Smart Home Automation

Your Name

March 17, 2025

Abstract

Home automation using Raspberry Pi has gained popularity due to its low cost and capacity to improve convenience and energy efficiency. This concept combines a smart home automation system with blockchain technology to enable safe and transparent energy invoicing. The system gathers energy usage data from a smart energy meter linked to an Arduino and sends it to a cloud-based IoT platform via MQTT and REST APIs for real-time monitoring.This data is retrieved by a Web3 application, which then interacts with an Ethereum smart contract to securely produce and handle energy invoices. Using blockchain technology, the system assures transparency, security, and decentralization in energy transactions. The integration of IoT and smart contracts eliminates intermediaries, lowering operating costs and increasing efficiency. This study exhibits a seamless transition from data collecting to invoicing and payment in a decentralized context, which helps to enhance smart energy management solutions.

Contents

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 automatic meter reading, real-time monitoring, and remote control of electrical connections without the need for personal involvement[2]. The integration of Arduino controllers with GSM modules improves data transmission, allowing power companies to track energy use in kilowatt-hours (kWh) and create billing information[3]. In addition to energy monitoring, home automation systems include sensors, cameras, and web-based applications to increase security and device control[4]. Real-time data gathering and storage in databases, such as MySQL, ensures accurate monitoring and analysis. The use of the MQTT protocol improves data quality and dependability in IoT-based systems[5].

Blockchain and smart contracts have emerged as disruptive technologies with the potential to transform many sectors. Smart contracts are self-executing programs that autonomously enforce contractual conditions without the need for intermediaries, improving efficiency and lowering operational expenses[6]. These technologies offer advantages such as transparency, security, and decentralization, making them appropriate for a wide range of applications, from identity management to business process automation[2].

Home automation systems that combine IoT with blockchain-based smart contracts can improve trust and security in energy management. This project investigates the integration of a smart energy meter with a blockchain-based billing system, using Raspberry Pi, IoT platforms, and Ethereum smart contracts to create a decentralized, transparent, and automated energy billing solution.

Chapter 2

Project Workflow

2.1 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[5].

A Web3 application obtains this information and communicates with an Ethereum smart contract to generate energy bills in a secure and transparent manner[6][7]. By merging IoT and blockchain, the solution automates billing and payments, increasing efficiency and security.

2.2 System Architecture

The architecture consists of multiple interconnected components, including sensors, microcontrollers, cloud platforms, and blockchain technology. The data flow and interactions are illustrated in Figure 2.1.

2.3 Components and Data Flow

2.3.1 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.
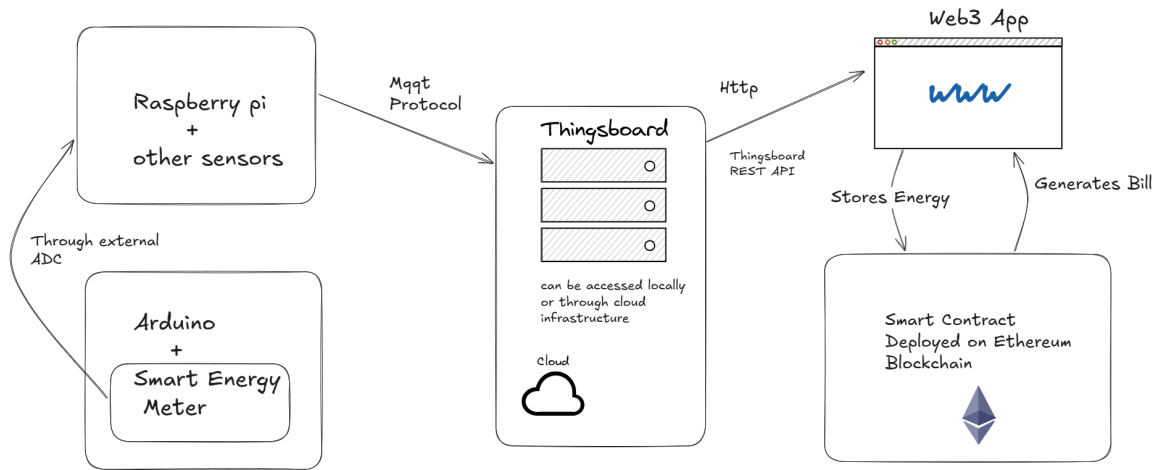
Figure 2.1: System Workflow

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 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 3

Proteus Schematic Design and Simulation

3.1 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[8].

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 [9]. Additionally, power factor measurement is achieved using LM358 operational amplifiers and XOR gates to analyze phase differences between voltage and current waveforms [10]. 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.

