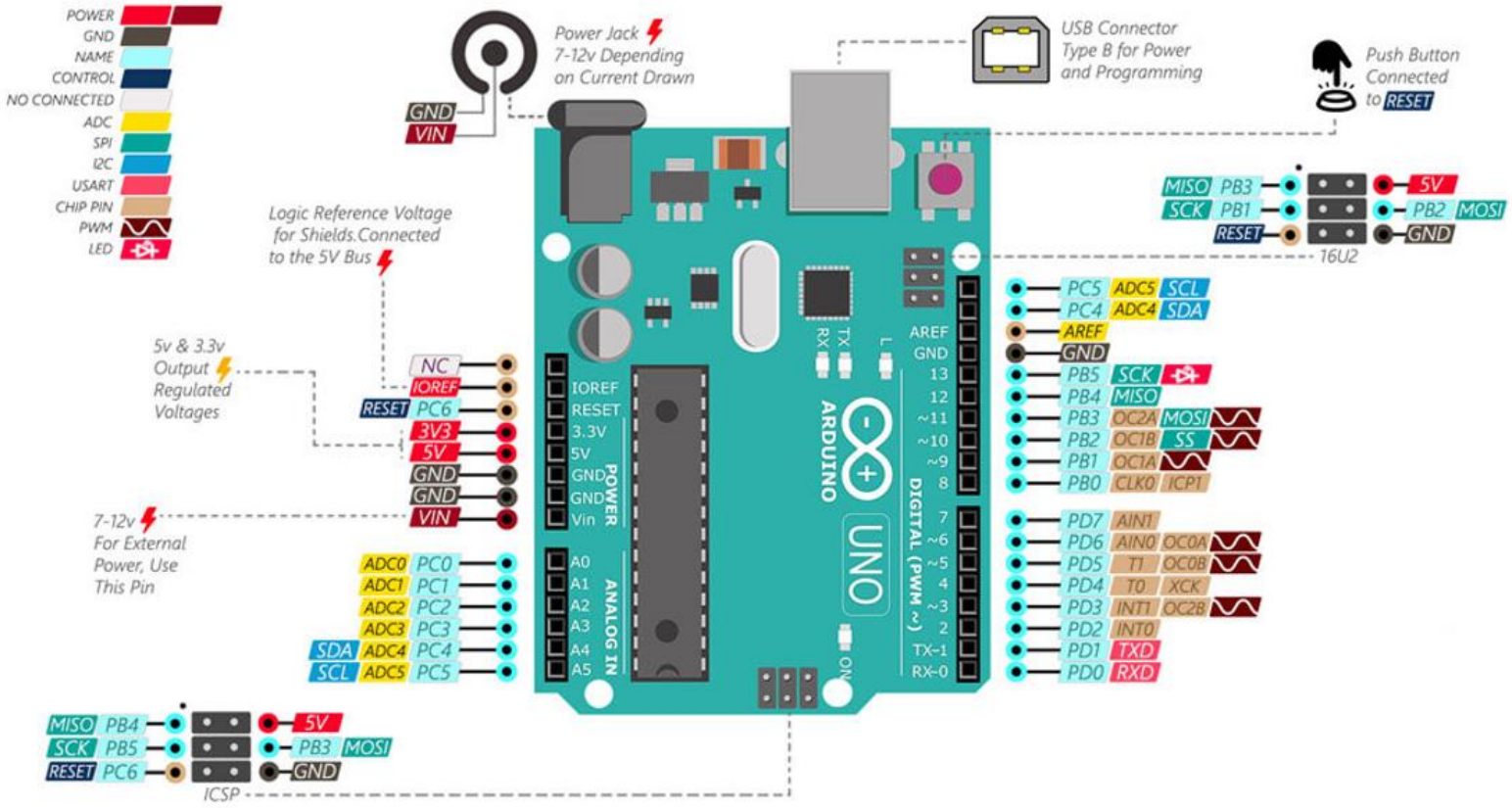


Figure 7) Pin diagram for the Arduino Uno Rev3 GMS Shield Sim900f. [10]

3. DS3231 Precision Clock Module (RTC)

The DS3231 Precision Clock Module (RTC) is an extremely accurate, low-power, real-time clock module. It provides the system with precise time and date information, which is particularly useful for scheduling tasks like automated lighting controls or temperature adjustments.

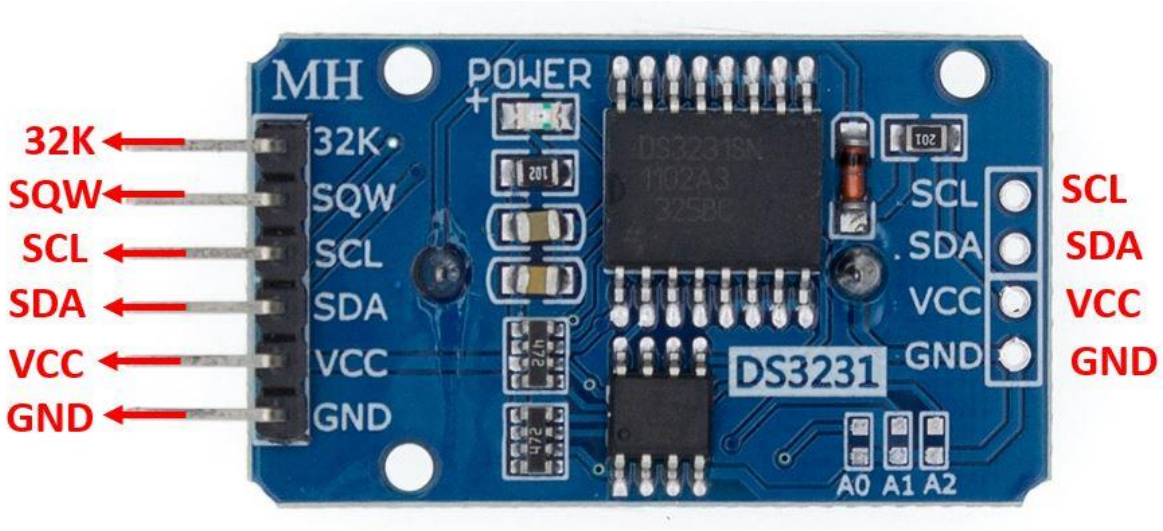


Figure 8) Depicts pins of the DS3231 Precision Clock Module (RTC). [21]

4. Humidity and Temperature Sensor - DHT11

The DHT11 is a high-accuracy, digital temperature and humidity sensor. It interfaces seamlessly with the ESP8266 NodeMCU and measures the ambient temperature and humidity levels in the home. This data is essential for maintaining optimal indoor conditions, triggering actions like adjusting the thermostat or activating the dehumidifier.

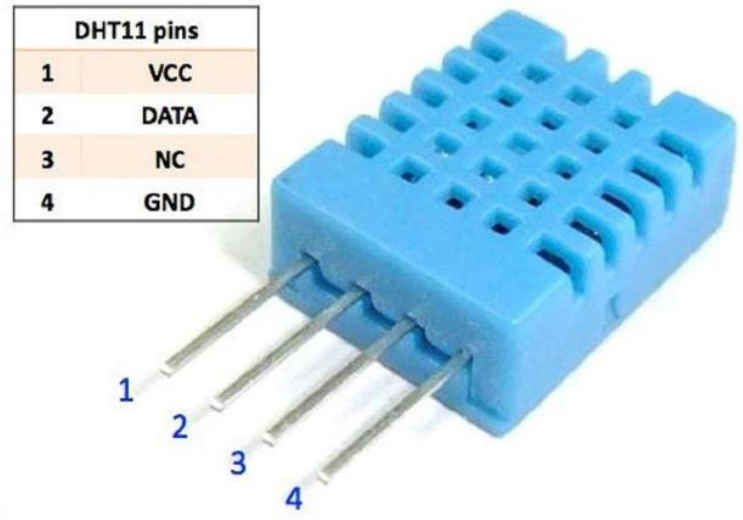


Figure 9) Depicts the DHT11 - AHT20 Pin Module - I2C Temperature and Humidity Sensor [5]

5. **Smoke Detection: MQ2 Gas Sensor**

   The MQ2 Gas Sensor is an essential part of our home safety system. It continuously monitors the air for traces of smoke and various combustible gases. In the event of a dangerous rise in gas levels, the system can alert the user and trigger necessary actions, providing crucial protection against fire hazards.

Figure 10) Depicts the Grove - Gas Sensor (MQ2). [7]

6. **Hall Effect Magnetic Sensor Module (3144E A3144 KY-003)**

   The Hall Effect Magnetic Sensor Module is a crucial component of our home automation system's security and automation features. It detects the presence of magnetic fields, which are typically associated with the operation of certain devices and mechanisms, such as door/window openings and closings. Upon detecting a change in a magnetic field (for example, when a door equipped with a magnet is opened), the system can send an alert to the user via the mobile application or trigger an alarm, providing additional security. This versatile sensor by MUZHI can be easily integrated into various Arduino, PIC, AVR, and Smart Car systems.

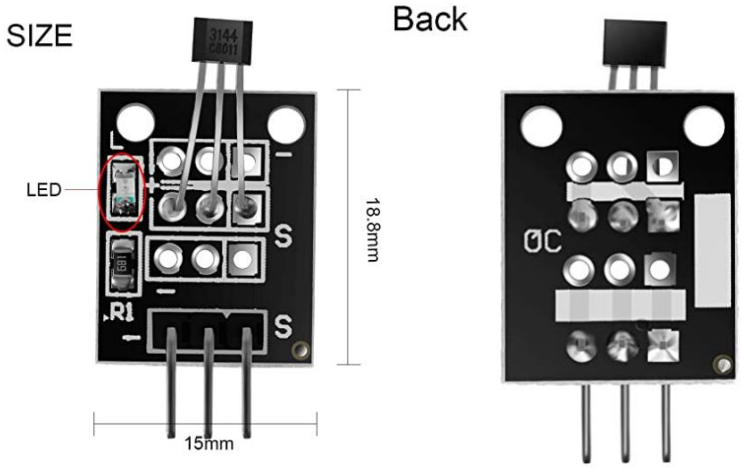


Figure 12) Depicts the Hall Effect Magnetic Sensor Module (3144E A3144 KY-003) [20]

7. **LIBO Electric Bolt Lock 2 Wires Lines DC 12V Fail-Safe Normal Temperature Electronic Lock**

The LIBO Electric Bolt Lock, a DC 12V 2-wire fail-safe electronic locking device, plays a critical role in our smart home automation system by providing secure, efficient control of the smart door lock mechanism. This robust lock, designed to operate with glass, wooden, and iron doors, is engineered to actuate upon receiving commands from the mobile app interface. Its fail-safe mechanism ensures that the lock automatically disengages during a power failure, providing an additional layer of security in emergency situations. With normal temperature operation, this electronic lock functions reliably in a wide range of ambient conditions. Its low power consumption, coupled with its high durability, makes it an ideal choice for a home automation system where the emphasis is on both security and energy efficiency.

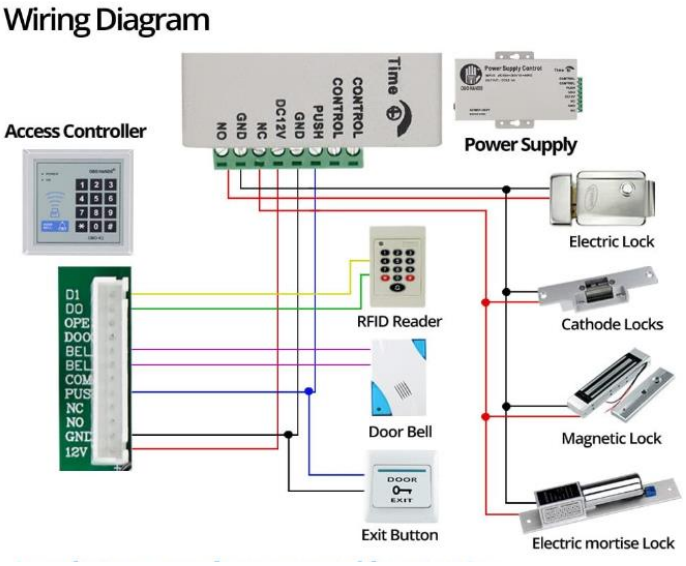


Figure 13) Depicts the wiring diagram of the deadbolt used.

8. **Liquid Crystal Display**

The LCD screen serves a pivotal role in conveying crucial information to the user, ensuring a clear and concise display of significant system statuses, temperature readings, and other pertinent details. This enhances the overall intuitiveness and user-friendly aspect of the interface. Furthermore, it offers convenience when inputting a passcode into the 4x4 keypad, as users can visually confirm their input, made possible with a 16x2 LCD. While the connections for the LCD may initially seem complex, as illustrated in FigureLCD, this complexity is significantly mitigated by employing an i2c backpack that can be directly attached to the LCD, simplifying the connections down to just 4 pins. This allows for an effortless connection of the LCD with the I2C backpack to the Arduino Uno, as shown in FigureLCDtoArduino. Additionally, our choice of an LCD display integrated with an I2C chip is not only cost-effective but also contributes to savings in the final budget. [16]

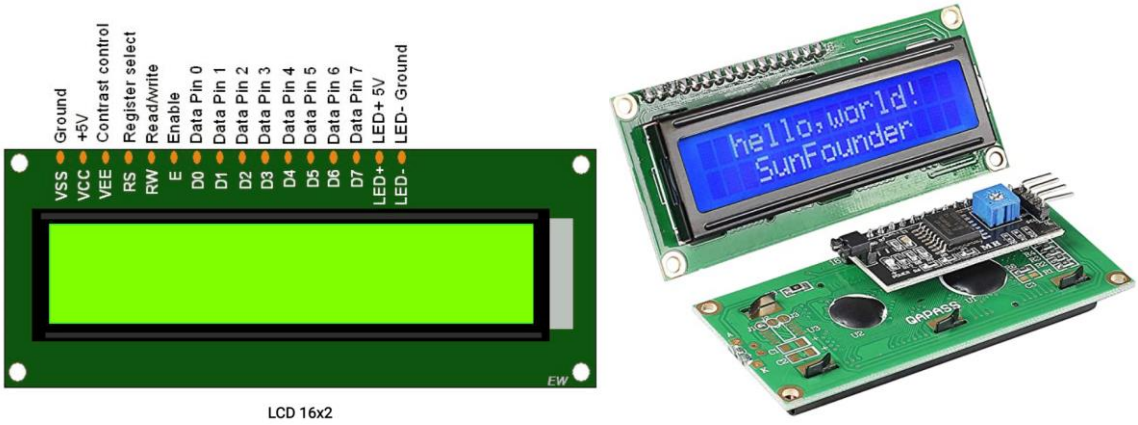


Figure 14) There are 16 pins on the LCD. The 16 pins are soldered onto the i2c chip, condensing our connections to only 4 pins (Vcc, Ground, SCL, SDA). [16]

9. 4x4 Keypad 16 Key Matrix Array

This 16-button keypad offers a straightforward method for user interaction with the system. It enables users to enter security passcodes and to control various aspects of the smart home system, from temperature to lighting.

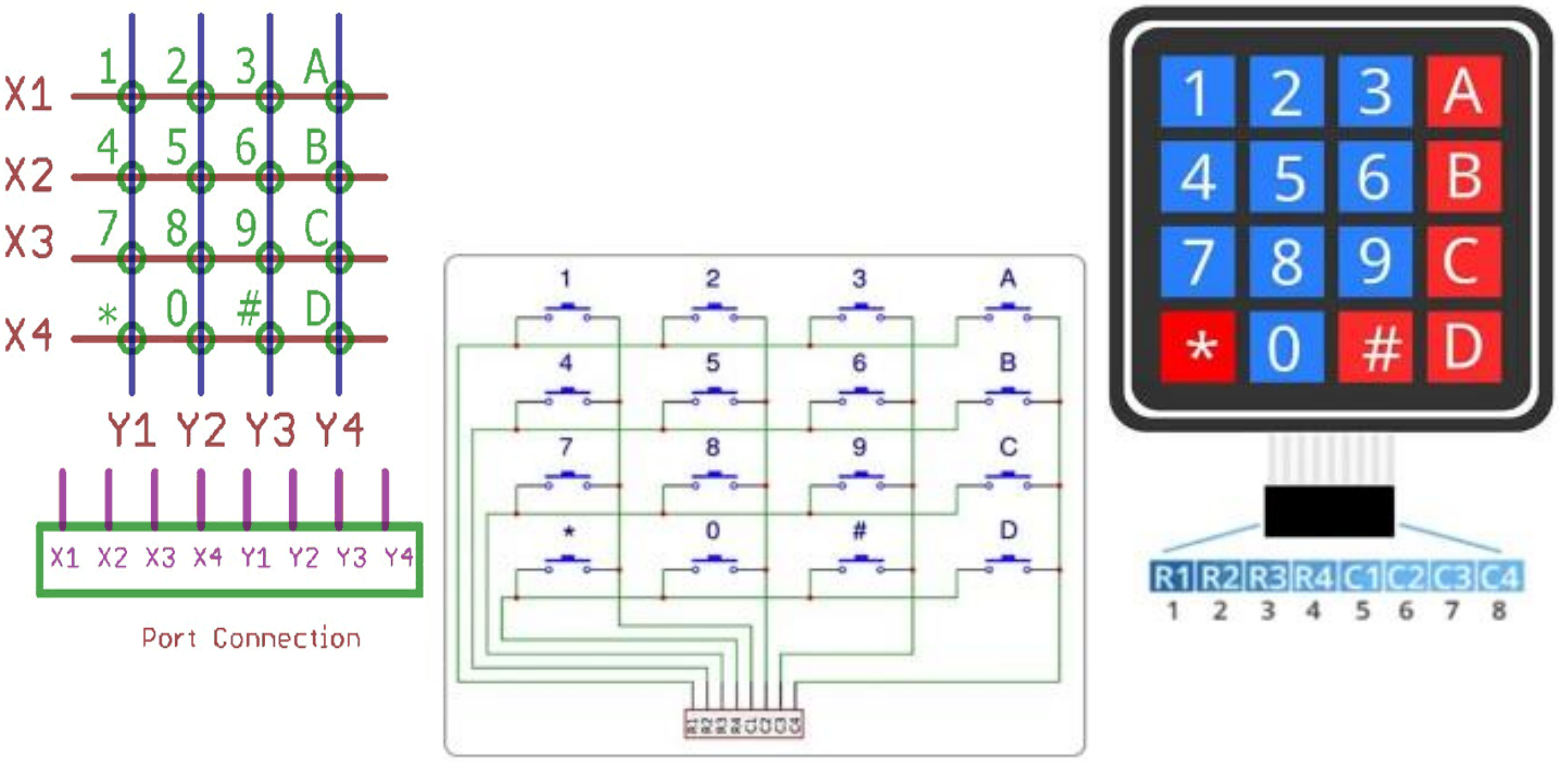


Figure 15) Pictures of the DIYmalls 4x4 keypad and its accompanying connections diagram depicting the 4 rows and columns that correspond to the 8 pins. [14]

10. 5V 1 Channel Level Trigger Optocoupler Relay Module

The Optocoupler Relay Module serves as a crucial link between our low voltage microcontrollers and higher voltage or current devices in our system. By allowing the low-voltage signals from the microcontroller to control higher voltage operations, it plays a crucial role in our home automation system.

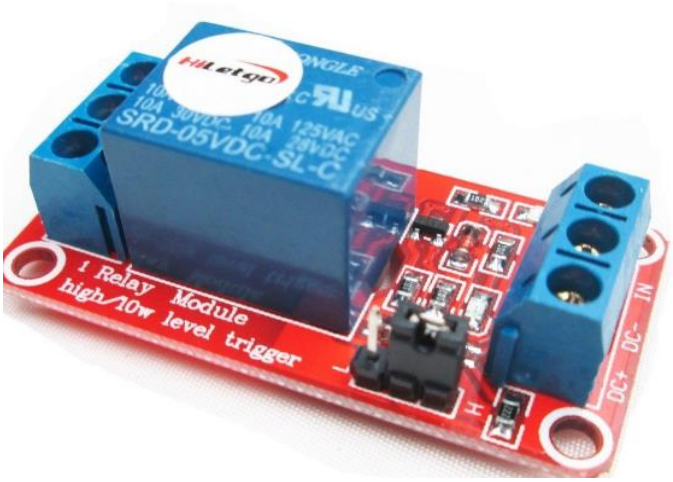


Figure 16) Depicts the 5V 1 Channel Relay Module With optocoupler which supports a High or Low-Level Trigger. [15]

11. LED – RGB Clear Common Cathode

This RGB LED is a multi-color light source that we'll use to provide color control in our smart lighting system. Each of the LED's red, green, and blue elements can be individually controlled through the ESP8266 NodeMCU, allowing users to customize their home lighting's color via the mobile application.

Figure 17) Depicts an example of the LED used.

12. 9V-MN1604 Battery

The 9V-MN1604 Battery plays a pivotal role in ensuring the continuous operation of our home automation system. As a reliable power source, it provides the necessary energy for various components, such as the Hall Effect Magnetic Sensor Module and the Arduino Uno Rev3, to function. It ensures that the system remains operational even during power outages or disruptions, adding to the overall reliability and convenience of the smart home system. Moreover, its widespread availability and long life span make it an ideal choice for such an application.

Figure 18) Depicts the 9V Alkaline battery along with the Snap Connector I-Type to PH2.0 Connector Test Cable.

