

# Smart Home Automation

*Major Project Report*



**National Institute of Technology, Kurukshetra**

**Submitted By:**

**Armaan Dhillon**

Roll No: 12114058

Department of Electrical  
Engineering

**Submitted To:**

**Dr. Monika Mittal**

Department of Electrical  
Engineering

January 2025 - May 2025

## CANDIDATE DECLARATION

---

I hereby declare that the work presented in this report, “**Smart Home Automation**”, is an original work that was completed under the supervision of **Dr. Monika Mittal** and submitted to the **Department of Electrical Engineering**.

I have not submitted the work described in this dissertation for the grant of any other degree from this or any other institute. I have respected intellectual property rights in every possible way and have assisted others in using them for academic purposes.

I shall be held responsible for any complete or partial copyright violation or intellectual property rights violation discovered at any time after the award of my degree, and not my Supervisor/- Head of the Department/Institute.

Armaan Dhillon

12114058

Electrical Engineering Department  
National Institute of Technology Kurukshetra

## CERTIFICATE

---

Certified that the work described by Mr. **Armaan Dhillon** (Roll No. **12114058**), “**Smart Home Automation**”, was completed under my supervision during the **8th Semester**. According to our knowledge, this work has not been submitted for a degree elsewhere.

**Dr. Monika Mittal**

Associate Professor

NIT Kurukshetra

# ACKNOWLEDGMENTS

---

My heartfelt gratitude goes out to my supervisor, **Dr. Monika Mittal**, for her helpful assistance and unwavering support throughout this project. Her technical expertise, accurate recommendations, and timely guidance were invaluable. Her wisdom and insight provided me with a great deal of motivation.

Additionally, I would like to express my sincere gratitude to **Prof. B.V. Rama Reddy**, Director of NIT Kurukshetra, and **Prof. Jyoti Ohri**, Head of the Department, for creating a conducive environment and inspiring us to pursue constructive research efforts.

I also want to express my gratitude to all the instructors and staff at the NIT Kurukshetra Electrical Engineering Department for their invaluable assistance and support.

I would like to extend my sincere gratitude to my colleagues and seniors for their valuable suggestions and encouragement, which helped me complete my thesis work.

## **Abstract**

Home automation utilizing Raspberry Pi has grown in popularity due to its low cost and capacity to improve convenience and energy efficiency. However, many existing home automation systems suffer severe issues, such as inadequate security, restricted interoperability, lengthy configuration processes, and insufficient real-time energy monitoring. This project overcomes these restrictions by creating a smart home automation system that combines Raspberry Pi, ThingsBoard, Docker, and real-time sensor data to deliver a safe, modular, and scalable solution.

The system collects energy usage data from sensors linked to a smart energy meter and delivers it to a cloud-based IoT platform using MQTT and REST APIs. ThingsBoard enables real-time visualization and control via customisable dashboards, while Docker ensures modular deployment and service separation. The use of encrypted communication protocols improves security, while the open-source, protocol-agnostic architecture allows for smooth integration of numerous devices. This project provides a realistic and efficient solution to modern home automation by simplifying system setup and enabling real-time monitoring and automation, resulting in improved energy management and user accessibility.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Problem Statement and Proposed Solution</b>	<b>6</b>
2.1	Problem Statement . . . . .	6
2.2	Proposed Solution . . . . .	7
<b>3</b>	<b>Literature Survey</b>	<b>8</b>
3.1	Introduction . . . . .	8
3.2	Overview of the Research Domain . . . . .	8
3.3	Review of Existing Work . . . . .	9
3.3.1	Security Concerns . . . . .	9
3.3.2	Interoperability Issues . . . . .	9
3.3.3	User Experience . . . . .	9
3.3.4	Energy Efficiency . . . . .	10
3.4	Research Gaps and Motivation . . . . .	10
3.5	Comparative Analysis with the Proposed Solution . . . . .	10
3.6	Conclusion . . . . .	11
<b>4</b>	<b>Project Workflow</b>	<b>12</b>
4.1	Overview . . . . .	12
4.2	System Architecture . . . . .	12
4.3	Components and Data Flow . . . . .	12
4.3.1	Energy Measurement and Data Acquisition . . . . .	12
4.3.2	Data Transmission and Storage . . . . .	13
4.3.3	Web3 Integration and Smart Contracts . . . . .	13
4.3.4	Billing and Payment System . . . . .	13
4.4	Conclusion . . . . .	13

<b>5</b>	<b>Proteus Schematic Design and Simulation</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.2	Schematic Design in Proteus . . . . .	16
5.2.1	Smart Home Automation System . . . . .	16
5.2.2	Complete Schematic of Smart Home Automation System . . . . .	17
5.2.3	Smart Energy Meter and Power Factor Measurement . . . . .	18
5.2.4	Complete Schematic of Smart Energy Meter . . . . .	21
5.3	Code Implementation . . . . .	21
5.3.1	Arduino Code for Smart Energy Meter . . . . .	21
5.3.2	Raspberry Pi Code for Home Automation . . . . .	22
5.4	Simulation and Testing . . . . .	24
5.4.1	Home Automation Testing . . . . .	24
5.4.2	Smart Meter Testing . . . . .	24
5.5	Conclusion . . . . .	25
<b>6</b>	<b>ThingsBoard Integration</b>	<b>26</b>
6.1	Introduction to ThingsBoard . . . . .	26
6.2	Prerequisites for ThingsBoard MQTT Integration . . . . .	27
6.3	Connecting ThingsBoard with MQTT . . . . .	28
6.3.1	Step 1: Define Connection Parameters . . . . .	28
6.3.2	Step 2: Install and Import MQTT Library . . . . .	28
6.3.3	Step 3: Create MQTT Client and Connect . . . . .	29
6.3.4	Step 4: Publish Sensor Data . . . . .	29
6.3.5	Step 5: Verify Data in ThingsBoard . . . . .	30
6.3.6	Step 6: Creation of Dashboard in ThingsBoard . . . . .	30
6.4	Prerequisites for ThingsBoard API Integration . . . . .	31
6.5	Connecting to ThingsBoard API . . . . .	31
6.5.1	Step 1: Generate Access Token . . . . .	32
6.5.2	Step 2: Extract JWT Token . . . . .	32
6.5.3	Step 3: Fetch Telemetry Data . . . . .	32
6.5.4	Step 4: Process Response Data . . . . .	33
6.6	Conclusion . . . . .	33

<b>7</b>	<b>Blockchain and Smart Contracts</b>	<b>34</b>
7.1	Introduction to Blockchain . . . . .	34
7.2	Advantages of Blockchain and Smart Contracts . . . . .	34
7.3	Solidity Basics . . . . .	35
7.4	Smart Contract Deployed in the Project . . . . .	35
7.5	Ethereum and Gas Estimation . . . . .	36
7.6	Remix IDE and Smart Contract Deployment . . . . .	37
7.7	Conclusion . . . . .	38
<b>8</b>	<b>Building the Web3 Application</b>	<b>39</b>
8.1	Introduction . . . . .	39
8.2	Project Setup . . . . .	39
8.3	Connecting to Blockchain . . . . .	40
8.4	Fetching Energy Data . . . . .	40
8.5	Interacting with the Smart Contract . . . . .	40
	8.5.1 Storing Energy Data . . . . .	40
	8.5.2 Fetching and Paying Bills . . . . .	41
8.6	Frontend Interface . . . . .	42
8.7	Transaction Logs and User Interaction . . . . .	43
8.8	Conclusion . . . . .	43
<b>9</b>	<b>Conclusion and Future Scope</b>	<b>44</b>



# List of Figures

4.1	System Workflow . . . . .	13
5.1	Raspberry Pi 3 Schematic . . . . .	16
5.2	ADC (MCP3208) . . . . .	17
5.3	Complete Schematic . . . . .	18
5.4	Arduino Uno . . . . .	18
5.5	Current Sensor . . . . .	19
5.6	Voltage Measurement Circuit . . . . .	20
5.7	PowerFactor Measurement Circuit . . . . .	20
5.8	Complete Schematic of Smart Energy meter . . . . .	21
6.1	ThingsBoard's Home . . . . .	27
6.2	Device creation process in ThingsBoard . . . . .	28
6.3	Latest Telemetry after Publishing Data . . . . .	30
6.4	Dashboard Creation with ThingsBoard Widgets . . . . .	31
7.1	Gas Tracker for Ethereum Sepolia . . . . .	37
7.2	Smart contract deployment in Remix IDE . . . . .	37
8.1	User interface of the Energy Billing System . . . . .	42
8.2	Transaction log from frontend interactions with the smart contract . . . . .	43

# Chapter 1

## Introduction

Home automation using Raspberry Pi has gained popularity due to its numerous advantages and cost-effectiveness. These systems provide users with the ability to control household appliances through local networks or remote access, thereby enhancing convenience and energy efficiency[1]. Modern home automation technologies offer features such as automatic meter reading, real-time monitoring, and remote control of electrical connections without the need for personal involvement[2]. In addition to energy monitoring, home automation systems commonly include sensors, cameras, and web-based applications to increase security and enable comprehensive device control[3]. The MQTT protocol, known for its lightweight and reliable messaging, is frequently used in IoT-based systems to improve data quality and communication reliability[4].

Despite these developments, there are still certain limits in existing home automation options. Many systems continue to have insufficient encryption and access control, leaving them vulnerable to illegal access. Proprietary communication protocols and restricted integration possibilities further impede interoperability across devices from different manufacturers. Furthermore, extensive setup processes and non-intuitive user interfaces make it difficult for non-technical individuals to access and operate.

This project addresses these challenges by creating a smart home automation system using Raspberry Pi, Docker, ThingsBoard, and real-time sensor data. The use of encrypted MQTT assures safe communication, while Docker containers enable modular deployment and service isolation. ThingsBoard supports protocol-agnostic integration and user-friendly dashboards, easing system maintenance and customization. The system's real-time monitoring and automation aims to increase energy efficiency, user experience, and support for a scalable, cost-effective, and open-source approach to home automation.

# Chapter 2

## Problem Statement and Proposed Solution

### 2.1 Problem Statement

Smart home automation has received a lot of interest in recent years because of its ability to improve energy efficiency, increase convenience, and enable cognitive management over household appliances. Despite significant development in this area, current systems suffer a number of ongoing problems that restrict their effectiveness and widespread acceptance.

One of the primary considerations is security. Existing systems frequently use inadequate encryption standards and lack effective access control measures. This makes the system susceptible to unwanted access, data breaches, and manipulation of essential home systems. Furthermore, interoperability presents a substantial challenge. Many commercial smart home solutions use proprietary communication protocols and closed designs, making it impossible to integrate devices from other suppliers or expand the system without encountering compatibility concerns.

Another important factor to consider is user experience. Many smart home systems are difficult to set up and provide little customization, discouraging non-technical consumers from embracing and efficiently utilizing new technologies. Finally, energy efficiency remains a significant concern in many systems. Limited support for real-time monitoring and control of energy consumption frequently leads to wasteful power consumption and higher operational costs.