3.2 Schematic Design in Proteus

3.2.1 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)[8].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[8].
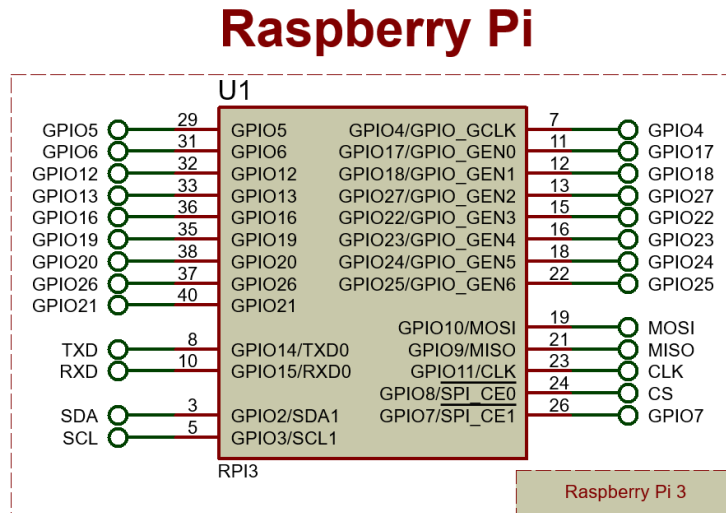
Figure 3.1: Raspberry Pi 3 Schematic

ADC (MCP3208): Analog-to-digital converters (ADCs) transform analog signals into digital data, allowing microcontrollers and computers to process real-world inputs.The MCP3208 is a 12-bit ADC with 8 input channels. It provides high accuracy (±1 LSB DNL, ±1-2 LSB INL) and low power consumption. It runs on 2.7V-5.5V, has an SPI interface, and is commonly used for sensor applications and data gathering[11].
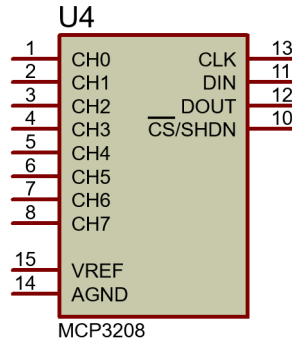
Figure 3.2: ADC (MCP3208)

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[12].

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.

3.2.2 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 ans involves following connections.
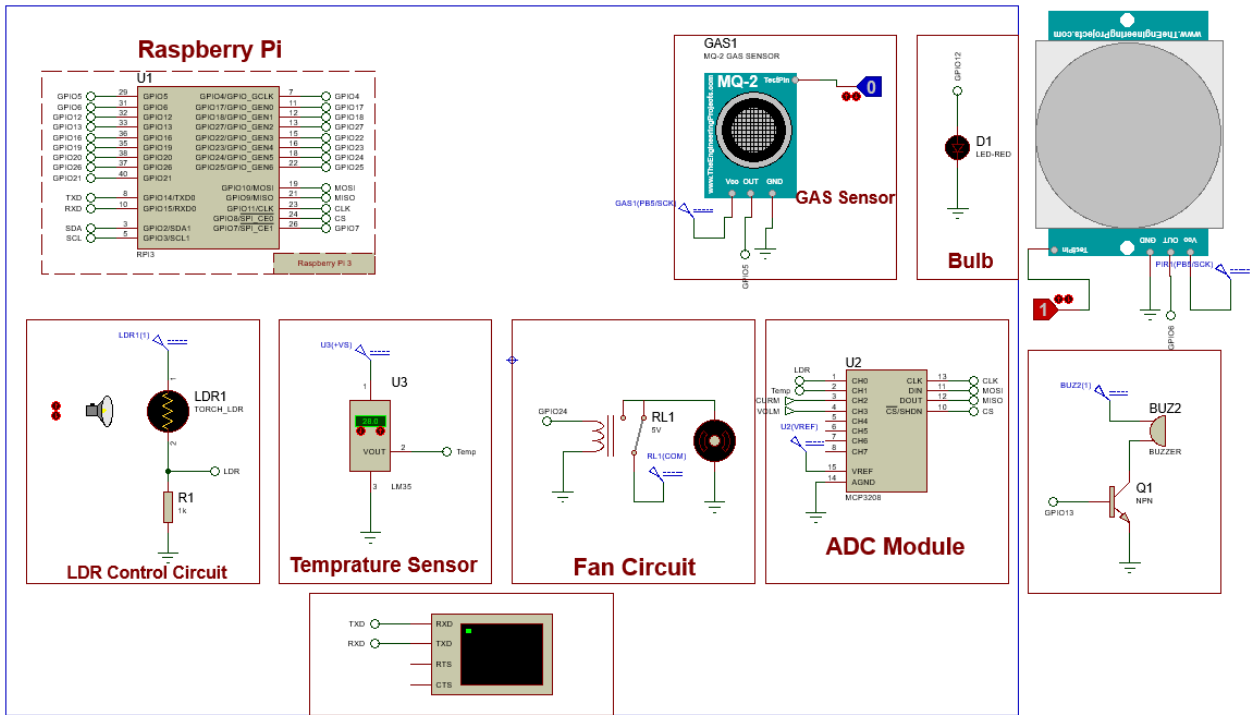
Figure 3.3: Complete Schematic

3.2.3 Smart Energy Meter and Power Factor Measurement

The smart energy meter circuit consists of:

Arduino Uno: The Arduino Uno, built around the AVR microcontroller, has developed as a versatile platform for teaching and implementing digital control systems.
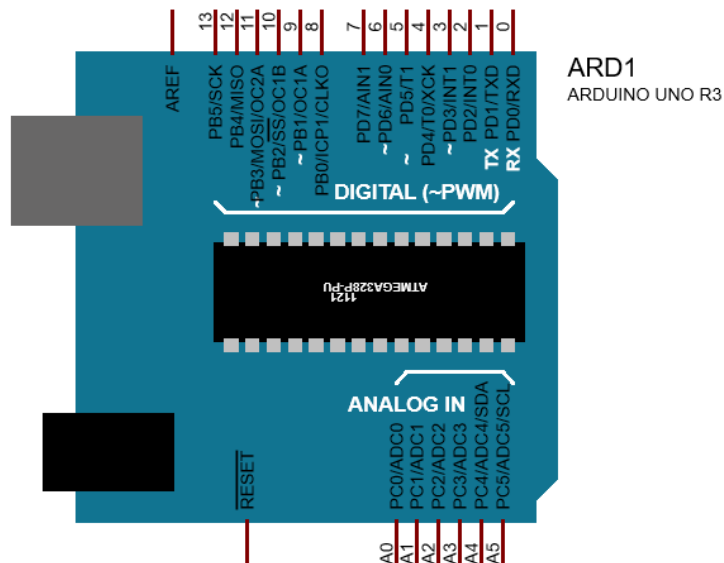
Figure 3.4: Arduino Uno

It has characteristics suited for power electronics applications, providing switching frequencies

of up to 100 kHz with extra libraries[13]. The Arduino Uno has been shown to be effective in controlling industrial-scale instruments, performing similarly to industrial-class controllers that use PID algorithms[14].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[15].

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[9]. 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 µs) and low noise improve performance for smart energy management[10].
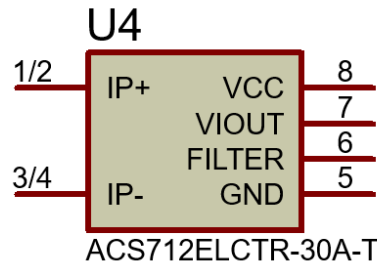
Figure 3.5: Current Sensor

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 500µF 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.
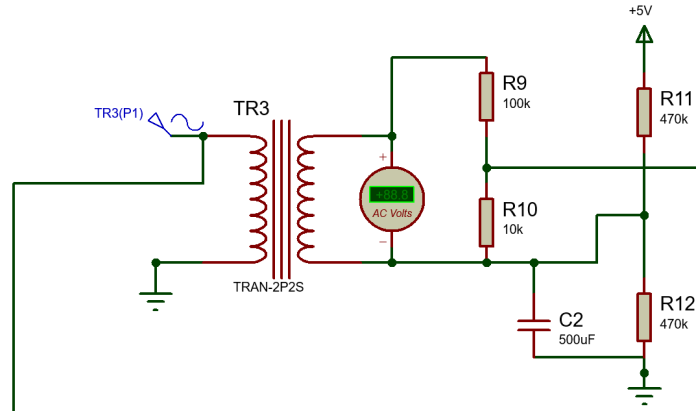
Figure 3.6: Voltage Measurement Circuit

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 θ is the phase angle.
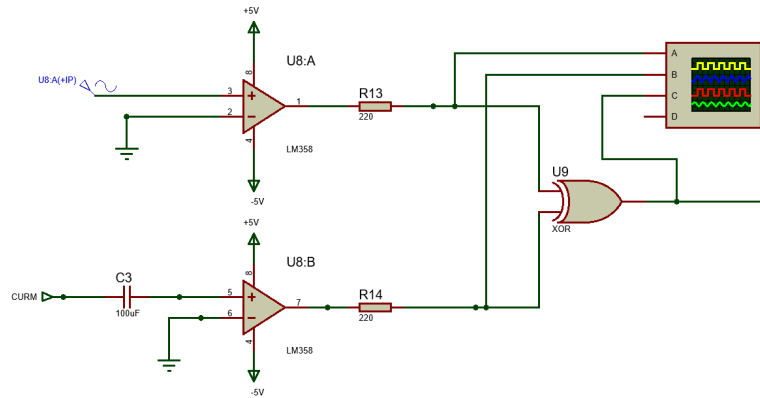
Figure 3.7: PowerFactor Measurement Circuit

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).