13. Sim800L GPRS Module

The SIM800L GPRS Module serves as a pivotal element in our smart home automation system, bridging the gap between local operation and global communication. This compact module, capable of processing GSM/GPRS data, facilitates the transmission and reception of commands and data between the system and the user's mobile application. Upon receiving signals from the Arduino, the SIM800L uses its GPRS functionality to transmit these data to a cloud server, allowing real-time interaction with the user interface regardless of geographical location. Conversely, it can receive commands from the cloud, interpreted by the Arduino to perform relevant tasks in the home automation system. In case of a security alert or change in environmental conditions, this module plays a critical role in rapidly communicating the information to the user, enhancing the system's efficiency and reliability. Additionally, its low power consumption and compact size make it an ideal choice for an IoT-based home automation system.

Figure 19) Depicts the Sim800L GPRS Module

II. ASSEMBLY

Connection of the NodeMCU with other Components

The NodeMCU ESP8266 acts as the central controller for the smart home automation system, interfacing with the RGB LEDs, gas sensors, temperature sensors, an Arduino Uno board, and Google Firebase. Here's how these connections are established:

- *RGB LEDs*:
  The RGB LEDs have four pins each - Red, Green, Blue, and Ground. The Red, Green, and Blue pins of the LEDs are connected to three PWM capable digital output pins on the NodeMCU (e.g., D1, D2, and D3, respectively). Each of these connections requires a 220-ohm resistor to limit the current flow to the LEDs, protecting them from damage. The Ground pin of the LEDs is connected to a GND pin on the NodeMCU.

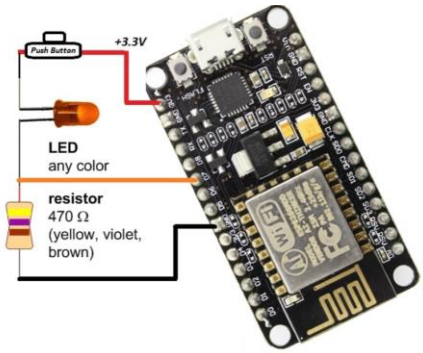


Figure 20) Depicts the connections between the LED lights and the NodeMCU.

- *MQ2 Gas Sensor*:
  The MQ2 Gas Sensor has four pins - VCC, GND, Digital Output (DO), and Analog Output (AO). The VCC pin is connected to a 3.3V pin on the NodeMCU and the GND pin to one of the NodeMCU's GND pins. The DO pin isn't used in this setup, and the AO pin is connected to an analog input pin on the NodeMCU (e.g., A0).

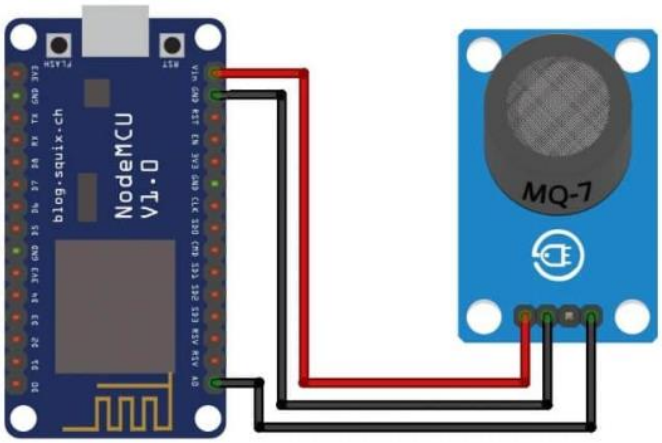


Figure 21) Depicts the connections between the Gas Sensor and the NodeMCU. [27]

- *Temperature and Humidity Sensor (DHT11):*
  The DHT11 sensor has three pins - VCC, GND, and Data. The VCC pin is connected to a 3.3V pin on the NodeMCU, the GND pin to a GND pin on the NodeMCU, and the Data pin to a digital input/output pin on the NodeMCU (e.g., D4). A 10k-ohm pull-up resistor is placed between the VCC and Data pins to ensure reliable data transmission.

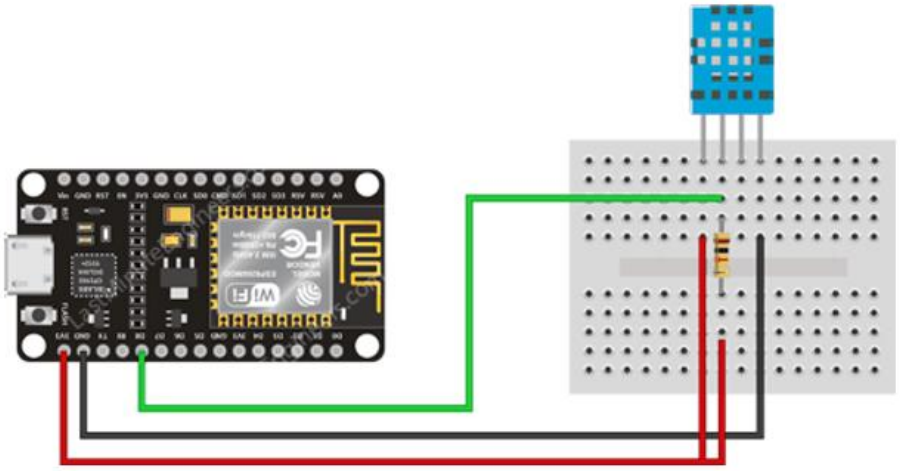


Figure 22) Depicts the connections between the Temp. and Humidity Sensor with the NodeMCU. [27]

- *Arduino Uno Board*:
  The NodeMCU ESP8266 module and the Arduino Uno with a SIM900f GMS Shield are both integral components of our IoT-based smart home automation project. The NodeMCU acts as the master microprocessor, handling data collection, processing, and transmission to Firebase, while the Arduino interfaces with additional sensors and modules that require more substantial power or resources. The NodeMCU communicates with the Arduino Uno via the UART serial communication interface, allowing both devices to exchange data seamlessly.

  1. *UART Serial Communication*:
     The NodeMCU ESP8266 connects to the Arduino Uno through a UART interface, which facilitates asynchronous serial communication between the two devices. Specifically, the RX pin (pin D1) of the NodeMCU should be connected to the TX pin (pin 1) of the Arduino, while the TX pin (pin D2) of the NodeMCU should be connected to the RX pin (pin 0) of the Arduino. This wiring setup allows the NodeMCU to transmit data to the Arduino and vice versa, creating an effective two-way communication channel.

  2. *Voltage Divider for TX Line:*
     Since the NodeMCU operates at 3.3V logic and the Arduino Uno operates at 5V logic, a voltage divider is needed to step down the voltage from the NodeMCU's TX pin to avoid damaging the Arduino's RX pin. This can be achieved by using two

resistors. For example, you can connect a 10kΩ resistor between the NodeMCU's TX pin and the Arduino's RX pin, and a 20kΩ resistor between the Arduino's RX pin and GND. This configuration will drop the voltage down to a safe level for the Arduino.

3. *Power Supply:*
   Both devices are powered separately to ensure stable operation. The NodeMCU can be powered via its micro-USB port, whereas the Arduino Uno can be powered via its barrel jack connector or USB port.

4. *Common Ground:*
   The GND pin of the NodeMCU is connected to a GND pin on the Arduino Uno. Establishing a common ground is essential for proper voltage level referencing during serial communication.

Once the NodeMCU and Arduino are interconnected, they can effectively collaborate, with the NodeMCU handling data processing and cloud connectivity while the Arduino manages the auxiliary components and modules. Care should be taken to ensure the right voltage level compatibility and correct pin connections, to avoid damage to either of the devices.

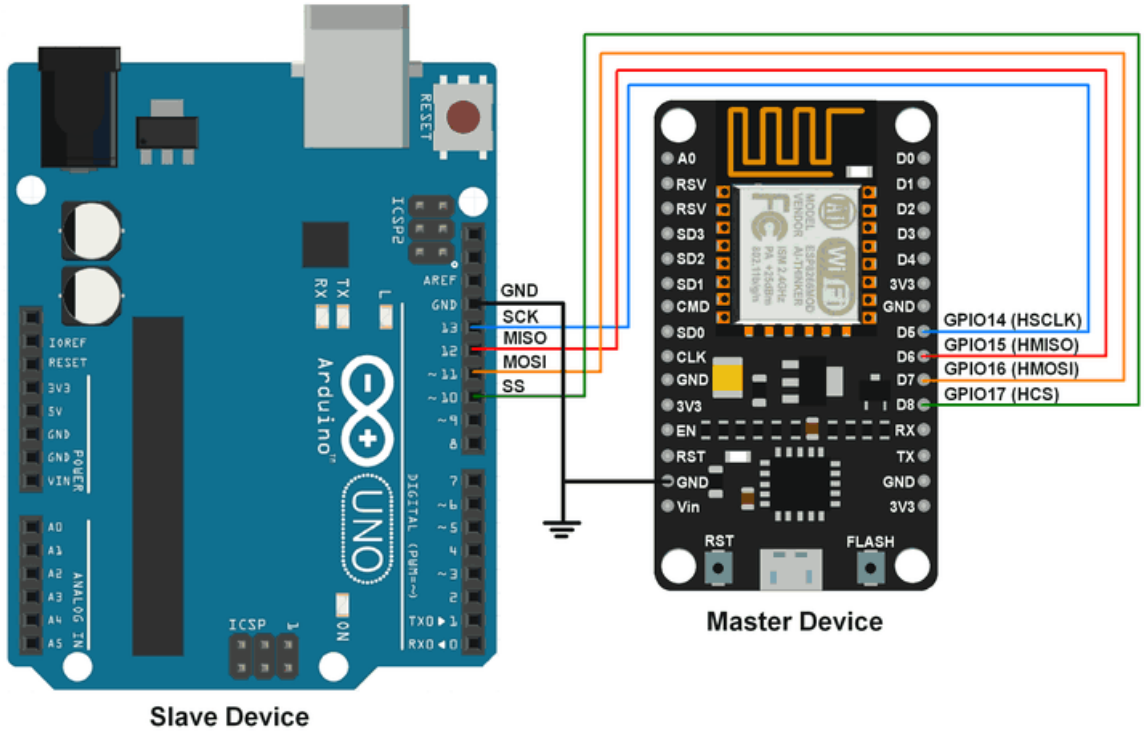


Figure 23) Depicts how once could create a master slave connection between the Node MCU and Arduino UNO. [26]

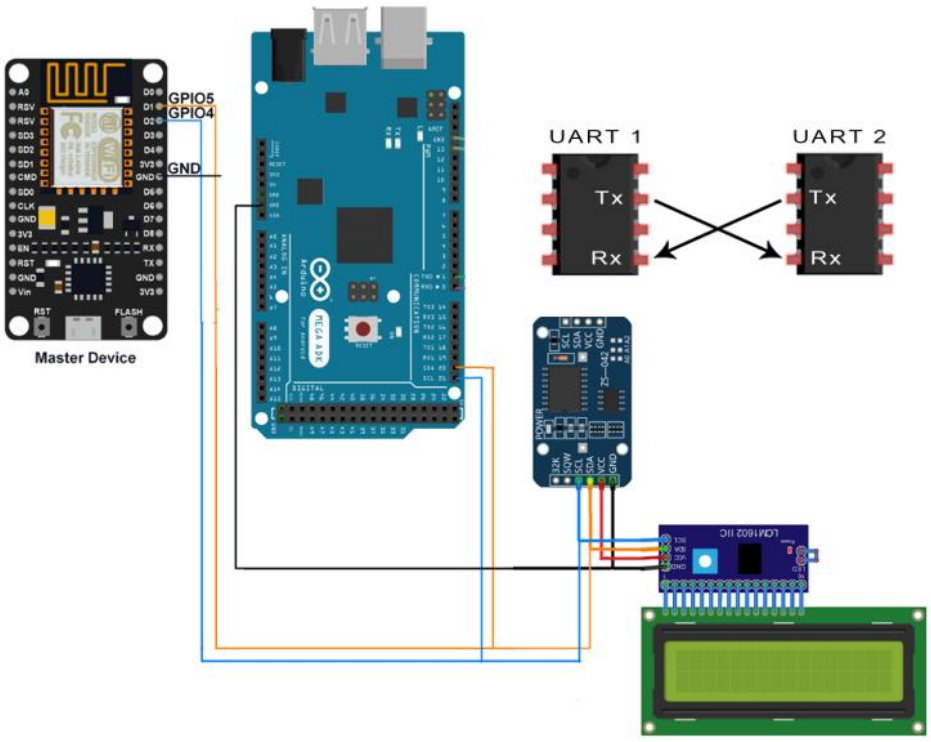


Figure 24) Illustrates the implementation of UART (Universal Asynchronous Receiver Transmitter) communication between two microcontrollers. In this configuration, the TX (transmit) pin of one microcontroller is linked with the RX (receive) pin of the other, and vice versa. This facilitates a bidirectional communication channel, allowing the Arduino to share its collected data with the NodeMCU, and likewise the NodeMCU to transmit data to the Arduino. [26]

To avoid damage, we confirm that all connections are secure and proper before turning up the NodeMCU and other components. Because the NodeMCU works at a 3.3V logic level, verify that any attached devices are voltage level compliant.

Connecting the SIM800L GPRS Module to the Arduino Uno

The SIM800L GPRS module is interconnected with the Arduino Uno by acting as a backpack that snugly sits atop the microprocessor, eliminating the need for additional wires and ensuring a streamlined integration. This crucial element in the smart home automation system interfaces using a UART serial connection.

The SIM800L module's RXD pin (Receive Data) directly syncs with the Arduino's TX (Transmit) pin, forming a data transmission link, and its TXD pin (Transmit Data) likewise is connected to the Arduino's RX (Receive) pin to complete a two-way data communication channel. Both the GND (Ground) pins of the SIM800L and Arduino are unified to establish a common grounding point.

Powering the SIM800L involves connecting its VCC pin to the 5V pin on the Arduino, necessitating a voltage regulator or voltage divider circuit to maintain the module's power requirement within a safe range of 3.4V - 4.4V. Stability of power supply is further ensured by attaching a 1000µF capacitor between the VCC and GND pins of the SIM800L module, serving to neutralize any power fluctuations.

The DS3231 Real-Time Clock Module is plugged into the breadboard with its VCC and ground pins connected to the breadboard's positive and negative terminals respectively. The SDA pin on the DS3231 routes to the Arduino's analogue 4 pin, accessible through the GSM shield, and the SCL pin to analogue pin 5 of the Arduino. This setup allows the DS3231 to maintain accurate timekeeping and feed that data back to the Arduino.

Finally, the PIR motion sensor integrates into the system with its Vcc and ground pins routed to the breadboard's positive and negative terminals respectively. The output pin of the PIR sensor interfaces with a digital pin on the Arduino, enabling the Arduino to read and process motion detection signals from the sensor.

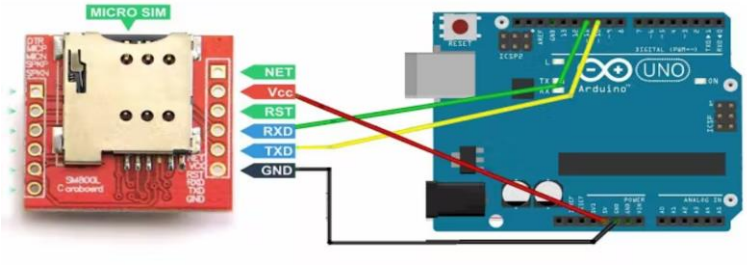


Figure 25) Sim to Arduino: The RXD and TXD pins are connected to any 2 digital pins from the Arduino, the Vcc and ground pins are connected to the Arduino's +5v and Ground pins respectively.

Connecting the GSM Shield to the Arduino Uno

The GSM Shield will plug directly into the pins of the Arduino Uno as both hardware components have the exact same pin layout and the GSM Shield we are using is built for the Arduino Uno compatibility.

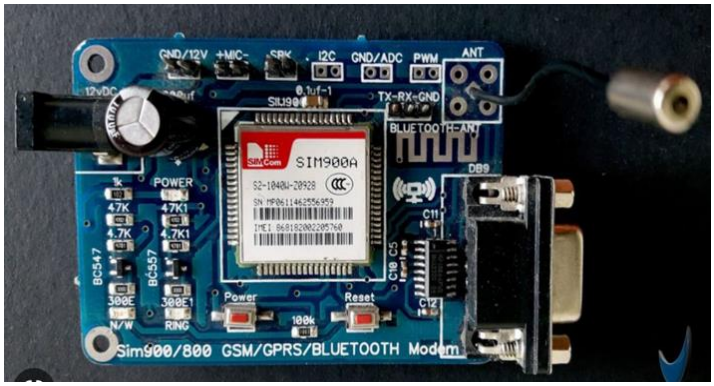


Figure 26) Depicts the GSM Shield that goes directly on top of Arduino.

Connection of the Arduino Uno Rev3 GMS Shield Sim900f with other Components

The Arduino Uno Rev3 GMS Shield Sim900f acts as an interface for the Hall Effect Magnetic Sensor, the High Torque Metal Gear Servo, the Liquid Crystal Display, the 4x4 Keypad 16 Key Matrix Array, the Optocoupler Relay Module, the RGB LED, and the 9V-MN1604 Battery. The GMS Shield is directly connected to the Arduino, which serves as a bridge for all these devices.

- *DS3231 Precision Clock Module (RTC):*

  The DS3231 connects to a breadboard where the VCC pin runs to the breadboard +, the ground pin runs to the breadboard -, and the SDA pin goes to the analog 4 pin on the Arduino which is accessible through the GSM shield on top of the Arduino. Lastly, the SCL pin goes to analog pin 5 of the Arduino Uno.

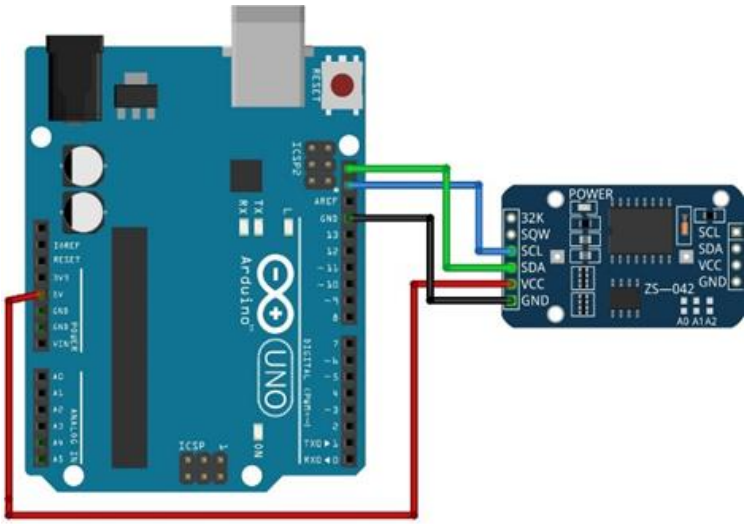


Figure 27) Connection Diagram showing DS3231 sensor to Arduino Uno [23]

- *The Hall Effect Magnetic Sensor Module*:

  The VCC pin of the sensor is connected to the 5V pin on the Arduino, and the GND pin is connected to one of the GND pins on the Arduino. The signal pin from the sensor is connected to a digital input pin on the Arduino (e.g., D2).

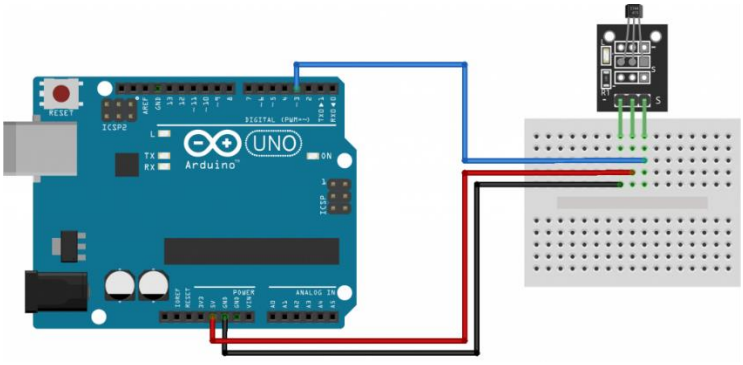


Figure 28) Depicts the Arduino to Magnetic senor connection. [23]

- *LIBO Electric Bolt Lock*

The LIBO Electric Bolt Lock functions at 12V DC, hence directly connecting it to the 5V output of the Arduino Uno will not provide adequate power. Therefore, a relay is used to facilitate this connection and safely control the high-voltage lock mechanism from the low-voltage Arduino. Here are the specific connections:

1. ***Connecting the Relay to the Arduino Uno:*** The relay module's VCC pin is connected to the Arduino Uno's 5V pin, and the GND pin is connected to the GND pin on the Arduino Uno. The relay module's IN pin, which accepts the signal from the Arduino, is connected to a digital output pin on the Arduino Uno (e.g., D2). This allows the Arduino to send commands that control the relay and subsequently the electric bolt lock.
2. ***Connecting the Relay to the LIBO Electric Bolt Lock:*** The electric bolt lock consists of two wires, positive (red) and negative (black). The positive wire is connected to the Normally Open (NO) terminal of the relay module. The negative wire is connected directly to the negative terminal of the 12V power source (battery).
3. ***Providing Power:*** The 12V power source is necessary to power the bolt lock. The positive terminal of the 12V power source is connected to the Common (COM) terminal of the relay module. When the relay is activated by the Arduino, it connects the COM and NO terminals together, effectively connecting the bolt lock to the 12V power source and triggering the locking/unlocking mechanism.