## 2.2 Proposed Solution

To overcome these issues, a smart home automation solution was created that combines Raspberry Pi, ThingsBoard, Docker, and real-time sensor data. This system seeks to be affordable, secure, flexible, and adaptable to a wide range of residential situations.

The Raspberry Pi is the primary hardware platform, operating as a local controller and managing edge computing duties. It supports sensor data gathering and communication with connected devices. ThingsBoard, an open-source IoT platform, is used for device administration, real-time data visualization, and building interactive dashboards. This platform supports several protocols and enables for simple customisation, addressing interoperability and user experience issues.

Docker uses containerization to isolate services, improve system security, and simplify deployment and upgrades. Each service operates in its own container, making the system more flexible and simple to manage or scale. Real-time sensor integration enables continuous monitoring of environmental conditions and energy consumption, resulting in automation and increased energy efficiency.

From a security standpoint, encrypted MQTT protocols are employed for data transmission, establishing a safe communication channel between devices and the cloud platform. Docker-based service isolation provides an additional degree of safety, reducing possible risks from system flaws.

In terms of interoperability, the system is intended to be protocol-agnostic and completely open source. This enables the smooth integration of several devices, independent of manufacturer, and encourages vendor independence. To improve the user experience, the platform includes visual dashboards with drag-and-drop widgets, allowing users to design and monitor their smart home environment with little technical knowledge.

Real-time sensor data enables dynamic decision-making, allowing the system to respond immediately to changes in ambient conditions. This improves energy efficiency by enabling context-aware automation, such as changing lighting or shutting off equipment when not in use.

# Chapter 3

## Literature Survey

### 3.1 Introduction

Smart home automation has become a vital part of modern living, offering benefits such as increased convenience, energy optimization, and enhanced security. With the advancement of Internet of Things (IoT) technologies, homes are now equipped with interconnected sensors, controllers, and cloud platforms that enable real-time automation and monitoring. However, as noted in numerous studies, the journey toward widespread adoption remains hindered by technical, financial, and usability-related challenges [5][6][7].

While these technologies show promise, real-world implementations frequently fall short due to constraints such as inadequate device interoperability, security risks, a lack of customisation, and exorbitant installation costs. Existing systems sometimes rely on closed ecosystems, further complicating integration and limiting user flexibility. This chapter explores major literature in the field, identifies the primary barriers to smart home automation, and compares those results to our system's implementation, which uses Raspberry Pi, ThingsBoard, Docker, and real-time sensor-cloud connection.

### 3.2 Overview of the Research Domain

The smart home ecosystem comprises a diverse set of devices and platforms that collaborate to automate household operations. Temperature control, lighting, security systems, and energy monitoring are all examples of IoT-enabled applications. As seen in recent studies, the integration of different sensors and platforms brings both functional potential and additional complications[8].

Security breaches, data privacy issues, device incompatibilities, and poor user engagement remain significant impediments to adoption. Although open-source systems and current edge computing devices such as the Raspberry Pi provide promising possibilities, much of the research focuses on systems that are either highly centralized or dispersed due to proprietary constraints. This assessment divides the main challenges into four categories: security, interoperability, user experience, and energy efficiency.

## **3.3 Review of Existing Work**

### **3.3.1 Security Concerns**

Security in smart home automation is a serious concern raised in various research. As the number of internet-connected gadgets grows, so does the attack surface for cyber attacks. According to multiple sources, many IoT devices lack strong encryption and authentication, leaving them vulnerable to unauthorized access[8][9]. Others raise worries about user data being collected or modified owing to inadequate or nonexistent privacy protections[10]. Furthermore, the lack of common security methods across different manufacturers increases these risks[11][12].

### **3.3.2 Interoperability Issues**

Interoperability refers to the ability of equipment from various vendors to communicate seamlessly. According to research, many smart home systems fail to integrate effectively because they use proprietary communication protocols[13][11]. Incompatibility between platforms frequently causes users to rely on vendor-specific hubs or apps, resulting in a disjointed user experience. This difficulty is exacerbated by the lack of globally accepted IoT standards[12].

### **3.3.3 User Experience**

Despite their functional potential, smart home systems can provide a terrible user experience. According to studies, setup methods are typically technically complex, preventing non-expert users[14][15]. Other findings indicate that many systems fail to tailor interactions depending on user behavior, leaving automation seeming stiff and detached[16]. Furthermore, non-intuitive interfaces and a lack of transparency increase user irritation and slow adoption[17].

### **3.3.4 Energy Efficiency**

Energy efficiency is usually highlighted as a significant motivator for smart home adoption, however research shows that many systems fall short of meeting this expectation. Several studies address how real-time energy monitoring and management are underutilized due to the high system complexity[14]. Others point out that balancing user comfort with energy-saving automation is a continuing challenge[11][16], and that consumers frequently lack awareness into their own consumption habits[18].

## **3.4 Research Gaps and Motivation**

While the present literature thoroughly examines the theoretical and technological underpinnings of smart home automation, the majority of solutions are either too expensive or do not provide end-to-end capability. Systems that rely significantly on centralized cloud services are vulnerable to latency, data breaches, and platform dependencies. On the other hand, decentralized systems frequently lack the coordination and integration needed for comprehensive automation.

The proposed project was motivated by the need for a modular, cost-effective, and fully adaptable system that could be readily installed and maintained. The project takes a balanced approach, using a Raspberry Pi as an edge controller, ThingsBoard for visualization and device management, and Docker for scalable deployment, to bridge the gap between academic research and real-world applications.

## **3.5 Comparative Analysis with the Proposed Solution**

The proposed smart home automation solution utilizes Raspberry Pi, ThingsBoard, Docker, and real-time sensor data integration to address limitations identified in existing literature. By leveraging open-source and containerized technologies, it aims to offer a low-cost, secure, and modular framework adaptable to varied use cases. Table 3.1 summarizes how this solution compares with the challenges found in previous research.

Challenge	Literature Observations	Proposed Solution
Security	Weak encryption, poor access control	Encrypted MQTT, Docker-based isolation
Interoperability	Proprietary protocols, poor integration	Open-source, protocol-agnostic stack
User Experience	Complex setup, limited personalization	Visual dashboards, easy configuration
Energy Efficiency	Limited real-time monitoring	Live sensor data, extensible dashboards

Table 3.1: Comparison of Literature Challenges and Proposed Solution

## 3.6 Conclusion

Smart home automation is always evolving, with substantial advances in device capabilities, cloud integration, and automation intelligence. However, difficulties such as security vulnerabilities, device compatibility, usability, and energy management continue to impede large-scale adoption. Existing literature identifies these difficulties, but often lacks practical remedies.

This project provides a comprehensive, technically competent response to these difficulties. It provides a realistic framework that corresponds with academic research concepts while boosting scalability, dependability, and user empowerment by integrating open-source platforms, containerized infrastructure, and real-time sensor-cloud communication.



# Chapter 4

## Project Workflow

### 4.1 Overview

This chapter depicts the workflow of a smart home automation system that is coupled with blockchain for energy billing. The smart meter captures energy usage data, which is then transmitted to a cloud-based IoT platform for real-time monitoring[1][2]. Data is processed using MQTT and REST APIs to ensure reliability[4].

A Web3 application obtains this information and communicates with an Ethereum smart contract to generate energy bills in a secure and transparent manner[19][20]. By merging IoT and blockchain, the solution automates billing and payments, increasing efficiency and security.

### 4.2 System Architecture

The architecture consists of multiple interconnected components, including sensors, micro-controllers, cloud platforms, and blockchain technology. The data flow and interactions are illustrated in Figure 4.1.

### 4.3 Components and Data Flow

#### 4.3.1 Energy Measurement and Data Acquisition

The system starts with an Arduino connected to a smart energy meter, which measures power consumption. Since Arduino lacks a built-in analog-to-digital converter (ADC) with the required resolution, an external ADC is used for accurate readings. The measured data is then transmitted to a Raspberry Pi for further processing.

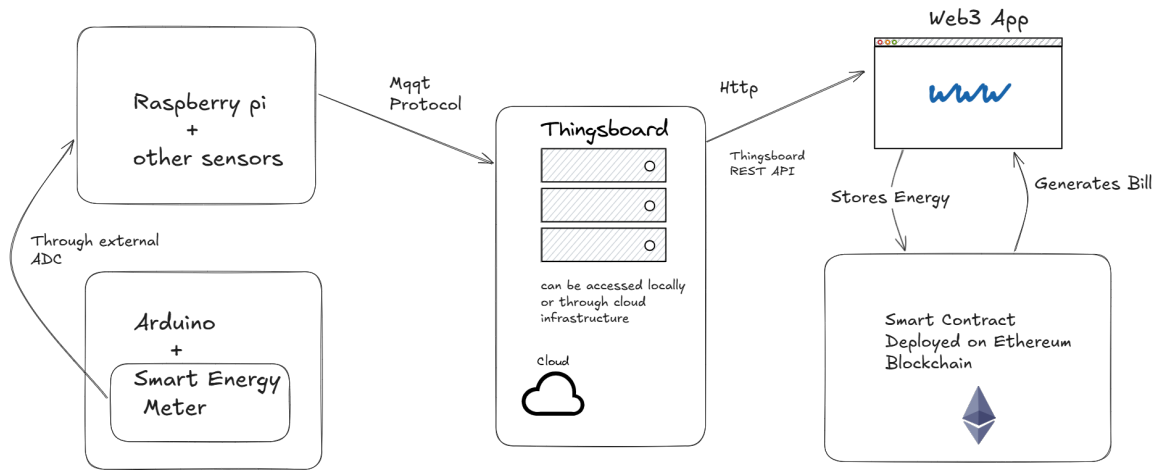


Figure 4.1: System Workflow

### 4.3.2 Data Transmission and Storage

This chapter illustrates the workflow of a smart home automation system that is integrated with blockchain for energy billing. The system's goal is to collect energy consumption data, store it in a cloud-based IoT platform, and bill and pay using a Web3 application that interacts with an Ethereum blockchain smart contract.

### 4.3.3 Web3 Integration and Smart Contracts

A Web3 application fetches energy consumption data from ThingsBoard via HTTP requests. The retrieved energy data is then stored in a smart contract deployed on the Ethereum blockchain. The smart contract automatically generates an energy bill based on the stored consumption values.

### 4.3.4 Billing and Payment System

Using HTTP queries, a Web3 application retrieves energy usage data from ThingsBoard. A smart contract that is implemented on the Ethereum blockchain then stores the energy data that has been recovered. Using the saved consumption values, the smart contract automatically creates an energy bill.

## 4.4 Conclusion

This process combines blockchain, cloud computing, and the Internet of Things to produce an automated and decentralized energy billing system. An effective and transparent billing

procedure is made possible by the modular architecture, which facilitates smooth data flow from energy monitoring to smart contract interactions.