3.2.4 Complete Schematic of Smart Energy Meter

The smart energy meter circuit is designed to measure and monitor power consumption while ensuring efficient energy usage.
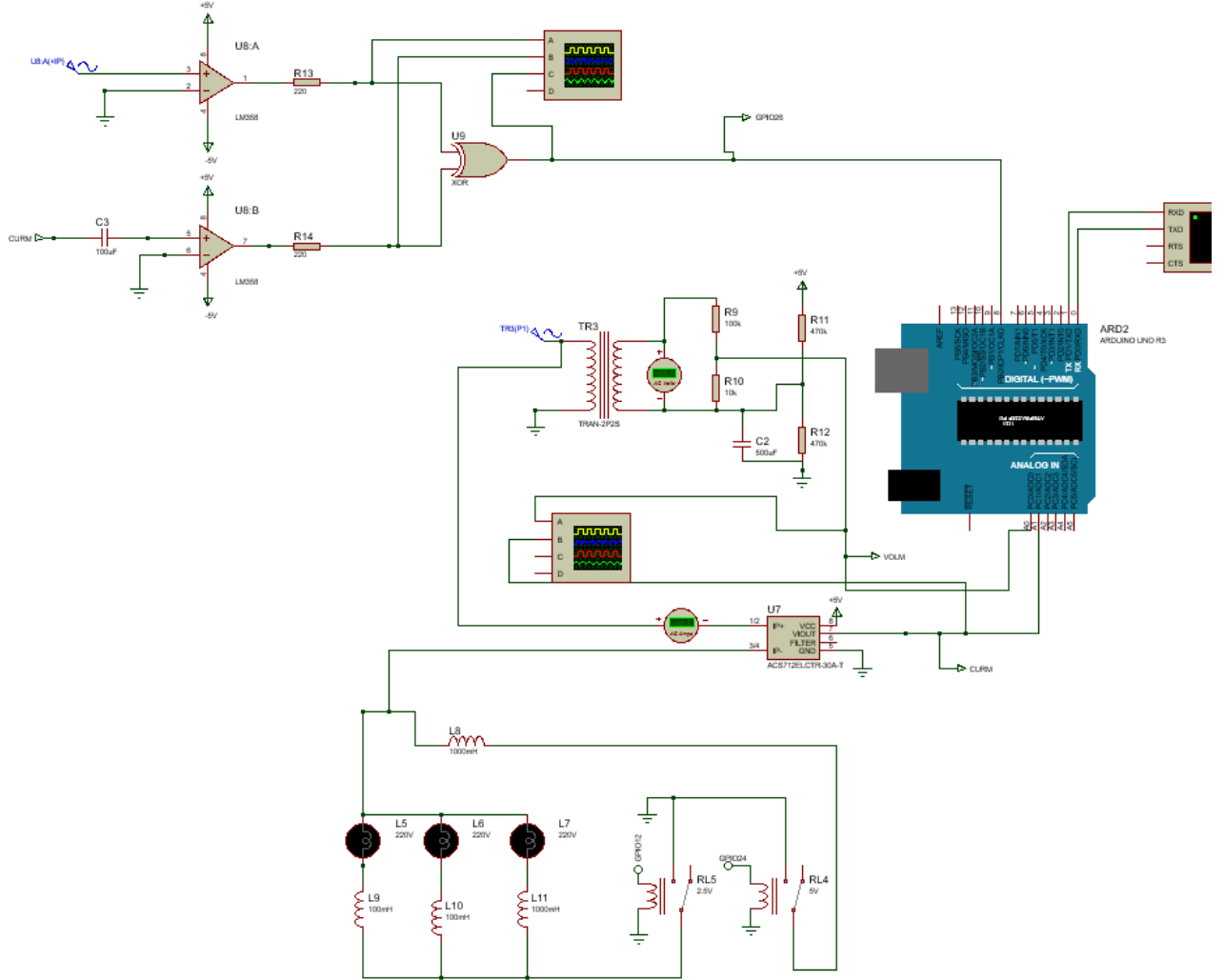
Figure 3.8: Complete Schematic of Smart Energy meter

3.3 Code Implementation

3.3.1 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.

```cpp
// Pin configuration
const int voltagePin = A0; // Voltage sensor input
const int currentPin = A1; // Current sensor input

// Constants for sensor calibration
const float voltageMultiplier = 230.0 / 1023.0;  // Adjust based on sensor
    specs
const float currentMultiplier = 10.0 / 1023.0;   // Adjust for CT sensor
    scaling

void setup() {
    Serial.begin(9600);  // Start serial communication
}

void loop() {
    // Read raw analog values from sensors
    int rawVoltage = analogRead(voltagePin);
    int rawCurrent = analogRead(currentPin);

    // Convert raw values to real-world measurements
    float voltage = rawVoltage * voltageMultiplier;
    float current = rawCurrent * currentMultiplier;
    float power = voltage * current; // Active power calculation

    // Print the measurements to serial monitor
    Serial.print("Voltage: "); Serial.print(voltage); Serial.print(" V, ");
    Serial.print("Current: "); Serial.print(current); Serial.print(" A, ");
    Serial.print("Power: "); Serial.print(power); Serial.println(" W");

    delay(1000);  // Wait 1 second before next reading
}
```

3.3.2 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.

```python
import RPi.GPIO as GPIO
import time
import json
import spidev
import math
from paho.mqtt.client import Client

# GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)  # LDR sensor for light detection
GPIO.setup(27, GPIO.OUT) # Relay control for light switching

# SPI setup for ADC communication (e.g., MCP3008)
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1350000

# MQTT setup
THINGSBOARD_HOST = "localhost"
ACCESS_TOKEN = "your_access_token"
client = Client()
client.username_pw_set(ACCESS_TOKEN)
client.connect(THINGSBOARD_HOST, 1883, 60)

# Read ADC channel for sensor values
def read_channel(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    return ((adc[1] & 3) << 8) + adc[2]

# Calculate RMS voltage from multiple samples
def calculate_rms_voltage():
    sum_squares = sum((read_channel(3) * 3.3 / 1023.0) ** 2 for _ in range
    (200))
    return math.sqrt(sum_squares / 200) * 11  # Voltage divider correction

# Publish sensor data to MQTT
def publish_sensor_data():
    while True:
        light_level = read_channel(0)  # Read LDR sensor value
        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 ()
```

3.4 Simulation and Testing

3.4.1 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.

3.4.2 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.

3.5 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 4

ThingsBoard Integration

4.1 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[16]. 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[17][18]. 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[19], CoAP[20], and HTTP[21]. 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[19]. In contrast, CoAP is an UDP-based protocol designed for limited contexts, with lower overhead but lesser dependability than MQTT[20]. 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[17].