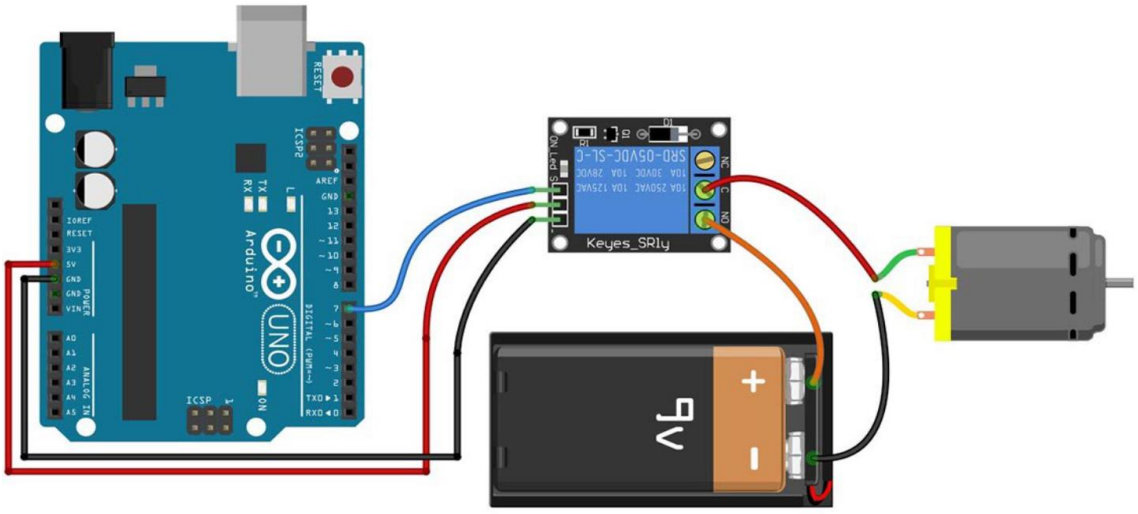


Figure 29) Basic Connection Diagram depicting the connection of the Relay to the Locking mechanism. (In this figure a magnetic deadbolt is used rather than a Servo motor) [24]

- *The Liquid Crystal Display (LCD)*:

The SDA and SCL pins of the LCD are connected to the A4 and A5 pins of the Arduino respectively, which are the I2C interface pins.

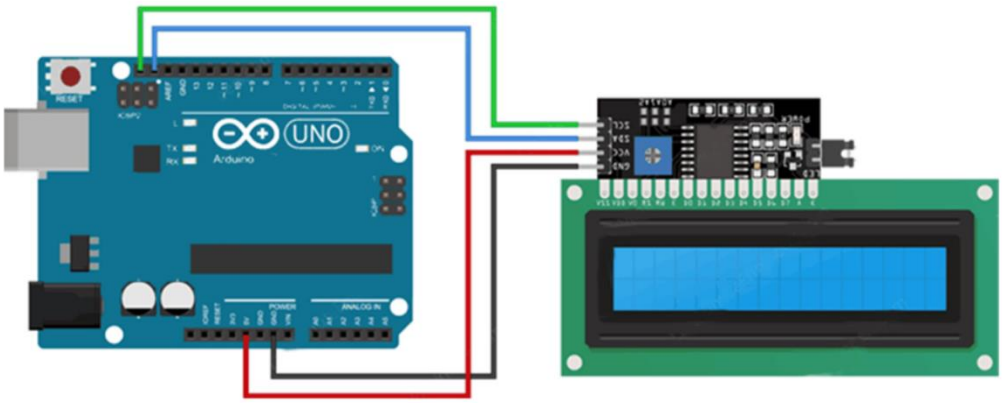


Figure 30) LCD to Arduino: we can refer to Figure 31 for the wiring chart. [25]

Arduino Uno	LCD1602 I2C display module
5V	VCC
GND	GND
A4 or SDA	SDA
A5 or SCL	SCL

Figure 31) Wiring chart between the Arduino Uno and the 16x2 LCD i2c Display module.

- *The 4x4 Keypad 16 Key Matrix Array*:

  This connects to digital pins on the Arduino (e.g., D4 to D11).

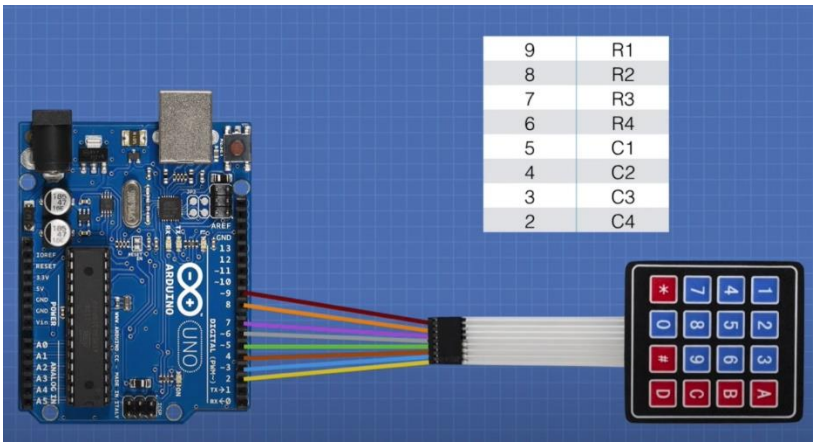


Figure 32) Depicts the Arduino to Keypad connection.

- *The Optocoupler Relay Module*: The VCC and GND of the module connect to the 5V and GND pins of the Arduino respectively, while the input pin connects to a digital output pin on the Arduino (e.g., D12).

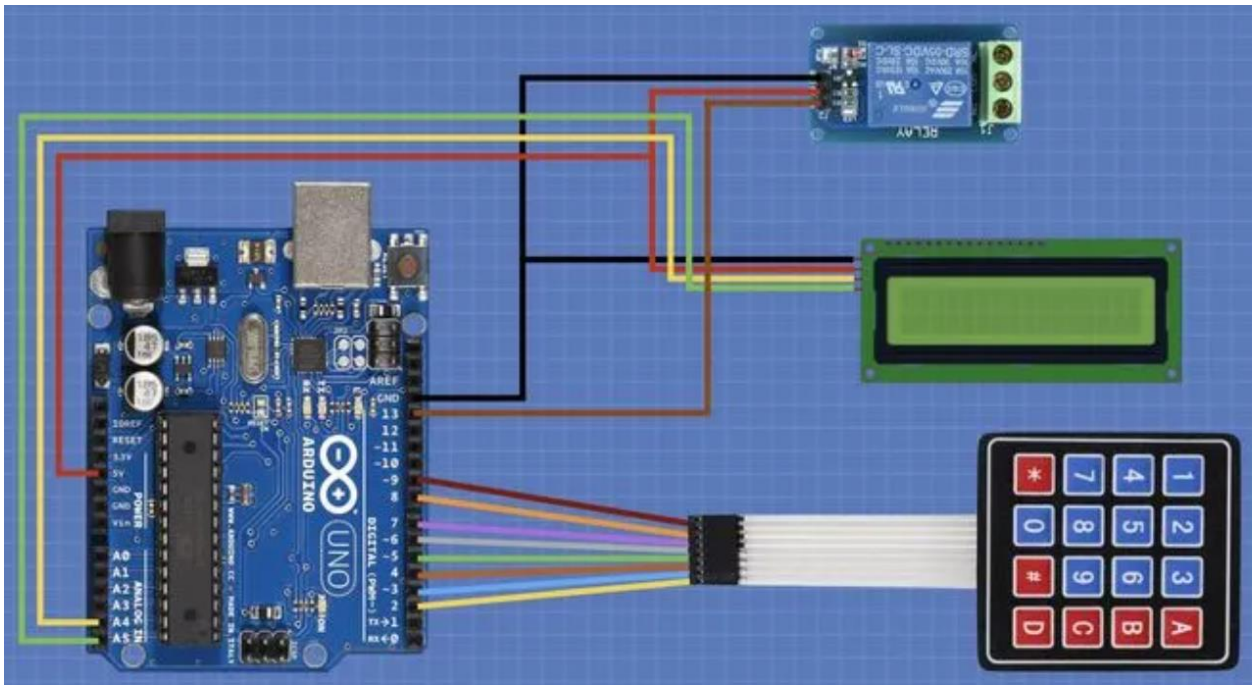


Figure 33) Depicts the Arduino connected to the keypad, LCD display and Optocoupler Relay Module.

- *The RGB LED*: The RGB pins of the LED are connected to PWM capable digital output pins on the Arduino (e.g., D9, D10, and D11 for Red, Green, and Blue, respectively), and the cathode (GND) is connected to one of the GND pins on the Arduino.
- *The 9V-MN1604 Battery*: The battery connects to the Vin pin on the Arduino Uno and the GND pin to power the Arduino and its attached devices.

Connecting to the Internet

The NodeMCU ESP8266, with its built-in Wi-Fi capabilities, connects our system to the internet. By connecting to a Wi-Fi network, the NodeMCU can communicate with the Firebase cloud service, enabling real-time data exchange. The Arduino Uno Rev3 GMS Shield Sim900f also provides internet connectivity through the cellular network when Wi-Fi isn't available.

The NodeMCU and the Arduino communicate via UART. The NodeMCU reads sensor data from the Arduino, connects to the internet, and uploads the data to Firebase. This data can then be accessed in real time from any device connected to the internet. The NodeMCU can also receive commands from Firebase, and relay them to the Arduino, to control the connected devices.

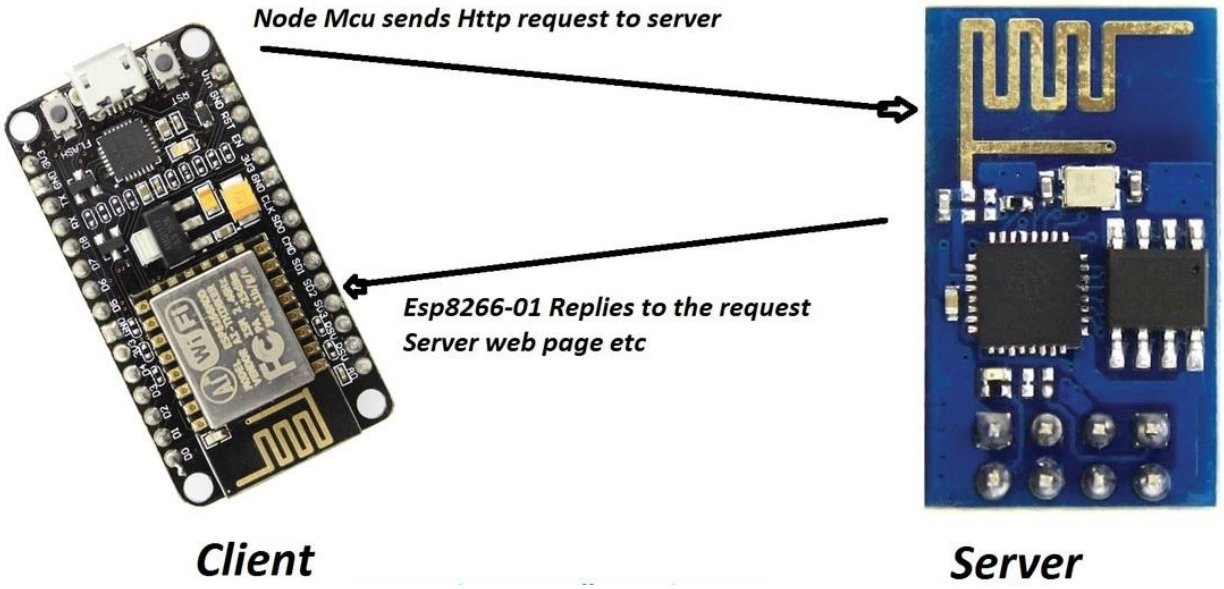


Figure 34) Depicts the NodeMCU connection to the Server.

Final Constructed Circuits

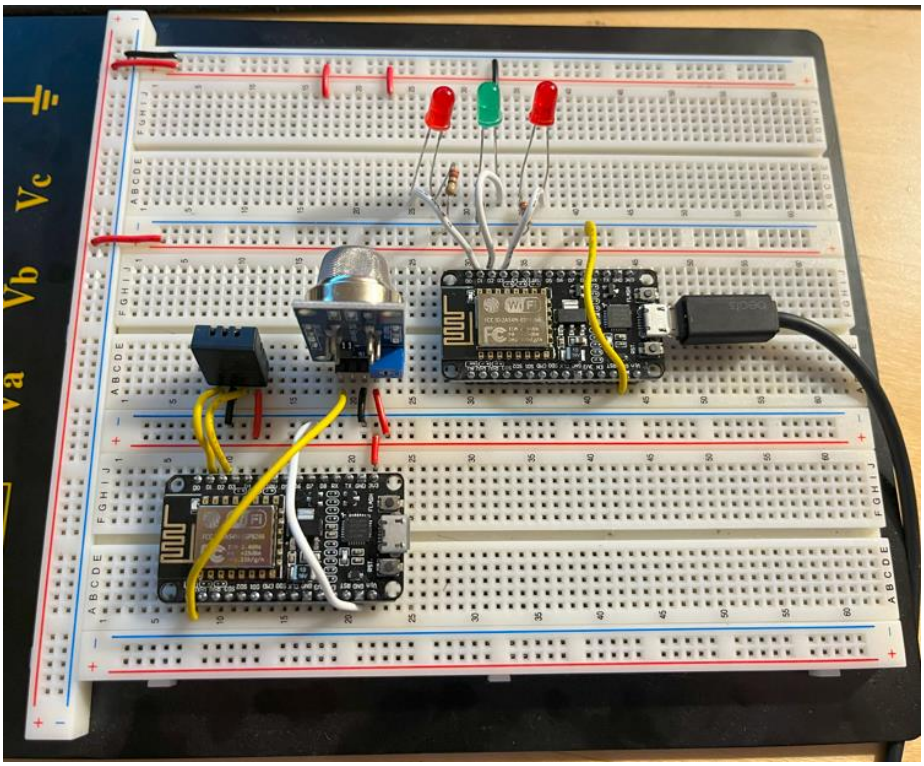


Figure 35) The GSM shield communicates via the Arduino's inbuilt USB port which is then connected to the NodeMCU, forming a bridge for the lock control system. This communication link uses the Wi-Fi capabilities of the NodeMCU, enabling us to have a successfully integrated IoT smart home automation assembly as seen in our final circuit diagram.

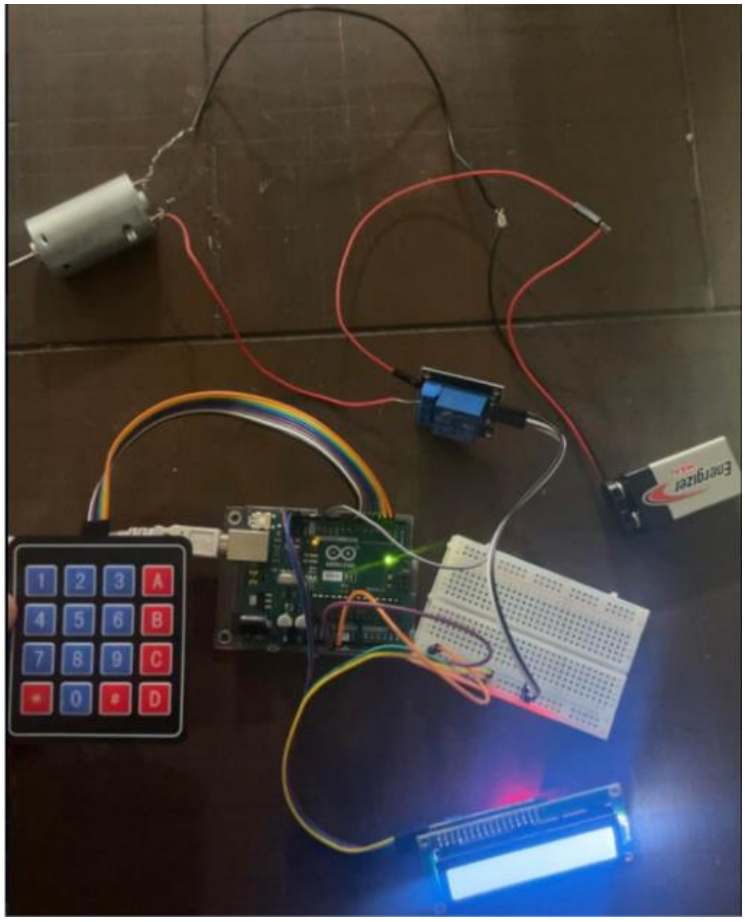


Figure 36) Depicts the constructed circuit of Figure 33.

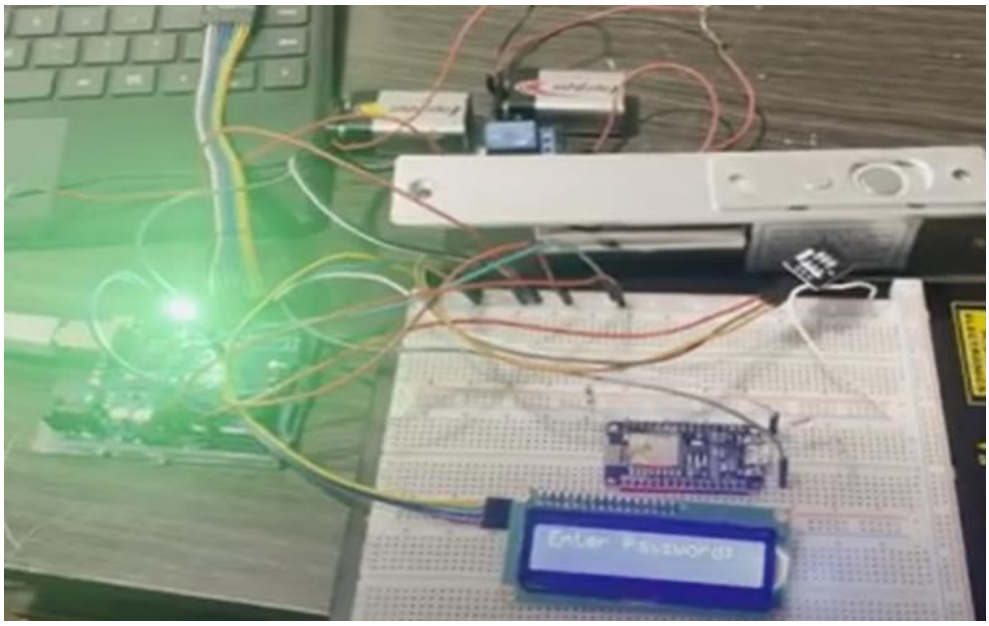


Figure 37) Depicts the completed locking and messaging system with the Arduino connected to the NodeMCU.

III. WIRELESS COMMUNICATION

A. MQTT (Message Queuing Telemetry Transport) Protocol

The MQTT is a crucial messaging protocol used for IoT-based designs, including our Smart Home Automation system. This protocol facilitates the remote devices' connection via a publish-and-subscribe messaging scheme with a small code footprint and nominal network bandwidth.

In MQTT, the clients can be categorized into publishers and subscribers. Publishers transmit data to a server (broker), while subscribers receive data from the server. Multiple clients can connect to a broker and subscribe to topics that interest them. An MQTT broker/library is essential for bridging communication between subscribers and publishers.

An MQTT session goes through four stages: connection, authentication, communication, and termination. Connection is initiated by establishing a Transmission Control Protocol/Internet Protocol (TCP/IP) connection to the broker via a standard port (1883). Session termination involves the subscriber or publisher sending a DISCONNECT message to the MQTT broker, thus closing the connection.

Our IoT-based Smart Home Automation system implements the MQTT protocol for communicating between various sensors, actuators, and the NodeMCU microcontroller. The microcontroller publishes sensor data and receives control commands from the cloud server, which we'll discuss in more detail later.

B. Google Firebase

Google Firebase serves as our app's backend. Firebase offers a suite of cloud-based tools, including a Cloud Firestore, a cloud-hosted NoSQL database that syncs data across all connected clients in real-time.

Firebase SDKs for various platforms (including Web, iOS, and Android) ensure that we can integrate Firebase into our web and mobile apps. These SDKs also facilitate user authentication, cloud messaging, analytics, and other features critical for our IoT system.

Firebase Authentication helps us build a secure authentication system for our app and website, supporting authentication methods such as email and password, Google, and more. Firebase Cloud Messaging allows us to send messages and notifications to our users, vital for real-time control and status updates of our home automation system.

C. Cloud Server

Our IoT system employs a cloud server to store sensor data and facilitate communication between the mobile app, website, and our smart home devices. Specifically, we're using a combination of Google Firebase and an MQTT broker.

Google Firebase plays a central role in our IoT system. It allows for real-time data updates across all connected clients (web, mobile app, and our smart home devices). Firebase's Firestore database stores the sensor data and the current state of all our home automation devices.

On the other hand, the MQTT broker (such as Mosquitto) handles real-time messaging between our home automation devices and the Firebase cloud server. The broker receives data from our devices, publishes this data to the appropriate Firebase database entries, and pushes control commands from the app or website to the devices.

D. Coding

This section entails the technical development and implementation of the IoT smart home project. This includes establishing efficient data structures, developing a mobile app and a website, ensuring real-time data connectivity, scaling for multiple rooms and properties, providing user-friendly navigation, and preparing the system for numerous users and interactions. The use of Firebase Firestore, an online NoSQL database, is a significant aspect of the project.

1. ***Data Structures***:
   For organizing and storing data related to the IoT smart home, appropriate data structures are crucial. A combination of dictionaries and lists is typically used. For instance, each room's sensor data could be stored in a dictionary with the sensor type as the key and the sensor readings as the value. Multiple rooms could be organized into a list. Here's an example:

```
account_data = {
    'user_id': 1,
    'properties': [
        {
            'property_id': 1,
            'name': 'Home',
            'rooms': [
                {
                    'room_id': 1,
                    'name': 'Living Room',
                    'measurements': {
                        'temperature': 72,
                        'air_quality': 0.95,
                        'lighting': 70,
                        'infrared_detection': False,
                        'door_lock_status': True
                    }
                }
            ]
        }
    ]
}
```

Figure 38) Example of a user's account with multiple properties and rooms

2. ***Firebase Setup***:
   For both the website and the app, we first need to set up Firebase. This involves creating a new project in Firebase, adding the apps to the project, and integrating the Firebase SDKs. The SDK allows us to interact with the Firebase services, including Firestore and Firebase Authentication. We use Firestore for storing our sensor data and device states, while Firebase Authentication handles user logins.

```
// Import the Firebase library
import * as firebase from 'firebase/app';
import 'firebase/auth';
import 'firebase/firestore';

// Initialize Firebase with your configuration
const firebaseConfig = {
  apiKey: 'your-api-key',
  authDomain: 'your-auth-domain',
  projectId: 'your-project-id',
  storageBucket: 'your-storage-bucket',
  messagingSenderId: 'your-messaging-sender-id',
  appId: 'your-app-id'
};

firebase.initializeApp(firebaseConfig);

// Get references to the authentication and database services
const auth = firebase.auth();
const db = firebase.firestore();
```

Figure 39) Depicts the setup of Firebase.