# Chapter 5

## Proteus Schematic Design and Simulation

### 5.1 Introduction

Proteus is a powerful software program that is commonly used to develop and simulate embedded systems prior to their real implementation. It allows you to efficiently test circuits, detect design problems, and enhance performance without the need for actual hardware. This chapter focuses on the schematic design and modeling of a smart home automation system and a smart energy meter, which integrate sensors, actuators, and communication modules to provide real-time monitoring and control.

The Raspberry Pi 3 serves as the core processing unit for the smart home automation system, which communicates with a variety of sensors such as Light Dependent Resistors (LDR), Passive Infrared (PIR) sensors, and gas detectors to automate household appliances. The Raspberry Pi lacks built-in analog input capabilities, thus an external ADC (MCP3208) is used to interpret analog sensor data and enabling IoT-based automation[21].

The smart energy meter focuses on real-time power consumption monitoring, incorporating an Arduino Uno for data acquisition, ACS712 current sensors for current measurement, and voltage dividers for accurate voltage monitoring [22]. Additionally, power factor measurement is achieved using LM358 operational amplifiers and XOR gates to analyze phase differences between voltage and current waveforms [23]. These measurements are transmitted to the IoT platform, enabling remote energy management and optimization.

This chapter details the schematic design, component selection, and software implementation in Proteus. The proposed system provides not just automation and remote monitoring, but

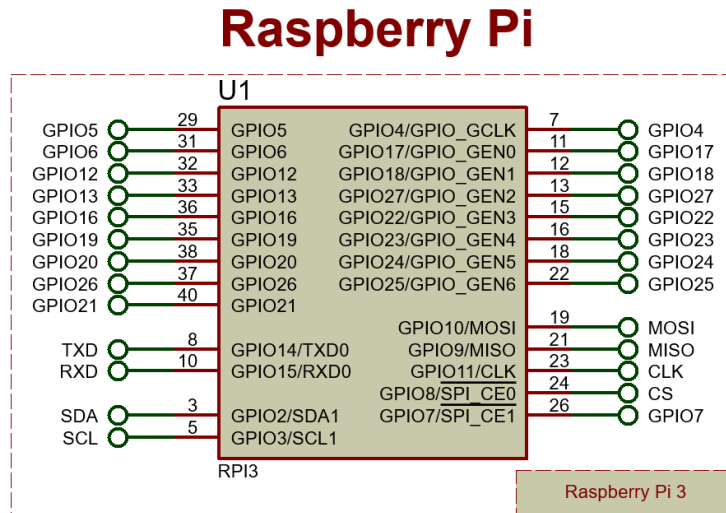
also efficient energy utilization and billing. Proteus simulations evaluate the system’s many features prior to real-world deployment, ensuring reliability and performance.

## 5.2 Schematic Design in Proteus

### 5.2.1 Smart Home Automation System

The home automation circuit includes:

**Raspberry Pi 3:** The Raspberry Pi 3 B+ was chosen for this home automation project because of its powerful capabilities and simplicity of integration. It is driven by a Broadcom BCM2837B0 quad-core Cortex-A53 (ARMv8) processor clocked at 1.4GHz, with 1GB LPDDR2 SDRAM and a Broadcom Videocore-IV GPU. Networking options include Gigabit Ethernet (by USB), dual-band Wi-Fi (2.4GHz and 5GHz 802.11b/g/n/ac), and Bluetooth 4.2 (BLE)[21]. It includes a 40-pin GPIO header, HDMI, a 3.5mm audio connector, four USB 2.0 ports, Ethernet, CSI, and DSI. The Micro-SD storage format and compact dimensions (82mm × 56mm × 19.5mm, 50g) make it ideal for embedded applications[21].



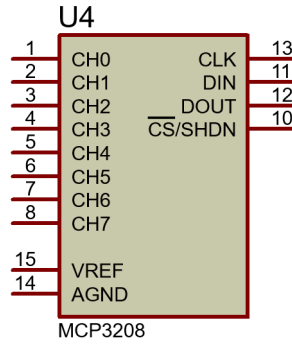


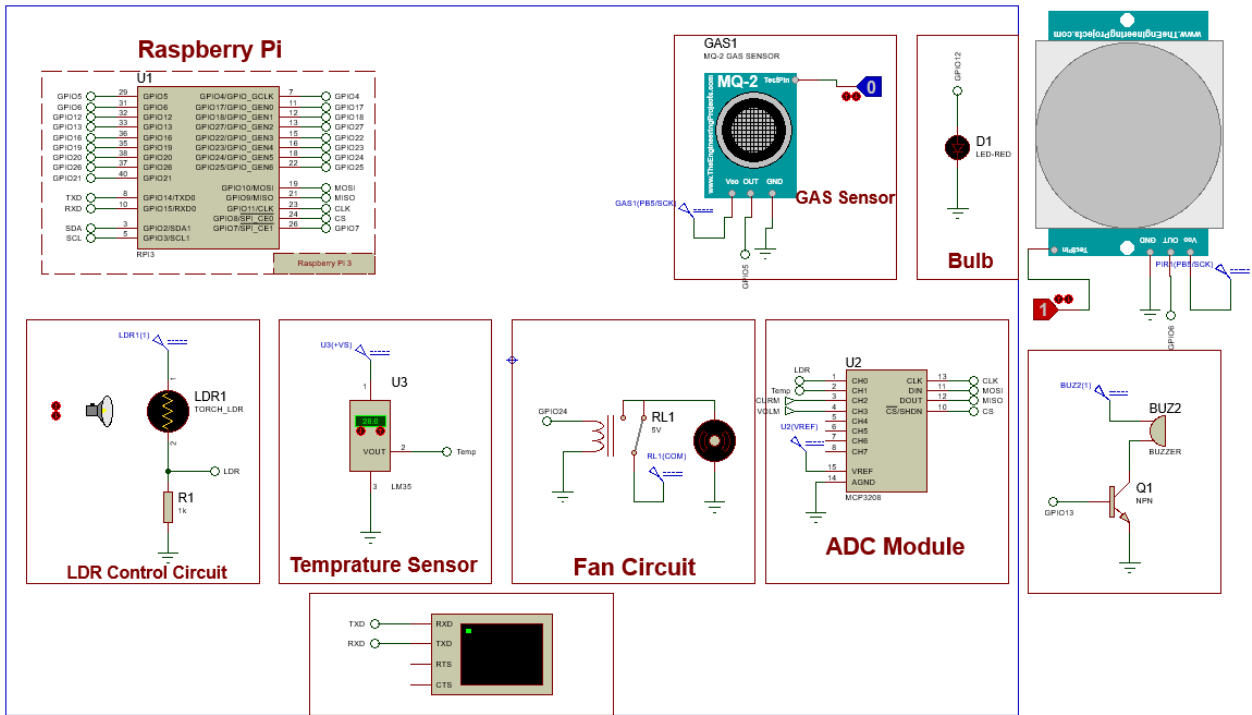
Figure 5.2: ADC (MCP3208)

Because Raspberry Pi lacks built-in analog inputs, an external ADC such as the MCP3208 is required to read analog sensor data. Interfacing via SPI allows for real-time monitoring in IoT applications and industrial automation[25].

**Other Sensors:** We used a variety of sensors and actuators to allow for sophisticated control and monitoring. The system uses an LDR (Light Dependent Resistor) to detect ambient light levels, allowing lights to be turned on or off based on the brightness of the surroundings. A PIR (Passive Infrared) sensor was used to detect motion and human presence. This sensor is essential for security and automated lighting because it activates lights or alerts when motion is detected. Furthermore, a MQ-2 gas sensor was used to monitor air quality and identify combustible gases such as LPG and CO, assuring safety by sending alarms in the event of a gas leak. The LM35 temperature sensor was included to properly monitor room temperature and allow for automatic cooling control.

### 5.2.2 Complete Schematic of Smart Home Automation System

The smart home automation circuit integrates multiple sensors and actuators to automate home appliances based on environmental conditions and involves following connections.



of up to 100 kHz with extra libraries[26]. The Arduino Uno has been shown to be effective in controlling industrial-scale instruments, performing similarly to industrial-class controllers that use PID algorithms[27]. The Arduino Uno, which is based on the Atmega328 processor, supports Assembly and C programming, allowing students to learn about microcontroller architecture and interact with real-world devices like LCDs, motors, and sensors[28].

**Current Measurement:** In our smart home automation project, we employ the ACS712 Hall-effect-based current sensor to monitor AC/DC power levels. It has low resistance (1.2 mohm) and 2.1 kVRMS isolation for precise and safe current measuring[22]. The 30A variant with 66mV/A sensitivity is integrated with an arduino uno, allowing for real-time power tracking and energy consumption analysis via ThingsBoard. Calibration improves accuracy and reduces measurement error, making it ideal for IoT-based billing and automation systems. Its rapid reaction (5  $\mu$ s) and low noise improve performance for smart energy management[23].

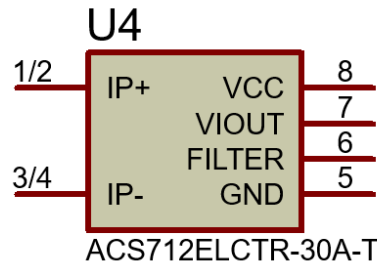


Figure 5.5: Current Sensor

**Voltage Measurement:** We monitor AC voltage in our smart home automation project with a step-down transformer, a voltage divider network, and a filtering capacitor. The step-down transformer (TR3) converts the high AC mains voltage to a lower, safer level suited for measurement. The voltage divider circuit (R9, R10, R11, and R12) reduces the voltage to a range compatible with the Arduino ADC, ensuring precise readings while protecting the microcontroller. A 500 $\mu$ F capacitor (C2) filters, reduces noise, and stabilizes the signal for accurate RMS voltage measurement. This configuration enables the Arduino to process voltage data and send it to ThingsBoard, allowing for real-time monitoring and effective energy management in our smart home system.



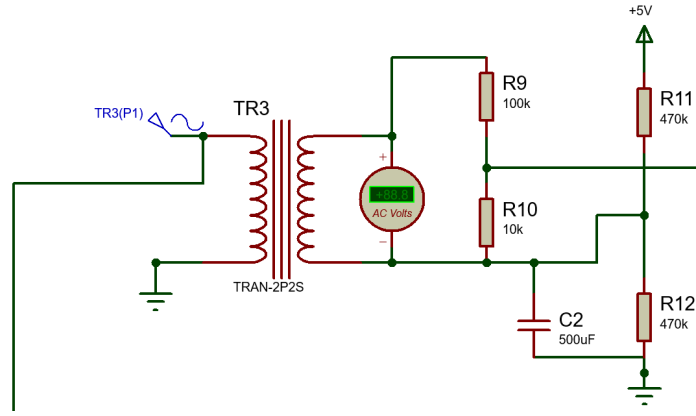


Figure 5.6: Voltage Measurement Circuit

**Power Factor Measurement:** we monitor power factor with LM358 operational amplifiers and an XOR gate. The LM358 op-amps act as zero-crossing detectors, transforming AC voltage and current waveforms to square waves. The first op-amp (U8:A) detects zero crossings in the voltage waveform, whereas the second op-amp (U8:B) detects current waveform. These processed signals are then routed through an XOR gate (U9), which generates a pulse-width output proportional to the phase difference between the voltage and current waves. The Arduino measures pulse width and calculates power factor as  $\cos(\theta)$ , where  $\theta$  is the phase angle.