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

19

alert users or initiate automated replies. Furthermore, the platform supports both lightweight communication protocols, such as MQTT and CoAP, and classic RESTful services[20].
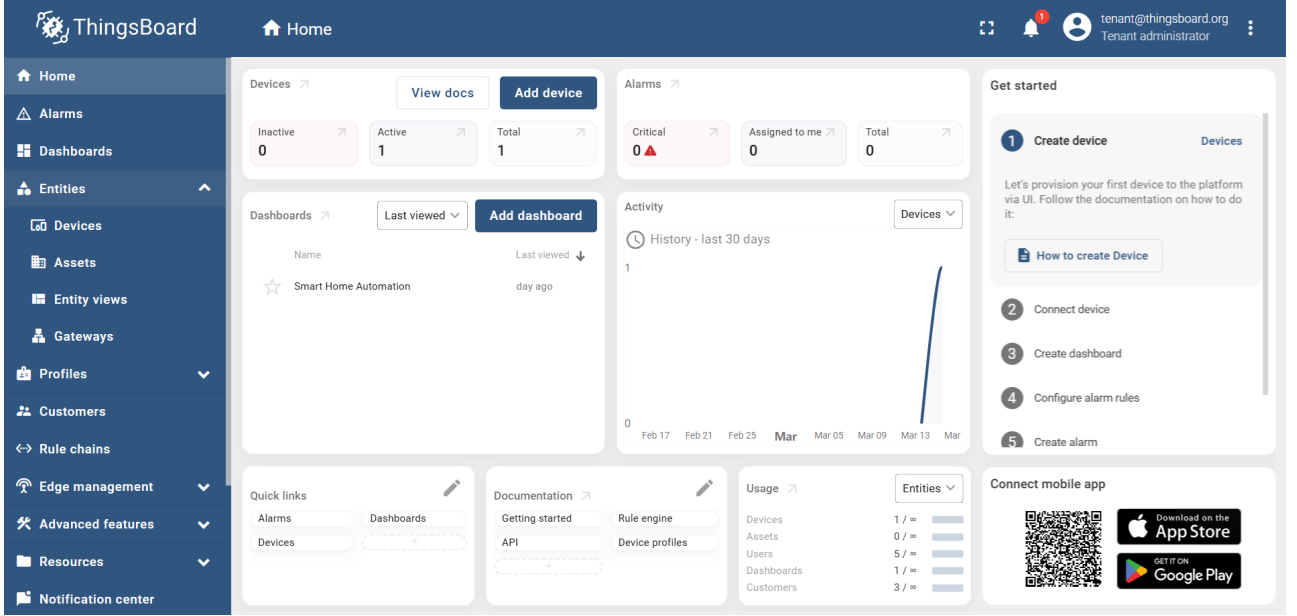
Figure 4.1: ThingsBoard's Home

4.2 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 4.2. 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 4.2: Device creation process in ThingsBoard

4.3 Connecting ThingsBoard with MQTT

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

4.3.1 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.

```
THINGSBOARD_HOST = "localhost"
ACCESS_TOKEN = "dU6S0YIAPX5WwfmB3wUi"  # Replace with your actual token
MQTT_PORT = 1883
```

4.3.2 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.

```
import paho.mqtt.client as mqtt
import json
```

4.3.3 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.

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

```

4.3.4 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.