3. ***Laravel Backend and Firebase Integration***:
   The Laravel backend connects to Firebase for real-time data exchange. You use the Kreait\Firebase\Factory class to establish a connection with Firebase, and manipulate data using Laravel. Below is an example of how it can be done in an EspLogicController:

```
# Laravel Backend and Firebase Integration (php)
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Kreait\Firebase\Factory;

class EspLogicController extends Controller
{
    public function firebaseRead(Request $request)
    {
        $firebase = (new Factory)
            ->withServiceAccount(__DIR__.'/webautomation@01-firebase-adminsdk-5vtzr-e817d9ale6.json')
            ->withDatabaseUri('https://webautomation001-default-rtdb.firebaseio.com/');
        $database = $firebase->createDatabase();
        $getData = $database->getReference();
        return json_encode($getData->getValue());
    }

    public function firebaseSet($state)
    {
        // Similar to firebaseRead, establish a connection first
        // Then set data based on the state value
    }
}
```

Figure 40) Depicts the Laravel Backend and Firebase Integration

4. ***REST API using Flask***

Here's an example of how you can create a RESTful API using the Flask framework in Python to manage properties.

```python
# REST API using Flask
from flask import Flask, request, jsonify
from firebase_admin import firestore

app = Flask(__name__)

db = firestore.client()

@app.route('/properties', methods=['POST'])
def add_property():
    data = request.get_json()
    doc_ref = db.collection('users').document(data['user_id']).collection('properties')
    doc_ref.add(data['property'])
    return jsonify({'msg': 'Property added successfully'}), 201

@app.route('/properties/<property_id>', methods=['GET'])
def get_property(property_id):
    doc_ref = db.collection('properties').document(property_id)
    property = doc_ref.get().to_dict()
    return jsonify(property), 200
```

Figure 41) Depicts RESTful API using the Flask framework

5. ***MQTT Broker Setup:***

Next, we set up our MQTT broker, Mosquitto. The broker runs on a cloud server and has its own domain name that our devices, website, and app can connect to. The broker's main task is to route messages between our devices and the Firebase Firestore database.

```python
# Import the required modules
import paho.mqtt.client as mqtt

# Define Variables
MQTT_BROKER = "your_broker_url_or_ip"
MQTT_PORT = 1883
KEEP_ALIVE_INTERVAL = 60

# Define on connect event function
# We shall subscribe to our Topic in this function
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT Broker!")
    client.subscribe("home/room/temperature", qos=1)
    client.subscribe("home/room/humidity", qos=1)

# Define on message event function
# We shall print the received messages in this function
def on_message(client, userdata, msg):
    print("MQTT Data Received...")
    print("MQTT Topic: " + msg.topic)
    print("Data: " + str(msg.payload))

def on_subscribe(client, userdata, mid, granted_qos):
    print("Subscribed to Topic: " + str(mid) + " with Qos: " + str(granted_qos))

# Initiate MQTT Client
mqttc = mqtt.Client()

# Register event callbacks
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_subscribe = on_subscribe

# Connect with MQTT Broker
mqttc.connect(MQTT_BROKER, MQTT_PORT, KEEP_ALIVE_INTERVAL)

# Continue monitoring the incoming messages for subscribed topic
mqttc.loop_forever()
```

Figure 42) Depicts how a client connects to an MQTT broker

6. ***Firestore Security Rules***:

   Here's an example of how you can configure Firestore security rules for your application.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth.uid == userId;

      match /properties/{propertyId} {
        allow read, write: if request.auth.uid == userId;

        match /rooms/{roomId} {
          allow read, write: if request.auth.uid == userId;
        }
      }
    }
  }
}
```

Figure 43) Depicts us incorporating the Firestore Security Rules

7. ***Device Firmware***:

   The NodeMCU microcontroller firmware is written in Arduino C++. It uses the PubSubClient library for MQTT communication. The firmware reads sensor data, publishes it to the MQTT broker, and subscribes to topics for receiving control commands.

```
# Importing the required libraries
from umqtt.simple import MQTTClient
import machine

# MQTT client setup
client_id = 'nodeMCU' # create a unique id for your NodeMCU
mqtt_server = 'your_broker_url_or_ip' # replace with your MQTT broker's url or ip
client = MQTTClient(client_id, mqtt_server)

# Function to read sensor data
def read_sensor():
    # Put your sensor reading logic here
    pass

# Function to publish sensor data to MQTT
def publish_data():
    data = read_sensor() # Read sensor data
    client.publish("nodeMCU/sensor_data", str(data)) # Publish data

# Function to subscribe to a topic
def sub_cb(topic, msg):
    print((topic, msg))

client.set_callback(sub_cb)
client.subscribe("nodeMCU/commands")

# Main function
while True:
    client.check_msg() # Check for new messages
    publish_data() # Publish sensor data
```

Figure 44) Depicts examples of firmware code.

8. ***App and Website Development***:

   Our mobile app and website are designed to display live data from the Firebase Firestore database. They use Firebase's Firestore SDK, which provides real-time updates whenever database entries change. Both the app and website include an interface for users to send control commands, which are pushed to the Firestore database and then published to the appropriate MQTT topic by the MQTT broker.

9. ***Timing Function:***

   In our system, we track command execution times by logging timestamps at various stages of the command execution process. When a user sends a command via the website or app, we log the current time. When the command reaches the MQTT broker, we log the time again. Finally, when the NodeMCU executes the command, it sends a confirmation message back to the broker, which logs the time one more time. By comparing these timestamps, we can track how long it takes for commands to travel through our system and be executed.

```python
import time

# Function to log timestamp
def log_time(event):
    timestamp = time.time()
    print(f"{event} happened at {timestamp}")

# Log command sent time
log_time("Command sent")

# Log command received time in MQTT broker
log_time("Command received in MQTT broker")

# Log command execution time in NodeMCU
log_time("Command executed in NodeMCU")
```

Figure 44) Depicts us incorporating a Timer into our code.

8. ***Mobile App Development:***

   Languages commonly used for mobile app development include Swift (iOS), Java/Kotlin (Android), and JavaScript (React Native for cross-platform apps). Firebase Firestore SDK is used to interact with Firebase in these environments. A simple code snippet in Swift (iOS) to interact with Firestore might look like:

```
// Import the Firebase module
import Firebase

// Configure an instance
let db = Firestore.firestore()

// Write data
db.collection("devices").document("device1").setData(["led": "ON"]) { err in
    if let err = err {
        print("Error writing document: \(err)")
    } else {
        print("Document successfully written!")
    }
}
```

Figure 45) Depicts snippet of code in Swift (iOS) to interact with Firestore

9. *IoT Device Control:*

The firebaseSet method in EspLogicController enables controlling IoT devices such as LEDs and Servos through Firebase. It sends commands to Firebase, which are then forwarded to the respective devices. For instance, $database->getReference('/esp32read1/led')->set("ON"); will turn on an LED connected to ESP32.

10. *Website:*

JavaScript or a JavaScript-based framework, such as React.js, Vue.js, or Angular, is often used for website development. The interface with the Firestore would be identical to that of the mobile app, but in JavaScript. As an example:

```
# Website (html)
@extends('template.main')

@section('title')
    Dashboard
@endsection

@section('content')
<section class="content">
    <div class="box" style="width: 75%;">
        <div class="box-body table-responsive">
            <table class="sensortable table-striped table-bordered table-dashboard datatable">
                <thead>
                    <th width="25%">Sensor</th>
                    <th width="20%">Value</th>
                </thead>
                <tbody>
                    <tr>
                        <td>Temperature Sensor</td>
                    </tr>
                    <!-- Other sensor data here -->
                </tbody>
            </table>
        </div>
    </div>
</section>
@endsection
```

Figure 46) Depicts an example of how we'd construct the website.

11. Notifications / SMS messaging

Notifications and alerts are essential components of any IoT-powered smart home automation system. We've built a "Security Notifications" feature into our system that uses the capabilities of Firebase Cloud Messaging (FCM) to deliver real-time notifications to users about various incidents in their home.

These messages can be generated by many events such as PIR sensor motion detection, light status changes, temperature variations, smoke detection, and so on. When such occurrences occur, a notification message is created and delivered to the user's mobile device, providing real-time information on the status of their house.

Here's a Python script that acts as a Firebase cloud function, listening for changes in Firestore and sending alerts as needed:

```python
from firebase_admin import firestore, initialize_app, messaging
from flask import Flask

app = Flask(__name__)

# Initialize Firestore
default_app = initialize_app()
db = firestore.client()

# Function to send a notification
def send_notification(title, body):
    message = messaging.Message(
        notification=messaging.Notification(
            title=title,
            body=body
        ),
        topic='alerts',
    )
    response = messaging.send(message)
    print('Successfully sent message:', response)

# Firestore change listener
@app.route('/listenFirestore', methods=['POST'])
def listen_firestore():
    doc_ref = db.collection(u'rooms').document(u'bedroom')

    # Watch the document
    doc_watch = doc_ref.on_snapshot(lambda doc_snapshot, changes, read_time:
        send_notification("Alert", "Motion detected by your PIR sensor in Bedroom!") if doc_snapshot[0].get("pir") else None)

    return 'Started listening to Firestore!', 200
```

Figure 47) Depicts an example of the code that sends notifications.

To allow users to control whether they receive notifications or not, you could add an "enableNotifications" field to each user's data in Firestore. Then, check this field before sending a notification.

```python
def listen_firestore():
    doc_ref = db.collection(u'rooms').document(u'bedroom')
    user_ref = db.collection(u'users').document(u'user1')  # Fetch user data

    # Check if notifications are enabled
    user = user_ref.get().to_dict()
    if user.get('enableNotifications'):
        doc_watch = doc_ref.on_snapshot(lambda doc_snapshot, changes, read_time:
            send_notification("Alert", "Motion detected by your PIR sensor in Bedroom!") if doc_snapshot[0].get("pir") else None)
    else:
        print("Notifications are disabled for this user")

    return 'Started listening to Firestore!', 200
```

Figure 48) Depicts an example of the code that activates/deactivates notifications.

```php
# Controller (php)
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Kreait\Firebase\Factory;

class EspLogicController extends Controller
{
    public function index()
    {
    }

    public function create()
    {
    }

    public function store(Request $request)
    {
    }

    public function show($id)
    {
    }

    public function edit($id)
    {
    }

    public function update(Request $request, $id)
    {
    }

    public function destroy($id)
    {
    }
```

```php
    public function firebaseRead(Request $request)
    {
        $firebase = (new Factory)
            ->withServiceAccount(__DIR__.'/webautomation001-firebase-adminsdk-5vtzr-e817d9ale6.json')
            ->withDatabaseUri('https://webautomation001-default-rtdb.firebaseio.com/');
        $database = $firebase->createDatabase();
        $getData = $database->getReference();
        return json_encode($getData->getValue());
    }

    public function firebaseSet($state)
    {
        $firebase = (new Factory)
            ->withServiceAccount(__DIR__.'/webautomation001-firebase-adminsdk-5vtzr-e817d9a16.json')
            ->withDatabaseUri('https://91-default-rtb.firebaseio.com/');
        $database = $firebase->createDatabase();
        $getData = $database->getReference();

        switch($state){
            case "1":
                $database->getReference('/esp32read1/led')->set("ON");
                break;
            case "2":
                $database->getReference('/esp32read1/led')->set("OFF");
                break;
            case "3":
                $database->getReference('/esp32read1/servo')->set("ON");
                break;
            case "4":
                $database->getReference('/esp32read1/servo')->set("OFF");
                break;
        }
    }
}
```

Figure 49) Depicts the Controller Code.

```
# View (Dashboard.blade.php) (html)
@extends('template.main')

@push('css')
<style>
    .btn1 {
        width: 75px;
    }
    table {
        width: 100%;
        font-size: 120%;
    }
    th, td {
        padding: 5px;
        text-align: center;
    }
    .table-dashboard tbody tr:nth-child(even) {
        background-color: #cefebi;
    }
    .table-dashboard thead {
        background-color: #96bf73;
    }
</style>
@endpush

@section('title')
    Dashboard
@endsection

@section('breadcrumb')
    @parent
@endsection
```

```
@section('content')
<section class="content">
    <div class="box" style="width: 75%;">
        <div class="box-header with-border"></div>
        <div class="box-body table-responsive">
            <table class="sensortable table-striped table-bordered table-dashboard datatable">
                <thead>
                    <th width="25%">Sensor</th>
                    <th width="20%">Value</th>
                </thead>
                <tbody>
                    <tr>
                        <td>Temperature Sensor</td>
                    </tr>
                    <tr>
                        <td>Gas Sensor</td>
                        <td id="gassensor_value"></td>
                    </tr>
                    <tr>
                        <td>PIR Sensor</td>
                        <td id="pirsensor_value"></td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
    <div class="box" style="width: 75%;">
        <div class="box-header with-border">
            <!-- Additional Content Here -->
        </div>
    </div>
</section>
@endsection

@push('js')
<script>
    // Your JavaScript logic here
</script>
@endpush
```

Figure 50) Depicts the (Dashboard.blade.php) (html) Code.

```
#include <Firebase.h>
#include <FirebaseArduino.h>
#include <FirebaseCloudMessaging.h>
#include <FirebaseError.h>
#include <FirebaseHttpClient.h>
#include <FirebaseObject.h>
#include <ArduinoJson.h>
#include <memory>

#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

#define FIREBASE_HOST "seniordesigndoorv02-default-rtdb.firebaseio.com" // the project name address from firebase id
#define FIREBASE_AUTH "5gT5CsBXZ0XTMq68ywCKmD1AmLcdQMCZa5GShAe1"   // the secret key generated from firebase
#define WIFI_SSID "Fios-mFBX2" //Input Wifi SSID
#define WIFI_PASSWORD "doe569mug359bey" //Input Wifi Password


void setup() {
  Serial.begin(9600);
  delay(1000);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    //Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  if (Firebase.failed())  // connect to firebase
  {
    Serial.print(Firebase.error());
  } else {
    Serial.print("Firebase Connected");
  }

}
```

Figure 51) NodeMCU ESP 8266 Connection to Wi-Fi and Firebase

For each program the Nodemcu must make a connection to the home Wi-Fi and the firebase. This is done in a few steps. First it takes the domain of the firebase server which acts as the door and the authentication key allows the door to open thus the mcu can send data to and from the cloud. From there we also connect to the home Wi-Fi which creates the bridge from the mcu to the cloud. This setup occurs for all the applications.

```
58  void loop() {
59    {
60      fireStatusLight1 = Firebase.getString("PWM_Light1/Value");  // get ld status input from firebase
61      Serial.println(fireStatusLight1);
62      analogWrite(led1, fireStatusLight1.toInt());
63      Serial.println(fireStatusLight1);
64      delay(1000);
65    }
66    {
67      fireStatusLight2 = Firebase.getString("PWM_Light2/Value");  // get ld status input from firebase
68      Serial.println(fireStatusLight2);
69      analogWrite(led2, fireStatusLight2.toInt());
70      Serial.println(fireStatusLight2);
71      delay(1000);
72    }
73    {
74      fireStatusLight3 = Firebase.getString("PWM_Light3/Value");  // get ld status input from firebase
75      Serial.println(fireStatusLight3);
76      analogWrite(led3, fireStatusLight3.toInt());
77      Serial.println(fireStatusLight3);
78      delay(1000);
79    }
80  }
```

Figure 52) Pushing and subscribing from the Firebase Realtime Database for the Light Switch.

In the loop which will repeat on the MCU will reach into the database, look for the location of the data and send that data to the mcu. This is done with the Firebase.getString which pulls the string from the database. In the light switch application, the on and off are pulled and allows the mcu to make the decision whether to turn the light on and off. From there it cycles to the next light and continues indefinitely.

```
91
92  void loop() {
93      float temp_hum_val[2] = {0};
94      // Reading temperature or humidity takes about 250 milliseconds!
95      // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
96
97
98      if (!dht.readTempAndHumidity(temp_hum_val)) {
99
100       //Get current timestamp
101       timestamp = getTime();
102       Serial.print ("time: ");
103       Serial.println (timestamp);
104
105       parentPath= databasePath + "/" + String(timestamp);
106
107         debug.print("Humidity: ");
108         debug.print(temp_hum_val[0]);
109         debug.print(" %\t");
110         String fireHumid = String(temp_hum_val[0]) + String("%");
111         Firebase.pushString(parentPath + "/Humditiy", fireHumid);
112
113         debug.print("Temperature: ");
114         debug.print(temp_hum_val[1]);
115         debug.println(" *C");
116         String fireTemp = String(temp_hum_val[1]) + String("°C");
117         Firebase.pushString(parentPath + "/Temperature", fireTemp);
118
119     } else {
120         debug.println("Failed to get temprature and humidity value.");
121     }
122
123     delay(60000);
124 }
```

Figure 53) Pushing and subscribing from the Firebase Realtime Database for the DHT20 and MQ2 Sensors

In the case of the DHT20 and MQ2 sensors it is only pushing data to the firebase via the Firebase.pushString function. The code works simply by accessing the data from the sensor and turning the data into a string and adding a timestamp to the data. Finally, the data is pushed onto the database.

12. Sensors (Live Data Transmission)

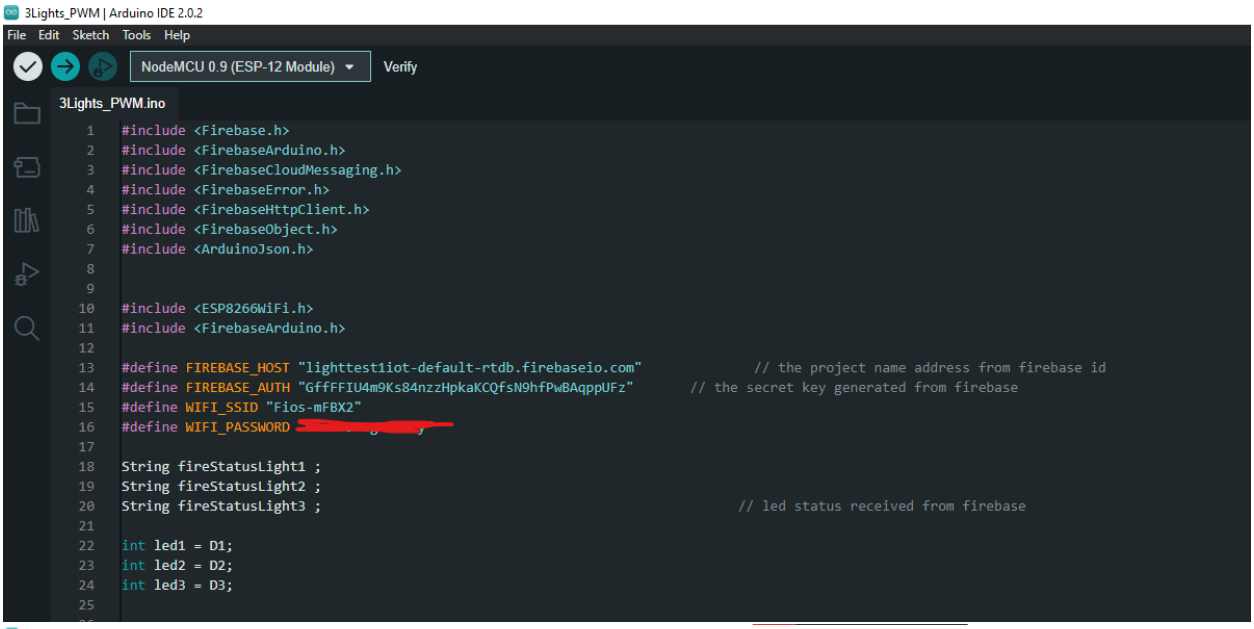
a) Light Switch using Pulse Width Modulation

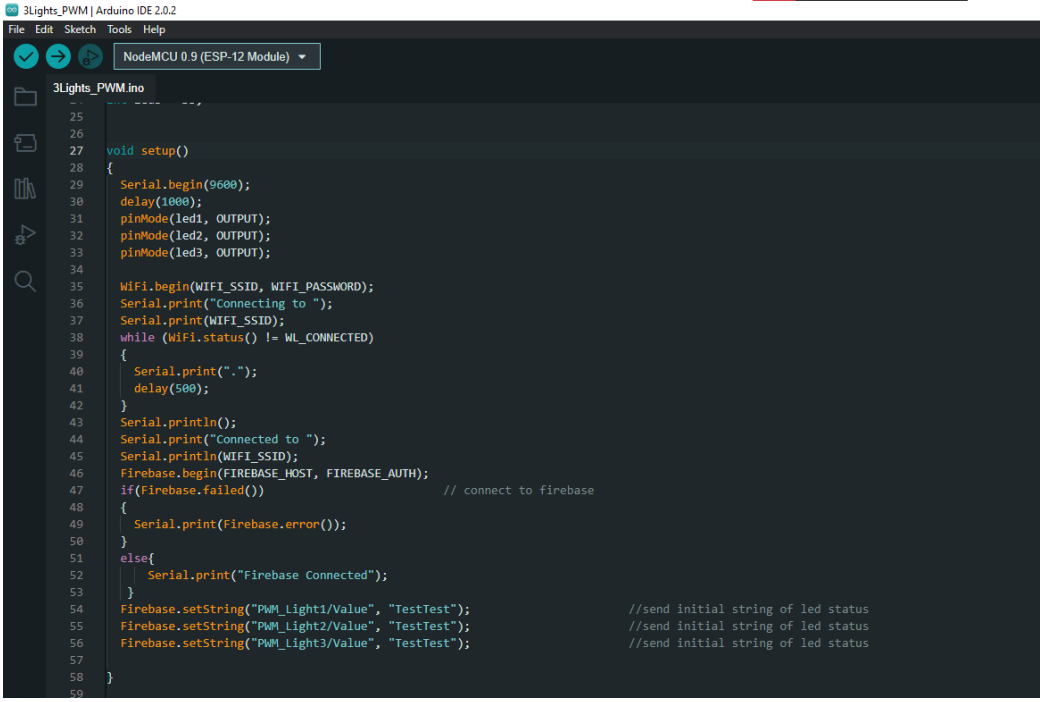
With the cutting-edge technology of Pulse Width Modulation, we can improve the functioning of a light switch. We can easily adjust the brightness of a light using this technology by sending a numerical signal to the cloud, which is then transmitted over the internet to the router, and finally to the NodeMCU that controls the light.

To enhance the user experience even further, we can incorporate a feedback loop into the Light Switch mechanism. We can accurately determine whether the switch is turned on or off by using a light sensor and relaying this data back to the cloud for further analysis.

We can easily manipulate the intensity of the light to suit our preferences using inputs ranging from 1 to 255, making this innovation an ideal choice for those looking for modern, high-tech solutions to everyday problems.

We can use Pulse Width Modulation to adjust not only the intensity of a single light, but also up to three lights from the same board at the same time. This cutting-edge technology enables us to easily control many light sources, making it a practical and efficient option for both homes and companies. We can easily manage the brightness levels of all three lights by simply inputting the required numerical numbers into the cloud, delivering a comfortable and tailored lighting atmosphere at all times.





3Lights_PWM | Arduino IDE 2.0.2

File Edit Sketch Tools Help

NodeMCU 0.9 (ESP-12 Module)

3Lights_PWM.ino