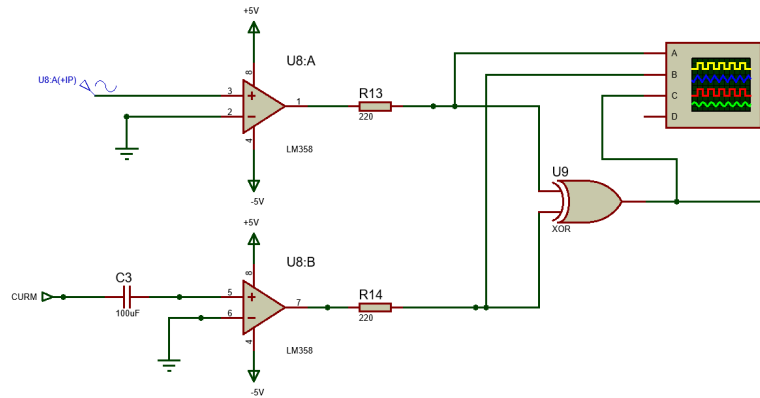


Figure 5.7: PowerFactor Measurement Circuit

**relay-based load control** electrical appliances are managed remotely using a relay control system. The circuit consists of electromagnetic relays (RL4, RL5) that are controlled by the microcontroller's GPIO pins (for example, Raspberry Pi or Arduino). When a GPIO pin is set to HIGH, the accompanying relay coil is activated, shutting the switch and enabling current to flow to the attached load (such as lights or motors).

## 5.2.4 Complete Schematic of Smart Energy Meter

The smart energy meter circuit is designed to measure and monitor power consumption while ensuring efficient energy usage.

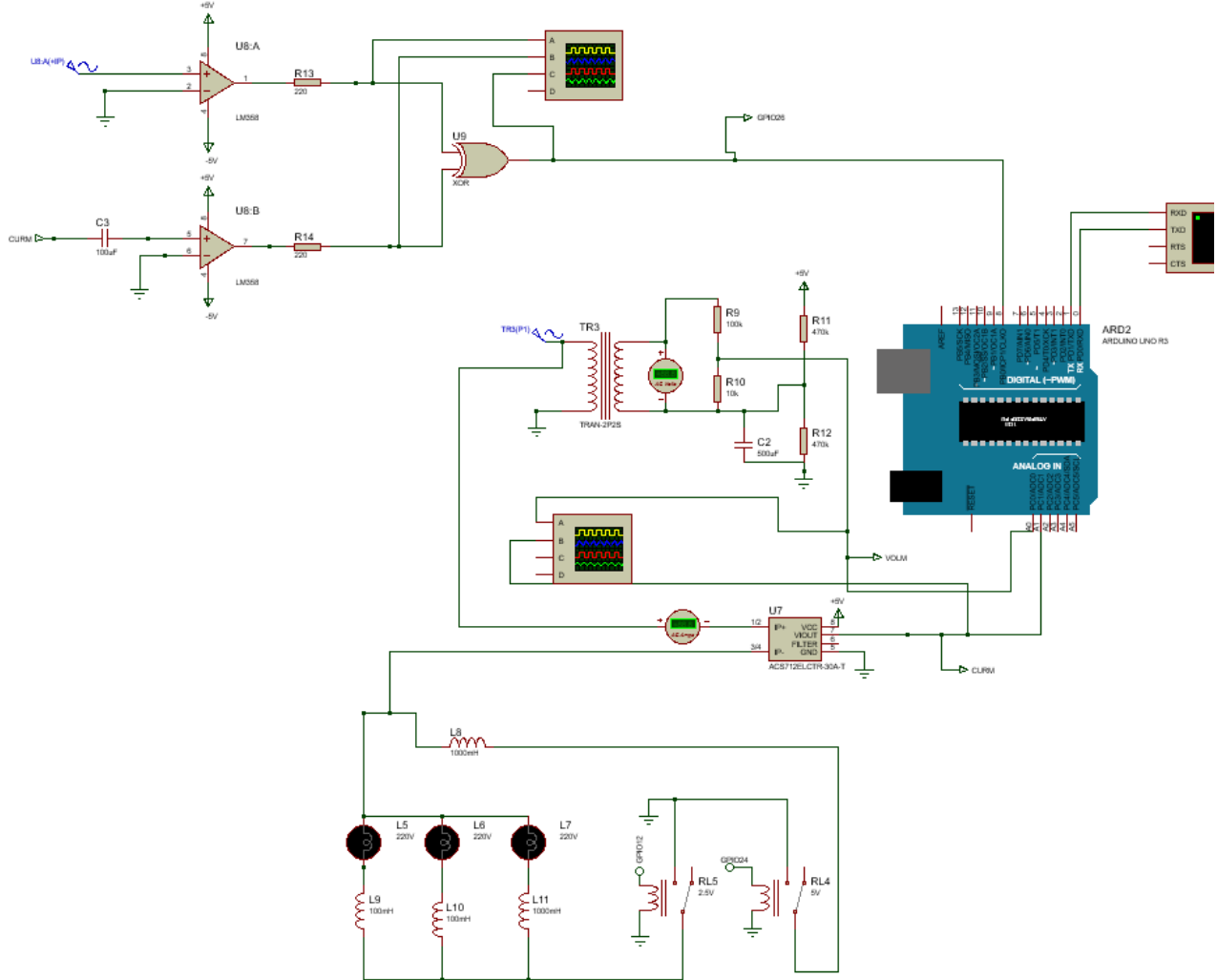


Figure 5.8: Complete Schematic of Smart Energy meter

## 5.3 Code Implementation

### 5.3.1 Arduino Code for Smart Energy Meter

The Arduino is responsible for measuring voltage, current, and calculating power consumption in real-time. The analog pins read voltage and current sensor outputs, which are then converted into meaningful electrical values. The calculated power is then transmitted over serial communication.

```

1 // Pin configuration
2 const int voltagePin = A0; // Voltage sensor input
3 const int currentPin = A1; // Current sensor input
4
5 // Constants for sensor calibration
6 const float voltageMultiplier = 230.0 / 1023.0; // Adjust based on sensor
    specs
7 const float currentMultiplier = 10.0 / 1023.0; // Adjust for CT sensor
    scaling
8
9 void setup() {
10     Serial.begin(9600); // Start serial communication
11 }
12
13 void loop() {
14     // Read raw analog values from sensors
15     int rawVoltage = analogRead(voltagePin);
16     int rawCurrent = analogRead(currentPin);
17
18     // Convert raw values to real-world measurements
19     float voltage = rawVoltage * voltageMultiplier;
20     float current = rawCurrent * currentMultiplier;
21     float power = voltage * current; // Active power calculation
22
23     // Print the measurements to serial monitor
24     Serial.print("Voltage: "); Serial.print(voltage); Serial.print(" V, ");
25     Serial.print("Current: "); Serial.print(current); Serial.print(" A, ");
26     Serial.print("Power: "); Serial.print(power); Serial.println(" W");
27
28     delay(1000); // Wait 1 second before next reading
29 }

```

Code 5.1: arduino code for energy calculations

### 5.3.2 Raspberry Pi Code for Home Automation

The Raspberry Pi is responsible for reading sensor data, controlling home appliances based on environmental conditions, and sending data to an IoT platform (ThingsBoard) using MQTT.

```

1 import RPi.GPIO as GPIO
2 import time
3 import json
4 import spidev
5 import math
6 from paho.mqtt.client import Client
7
8 # GPIO setup
9 GPIO.setmode(GPIO.BCM)
10 GPIO.setup(17, GPIO.IN) # LDR sensor for light detection
11 GPIO.setup(27, GPIO.OUT) # Relay control for light switching
12
13 # SPI setup for ADC communication (e.g., MCP3008)
14 spi = spidev.SpiDev()
15 spi.open(0, 0)
16 spi.max_speed_hz = 1350000
17
18 # MQTT setup
19 THINGSBOARD_HOST = "localhost"
20 ACCESS_TOKEN = "your_access_token"
21 client = Client()
22 client.username_pw_set(ACCESS_TOKEN)
23 client.connect(THINGSBOARD_HOST, 1883, 60)
24
25 # Read ADC channel for sensor values
26 def read_channel(channel):
27     adc = spi.xfer2([1, (8 + channel) << 4, 0])
28     return ((adc[1] & 3) << 8) + adc[2]
29
30 # Calculate RMS voltage from multiple samples
31 def calculate_rms_voltage():
32     sum_squares = sum((read_channel(3) * 3.3 / 1023.0) ** 2 for _ in range
33                       (200))
34     return math.sqrt(sum_squares / 200) * 11 # Voltage divider correction
35
36 # Publish sensor data to MQTT
37 def publish_sensor_data():
38     while True:
39         light_level = read_channel(0) # Read LDR sensor value
40         GPIO.output(27, light_level < 100) # Turn on relay if dark

```

```

40
41     payload = json.dumps({
42         "light": light_level,
43         "voltage": calculate_rms_voltage()
44     })
45     client.publish("v1/devices/me/telemetry", payload)
46     time.sleep(0.5) # Wait before next update
47
48 publish_sensor_data()

```

Code 5.2: Raspberry Pi Code Collects Sensor Data and Sends it over using MQTT

## 5.4 Simulation and Testing

### 5.4.1 Home Automation Testing

Several tests were run in the Proteus simulation environment to validate the smart home automation system. The LDR-based light control circuit was tested under various lighting situations. The simulation confirmed that when the light intensity fell below a particular threshold, the relay was activated, which turned on the linked bulb. When the light intensity increased, the relay deactivated and the bulb turned off. This behavior provided automatic lighting management based on ambient light levels.

The PIR sensor was tested by mimicking motion within its detecting range. When motion was detected, the buzzer sounded an alert. When there was no motion, the buzzer remained off. This proved that the motion detection system was working properly, making it appropriate for security applications.

The MQTT communication between the Raspberry Pi and ThingsBoard Cloud was also tested. Sensor data, including light intensity and relay state, was successfully transmitted and displayed on the ThingsBoard dashboard in real time. This validated the proper integration of IoT communication in the system.

### 5.4.2 Smart Meter Testing

The smart energy meter system was evaluated for accuracy and dependability in measuring electrical characteristics. The Arduino successfully obtained voltage and current readings from the sensors. These readings were compared to expected values, and the findings revealed an

acceptable margin of error, indicating measurement accuracy.