```
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}
client.publish("v1/devices/me/telemetry", json.dumps(telemetry_data))
print("Data sent successfully!")
client.disconnect()

```

Figure 4.3: Latest Telemetry after Publishing Data

4.3.5 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.

4.3.6 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 4.4.
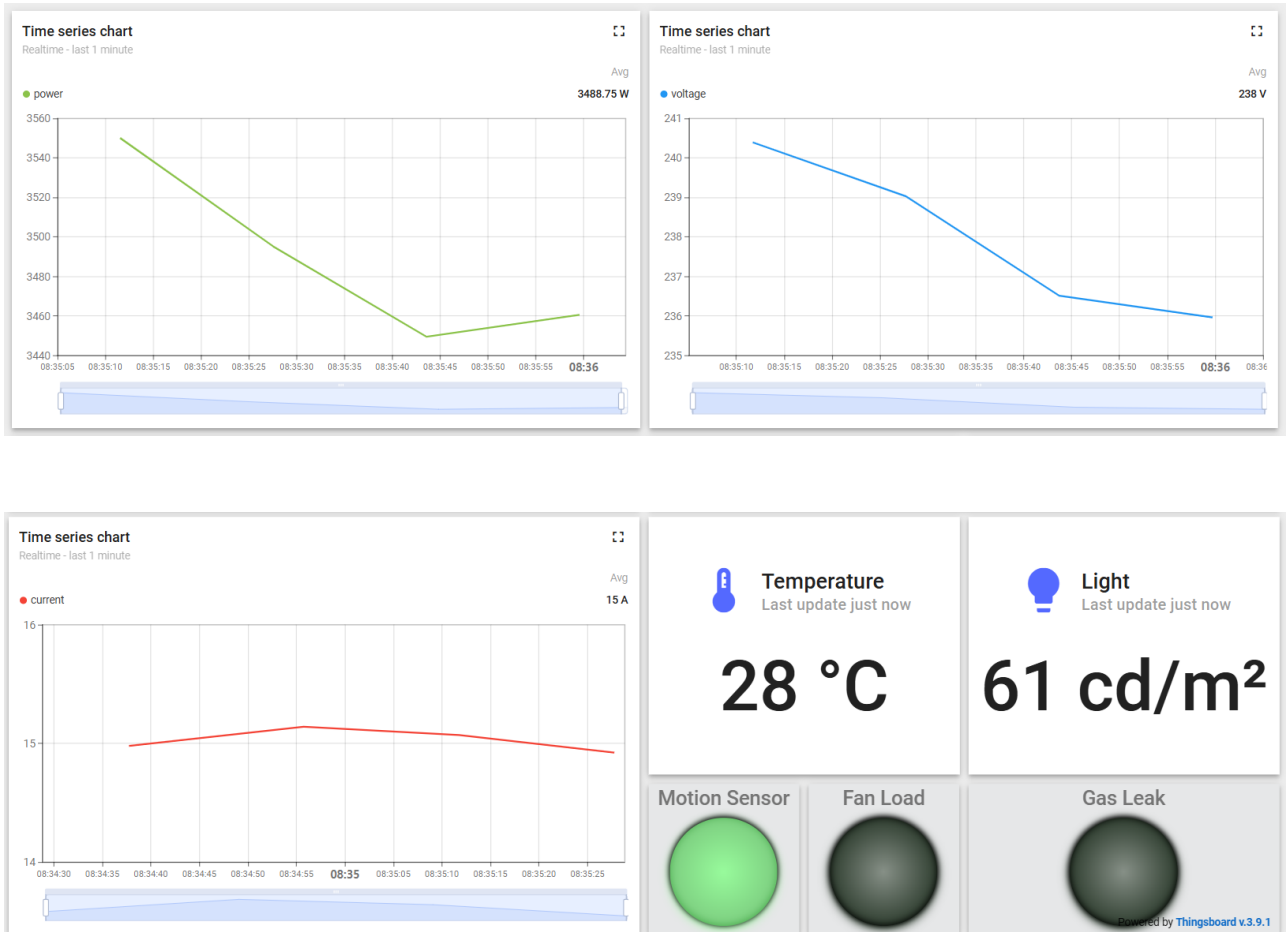
Figure 4.4: Dashboard Creation with ThingsBoard Widgets

4.4 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.

4.5 Connecting to ThingsBoard API

To fetch telemetry data from ThingsBoard, follow these steps:

4.5.1   Step 1: Generate Access Token

The first step is to authenticate with ThingsBoard and obtain an access token. This is done by sending a POST request to the authentication API with valid credentials. The request should be formatted in JSON, containing a username and password.

```
POST http://localhost:9090/api/auth/login
Content-Type: application/json


{
    "username": "tenant@thingsboard.org",
    "password": "tenant"
}

```

4.5.2   Step 2: Extract JWT Token

Once the authentication request is processed, the API will return a response containing a JWT token. This token is essential for making authorized requests to ThingsBoard. The response will be in JSON format, and the token can be extracted from the `token` field.

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsIn..."
}

```

4.5.3   Step 3: Fetch Telemetry Data

After obtaining the authentication token, the next step is to retrieve telemetry data from a specific device. The device ID must be specified in the request URL, and the JWT token must be included in the request headers under the `X-Authorization` field. The following JavaScript code demonstrates how to fetch power telemetry data from a ThingsBoard device.

```
const response = await fetch(
    `http://localhost:9090/api/plugins/telemetry/DEVICE/0eb8ae80-ffe6-11ef-
    b36a-71656502eb9c/values/timeseries?keys=power`,
    {
        headers: { "X-Authorization": `Bearer ${thingsboardToken}` },
    }
```

```
6   );
7   const data = await response.json();
8   console.log(data);
9
```

4.5.4   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.

4.6   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 5

Blockchain and Smart Contracts

5.1 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[22]. Transactions are encrypted with cryptographic keys, validated by distributed nodes, and kept immutably, making blockchain immune to fraud and illegal changes[23]. 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[24]. These self-executing contracts specify the terms and circumstances of a transaction while ensuring efficiency, security, and dependability[25].

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[26]. Blockchain and smart contracts are becoming increasingly popular, opening up new opportunities for safe and efficient digital transactions.

5.2 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.

5.3   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.

5.4   Smart Contract Deployed in the Project

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

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EnergyBilling {
    address public owner;
    mapping(address => uint256) public userEnergy; // Stores energy
  consumption per user
    mapping(address => uint256) public userBills; // Stores the bill amount
     per user

    uint256 public constant RATE_PER_KWH = 2; // Cost per kWh (Example: 2
  Wei per kWh)

    event EnergyStored(address indexed user, uint256 energy);
    event BillPaid(address indexed user, uint256 amount);

    modifier onlyOwner() {
        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.