```
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  if(Firebase.failed())                       // connect to firebase
  {
    Serial.print(Firebase.error());
  }
  else{
      Serial.print("Firebase Connected");
   }
  Firebase.setString("PWM_Light1/Value", "TestTest");                    //send initial string of led status
  Firebase.setString("PWM_Light2/Value", "TestTest");                    //send initial string of led status
  Firebase.setString("PWM_Light3/Value", "TestTest");                    //send initial string of led status

}

void loop()
{
  {
  fireStatusLight1 = Firebase.getString("PWM_Light1/Value");                                  // get ld status input from firebase
  Serial.println(fireStatusLight1);
  analogWrite(led1, fireStatusLight1.toInt());
  Serial.println(fireStatusLight1);
  delay(1000);
  }
  {
  fireStatusLight2 = Firebase.getString("PWM_Light2/Value");                                  // get ld status input from firebase
  Serial.println(fireStatusLight2);
  analogWrite(led2, fireStatusLight2.toInt());
  Serial.println(fireStatusLight2);
  delay(1000);
  }
  {
  fireStatusLight3 = Firebase.getString("PWM_Light3/Value");                                  // get ld status input from firebase
  Serial.println(fireStatusLight3);
  analogWrite(led3, fireStatusLight3.toInt());
  Serial.println(fireStatusLight3);
  delay(1000);
  }
}
```

Figure 54) Depicts the code that controls the lights.

The code for the Cloud is as follows:

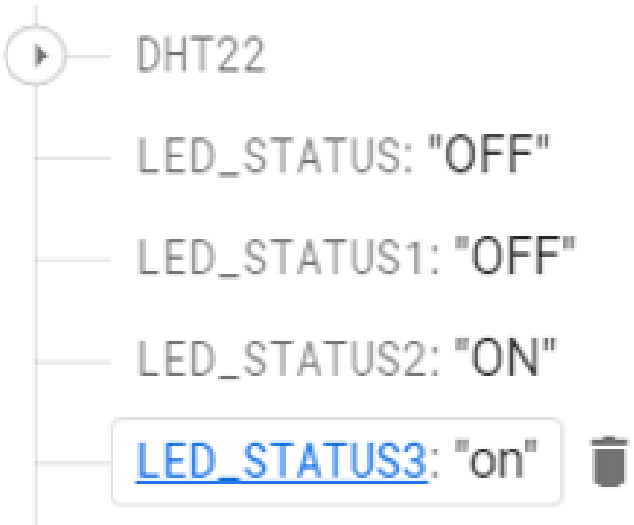


Figure 55) Depicts the code for the lights within the database.

b) Gas & Temperature Sensors

```
#include "Wire.h"
#include <DHT.h>                                                    //
dht11 temperature and humidity sensor library
#include <Firebase.h>
#include <FirebaseCloudMessaging.h>
#include <FirebaseError.h>
#include <FirebaseHttpClient.h>
#include <FirebaseObject.h>
#include <Adafruit_Sensor.h>


#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

#define FIREBASE_HOST "lighttest1iot-default-rtdb.firebaseio.com"              //
the project name address from firebase id
#define FIREBASE_AUTH "GffFFIU4m9Ks84nzzHpkaKCQfsN9hfPwBAqppUFz"       // the
secret key generated from firebase
#define WIFI_SSID "Fios-mFBX2"
#define WIFI_PASSWORD "xxx"


#define DHTTYPE DHT20                                               //

DHT dht(DHTTYPE);

#if defined(ARDUINO_ARCH_AVR)
    #define debug  Serial

#elif defined(ARDUINO_ARCH_SAMD) ||  defined(ARDUINO_ARCH_SAM)
    #define debug  SerialUSB
#else
    #define debug  Serial
#endif



void setup()
{
  Wire.begin();
  Serial.begin(9600);
  delay(1000);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to ");
```

```
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  if(Firebase.failed())                          // connect to firebase
  {
    Serial.print(Firebase.error());
  }
  else{
      Serial.print("Firebase Connected");
   }
  dht.begin();                                                          //St
art reading dht sensor
}


void loop() {
    float temp_hum_val[2] = {0};



    if (!dht.readTempAndHumidity(temp_hum_val)) {
        debug.print("Humidity: ");
        debug.print(temp_hum_val[0]);
        debug.print(" %\t");
        String fireHumid = String(temp_hum_val[0]) + String("%");
        Firebase.pushString("DHT22/Humidity", fireHumid);
        debug.print("Temperature: ");
        debug.print(temp_hum_val[1]);
        debug.println(" *C");
        String fireTemp = String(temp_hum_val[1]) + String("°C");
        Firebase.pushString("DHT22/Temperature", fireTemp);
    } else {
        debug.println("Failed to get temprature and humidity value.");
    }

    delay(1500);
}
```

Figure 56) Depicts the code that reads the temperature and gas values.

c) Door Locking

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>

// Length of password + 1 for null character
#define Password_Length 8
// Character to hold password input
char Data[Password_Length];
// Password
char Master[Password_Length] = "1234567";

int lockOutput = 12;
int greenLed = 11;
int redLed = 10;
int hallEffectPin = A0;
int state = 0;

// Counter for character entries
byte data_count = 0;

// Character to hold key input
char customKey;

// Constants for row and column sizes
const byte ROWS = 4;
const byte COLS = 4;

// Array to represent keys on keypad
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```

```
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

// Create keypad object
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

// Create LCD object
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  // Setup LCD with backlight and initialize
  lcd.backlight();
  lcd.init();

  // Set lockOutput as an OUTPUT pin
  pinMode(lockOutput, OUTPUT);

  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(hallEffectPin, INPUT);
}
void loop() {

  // Initialize LCD and print
  lcd.setCursor(0, 0);
  lcd.print("Enter Password:");

  // Look for keypress
  customKey = customKeypad.getKey();
  if (customKey) {
    // Enter keypress into array and increment counter
    Data[data_count] = customKey;
    lcd.setCursor(data_count, 1);
    lcd.print(Data[data_count]);
    data_count++;
  }

  // See if we have reached the password length
  if (data_count == Password_Length - 1) {
    lcd.clear();

    if (!strcmp(Data, Master)) {
      // Password is correct
      lcd.print("Correct");
      // Turn on relay for 5 seconds
      digitalWrite(lockOutput, HIGH);
      delay(5000);
      digitalWrite(lockOutput, LOW);
    }
    else {
      // Password is incorrect
      lcd.print("Incorrect");
      delay(1000);
    }

```

```
93      // Clear data and LCD display
94      lcd.clear();
95      clearData();
96
97    state = digitalRead(hallEffectPin);
98
99     if (state == LOW)  {
100      digitalWrite(redLed, HIGH);
101      digitalWrite(greenLed, LOW);
102    }
103    else{
104      digitalWrite(redLed, LOW);
105      digitalWrite(greenLed, HIGH);
106    }
107
108    }
109  }
110
111  void clearData() {
112    // Go through array and clear data
113    while (data_count != 0) {
114      Data[data_count--] = 0;
115    }
116    return;
117  }
```

Figure 57) Depicts the code that controls the locking mechanism.

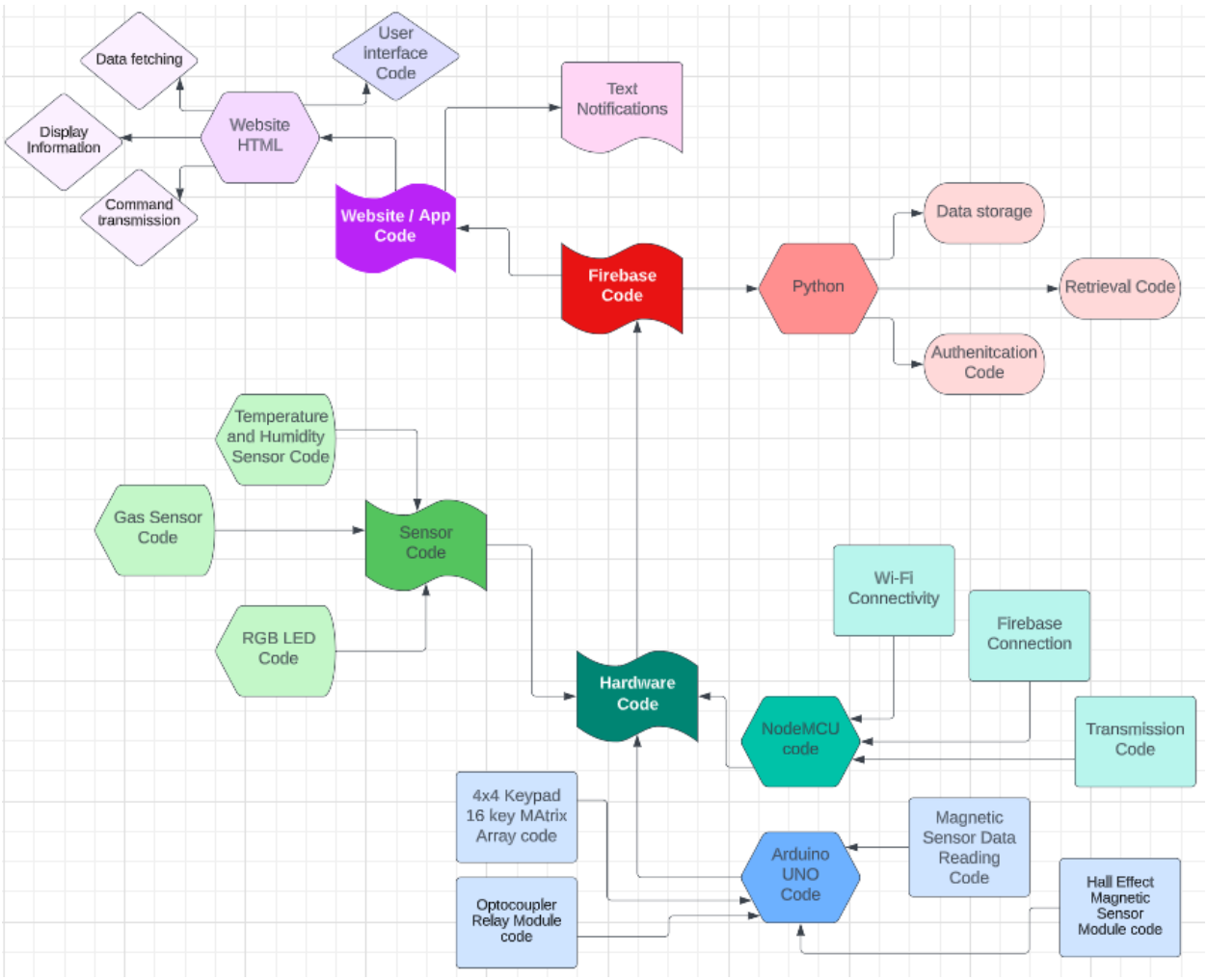


Figure 58) Depicts a simplified flowchart of the Code Hierarchy in its entirety.

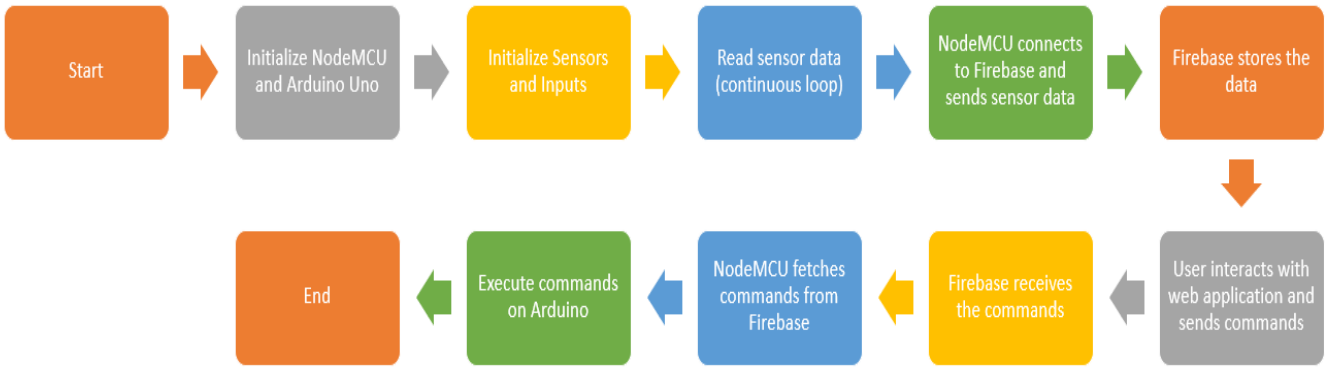


Figure 59) Depicts the Sensor Functionality and Communication Protocol.

IV. WEB APPLICATION

A. Functionality & Overview of Web Application Process

Our IoT smart home system's web application is designed to provide users with a seamless, real-time experience of interacting with their smart devices. The website updates with a frequency of 1-3 minutes or depending on how fast the device states change in real-time.

Below is a broad overview of how data flows within our system:

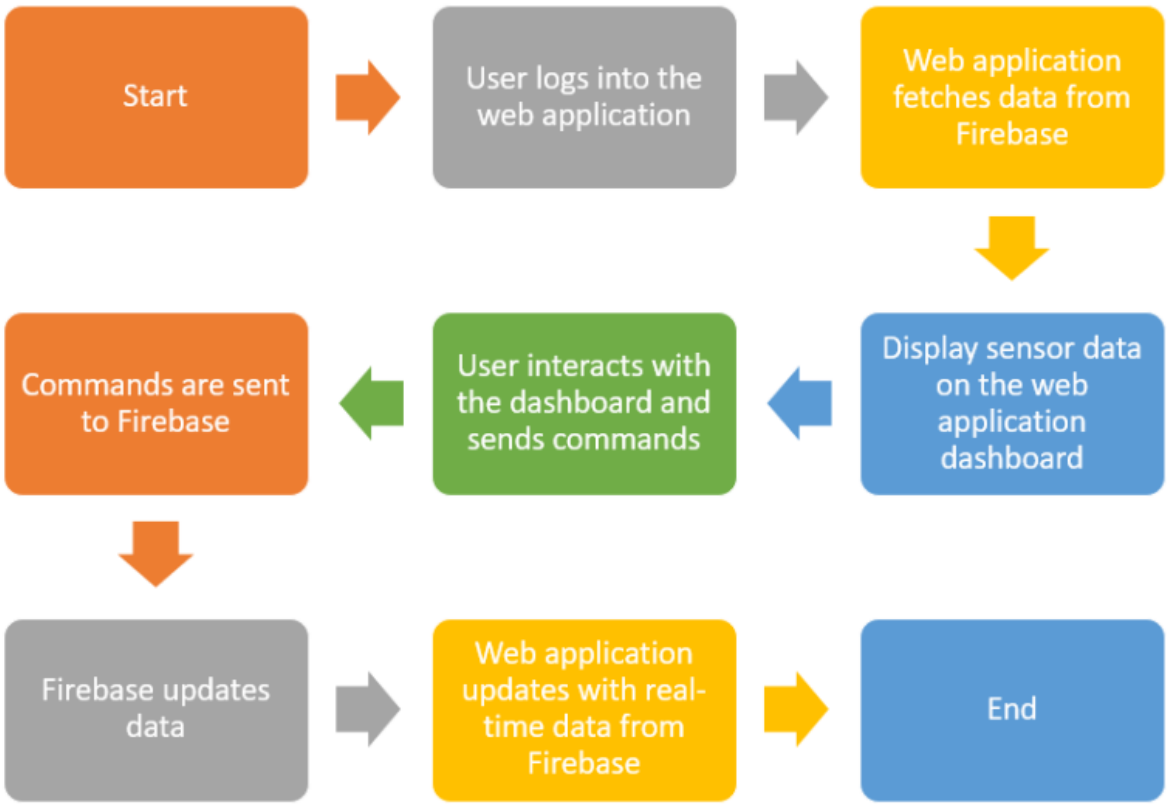


Figure 60) Depicts the Web Application Process Overview

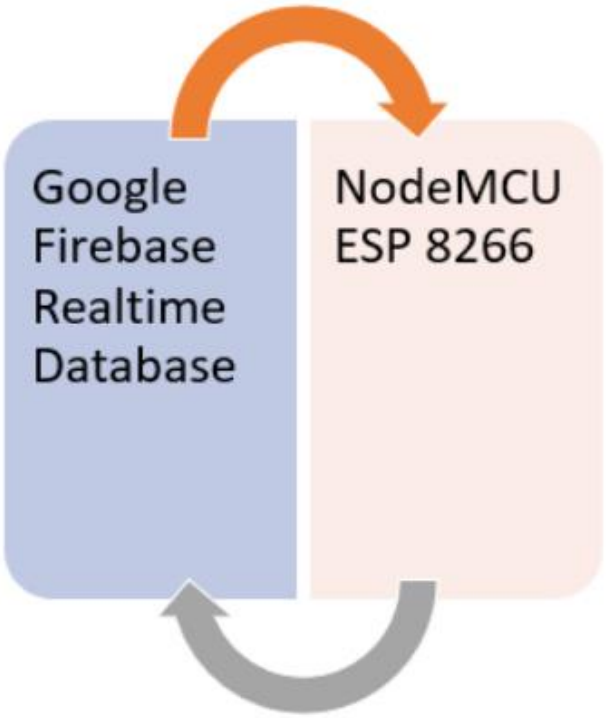


Figure 61) Illustrates the bidirectional data flow and interaction between the NodeMCU microcontroller and Google Firebase, a cloud-based NoSQL database. The process can be broken down into two main parts: data upload and command reception.

Data Upload: The NodeMCU microcontroller collects data from various connected sensors in real-time. This information may include temperature readings, light status, smoke detection, motion detection, and more. This data is then processed and published to specific topics using the MQTT protocol, to which the Google Firebase database is subscribed. Upon receiving the data, Firebase updates the corresponding fields in the Firestore database. This real-time synchronization feature of Firebase ensures that the data reflected on the database is always up-to-date, allowing for immediate user interaction and control via the website or mobile application.

Command Reception: The NodeMCU is also subscribed to certain topics on the MQTT broker. These topics correspond to control commands sent by the user via the mobile application or website. When a command is issued, it is pushed to Firestore, then to the MQTT broker. The broker then publishes these commands to the subscribed NodeMCU. Upon receiving a command, the NodeMCU executes it, such as turning on/off a light or activating a servo motor, and sends back a confirmation message to Firebase, indicating the successful execution of the command.

The described system thus enables real-time monitoring and control of IoT devices, underlining the interplay between the NodeMCU and Firebase in facilitating efficient and responsive smart home automation.

B. Cloud – Realtime Database

The Realtime Database is a NoSQL cloud database provided by Firebase, a Google platform for developing mobile and web applications. The capacity of this database to synchronize data in real-time across all connected clients is its defining characteristic. Because of this real-time functionality, Firebase is ideal for applications that require instant and dependable data exchange, such as chat applications, collaborative tools, and IoT systems.

Data is saved as JSON in the Firebase Realtime Database and is synced in real-time to every connected client. When you create cross-platform apps using the iOS, Android, and JavaScript SDKs, all of your clients share a single Realtime Database instance and are immediately updated with the most recent data.

The way it works is that whenever a piece of data in the database changes, an event is triggered, and the updated data is automatically transmitted to all interested clients. This guarantees that all users see the most up-to-date information without having to reload or poll the database. Furthermore, the Firebase Realtime Database can continue to operate even when network connectivity is poor or even unavailable. When the network connection is restored, it automatically synchronizes data, allowing for a seamless user experience.

For security, Firebase provides robust data validation and user authentication features. Rules can be defined for data validation and read/write access control, and these rules are stored in a JSON format on the Firebase servers. This ensures the security and integrity of data while still providing the flexibility and efficiency of a real-time database.

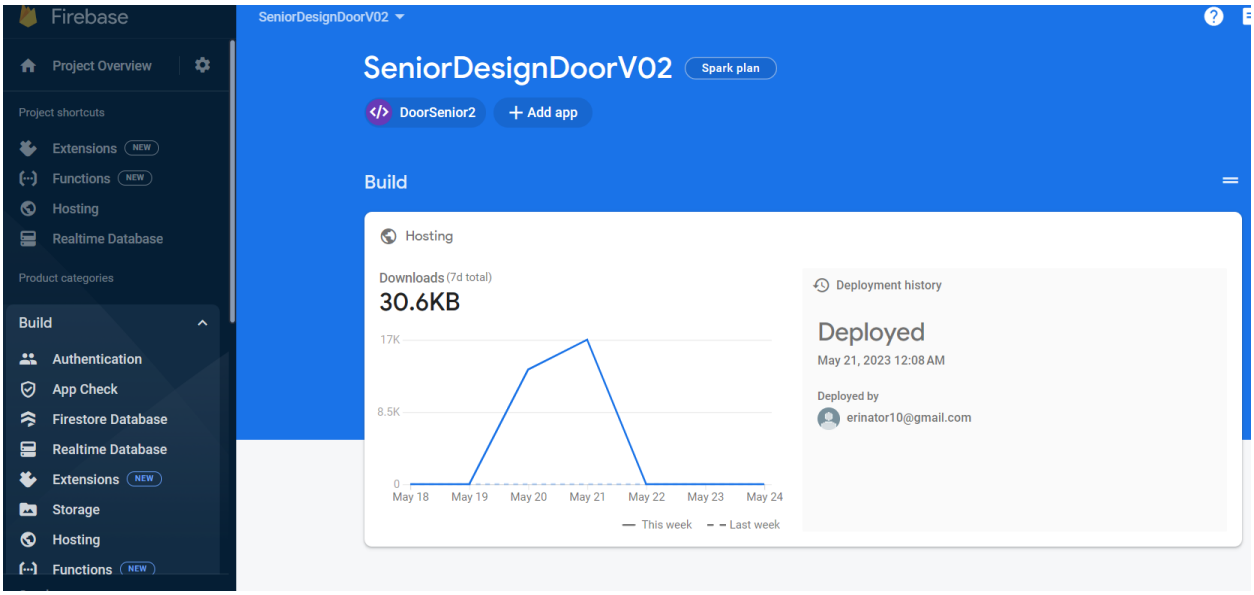


Figure 62) Depicts what the Firebase website looks like.

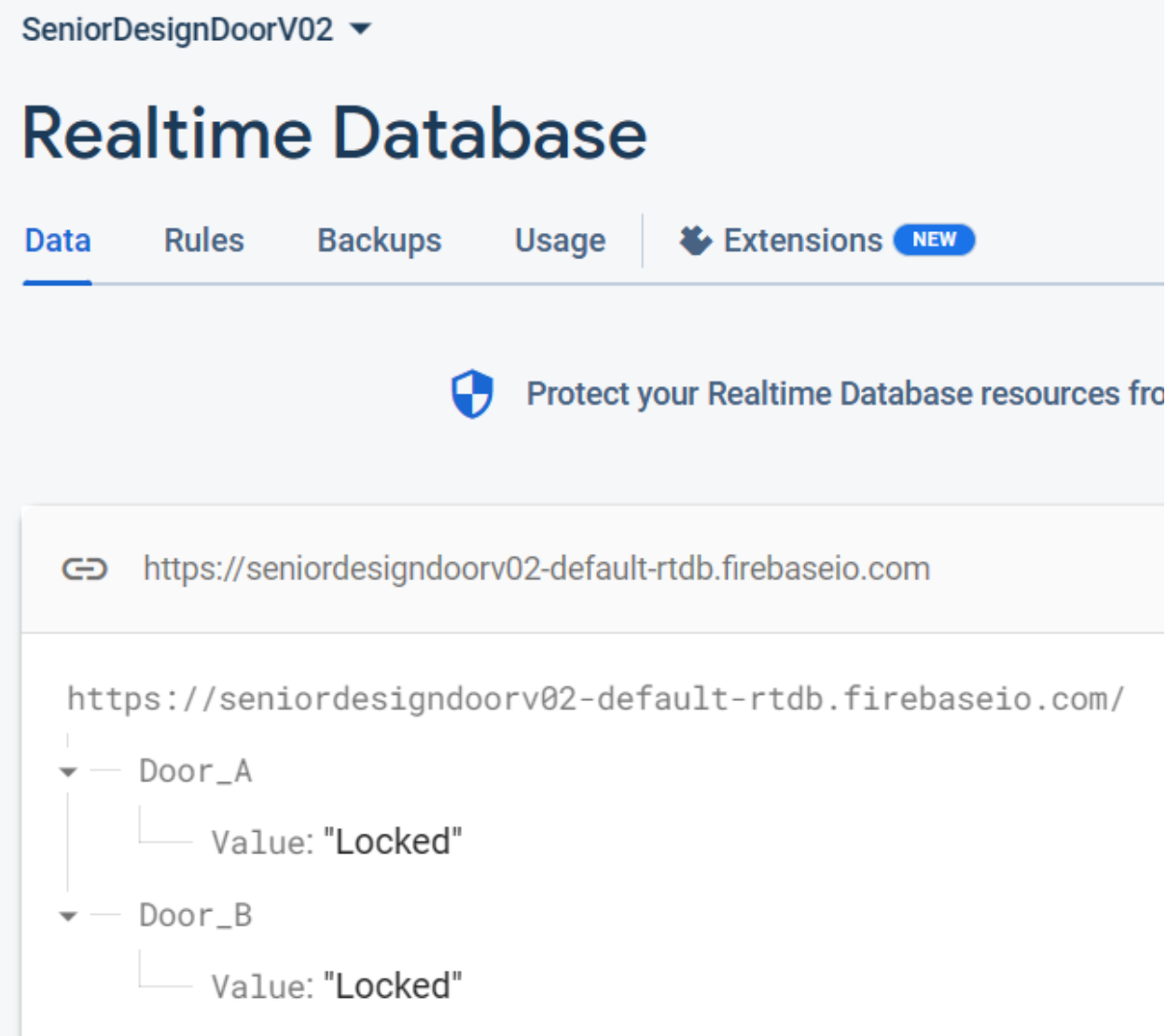


Figure 62) Depicts what the Backend Realtime Database for the Doors looks like, stored within our cloud storage.

C. Code for Web Application

1. *Code Hierarchy*

The structure of the website is relatively straightforward. Our index.blade.php file is the primary view where users interact with the devices. This view uses Vue.js for dynamic content updates and Bootstrap for the responsive design. We use Laravel's MVC (Model-View-Controller) architecture, which separates the application logic from the user interface, making it easier to maintain.

Below is a simple flowchart demonstrating the structure and connections within the application:

Figure 63) Depicts the code Hierarchy of the front and backend of the web development.

In this structure, the key elements are:

- **index.blade.php**: This is the primary interface that users will interact with. It displays the real-time data from sensors and provides controls for devices.
- **ESPLogicController**: This is the Laravel controller handling the business logic of communicating with Firebase and serves as a backend for our web application.
- **Firebase SDK**: This is the real-time database we use for exchanging data between the Laravel backend and the IoT devices.
- **Assets**: These are the design components of the application, including CSS for styling and JS for dynamic interactions.

2. *Source Code*

Let's dive into the code for our web application, focusing on the important aspects:

‘index.blade.php’:

This is the main interface where users interact with the application. It displays the real-time data from sensors and provides controls for the devices.