The power consumption was calculated using the formula  $P = V \times I$ , and the computed values were displayed on the serial monitor. The readings matched theoretical expectations, demonstrating the correct implementation of power measurement.

Furthermore, the power factor correction circuit was simulated by adding different loads. As reactive power varied, the circuit modified to enhance the power factor. This enabled the system to dynamically optimize power consumption, decreasing energy waste.

## 5.5 Conclusion

The Proteus simulation offered a dependable testing environment for both the smart home automation and smart energy metering systems. The home automation system operated as planned, automatically regulating lighting based on ambient conditions and detecting motion to trigger security warnings. The smart meter measured electrical characteristics precisely and estimated power usage with excellent reliability.

Both systems' successful simulation testing shows that they are ready for real-world implementation. With the right hardware, these systems can improve energy efficiency and automation in smart home situations.

# Chapter 6

## ThingsBoard Integration

### 6.1 Introduction to ThingsBoard

Rapid breakthroughs in semiconductor technology and wireless communication have resulted in the development of low-cost sensor-based devices, which serve as the foundation for the Internet of Things (IoT) ecosystem[29]. These sensors produce massive volumes of data, demanding effective collection, processing, and management frameworks. IoT platforms play an important role in managing this data by offering connectivity, security, data visualization, and analytics capabilities[30][31]. Among these platforms, ThingsBoard has emerged as an effective open-source solution for IoT data collecting and management.

ThingsBoard is a Java 8-based IoT platform that acts as a gateway for devices that communicate using MQTT[32], CoAP[33], and HTTP[34]. These protocols allow for lightweight communication between resource-constrained IoT devices and cloud services. MQTT is a publish/subscribe protocol for small, low-power devices that enables efficient message exchange via a broker with varying Quality of Service (QoS) levels[32]. In contrast, CoAP is an UDP-based protocol designed for limited contexts, with lower overhead but lesser dependability than MQTT[33]. One of ThingsBoard's key features is the ability to build rules and plugins for message processing. Rules contain data filters, metadata enrichment processors, and action triggers that change messages into new formats before sending them to plugins. This rule-based system supports basic data processing, including threshold-based notifications. However, the technology does not automatically support complex data aggregation over time or across several devices[30].

ThingsBoard allows you to configure alerts for both devices and assets, which improves real-time monitoring and event-based automation. When aberrant situations are recognized, these alerts

alert users or initiate automated replies. Furthermore, the platform supports both lightweight communication protocols, such as MQTT and CoAP, and classic RESTful services[33].

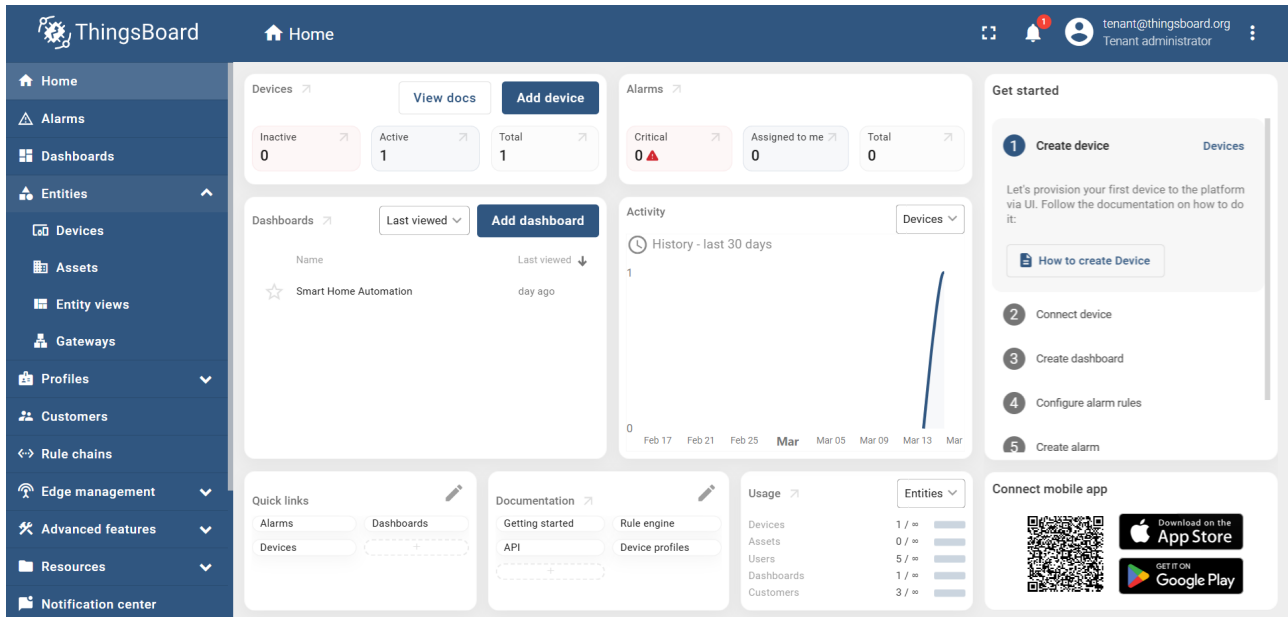


Figure 6.1: ThingsBoard’s Home

## 6.2 Prerequisites for ThingsBoard MQTT Integration

Before integrating ThingsBoard with MQTT, ensure that the required software components are installed and configured correctly. The ThingsBoard platform must be running on either a local system or a cloud server. Although ThingsBoard has a built-in MQTT broker, an external broker such as Mosquitto can also be used if needed. Each device must first be created in ThingsBoard before it can communicate via MQTT. To create a new device, navigate to the ThingsBoard dashboard, go to the **Devices** section, and click on **Add New Device**. Assign a meaningful name and select the appropriate device type. Once the device is created, an **Access Token** is generated, which will be required for authentication in MQTT communication. A visual representation of the device creation process is shown in Figure 6.2. Additionally, MQTT communication requires appropriate network settings. Ensure that port **1883** is open for unencrypted communication. Each device must authenticate with ThingsBoard using an **Access Token**, which is assigned when the device is created in ThingsBoard.



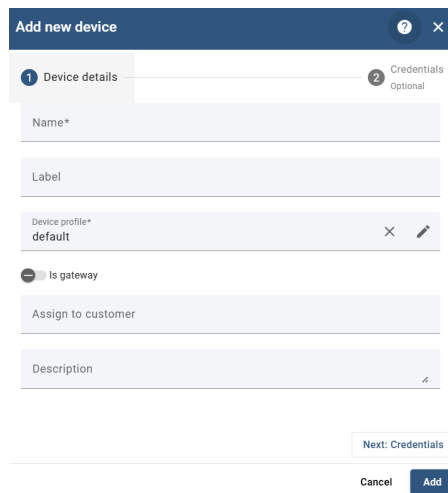


Figure 6.2: Device creation process in ThingsBoard

## 6.3 Connecting ThingsBoard with MQTT

To send telemetry data to ThingsBoard using MQTT, follow these steps:

### 6.3.1 Step 1: Define Connection Parameters

The first step is to configure the MQTT connection by specifying the ThingsBoard host, access token, and MQTT port. Replace the `ACCESS_TOKEN` with the token obtained from the ThingsBoard device.

```
1 THINGSBOARD_HOST = "localhost"
2 ACCESS_TOKEN = "dU6S0YIAPX5WwfmB3wUi" # Replace with your actual token
3 MQTT_PORT = 1883
4
```

Code 6.1: Connection Parameters as Environment Variables

### 6.3.2 Step 2: Install and Import MQTT Library

After setting up the connection parameters, ensure that the `paho-mqtt` library is installed. This library is required to establish an MQTT connection. Once installed, import the necessary modules in the Python script.

```
1 import paho.mqtt.client as mqtt
2 import json
3
```

Code 6.2: Importing MQTT using Python

### 6.3.3 Step 3: Create MQTT Client and Connect

The next step is to create an MQTT client instance and authenticate using the access token. The client must then connect to the ThingsBoard host on the specified port.

```
1 client = mqtt.Client()
2 client.username_pw_set(ACCESS_TOKEN) # Use Access Token for authentication
3 client.connect(THINGSBOARD_HOST, MQTT_PORT, 60)
4
```

Code 6.3: Connecting MQTT to ThingsBoard using Device Token

### 6.3.4 Step 4: Publish Sensor Data

Once connected, sensor data must be prepared in JSON format and published to the ThingsBoard MQTT topic. The following code snippet demonstrates how to send temperature and humidity data.

```
1 telemetry_data = {"current": 14.919, "fan_status": 0, "gas_detected": 0, "
    light": 61, "motion": 1, "power": 3493.48, "power_factor": 0.98, "temperature
    ": 27.85}
2 client.publish("v1/devices/me/telemetry", json.dumps(telemetry_data))
3 print("Data sent successfully!")
4 client.disconnect()
5
```

Code 6.4: Sending Telemetry Response in JSON

Telemetry				+	Q
<input type="checkbox"/>	Last update time	Key ↑	Value		
<input type="checkbox"/>	2025-03-17 08:36:47	current	14.919969877234003		
<input type="checkbox"/>	2025-03-17 08:36:47	fan_status	0		
<input type="checkbox"/>	2025-03-17 08:36:47	gas_detected	0		
<input type="checkbox"/>	2025-03-17 08:36:47	light	61		
<input type="checkbox"/>	2025-03-17 08:36:47	motion	1		
<input type="checkbox"/>	2025-03-17 08:36:47	power	3493.487321506501		
<input type="checkbox"/>	2025-03-17 08:36:47	power_factor	0.98		
<input type="checkbox"/>	2025-03-17 08:36:47	temperature	27.85923753665689		

Items per page: 10 1 – 9 of 9 << < > >>

Figure 6.3: Latest Telemetry after Publishing Data

### 6.3.5 Step 5: Verify Data in ThingsBoard

After publishing the data, it is essential to verify whether ThingsBoard has received it. This can be done by navigating to the ThingsBoard UI and checking the **Latest Telemetry** section of the configured device.

### 6.3.6 Step 6: Creation of Dashboard in ThingsBoard

After creating a device in ThingsBoard, the next step is to create a dashboard for monitoring and visualization. To create a new dashboard, navigate to the **Dashboards** section and click on **Create New Dashboard**. Provide a meaningful name and description for easy identification. Once the dashboard is created, widgets can be added to visualize device telemetry data. Click on **Edit Dashboard**, then use the **Add New Widget** option to select an appropriate widget type, such as charts, gauges, or tables. Link the widget to the corresponding device and telemetry keys. The dashboard creation process is illustrated in Figure 6.4.