5.5   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 5.1.
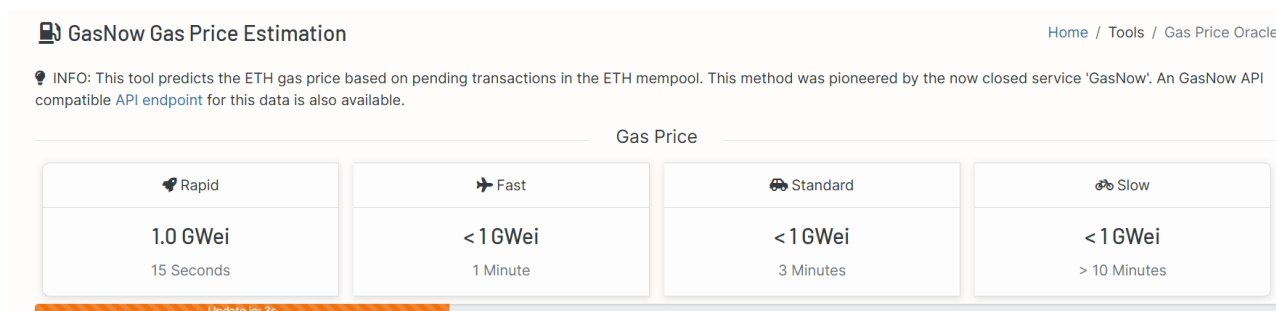
Figure 5.1: Gas Tracker for Ethereum Sepolia

5.6   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 5.2.
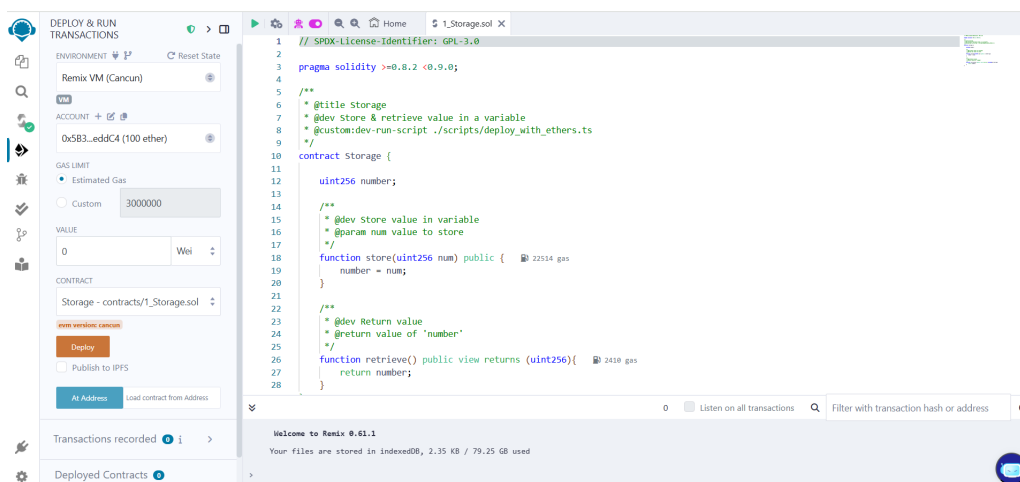
Figure 5.2: Smart contract deployment in Remix IDE

5.7   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 6

Ganache Toolchain

6.1   Introduction to Ganache

Ganache is a personal Ethereum blockchain designed for testing and development. It offers developers a local blockchain environment in which to build, test, and debug smart contracts before distributing them to a public or test network[27]. Ganache is offered in two flavors: Ganache UI, which has a graphical interface, and Ganache CLI, a command-line tool for automation and scripting[28]. It supports quick transactions, chain forking, and complete logging, making it a crucial tool for Ethereum development and testing[29].

One of the primary benefits of Ganache is its ability to mine transactions instantly. Unlike public testnets, which require confirmation time, Ganache handles transactions immediately, allowing for faster iterations during development. It also offers pre-funded accounts, making it simple to test transactions without spending real or test ETH. Furthermore, Ganache enables developers to change network characteristics like as gas price, block time, and transaction speed, providing greater flexibility for testing various scenarios.

6.2   Ethereum Testnets and Their Challenges

Ethereum testnets, such as Sepolia, Goerli, and Holesky, allow developers to test their smart contracts before releasing them to the Ethereum mainnet. These testnets imitate real-world Ethereum settings, such as transaction processing, block validation, and gas fees, without incurring any monetary charges.

Sepolia is now the most popular testnet for testing smart contracts. However, one of the most difficult aspects of using Ethereum testnets is obtaining test ETH, which is needed to deploy

contracts and execute transactions. Unlike a local blockchain, such as Ganache, testnets require developers to request ETH from faucets, which are frequently unstable or have limited availability. Furthermore, mining test ETH on proof-of-stake testnets such as Sepolia is impossible, forcing developers to rely on third-party sources.

Another limitation of testnets is the network congestion and transaction confirmation time. Since multiple developers use these networks simultaneously, transactions may take longer to be confirmed, especially during high activity periods. This delays testing and debugging, making it harder to iterate quickly on smart contract development.

6.3 Why Ganache is Beneficial for Development

Ganache offers a speedier, more regulated, and cost-effective alternative to Ethereum testnets. Unlike Sepolia and Goerli, which require real network confirmations, Ganache executes transactions instantly, allowing for faster testing and debugging.

Developers can use Ganache to generate numerous pre-funded accounts, avoiding the need for test ETH. This makes it much easier to test smart contract features like payments, ownership transfers, and gas calculation. Furthermore, because Ganache runs locally, there are no network congestion difficulties, resulting in a smoother development process.

Another key advantage of Ganache is the ability to reset the blockchain state at any time. When testing on a testnet like Sepolia, once a transaction is completed, it is permanently recorded on the blockchain. However, Ganache allows developers to reset the blockchain and start over, making it perfect for frequent testing and debugging.