```
# 'index.blade.php':
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Metadata, Styles, and Scripts -->
</head>
<body>
    <div id="app">
        <button v-on:click="firebaseSet('ON')">Turn ON</button>
        <button v-on:click="firebaseSet('OFF')">Turn OFF</button>

        <!-- Real-time sensor data -->
        <div>
            Temperature: {{ temperature }} °C
        </div>
    </div>

    <!-- Import Vue.js and other JavaScript -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
    <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>
```

*Figure 64) Depicts the **index.blade.php** file.*

In the **index.blade.php** file, the **firebaseSet** method sends commands to the backend, which are then sent to Firebase to control the devices.

The **app.js** script has the Vue.js code for updating the frontend in real-time:

```
# 'app.js':
new Vue({
    el: '#app',
    data: {
        temperature: 0
    },
    methods: {
        firebaseSet: function(status) {
            axios.post('/set-status', {
                status: status
            })
            .then(response => {
                console.log(response);
            });
        }
    },
    created: function() {
        this.firebaseListen();
    },
    methods: {
        firebaseListen: function() {
            axios.get('/get-status')
            .then(response => {
                this.temperature = response.data.temperature;
            });
        }
    }
});
```

*Figure 65) Depicts the **app.js** file.*

In this JavaScript code, the **firebaseSet** method is used to send commands to the backend. The **firebaseListen** method retrieves data from the backend and updates the Vue.js data accordingly.

The backend code in Laravel is responsible for receiving these HTTP requests and communicating with Firebase. The relevant controller (let's call it **ESPLogicController.php**) might look like:

```php
# 'ESPLogicController.php':
<?php

namespace App\Http\Controllers;

use Kreait\Firebase\Factory;
use Illuminate\Http\Request;

class ESPLogicController extends Controller
{
    private $database;

    public function __construct()
    {
        $factory = (new Factory)->withServiceAccount('/path/to/firebase/service/account.json');
        $this->database = $factory->createDatabase();
    }

    public function setStatus(Request $request)
    {
        $newStatus = $request->input('status');
        $this->database->getReference('device-status')->set($newStatus);
    }

    public function getStatus()
    {
        $temperature = $this->database->getReference('device-data/temperature')->getSnapshot()->getValue();
        return response()->json(['temperature' => $temperature]);
    }
}
```

*Figure 66) Depicts the **ESPLogicController.php** file.*

In this PHP controller, we use the Firebase SDK to set the device status in the **setStatus** method and get the device temperature in the **getStatus** method.

II. Real-Time Data Display and Command

To ensure real-time display of data, we use Vue.js and Firebase. Vue.js offers a reactive UI which automatically updates whenever the underlying data changes. Firebase is a real-time database service that pushes updates to connected clients as soon as data changes in the database.

For the command timing, we timestamp each command when it is created, sent, and acknowledged by the IoT device, and received back by the server. You can include timestamps in the data packet you send to Firebase and record the time at each key step. This way, you can calculate the time taken between these steps by comparing the timestamps.

Here's an example of how to modify the **setStatus** method to add a timestamp:

```
# 'setStatus':
public function setStatus(Request $request)
{
    $newStatus = [
        'command' => $request->input('status'),
        'timestamp' => now(),
    ];
    $this->database->getReference('device-status')->set($newStatus);
}
```

*Figure 67) Depicts the **setStatus** code.*

```
@extends('template.main')
@push('css')
    <style>
    .button1 {width: 75px;}
    table {
    width: 100%;
    font-size: 120%;

    }
    th {
    padding: 5px;
    text-align: center;
    }
    td {
    padding: 5px;
    text-align: center;
    }

    .table-dashboard tbody tr:nth-child(even) {
    background-color: #cef0b1;
        }
    .table-dashboard  thead{
    background-color: #96bf73;
    }
    </style>
@endpush
@section('title')
    Dashboard
@endsection

@section('breadcrumb')
    @parent
    <!--li class="active">Servo Control</li-->
@endsection
@section('content')
<section class="content">
  <div class="box" style="width:75%;">
      <div class="box-header with-border">
        <h3>Sensor</h3>
      </div>
      <div class="box-body table-responsive">
        <table class="sensortable table-stiped table-bordered table-dashboard datatable">
        <thead>
          <th width="25%">Sensor</th>
          <th width="20%">Value</th>
        </thead>
        <tbody>
```

```
            <tr>
              <td>Temperature Sensor</td>
              <td id="tempsensor_value"></td>
            </tr>
            <tr>
              <td>Gas Sensor</td>
              <td id="gassensor_value"></td>
            </tr>
            <tr>
              <td>PIR Sensor</td>
              <td id="pirsensor_value"></td>
            </tr>
          </tbody>
          </table>
        </div>
    </div>
    <div class="box" style="width:75%;">
        <div class="box-header with-border">
          <h3>Actuator</h3>
        </div>
        <div class="box-body table-responsive">
          <table class="actuatortable table-stiped table-bordered table-dashboard datatable">
          <thead>
            <th width="25%">Actuator</th>
            <th width="20%">Switch</th>
            <th width="25%">Command/Status</th>
          </thead>
          <tbody>
            <tr>
              <td>Servo Lock</td>
              <td>
                <div class="btn-group">
                <button onclick="set_firebase('{{route('esplogic.firebaseset', '3')}}')" class="btn btn-success  btn-flat btn-xs"><i class="fa fa-pla
                </div>
                <div class="btn-group">
                <button onclick="set_firebase('{{route('esplogic.firebaseset', '4')}}')" class="btn btn-danger  btn-flat btn-xs"><i class="fa fa-stop
                </div>
              </td>
              <td>
                <span id="servo_cmd" class="label label-success">ON</span>
                <span id="servo_stat" class="label label-danger">OFF</span>
              </td>
            </tr>
            <tr>
              <td>LED Control</td>
              <td>
                <div class="btn-group">
                <button onclick="set_firebase('{{route('esplogic.firebaseset', '1')}}')" class="btn btn-success  btn-flat btn-xs"><i class="fa fa-play
                </div>
                <div class="btn-group">
                <button onclick="set_firebase('{{route('esplogic.firebaseset', '2')}}')" class="btn btn-danger  btn-flat btn-xs"><i class="fa fa-stop'
                </div>
              </td>
              <td>
                <span id="led_cmd" class="label label-success">ON</span>
                <span id="led_stat"class="label label-danger">OFF</span>
              </td>
            </tr>
          </tbody>
          </table>
        </div>
    </div>
</section>

@endsection
@push('scripts')
<script>

$(document).ready(function() {
  read_firebase();
})
setInterval(function() {
  read_firebase()
},
3000);
function set_firebase(url) {
  $.get(url)
  .done((response)=>{
    alert('Command sent!');
  })
  .fail((errors)=>{
    alert('Error, can not delete data!');
  })
}

function read_firebase(){
  $.get('{{route('esplogic.firebaseread')}}')
  .done((response)=>{
    var jsondata = JSON.parse(response);
    //console.log(jsondata.esp32write1.led);
    //console.log(jsondata.esp32write1.servo);
    $("#tempsensor_value").text(jsondata.esp32write1.temp);
    $("#gassensor_value").text(jsondata.esp32write1.gas);
    $("#pirsensor_value").text(jsondata.esp32write1.pir);
```

```
    //======================command====================
    if(jsondata.esp32read1.led=="ON"){
      $('#led_cmd').text("ON");
      if ($('#led_cmd').hasClass('label-danger'))$('#led_cmd').removeClass('label-danger');
      if (!$('#led_cmd').hasClass('label-success'))$('#led_cmd').addClass('label-success');
    }
    else if(jsondata.esp32read1.led=="OFF"){
      $('#led_cmd').text("OFF");
      if ($('#led_cmd').hasClass('label-success'))$('#led_cmd').removeClass('label-success');
      if (!$('#led_cmd').hasClass('label-danger'))$('#led_cmd').addClass('label-danger');
    }
    if(jsondata.esp32read1.servo=="ON"){
      $('#servo_cmd').text("ON");
      if ($('#servo_cmd').hasClass('label-danger'))$('#servo_cmd').removeClass('label-danger');
      if (!$('#servo_cmd').hasClass('label-success'))$('#servo_cmd').addClass('label-success');
    }
    else if(jsondata.esp32read1.servo=="OFF"){
      $('#servo_cmd').text("OFF");
      if ($('#servo_cmd').hasClass('label-success'))$('#servo_cmd').removeClass('label-success');
      if (!$('#servo_cmd').hasClass('label-danger'))$('#servo_cmd').addClass('label-danger');
    }
    //======================status====================
    if(jsondata.esp32write1.led=="ON"){
      $('#led_stat').text("ON");
      if ($('#led_stat').hasClass('label-danger'))$('#led_stat').removeClass('label-danger');
      if (!$('#led_stat').hasClass('label-success'))$('#led_stat').addClass('label-success');
    }
    else if(jsondata.esp32write1.led=="OFF"){
      $('#led_stat').text("OFF");
      if ($('#led_stat').hasClass('label-success'))$('#led_stat').removeClass('label-success');
      if (!$('#led_stat').hasClass('label-danger'))$('#led_stat').addClass('label-danger');
    }
    if(jsondata.esp32write1.servo=="ON"){
      $('#servo_stat').text("ON");
      if ($('#servo_stat').hasClass('label-danger'))$('#servo_stat').removeClass('label-danger');
      if (!$('#servo_stat').hasClass('label-success'))$('#servo_stat').addClass('label-success');
    }
    else if(jsondata.esp32write1.servo=="OFF"){
      $('#servo_stat').text("OFF");
      if ($('#servo_stat').hasClass('label-success'))$('#servo_stat').removeClass('label-success');
      if (!$('#servo_stat').hasClass('label-danger'))$('#servo_stat').addClass('label-danger');
    }
    })
  .fail((errors)=>{
    return;
  })
}
      $('#servo_stat').text("OFF");
      if ($('#servo_stat').hasClass('label-success'))$('#servo_stat').removeClass('label-success');
      if (!$('#servo_stat').hasClass('label-danger'))$('#servo_stat').addClass('label-danger');
    }
    })
  .fail((errors)=>{
    return;
  })
}



</script>
@endpush
```

Figure 68) Depicts us combining various created methods to design the backend of the website architecture.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Kreait\Firebase;
use Kreait\Firebase\Factory;

class esplogicController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
```

```
48      public function show($id)
49      {
50          //
51      }
52
53      /**
54       * Show the form for editing the specified resource.
55       *
56       * @param  int  $id
57       * @return \Illuminate\Http\Response
58       */
59      public function edit($id)
60      {
61          //
62      }
63
64      /**
65       * Update the specified resource in storage.
66       *
67       * @param  \Illuminate\Http\Request  $request
68       * @param  int  $id
69       * @return \Illuminate\Http\Response
70       */
71      public function update(Request $request, $id)
72      {
73          //
74      }
75
76      /**
77       * Remove the specified resource from storage.
78       *
79       * @param  int  $id
80       * @return \Illuminate\Http\Response
81       */
82      public function destroy($id)
83      {
84          //
85      }
86
87      public function logicinit()
88      {
89
90      }
91
92      public function firebaseread(Request $request)
93      {
94          $firebase = (new Factory)
```

```php
    public function firebaseread(Request $request)
    {
        $firebase = (new Factory)
            ->withServiceAccount(__DIR__.'/webautomation001-firebase-adminsdk-5vtzr-e817d9a1e6.json')
            ->withDatabaseUri('https://webautomation001-default-rtdb.firebaseio.com');
        $database = $firebase->createDatabase();
        $getdata = $database->getReference();


        return(json_encode($getdata->getvalue()));
    }

    public function firebaseset($state)
    {
        $firebase = (new Factory)
            ->withServiceAccount(__DIR__.'/webautomation001-firebase-adminsdk-5vtzr-e817d9a1e6.json')
            ->withDatabaseUri('https://webautomation001-default-rtdb.firebaseio.com');
        $database = $firebase->createDatabase();
        $getdata = $database->getReference();
        if($state=="1"){
            $database->getReference('/esp32read1/led')->set('ON');;
        }
        else if($state=="2"){
            $database->getReference('/esp32read1/led')->set('OFF');;
        }
        else if($state=="3"){
            $database->getReference('/esp32read1/servo')->set('ON');;
        }
        else if($state=="4"){
            $database->getReference('/esp32read1/servo')->set('OFF');;
        }
    }

    /*$postData = [
        'sensor1'=>$request->sensor1,
        'sensor2'=>$request->sensor2,
    ];
    //$postRef = $this->database->getReference($ref_tablename)->push($postData);*/

}
```

Figure 64) Depicts code used to read from and write updated variable values from firebase to the website and vise versa.

III. Unifying Web and Mobile Applications

For unifying web and mobile applications, you would have to ensure both are connected to the same Firebase backend.

1. **Web Application:** As explained earlier, the Laravel backend communicates with Firebase, sending commands to and receiving updates from IoT devices.
2. **Mobile Application:** The mobile app, presumably built with a framework like Flutter or React Native, can use Firebase's SDK to directly interface with Firebase. This would allow the mobile app to send commands and receive updates just like the web app does.

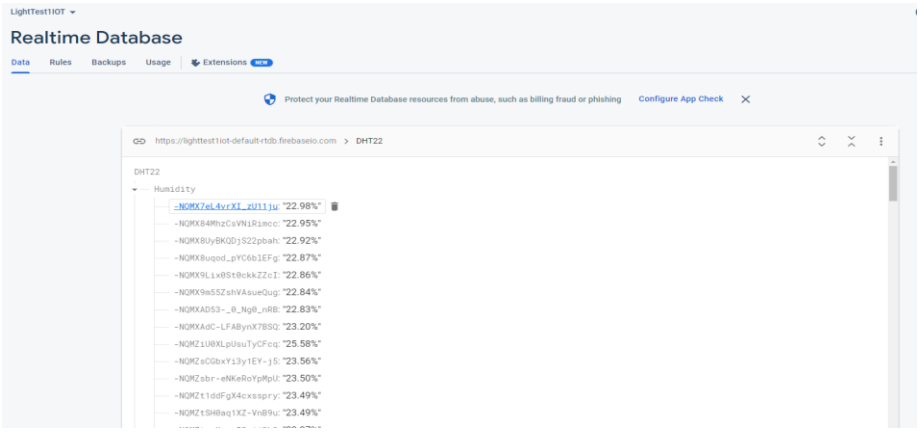
In this setup, users could switch seamlessly between the web and mobile applications while maintaining a consistent view of the IoT system state, as both are simply frontends for the same Firebase backend.