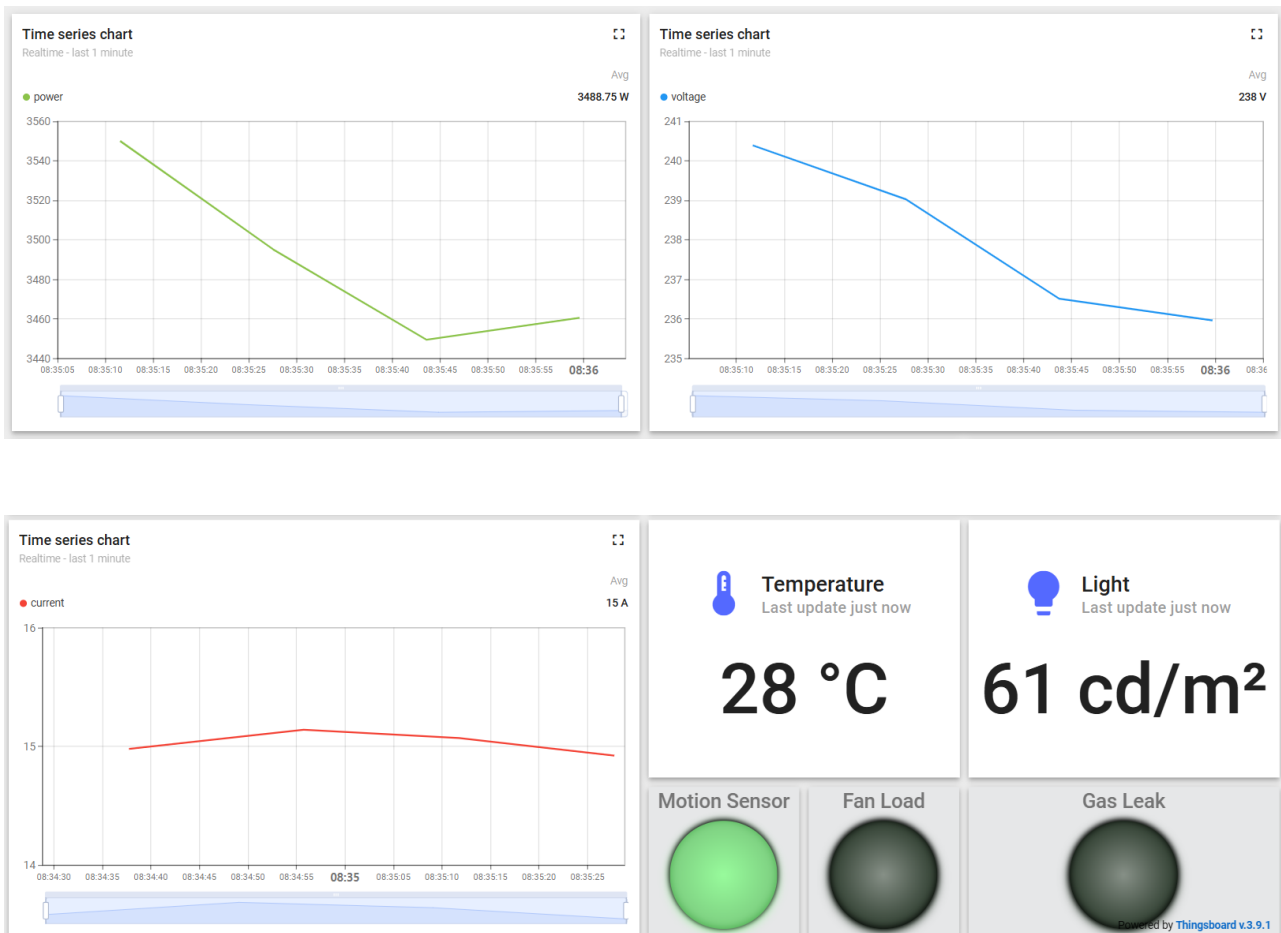


Figure 6.4: Dashboard Creation with ThingsBoard Widgets

## 6.4 Prerequisites for ThingsBoard API Integration

Before integrating ThingsBoard with the API, ensure that the required software and authentication mechanisms are set up properly. The ThingsBoard platform must be installed on a local system or a cloud server to facilitate communication. A REST client such as Postman or the `fetch` API in JavaScript is necessary for interacting with ThingsBoard's API. Additionally, proper network access should be configured to allow API requests.

To ensure security, ThingsBoard requires authentication via a JSON Web Token (JWT). The authentication process involves sending a username and password to ThingsBoard's login API. Upon successful authentication, a token is generated, which must be included in all subsequent API requests. Without this token, ThingsBoard will deny access to its resources.

## 6.5 Connecting to ThingsBoard API

To fetch telemetry data from ThingsBoard, follow these steps:



```
3      {
4          headers: { "X-Authorization": 'Bearer ${thingsboardToken}' },
5      }
6  );
7  const data = await response.json();
8  console.log(data);
9
```

Code 6.7: Fetching Data from ThingsBoard using Token Authentication method

#### 6.5.4 Step 4: Process Response Data

Once the telemetry data is retrieved, it will be returned in JSON format. The response will contain key-value pairs representing the requested telemetry values. The received data can then be processed, displayed, or stored as needed, depending on the application requirements.

## 6.6 Conclusion

Integrating ThingsBoard with MQTT allows for efficient and secure real-time data transmission for IoT devices. By following the methods mentioned, devices can authenticate, publish telemetry data, and take advantage of ThingsBoard's monitoring and automation capabilities. This lightweight and scalable strategy improves IoT installations by providing consistent connectivity while also allowing for future enhancements such as encryption and improved data handling.

# Chapter 7

## Blockchain and Smart Contracts

### 7.1 Introduction to Blockchain

Blockchain is a fast developing technology for safe, transparent, and decentralized data management. It is based on three core components: private key cryptography, peer-to-peer networking, and smart contracts[35]. Transactions are encrypted with cryptographic keys, validated by distributed nodes, and kept immutably, making blockchain immune to fraud and illegal changes[36]. Initially, blockchain was largely utilized for peer-to-peer financial transactions, as demonstrated by Bitcoin. However, in 2013, Ethereum launched smart contracts, which allow for the automatic implementation of agreements without the use of middlemen[37]. These self-executing contracts specify the terms and circumstances of a transaction while ensuring efficiency, security, and dependability[38].

Smart contracts have transformed businesses by allowing for smooth transactions in fields such as finance, supply chain management, and digital asset exchanges. By eliminating intermediaries, they save money, boost transparency, and build confidence among participants[39]. Blockchain and smart contracts are becoming increasingly popular, opening up new opportunities for safe and efficient digital transactions.

### 7.2 Advantages of Blockchain and Smart Contracts

Blockchain technology offers numerous advantages, including decentralization, security, transparency, and immutability. Since blockchain operates on a distributed ledger, there is no single point of failure, reducing the risk of downtime and attacks. Transactions recorded on the blockchain are secure due to cryptographic hashing, ensuring data integrity. Transparency

is another key advantage, as all transactions are publicly verifiable and tamper-proof. Additionally, blockchain eliminates intermediaries, making financial transactions more efficient and cost-effective.

Smart contracts further enhance the blockchain ecosystem by automating and enforcing agreements without requiring intermediaries. They execute predefined conditions and ensure that transactions occur only when the conditions are met. This eliminates the need for third parties such as banks, legal entities, or brokers, thus reducing costs and increasing efficiency. Smart contracts also provide trust and security as they are immutable once deployed on the blockchain.

## 7.3 Solidity Basics

Solidity is a high-level programming language used for writing smart contracts on Ethereum. It is statically typed and influenced by JavaScript, Python, and C++. Solidity contracts consist of state variables, functions, events, and modifiers.

## 7.4 Smart Contract Deployed in the Project

Below is an example of a Solidity smart contract implementing an energy billing system:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract EnergyBilling {
5     address public owner;
6     mapping(address => uint256) public userEnergy; // Stores energy
    consumption per user
7     mapping(address => uint256) public userBills; // Stores the bill amount
    per user
8
9     uint256 public constant RATE_PER_KWH = 2; // Cost per kWh (Example: 2
    Wei per kWh)
10
11     event EnergyStored(address indexed user, uint256 energy);
12     event BillPaid(address indexed user, uint256 amount);
13
14     modifier onlyOwner() {
15         require(msg.sender == owner, "Only owner can call this function");
```



```

16         -;
17     }
18
19     constructor() {
20         owner = msg.sender; // Set contract deployer as owner
21     }
22
23     function storeEnergy(uint256 _totalEnergy) public {
24         require(_totalEnergy > 0, "Energy must be greater than zero");
25
26         userEnergy[msg.sender] += _totalEnergy;
27         userBills[msg.sender] = userEnergy[msg.sender] * RATE_PER_KWH;
28
29         emit EnergyStored(msg.sender, _totalEnergy);
30     }
31
32     function getBill() public view returns (uint256) {
33         return userBills[msg.sender];
34     }
35
36     function payBill() public payable {
37         uint256 billAmount = userBills[msg.sender];
38         require(msg.value == billAmount, "Incorrect payment amount");
39
40         userBills[msg.sender] = 0; // Reset bill after payment
41
42         emit BillPaid(msg.sender, msg.value);
43     }
44 }
45

```

This contract allows users to store their energy consumption, calculate their bill, and make payments. It ensures security and automation of energy billing transactions.

## 7.5 Ethereum and Gas Estimation

Ethereum is a decentralized platform for smart contract execution. It uses Ether (ETH), the network's currency. To execute a smart contract, users must pay gas fees, which compensate miners for their computational efforts. Gas costs vary depending on network congestion and

contract complexity.

Each operation in Solidity has a distinct gas cost. For example, storing data on-chain is more expensive than performing computations. Gas estimation helps developers optimize contracts by identifying operations that consume excessive gas. Tools such as Remix IDE and Ethereum testnets provide gas estimation functionality to improve contract efficiency. A visual representation of gas estimation in Remix IDE is shown in Figure 7.1.

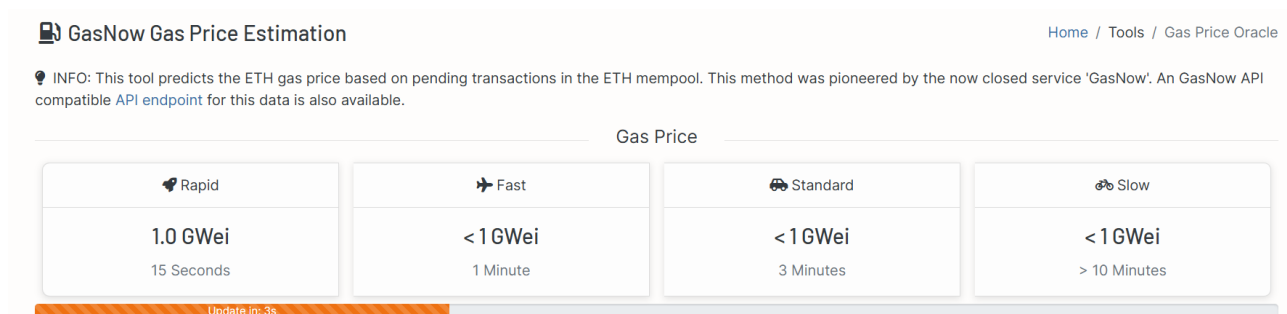


Figure 7.1: Gas Tracker for Ethereum Sepolia

## 7.6 Remix IDE and Smart Contract Deployment

Remix IDE is an online development environment for creating, building, deploying, and debugging smart contracts. It features a user-friendly interface and built-in Solidity compiler support, allowing developers to write and test contracts efficiently.

Remix also supports gas estimation, transaction debugging, and integration with Ethereum testnets. Developers can use Remix to deploy smart contracts, simulate transactions, and optimize gas usage before deploying them to the Ethereum mainnet. The deployment process in Remix IDE is illustrated in Figure 7.2.

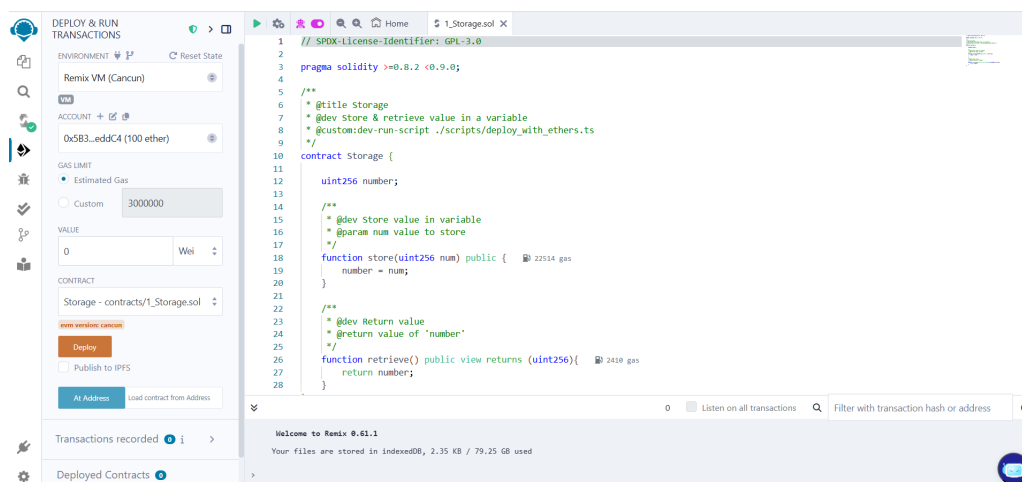


Figure 7.2: Smart contract deployment in Remix IDE

## 7.7 Conclusion

Blockchain and smart contracts offer a secure and decentralized method to automation across a variety of industries, including energy bills. Solidity allows for efficient contract construction, while Ethereum provides decentralized execution using gas estimation algorithms. Tools like Remix IDE make smart contract deployment easier, and combining IoT and blockchain improves automation and transparency in energy management. The combination of these technologies provides the door for a more efficient and secure system for handling digital transactions and automation.

# Chapter 8

## Building the Web3 Application

### 8.1 Introduction

The Internet has grown from a basic information-sharing platform to a global network that connects billions of people[40][41][42]. Web3 is emerging as the next phase, using blockchain technology to improve security, privacy, and user control[43][44].

Unlike previous centralized systems, Web3 allows for direct peer-to-peer connections, decreasing dependency on middlemen. It enables innovations such as decentralized apps (dApps), decentralized finance (DeFi), NFTs, and DAOs, which alter businesses[45].

Despite its potential, Web3 confronts obstacles such as scalability, restrictions, and environmental effect that must be addressed before widespread implementation[46][47]. This assessment examines Web3's accomplishments, prospects, and challenges, providing insight into its future.

### 8.2 Project Setup

We start by setting up a React project using Vite and installing the necessary dependencies:

```
1 npm create vite@latest web3-app --template react
2 cd web3-app
3 npm install
4 npm install web3 dotenv
5
```

## 8.3 Connecting to Blockchain

We use Web3.js to interact with a smart contract deployed on Ganache. The connection is initialized using environment variables:

```
1 import Web3 from "web3";
2
3 const web3 = new Web3(import.meta.env.VITE_GANACHE_RPC);
4 const contractAddress = import.meta.env.VITE_CONTRACT_ADDRESS;
5 const privateKey = import.meta.env.VITE_PRIVATE_KEY;
6 const account = import.meta.env.VITE_ACCOUNT;
7
```

## 8.4 Fetching Energy Data

To get energy consumption data, we retrieve it from ThingsBoard using an API call:

```
1 async function fetchEnergyData() {
2     const response = await fetch(
3         'http://localhost:9090/api/plugins/telemetry/DEVICE/${DEVICE_ID}/
4         values/timeseries?keys=power',
5         { headers: { "X-Authorization": 'Bearer ${thingsboardToken}' } }
6     );
7     const data = await response.json();
8     const power = data.power[0].value;
9     setEnergy(Math.floor(power));
10 }
```

## 8.5 Interacting with the Smart Contract

Our smart contract supports three main functions: storing energy data, fetching the bill, and making payments.

### 8.5.1 Storing Energy Data

The energy data is sent to the blockchain using the 'storeEnergy' function:

```
1 async function sendEnergyData() {
```

```

2    const tx = contract.methods.storeEnergy(energy);
3    const gas = await tx.estimateGas({ from: account });
4    const gasPrice = await web3.eth.getGasPrice();
5    const data = tx.encodeABI();
6    const nonce = await web3.eth.getTransactionCount(account, "latest");

7
8    const signedTx = await web3.eth.accounts.signTransaction(
9        { to: contractAddress, data, gas, gasPrice, nonce },
10       privateKey
11   );

12
13   await web3.eth.sendSignedTransaction(signedTx.rawTransaction);
14 }
15

```

## 8.5.2 Fetching and Paying Bills

To check the pending bill amount:

```

1  async function fetchBill() {
2      const billAmount = await contract.methods.getBill().call({ from:
3      account });
4      setBill(billAmount);
5  }

```

To pay the bill using the ‘payBill’ function:

```

1  async function payBill() {
2      const billAmount = await contract.methods.getBill().call({ from:
3      account });
4      const tx = contract.methods.payBill();
5      const gas = await tx.estimateGas({ from: account, value: billAmount });
6      const gasPrice = await web3.eth.getGasPrice();
7      const data = tx.encodeABI();
8      const nonce = await web3.eth.getTransactionCount(account, "latest");

9
10     const signedTx = await web3.eth.accounts.signTransaction(
11         { to: contractAddress, data, gas, gasPrice, nonce, value:
12         billAmount },
13         privateKey
14     );

```

```

13
14     await web3.eth.sendSignedTransaction(signedTx.rawTransaction);
15 }
16

```

## 8.6 Frontend Interface

The React frontend provides buttons to interact with the smart contract:

```

1  return (
2    <div className="container">
3      <h1>Energy Billing System</h1>
4      <p>Energy: {energy !== null ? `${energy} kWh` : "Fetching..."}</p>
5      <p>Bill: {bill !== null ? `${bill} wei` : "Not Fetched"}</p>
6
7      <button onClick={sendEnergyData}>Store Energy</button>
8      <button onClick={fetchBill}>Fetch Bill</button>
9      <button onClick={payBill}>Pay Bill</button>
10   </div>
11 );
12

```

The frontend allows seamless interaction with the smart contract, enabling users to track their energy consumption and settle their bills efficiently. The main interface of the React application is shown in Figure 8.1.

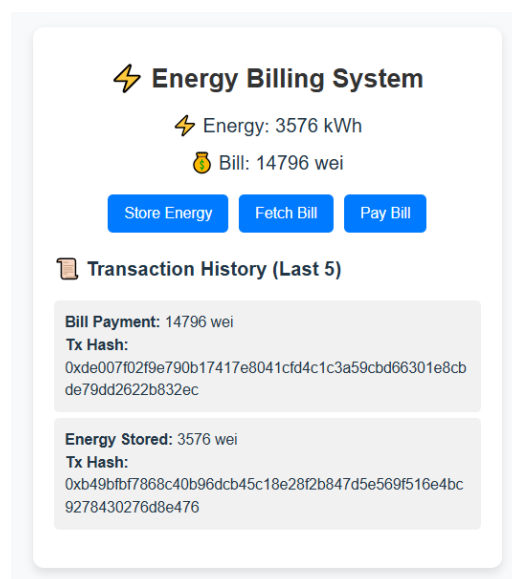


Figure 8.1: User interface of the Energy Billing System

## 8.7 Transaction Logs and User Interaction

To verify the proper execution of smart contract functions, transaction logs from the frontend interactions were analyzed. When a user stores energy data, retrieves the bill, or makes a payment, corresponding transactions are recorded on the blockchain. Figure 8.2 presents a transaction log demonstrating these interactions.

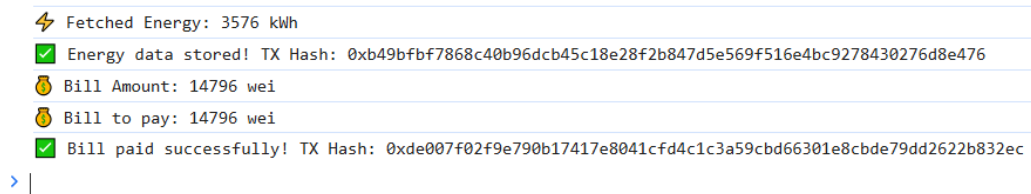


Figure 8.2: Transaction log from frontend interactions with the smart contract

By integrating React with Web3.js, users can seamlessly communicate with the Ethereum blockchain, ensuring a smooth and responsive experience.

## 8.8 Conclusion

This Web3 application successfully integrates React, Ganache, and ThingsBoard to enable decentralized energy billing. It demonstrates the potential of blockchain for transparent and automated energy management.



# Chapter 9

## Conclusion and Future Scope

This study highlights how combining IoT and smart energy management with home automation may revolutionize energy use, monitoring, and billing. The system uses real-time data and automation to assure transparency, efficiency, and accuracy in energy monitoring and invoicing, making energy management more streamlined and cost-effective.

Building on this foundation, the suggested smart home automation system can be greatly improved by using new technology. One interesting possibility is the integration of smart grid capabilities, which allows households to exchange excess solar energy within a local network. This peer-to-peer energy sharing approach encourages sustainable living while increasing overall energy efficiency.

Additional advances include the implementation of demand-response systems, which alter energy usage based on real-time grid circumstances, resulting in improved load control and less energy waste. Furthermore, combining solar energy and smart water pumping technologies can help to create a more self-sufficient household ecosystem. Solar panel monitoring would enable precise assessment of energy generation and automated distribution to home appliances, while intelligent water pumps could regulate consumption based on supply and demand.

Another area for improvement is updating the hardware infrastructure. Switching to more powerful microcontrollers, such as the ESP32 or other IoT-enabled devices, can greatly improve performance. These devices outperform the Raspberry Pi in terms of computing power, energy consumption, and built-in connection, making them ideal for real-time and scalable IoT applications.

# References

- [1] S. Jain, A. Vaibhav, and L. Goyal, “Raspberry pi based interactive home automation system through e-mail,” in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 277–280.
- [2] S. Chaudhari, P. Rathod, A. Shaikh, D. Vora, and J. Ahir, “Smart energy meter using arduino and gsm,” in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE, 2017, pp. 598–601.
- [3] V. Patchava, H. B. Kandala, and P. R. Babu, “A smart home automation technique with raspberry pi using iot,” in *2015 International conference on smart sensors and systems (IC-SSS)*. IEEE, 2015, pp. 1–4.
- [4] R. A. Atmoko, R. Riantini, and M. K. Hasin, “Iot real time data acquisition using mqtt protocol,” *Journal of Physics: Conference Series*, vol. 853, no. 1, p. 012003, may 2017. [Online]. Available: <https://dx.doi.org/10.1088/1742-6596/853/1/012003>
- [5] B. D. Julies and T. Zuva, “A review on internet of things smart homes, challenges, open issues and countermeasures,” in *Software Engineering Perspectives in Intelligent Systems*, R. Silhavy, P. Silhavy, and Z. Prokopova, Eds. Cham: Springer International Publishing, 2020, pp. 1073–1089.
- [6] B. Zhang and Y. Zhang, “The current situation and future challenges of smart home,” 2022.
- [7] D. A. and, “Anthropomorphizing artificial intelligence: towards a user-centered approach for addressing the challenges of over-automation and design understandability in smart homes,” *Intelligent Buildings International*, vol. 13, no. 4, pp. 227–240, 2021. [Online]. Available: <https://doi.org/10.1080/17508975.2020.1795612>

- [8] V. A, “Iot-based smart home automation systems: Enhancing energy efficiency and security,” *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 12, p. 2243–2253, Dec 2024.
- [9] A. Mazid, S. Kirmani, and Manaullah, “Iot enabled framework for smart home automation using artificial intelligence and blockchain technology,” in *International Conference on Artificial Intelligence of Things*. Springer, 2023, pp. 357–367.
- [10] Y. Li, A. M. Mandalari, and I. Straw, “Who let the smart toaster hack the house? an investigation into the security vulnerabilities of consumer iot devices,” *arXiv preprint arXiv:2306.09017*, 2023.
- [11] C. C. Sobin, “A survey on architecture, protocols and challenges in iot,” *Wireless Personal Communications*, vol. 112, no. 3, p. 1383–1429, Jan 2020.
- [12] I. Holguin and S. Errapotu, “Smart home iot communication protocols and advances in their security and interoperability,” *Proceedings of the International Conference on Cyber Security and Networking (CSNet)*, pp. 208–211, October 2023.
- [13] C. Portalés, S. Casas, and K. Kreuzer, “Challenges and trends in home automation: Addressing the interoperability problem with the open-source platform openhab,” in *Harnessing the Internet of Everything (IoE) for Accelerated Innovation Opportunities*. IGI Global, 2019, pp. 148–174. [Online]. Available: <https://www.igi-global.com/chapter/challenges-and-trends-in-home-automation/291662>
- [14] I. V. Sita and B. David, “Development and implementation of a comprehensive smart home automation system using home assistant and iot devices,” in *2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 2024, pp. 1–7.
- [15] X. Bai, “Research on smart home system design and user experience improvement strategies,” *Advances in Computer, Signals and Systems*, vol. 8, pp. 34–39, 2024. [Online]. Available: <https://www.clausiuspress.com/article/13197.html>
- [16] Y. A. Elmi, “Interoperable iot devices and systems for smart homes: A data analytics approach to enhance user experience and energy efficiency,” *Journal of Digitainability, Realism & Mastery (DREAM)*, vol. 2, no. 10, pp. 51–66, October 2023. [Online]. Available: <https://dreamjournal.my/index.php/DREAM/article/view/195>

- [17] S. Mare, L. Girvin, F. Roesner, and T. Kohno, "Consumer smart homes: Where we are and where we need to go," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19)*. Santa Cruz, CA, USA: ACM, Feb. 2019, pp. 117–122. [Online]. Available: <https://dl.acm.org/doi/10.1145/3301293.3302371>
- [18] G. Kaur, A. Kaur, N. Sharma, M. Singh, and K. Mittal, "Evolution of ai in smart home systems: A comprehensive review," *CGC International Journal of Contemporary Technology*, vol. 6, no. 2, pp. 388–395, October 2024. [Online]. Available: [https://cgcijctr.com/download.php?file=C20240401\\_Evolution+of+AI+in+Smart+Home+Systems+A+Comprehensive+Review\\_Final.pdf](https://cgcijctr.com/download.php?file=C20240401_Evolution+of+AI+in+Smart+Home+Systems+A+Comprehensive+Review_Final.pdf)
- [19] M. Abdelhamid and G. Hassan, "Blockchain and smart contracts," in *Proceedings of the 8th International Conference on Software and Information Engineering*, ser. ICSIE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 91–95. [Online]. Available: <https://doi.org/10.1145/3328833.3328857>
- [20] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. P. Seneviratne, "Blockchain-based smart contracts - applications and challenges," *arXiv: Computers and Society*, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:182952419>
- [21] N. Valov and I. Valova, "Home automation system with raspberry pi," in *2020 7th International Conference on Energy Efficiency and Agricultural Engineering (EE&AE)*. IEEE, 2020, pp. 1–5.
- [22] L. Li, Y. Chen, J. Liu *et al.*, "The application of hall sensors acs712 in the protection circuit of controller for humanoid robots," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 12. IEEE, 2010, pp. V12–101.
- [23] U. Khair, A. J. Lubis, I. Agustha, Dharmawati, and M. Zulfin, "Modeling and simulation of electrical prevention system using arduino uno,gsm modem, and acs712 current sensor," *Journal of Physics: Conference Series*, vol. 930, no. 1, p. 012049, dec 2017. [Online]. Available: <https://dx.doi.org/10.1088/1742-6596/930/1/012049>
- [24] C. Dout, "2.7v 4-channel/8-channel 12-bit a/d converters with spi ® serial interface," 1999.
- [25] G. Flurry, *An Analog-to-Digital Converter*. Berkeley, CA: Apress, 2021, pp. 397–412. [Online]. Available: [https://doi.org/10.1007/978-1-4842-7264-0\\_12](https://doi.org/10.1007/978-1-4842-7264-0_12)

- [26] L. Müller, M. Mohammed, and J. W. Kimball, “Using the arduino uno to teach digital control of power electronics,” in *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*. IEEE, 2015, pp. 1–8.
- [27] A. J. Taufiq, I. H. Kurniawan, and T. A. Y. Nugraha, “Analysis of arduino uno application on control system based on industrial scale,” *IOP Conference Series: Materials Science and Engineering*, vol. 771, no. 1, p. 012015, mar 2020. [Online]. Available: <https://dx.doi.org/10.1088/1757-899X/771/1/012015>
- [28] S. Naimi, S. Naimi, and M. A. Mazidi, “The avr microcontroller and embedded systems using assembly and c: Using arduino uno and atmel studio,” 2017.
- [29] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, “Next century challenges: Scalable coordination in sensor networks,” in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 263–270.
- [30] B. Hammi, R. Khatoun, S. Zeadally, A. Fayad, and L. Khoukhi, “‘Iot technologies;? show [aq id= q1]?¿ for smart cities,” *IET networks*, vol. 7, no. 1, pp. 1–13, 2018.
- [31] V. Gazis, M. Görtz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis, “A survey of technologies for the internet of things,” in *2015 international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2015, pp. 1090–1095.
- [32] Z. Kegenbekov and A. Saparova, “Using the mqtt protocol to transmit vehicle telemetry data,” *Transportation Research Procedia*, vol. 61, pp. 410–417, 2022.
- [33] Z. Shelby, K. Hartke, and C. Bormann, “Rfc 7252: The constrained application protocol (coap),” 2014.
- [34] M. B. Yassein, M. Q. Shatnawi *et al.*, “Application layer protocols for the internet of things: A survey,” in *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–4.
- [35] M. Swan, “Blockchain temporality: Smart contract time specifiability with blocktime,” in *Rule Technologies. Research, Tools, and Applications: 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings 10*. Springer, 2016, pp. 184–196.

- [36] S. Tern, “Survey of smart contract technology and application based on blockchain,” *Open Journal of Applied Sciences*, vol. 11, no. 10, pp. 1135–1148, 2021.
- [37] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie, “Shadoweth: Private smart contract on public blockchain,” *Journal of Computer Science and Technology*, vol. 33, pp. 542–556, 2018.
- [38] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov, “Smart contract invocation protocol (scip): A protocol for the uniform integration of heterogeneous blockchain smart contracts,” in *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32*. Springer, 2020, pp. 134–149.
- [39] R. Johari, K. Gupta, and A. S. Parihar, “Smart contracts in smart cities: Application of blockchain technology,” in *Innovations in Information and Communication Technologies (IICT-2020) Proceedings of International Conference on ICRIHE-2020, Delhi, India: IICT-2020*. Springer, 2021, pp. 359–367.
- [40] R. Aria, N. Archer, M. Khanlari, and B. Shah, “Influential factors in the design and development of a sustainable web3/metaverse and its applications,” *Future Internet*, vol. 15, no. 4, p. 131, 2023.
- [41] K. Nabben, “Web3 as ‘self-infrastructuring’: The challenge is how,” *Big Data & Society*, vol. 10, no. 1, p. 20539517231159002, 2023.
- [42] A. Murray, D. Kim, and J. Combs, “The promise of a decentralized internet: What is web3 and how can firms prepare?” *Business horizons*, vol. 66, no. 2, pp. 191–202, 2023.
- [43] D. Tennakoon, Y. Hua, and V. Gramoli, “Smart redbelly blockchain: Reducing congestion for web3,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 940–950.
- [44] J. Sadowski and K. Beegle, “Expansive and extractive networks of web3,” *Big Data & Society*, vol. 10, no. 1, p. 20539517231159629, 2023.
- [45] L. W. Cong, K. Tang, Y. Wang, and X. Zhao, “Inclusion and democratization through web3 and defi? initial evidence from the ethereum ecosystem,” National Bureau of Economic Research Cambridge, MA, USA, Tech. Rep., 2023.

- [46] M. Lacity, E. Carmel, A. G. Young, and T. Roth, “The quiet corner of web3 that means business,” *MIT Sloan Management Review*, vol. 64, no. 3, 2023.
- [47] G. Wang, R. Qin, J. Li, F.-Y. Wang, Y. Gan, and L. Yan, “A novel dao-based parallel enterprise management framework in web3 era,” *IEEE Transactions on Computational Social Systems*, vol. 11, no. 1, pp. 839–848, 2023.