V. TESTING

A. *Testing the Sensors*

The temperature and humidity sensor functions similarly to light switches, but with a notable difference: it doesn't require inputs from the cloud. Instead, this sensor measures both temperature and humidity levels in a designated area and transmits that information to the cloud. The resulting data can be communicated as either numerical or string-based values and can be retrieved at any desired interval. Furthermore, this data can be securely stored in the cloud for future analysis.

- When testing the sensors, it is crucial to ensure their proper functionality. In the case of the esp8266, the first step is to correctly code the output pin locations on the microcontroller. The esp8266 is a popular Wi-Fi module that can be used for IoT applications, including sensor monitoring and data transmission. By mapping the appropriate output pins on the esp8266 to the sensors, we establish the communication between the microcontroller and the sensors.
- Once the pin locations are configured, the next step is to write the sensor data to the Firebase platform. Firebase is a powerful cloud-based service that provides real-time data storage and synchronization. By integrating the esp8266 with Firebase, the sensor readings can be transmitted to the cloud and stored in a database.
- For example, when testing light switches, various inputs can be sent to the database, simulating different lighting conditions. By monitoring the changes in the light switch data recorded in the database, we can determine if the sensors are accurately detecting changes in ambient light and reflecting those changes in the recorded data. This test ensures that the light sensors are functioning properly and providing reliable data.
- Regarding gas sensors, it is essential to define a range within which the sensor should trigger an alert and send a message to the cloud. By setting a low threshold and introducing smoke near the gas sensor (e.g., by lighting a candle), we can observe whether the sensor detects the presence of smoke and initiates the appropriate response. This test ensures that the gas sensor is sensitive enough to detect potential hazards and effectively communicates that information to the cloud for further processing and action.



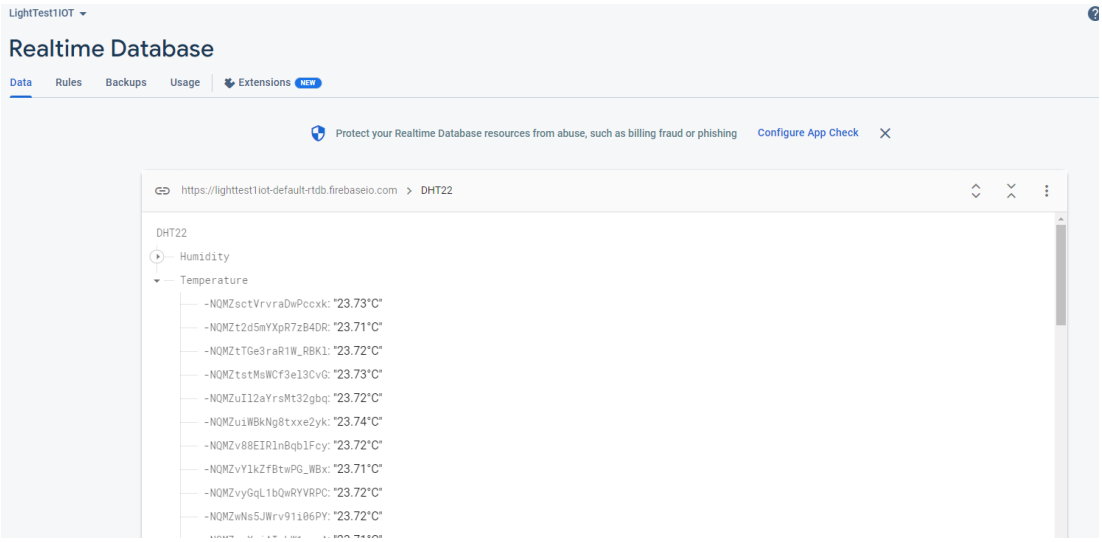


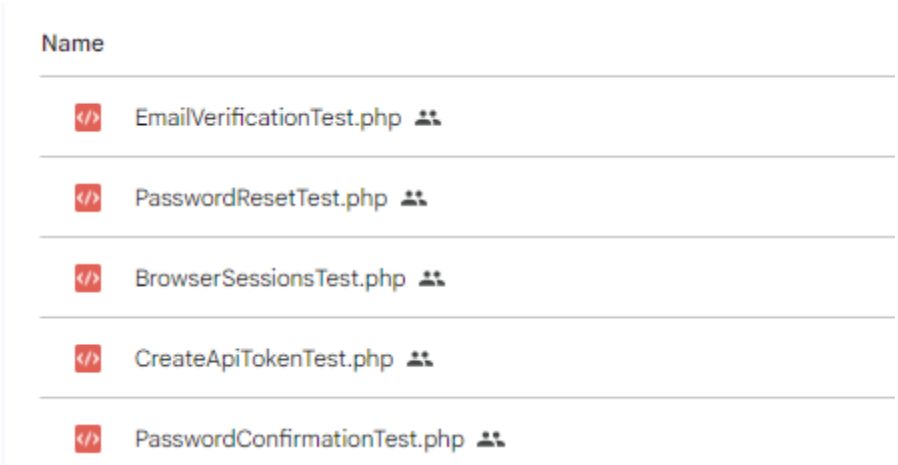
Figure 65) Depicts the Realtime data being captured for the Temperature and Gas Sensors during testing.

B. *Website Communications Test*

The extensive testing of the smart home automation system includes not only guaranteeing smooth data transfer between the cloud and the website, but also the operation of the website, the mobile application, and the SMS notification feature.

1. **Website Functionality Test**:

   This test is required to guarantee that all aspects of the website perform properly. All interactive buttons, forms, navigation links, and responsive designs must be tested. For example, if a button is intended to turn on and off a light, clicking the button should modify the state of the light in the real-time database, and the modified status should be represented on the website.



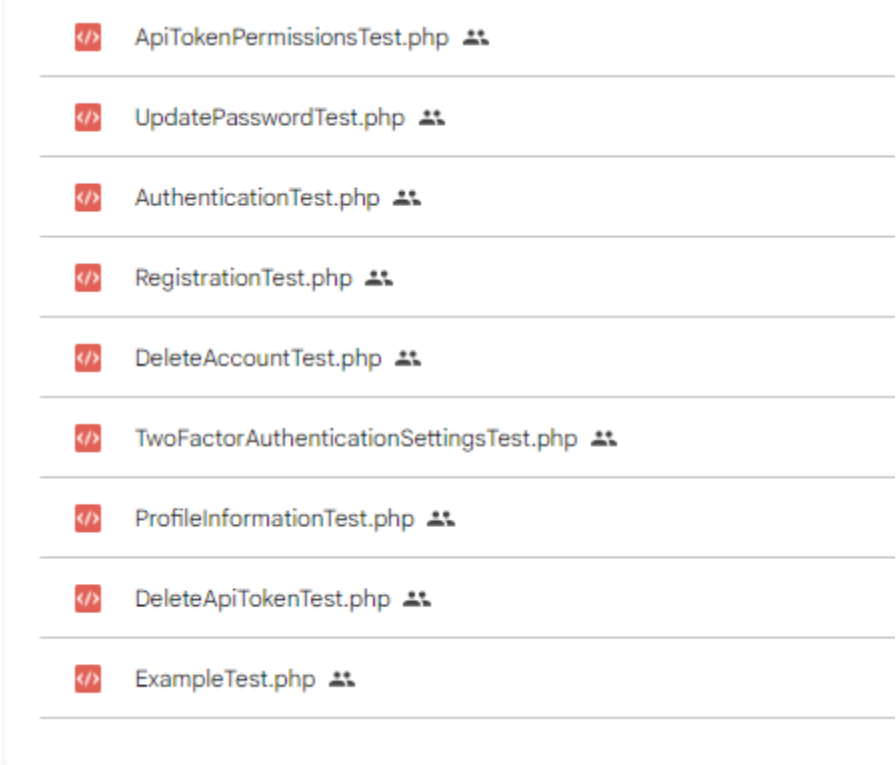


Figure 66) Depicts various tests that were created to test the functionality of the website.

```php
<?php

namespace Tests\Feature;

use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Jetstream\Http\Livewire\TwoFactorAuthenticationForm;
use Livewire\Livewire;
use Tests\TestCase;

class TwoFactorAuthenticationSettingsTest extends TestCase
{
    use RefreshDatabase;

    public function test_two_factor_authentication_can_be_enabled()
    {
        $this->actingAs($user = User::factory()->create());

        $this->withSession(['auth.password_confirmed_at' => time()]);

        Livewire::test(TwoFactorAuthenticationForm::class)
                ->call('enableTwoFactorAuthentication');

        $user = $user->fresh();

        $this->assertNotNull($user->two_factor_secret);
        $this->assertCount(8, $user->recoveryCodes());
    }
```

```php
    public function test_recovery_codes_can_be_regenerated()
    {
        $this->actingAs($user = User::factory()->create());

        $this->withSession(['auth.password_confirmed_at' => time()]);

        $component = Livewire::test(TwoFactorAuthenticationForm::class)
                ->call('enableTwoFactorAuthentication')
                ->call('regenerateRecoveryCodes');

        $user = $user->fresh();

        $component->call('regenerateRecoveryCodes');

        $this->assertCount(8, $user->recoveryCodes());
        $this->assertCount(8, array_diff($user->recoveryCodes(), $user->fresh()->recoveryCodes()));
    }

    public function test_two_factor_authentication_can_be_disabled()
    {
        $this->actingAs($user = User::factory()->create());

        $this->withSession(['auth.password_confirmed_at' => time()]);

        $component = Livewire::test(TwoFactorAuthenticationForm::class)
                ->call('enableTwoFactorAuthentication');

        $this->assertNotNull($user->fresh()->two_factor_secret);

        $component->call('disableTwoFactorAuthentication');

        $this->assertNull($user->fresh()->two_factor_secret);
    }
}
```

Figure 67) Depicts an example of one such test. The Two Factor Authentication Test.

2. **Mobile App Interface Test**:

   The mobile application is a crucial system component that must be extensively evaluated. This includes evaluating the capabilities of user authentication, data retrieval and display, and command transmission. The validity of real-time data synchronization is also checked, guaranteeing that data changed on the app is reflected on the cloud database and vice versa. To provide a seamless user experience, the app's responsiveness is tested across many devices and operating systems.

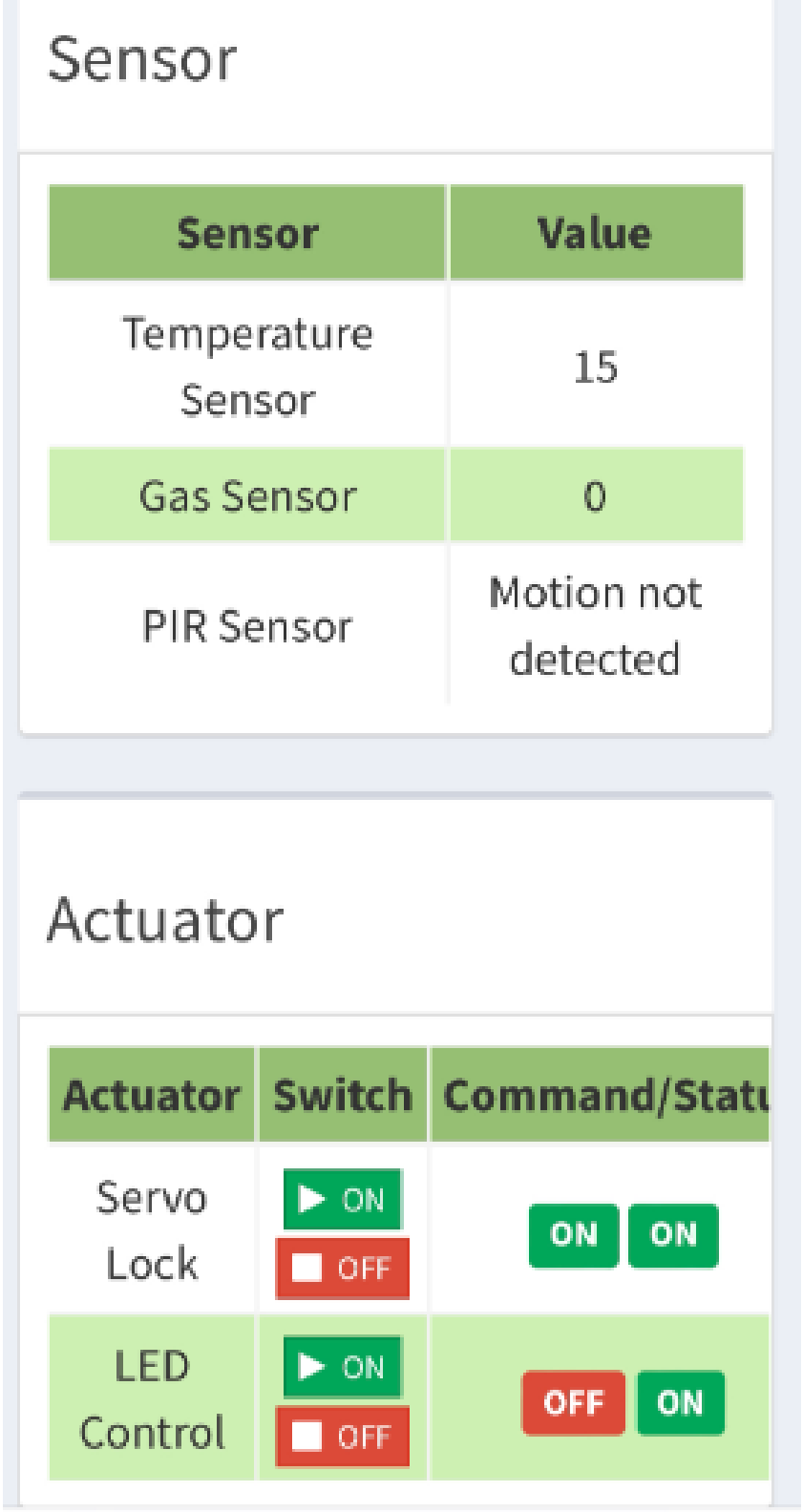


Figure 68) Depicts us testing the app's ability to gather data from Firebase.

3. **SMS Notification Feature Test**:

   This test validates that the SIM800L GPRS Module sends SMS alerts in response to particular occurrences. For example, if the PIR sensor detects motion, an SMS alert should be delivered to the associated cellphone number. Similarly, circumstances such as excessive temperature or smoke detection should cause SMS messages to be sent. The SMS notification functionality should also be tested for the ability to be toggled on and off based on user choice as stored in the database.

Figure 69) Depicts the SMS Test being successfully performed.

C. *Physical Implementation Test*

The Physical Implementation Test is an important milestone in the development of our intelligent home automation system. It enables us to determine whether our hardware components - in this example, the LED and the Hall Effect Magnetic Sensor - are operating as predicted outside of the controlled settings of a development environment.

a) **LED Functionality Test:**
The LED functionality test comprises sending commands from the app or internet interface to switch on and off the LED. The state of the LED should change in response to these commands. The test ensures that the LED is properly connected, that the proper voltages are given, and that the control orders from the Firebase Firestore database are efficiently translated into physical actions.

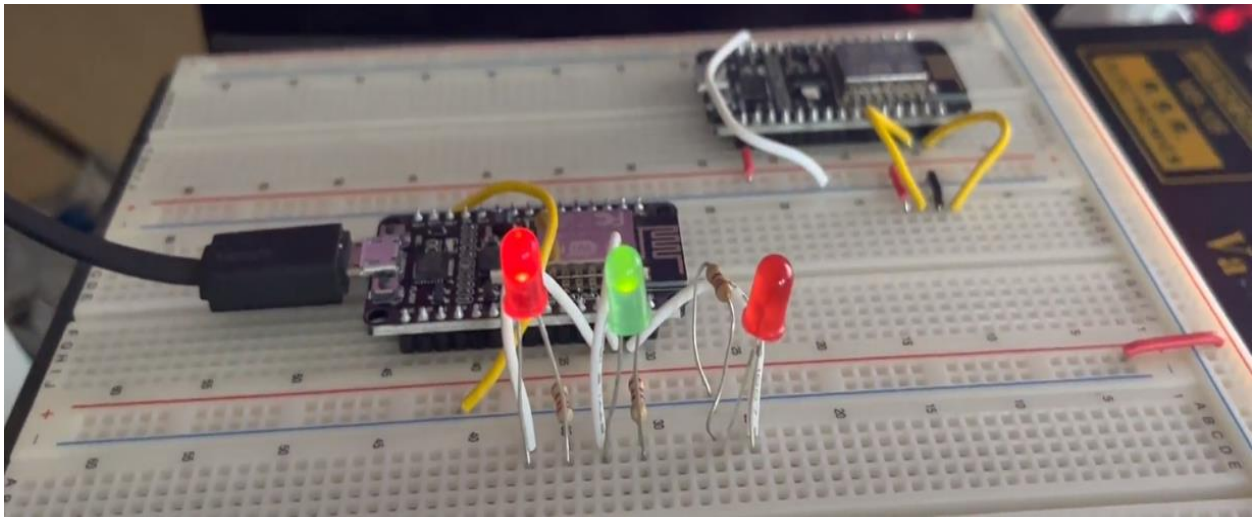


Figure 70) Depicts our LED lights successfully turning on.

b) **Deadlock and Hall Effect Magnetic Sensor Test:**
In this test, a magnet is moved close to the Hall Effect Magnetic Sensor to imitate the opening and shutting of a door or window. When the magnet is nearby, the sensor should detect the presence of a magnetic field and stop detecting when the magnet is moved away. The sensor's state should be updated in real-time in the Firebase Firestore database and presented in the app or on the website. The test is intended to check that the sensor is properly connected and calibrated, as well as that it can detect and report changes in magnetic field strength.

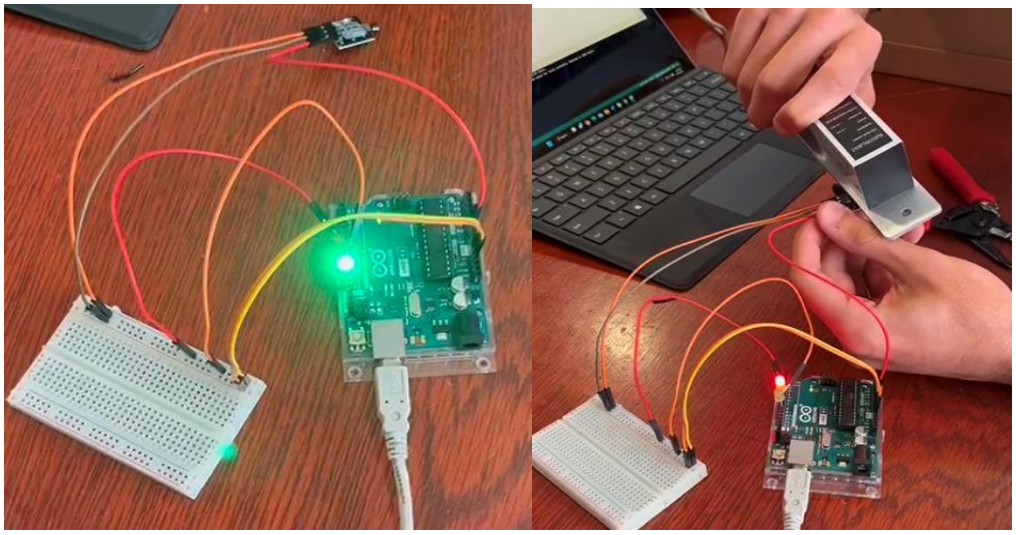


Figure 71) Depicts our Hall Effect Magnetic sensor successfully sensing the presence of the deadbolt and transmitting that information with a change in LED lighting.

D. *Web Application Demonstration & Testing.*

Web application demonstration and testing is an important step in evaluating our smart home automation system's real-time performance and responsiveness. This entails a thorough assessment of the system's capacity to appropriately display changes in device statuses (such as lights and door locks) on the user interface in real-time and vice versa.

1. Light Control Test:

For light control, we measure the response time - the duration from when a user interacts with the light control switch on the web application until the status change is reflected in Firebase and the light state changes physically. The test is performed by toggling the light from OFF to ON on the website and recording the time it takes for the LED to light up.

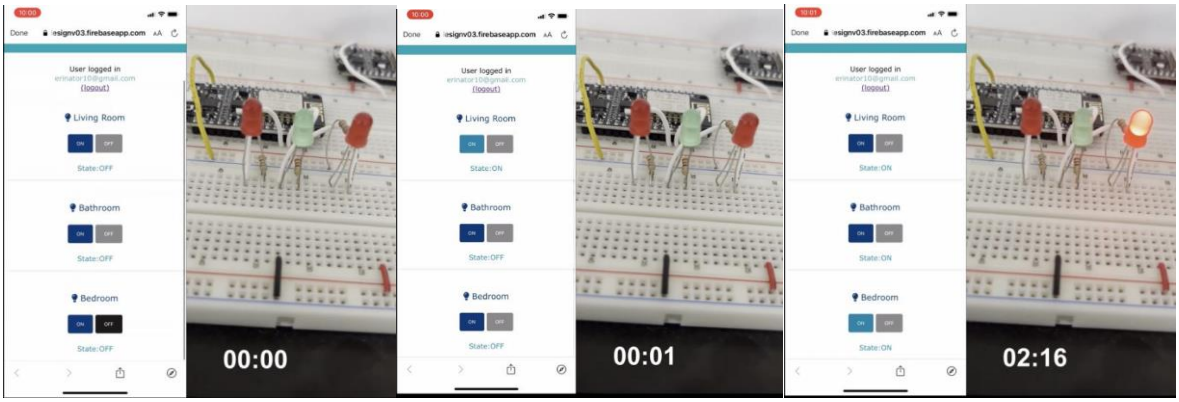


Figure 72) Depicts us successfully timing the change in lighting. It took 2.16 seconds to for the light to change after changing the status on the website.

2. Door Lock Control Test:

The door lock control test is slightly different. Here, we evaluate the time it takes for the web application to update the lock's status after a physical change. This simulates a real-world scenario where the door may be locked or unlocked manually and the system needs to reflect that change. In this test, the door is locked physically, and we time how long it takes for the website to update its status.

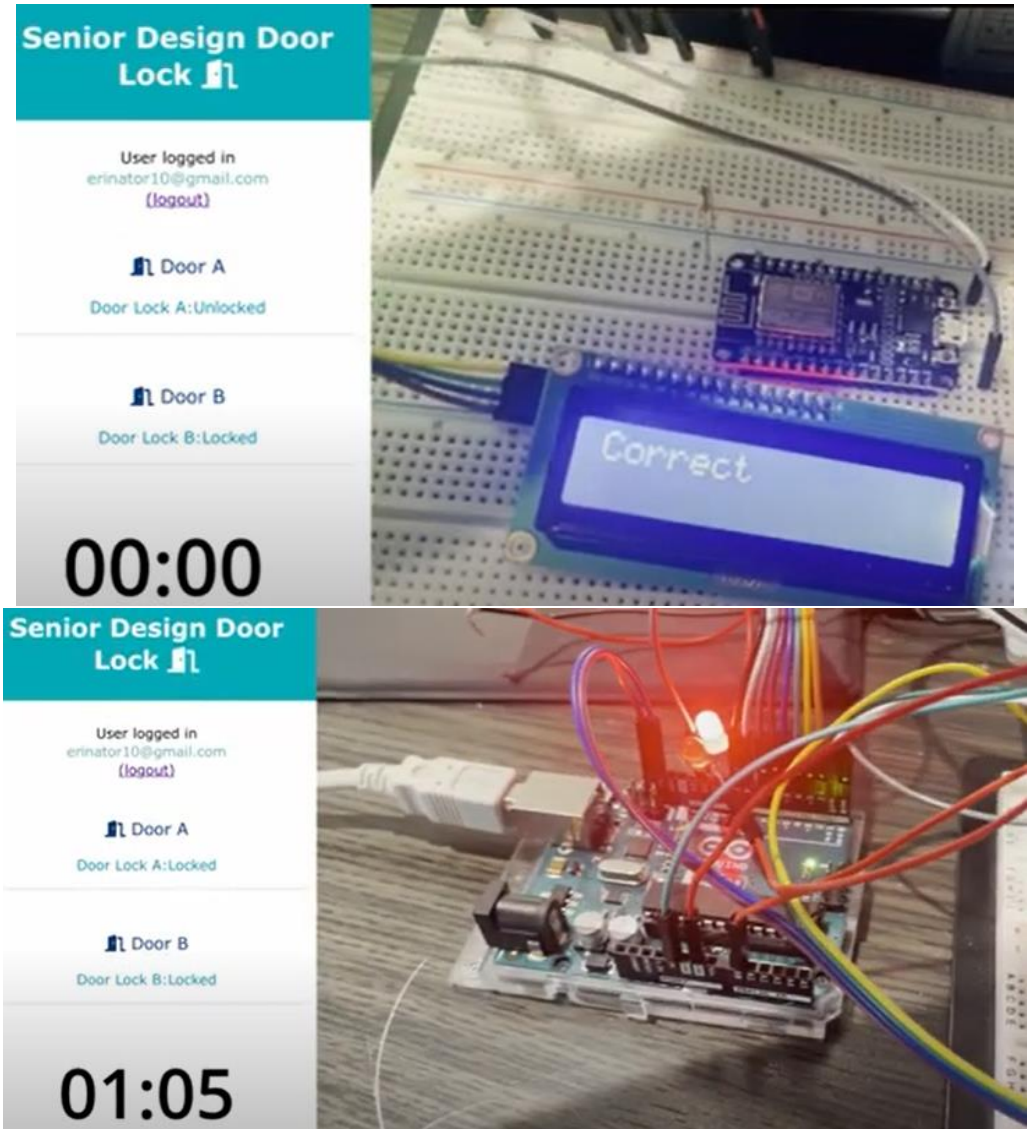


Figure 73) Depicts us successfully timing the lock change status. It took 1.05 seconds for the the website to update its status.

These tests demonstrate the system's capacity to deliver a unified user experience by guaranteeing near-instant feedback from physical devices and the web interface. Any delays or inconsistencies discovered during these testing can be examined and corrected to improve system performance.

Website's Front End Final Design

Figure 74) Depicts the login page of our website.

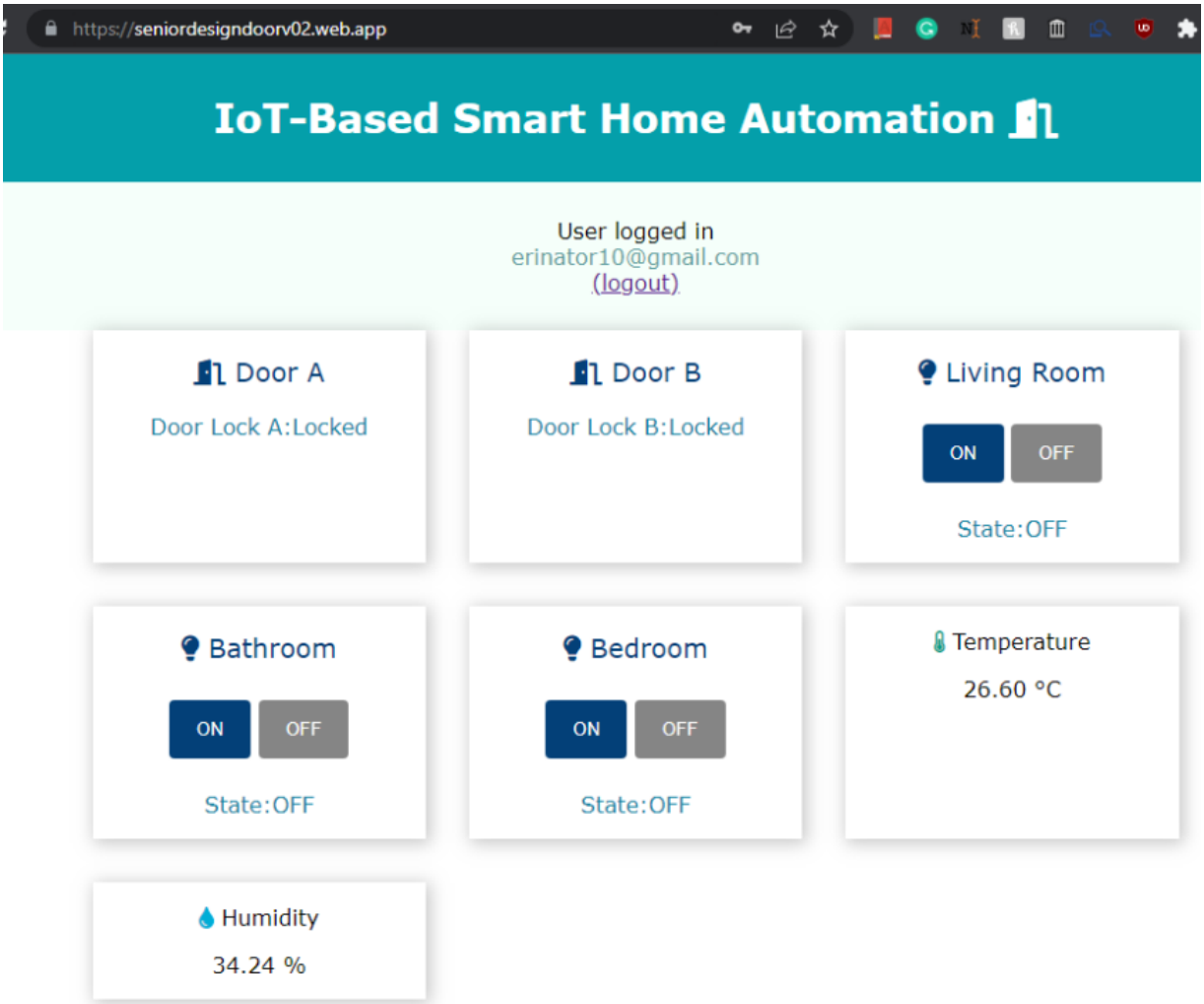


Figure 75) Depicts what the user would see after logging in..

VI. BILL OF MATERIALS

Table 3) Depicts our Bill of Materials for the Components used within this project.

Part Name	Quantity	Price/Unit	Total Price
5V 1 Channel Level Trigger Optocoupler Relay Module	2	*$6.99*	$13.98
9V-MN1604 Battery	2	*$2.46*	$4.92
uxcell 9V Battery Snap Connector I-Type to PH2.0 Connector Test Cable 14.5cm Long for Electric Testing Work, 3 Pcs	1	*$6.99*	$6.99
LED - RGB Clear Common Cathode	1	*$2.25*	$2.25
Humidity and Temperature Sensor - DHT20	1	*$6.50*	$6.50
ESP8266 NodeMCU	1	*$16.39*	$16.39
Gas Sensor MQ2	1	*$7.60*	$7.60
Arduino Uno Rev3	1	*$25.99*	$25.99
LIBO Electric Bolt Lock 2 Wires Lines DC 12V Fail Safe Electronic Lock Normal Temperature for Glass Wooden Iron Door Access Control System	1	*$39.99*	$39.99
DS3231 Precision clock Module (RTC)	1	*$2.04*	$2.04
Arduino Uno Rev3 GMS shield sim900f	1	*$39.99*	$39.99
Hall Effect Magnetic Sensor Module, 3144E A3144 Hall Effect Sensor KY-003 DC 5V for Arduino PIC AVR Smart Cars by MUZHI(6 Pcs)	1	*$1.69*	$1.69
DIYmalls 4x4 Keypad 16 Key Matrix Array Membrane Switch 8pin Keyboard for Arduino UNO R3 (Pack of 5)	1	*$7.95*	$7.95
SunFounder IIC I2C TWI 1602 Serial LCD Module Display Compatible with Arduino R3 Mega 2560 16x2	1	*$9.99*	$9.99
HiLetgo Smallest SIM800L GPRS GSM Breakout Module Quad-Band 850/900/1800/1900MHz SIM Card Slot Onboard with Antenna 3.7~4.2V	1	*$8.99*	$8.99
Grand Total	**-**	**-**	**$195.26**

The table above outlines the material cost breakdown necessary for the construction of our IoT-based Smart Home Automation System. Given our focus on building a project that is both scalable and easy to replicate, we opted for the most cost-effective, yet efficient, components such as the ESP8266 NodeMCU and the Arduino Uno Rev3. The total expense for one module of our project is approximately $1436.42. This amount, split among our team members, results in an equitable distribution of costs while ensuring high quality performance of the automation system. If provided, we might be reimbursed for a part of the total cost from our university's engineering department. As our project grows in scale and demand, we anticipate the unit cost will decrease due to bulk purchase benefits.

3. LIMITATIONS

A. Hardware

While many components used in our system are designed for seamless integration with Wi-Fi and cellular networks, there are certain inherent hardware limitations that can't be circumvented through programming.

i. The smart home devices we use have certain manufacturer-imposed limitations. For instance, some of the connected devices may not support real-time data updates, causing a slight delay in feedback received from these devices. Others may be designed to work primarily with a specific ecosystem and might not have full feature support when integrated with our smart home automation system.
ii. Wi-Fi and cellular connections, though widely available and easy to integrate, are susceptible to issues like signal interference and attenuation, network congestion, and physical obstructions, affecting their performance. Additionally, these connections also require a fair amount of power, which might not be ideal for battery-powered devices.

B. Communications

Our system's use of Wi-Fi and Cellular connections for device-cloud communications, while generally reliable, also has its own set of challenges.

i. Wi-Fi signals can be susceptible to interference from other household devices such as microwaves, cordless phones, or other Wi-Fi networks, leading to occasional signal dropouts or reduced performance.
ii. Cellular connections, while generally more stable, can be affected by factors such as network congestion, distance from the cellular tower, or obstructions like buildings and trees.
iii. Despite utilizing industry-standard security protocols, any form of wireless communication carries a risk of security vulnerabilities. If an unauthorized user gains access to the network, they could potentially intercept or tamper with the data being transmitted.

C. Web Application

Despite its robustness and ease of use, our web application isn't without its limitations.

i. Currently, our web application is hosted on a local server, meaning that it can only be accessed within the same local network. This poses a challenge when attempting to access the system remotely, which requires further considerations for security and reliable connectivity.
ii. The development and maintenance of a web application that can be accessed securely from any location, presents its own challenges. This includes ensuring that any data transmitted between the server and the user is securely encrypted, and managing the higher server load that comes with increased accessibility.

iii. The user interface of the web application, while designed to be intuitive and user-friendly, has a learning curve associated with it. Especially for users unfamiliar with smart home systems, getting accustomed to the controls and features may require some time.

D. Mobile Application

The development of a mobile application for our smart home system faced its unique challenges and limitations. Primarily, our group's limited bandwidth and the concurrent demands of other academic classes and project components impacted the full realization of our app.

i. The most significant constraint was time. We had to prioritize various areas of the project due to the intricacy of our undertaking and our academic responsibilities. While the mobile app worked, it did not receive the same degree of attention and development time as the other components. This resulted in a less polished and complete version than was originally intended.
ii. Due to the time restrictions indicated above, the mobile app was created with rudimentary functionality to operate the devices and check their condition. More complex capabilities or a more sophisticated user interface could not be created in the short span provided.
iii. Creating a mobile application that connects successfully with hardware and cloud services was a difficult undertaking. This high learning curve increased the project's development time and complexity.
iv. Another problem is ensuring the app's interoperability across multiple mobile devices and screen sizes. Optimizing the application's performance for multiple devices while maintaining a smooth and consistent user experience was a factor that required more resources than were available.

Despite these limitations, the mobile application successfully serves as a remote control panel for the smart home system. Users can control devices and monitor their status from the app, demonstrating the potential for a more comprehensive and polished version in the future, given more time and resources.

4. CONCLUSION

As we finish our partnership, we are pleased to announce the successful completion of our IoT-based Smart Home Automation System. Our technology connects diverse home appliances and smart devices into a cohesive network, allowing for greater control and flexibility in the domestic environment. These gadgets successfully interact with our cloud-based infrastructure via Wi-Fi and cellular connectivity, allowing homeowners to engage with their domestic environment in previously unreachable ways.

The guiding philosophy behind our design has been to keep things simple while also allowing for growth. By maintaining these goals at the forefront of our efforts, our system design enables for simple extension and the addition of new devices to the ecosystem as needed by users. Despite the complexities of device connectivity and communication, we have managed to keep the user interface simple to access and comprehend, providing the homeowner with a smooth experience.

Our strategic decision to use Wi-Fi and cellular connection for device-to-cloud communication has proven to be advantageous, resulting in a resilient, dependable, and secure network. Our system provides a reliable communication channel, backed by modern encryption and secure connection protocols - an essential component for any Smart Home Automation System.

Our online application, which offers real-time device data and allows homeowners to remotely operate the full smart home system, is at the heart of our project. Despite certain inherent constraints and obstacles, the functionality of the online application and the partially constructed mobile app shows the promise for future enhancement and extension. This degree of control and accessibility provides homeowners with a fresh approach to connect with their domestic operations, which is available at any time and from any location.

Furthermore, our system included security warnings through the SIM800L GPRS Module, which improved home security even further. This function automatically notifies homeowners of any detected movement or environmental irregularities in their houses. Despite its simplicity, it paves the way for more advanced alarm systems in the future.

In terms of limits, we effectively negotiated issues connected to hardware limitations, communication difficulties, and development constraints of the web and mobile application owing to time constraints. These constraints did not overshadow the achievement of developing a robust and dependable smart home automation system.

Finally, our project not only highlights the vision of a future in which smart homes are widespread, but it also serves as proof of technology's ability to construct systems that are both advanced and user-friendly. It exemplifies the possibilities that exist in the domain of smart technology. Despite the restrictions and obstacles we faced, the project's success represents a big step toward a future in which the benefits of smart homes, such as greater comfort, efficiency, convenience, and security, are widely available.

5. REFERENCES

[1] "NodeMCU ESP8266," Components101, [Online]. Available: https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet.

[2] "Connect your App to Firebase | Firebase Realtime Database," Firebase, [Online]. Available: https://firebase.google.com/docs/database/android/start.

[3] "Configure multiple projects | Firebase," Firebase, [Online]. Available: https://firebase.google.com/docs/projects/multiprojects.

[4] "Send real-time sensor data to Google Firebase with ESP8266," How To Electronics, Aug. 22, 2022. [Online]. Available: https://how2electronics.com/send-sensor-data-google-firebase-esp8266/.

[5] "DHT11 - AHT20 Pin Module - I2C Temperature and Humidity Sensor," Adafruit Industries, [Online]. Available: https://www.adafruit.com/product/5183.

[6] "DHT11 datasheet," Adafruit Industries, [Online]. Available: https://cdn-shop.adafruit.com/product-files/5183/5193_DHT11.pdf.

[7] B. Zuo, "Grove - Gas Sensor(MQ2) - Seeed Wiki," Seeed Studio, [Online]. Available: https://wiki.seeedstudio.com/Grove-Gas_Sensor-MQ2/.

[8] Cdaviddav, "MQ2 Gas Sensor Tutorial for Arduino, ESP8266 and ESP32," DIYI0T, May. 7, 2021. [Online]. Available: https://diyi0t.com/mq2-gas-sensor-arduino-esp8266-esp32/.

[9] "2Pcs 5V 1 Channel Level Trigger Optocoupler Relay Module," Banggood, [Online]. Available: https://usa.banggood.com/2Pcs-5V-1-Channel-Level-Trigger-Optocoupler-Relay-Module-p-1366337.html?imageAb=2.

[10] "Arduino Uno Rev3," Arduino Online Shop, [Online]. Available: https://store-usa.arduino.cc/products/arduino-uno-rev3.

[11] S. Campbell, "Basics of UART Communication," Circuit Basics, Nov. 14, 2021. [Online]. Available: https://www.circuitbasics.com/basics-uart-communication/.

[12] "Standard Size - High Torque - Metal Gear Servo," Adafruit Industries, [Online]. Available: https://www.adafruit.com/product/1142.

[13] "DIYmalls 4x4 Keypad 16 Key Matrix Array Membrane Switch 8pin Keyboard for Arduino UNO R3 (Pack of 5)," Amazon, [Online]. Available: https://www.amazon.com/DIYmalls-Keypad-Membrane-Keyboard-Arduino/dp/B086Z1ZXNJ/ref=mp_s_a_1_2_sspa?keywords=4x4+keypad.

[14] "4X4 Matrix switch keypad," DIYmalls, [Online]. Available: https://www.diymalls.com/4X4-Matrix-Array-16-Keys-Membrane-Keypad-Switch.

[15] "5V 1 Channel Relay Module With optocoupler Support High or Low Level Trigger," Shenzhen HiLetgo Technology Co., Ltd, [Online]. Available: http://www.hiletgo.com/ProductDetail/1958599.html.

[16] "I2C serial interface 1602 LCD module," Handson Tec, [Online]. Available: https://handsontec.com/dataspecs/module/I2C_1602_LCD.pdf.

[17] "i2c / SPI character LCD backpack," Kiwi Electronics, [Online]. Available: https://www.kiwi-electronics.com/en/i2c-spi-character-lcd-backpack-494.

[18] "SIM800l GSM Module," ElectroDuino, Mar. 17, 2022. [Online]. Available: https://www.electroduino.com/sim800l-gsm-module/.

[19] "GPRS GSM shield with antenna tested World wide store sim900 gsm shield DIY kit development board maker: Integrated Circuits," Aliexpress, [Online]. Available:

https://www.aliexpress.us/item/2251832620903838.html?gatewayAdapt=glo2usa4itemAdapt&_randl_shipto=US.

[20] "555-28027," Digikey, [Online]. Available: https://www.digikey.com/en/htmldatasheets/production/261959/0/0/1/555-28027.html.

[21] "DS3231 Etremel accrate I 2cinterate RTCTCXOCrstal," Maxim Integrated, [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf.

[22] "Gas Level Monitor using ESP8266 & Gas Sensor," How2Electronics, [Online]. Available: https://how2electronics.com/gas-level-monitor-esp8266-gas-sensor/.

[23] "Understanding Arduino UNO Hardware Design," All About Circuits, [Online]. Available: https://www.allaboutcircuits.com/technical-articles/understanding-arduino-uno-hardware-design/.

[24] "Hardware Structure of ARDUINO UNO," Instructables, [Online]. Available: https://www.instructables.com/Hardware-Structure-of-ARDUINO-UNO/.

[25] "NodeMCU Pinout," Circuits4You, [Online]. Available: https://circuits4you.com/2017/12/31/nodemcu-pinout/.

[26] "Arduino UNO and NodeMCU Interfacing," Arduino Forum, [Online]. Available: https://forum.arduino.cc/t/arduino-uno-and-nodemcu-interfacing/964507.

[27] "ESP8266 NodeMCU Input Output," Electronics Hub, [Online]. Available: https://www.electronicshub.org/esp8266-nodemcu-input-output/.

[28] "ESP8266 Inter-Communication Using Arduino IDE," Engineers Garage, [Online]. Available: https://www.engineersgarage.com/esp8266-inter-communication-using-arduino-ide/.

[29] "SIM900 GSM GPRS Shield Arduino," Random Nerd Tutorials, [Online]. Available: https://randomnerdtutorials.com/sim900-gsm-gprs-shield-arduino/.