6.4 Usage of Ganache in the Project

Ganache served as the primary testing environment for the **EnergyBilling** smart contract before deployment to the Ethereum testnet. Since Ethereum smart contracts require gas fees for each transaction, testing on a local blockchain helped reduce unnecessary costs. The project workflow began with launching Ganache,which creates a private Ethereum blockchain.

Several pre-funded accounts were used to simulate user interactions with the contract. The Remix IDE was then connected to Ganache for contract compilation and deployment. This configuration enabled efficient testing of contract functionalities such as energy storage, bill computation, and payment, as illustrated in Figure6.1.

Figure 6.1: Pre-funded accounts in Ganache

One of the key aspects analyzed using Ganache was gas estimation. Every smart contract function consumes gas, and understanding the cost of each action is crucial for optimization. Before deploying the contract to an actual testnet, modifications were made to minimize gas expenses using Ganache's transaction cost analysis, as shown in Figure 6.2.


```
Transaction: 0x9ed3f1b55b919ad81047d577040034205a30c218583d3da3142baf6602335998
Contract created: 0x345ca3e014aaf5dca488057592ee47305d9b3e10
Gas usage: 515687
Block number: 3
Block time: Thu Mar 13 2025 09:23:34 GMT+0000 (Coordinated Universal Time)
```

Figure 6.2: Transaction log in Ganache showing contract creation and function execution

Additionally, Ganache's instant transaction mining allowed for rapid testing iterations. Unlike Sepolia, where transactions take time to confirm, Ganache ensured that tests could be conducted quickly, leading to shorter development cycles. Figure 6.3 illustrates the blockchain view in Ganache, showing the sequence of transactions and blocks mined during testing.

Figure 6.3: Blockchain view in Ganache displaying mined blocks and transactions

6.5 Conclusion

Ganache is essential for Ethereum smart contract development because it provides a rapid, reliable, and cost-free testing environment. While testnets such as Sepolia are required for final testing prior to mainnet deployment, they present issues such as procuring test ETH and coping with network latency. Ganache addresses these concerns by offering a local blockchain with instant transactions, pre-funded accounts, and the option to reset the blockchain state.

Chapter 7

Building the Web3 Application

7.1 Introduction

The Internet has grown from a basic information-sharing platform to a global network that connects billions of people[30][31][32]. Web3 is emerging as the next phase, using blockchain technology to improve security, privacy, and user control[33][34].

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[35].

Despite its potential, Web3 confronts obstacles such as scalability, restrictions, and environmental effect that must be addressed before widespread implementation[36][37]. This assessment examines Web3's accomplishments, prospects, and challenges, providing insight into its future.

7.2 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
```

7.3 Connecting to Blockchain

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

```
import Web3 from "web3";


const web3 = new Web3(import.meta.env.VITE_GANACHE_RPC);
const contractAddress = import.meta.env.VITE_CONTRACT_ADDRESS;
const privateKey = import.meta.env.VITE_PRIVATE_KEY;
const account = import.meta.env.VITE_ACCOUNT;

```

7.4 Fetching Energy Data

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

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

```

7.5 Interacting with the Smart Contract

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

7.5.1 Storing Energy Data

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

```
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
```

7.5.2 Fetching and Paying Bills

To check the pending bill amount:

```
1  async function fetchBill() {

2      const billAmount = await contract.methods.getBill().call({ from:
   account });

3      setBill(billAmount);

4  }

5
```

To pay the bill using the 'payBill' function:

```
1  async function payBill() {

2      const billAmount = await contract.methods.getBill().call({ from:
   account });

3      const tx = contract.methods.payBill();

4      const gas = await tx.estimateGas({ from: account, value: billAmount });

5      const gasPrice = await web3.eth.getGasPrice();

6      const data = tx.encodeABI();

7      const nonce = await web3.eth.getTransactionCount(account, "latest");

8

9      const signedTx = await web3.eth.accounts.signTransaction(

10         { to: contractAddress, data, gas, gasPrice, nonce, value:
   billAmount },

11         privateKey

12     );
```

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

7.6   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 7.1.
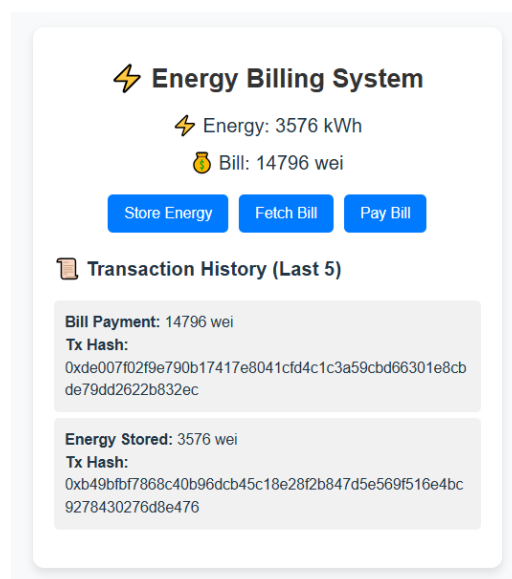

Figure 7.1: User interface of the Energy Billing System

7.7 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 7.2 presents a transaction log demonstrating these interactions.


```
⚡ Fetched Energy: 3576 kWh
✅ Energy data stored! TX Hash: 0xb49bfbf7868c40b96dcb45c18e28f2b847d5e569f516e4bc9278430276d8e476
🥇 Bill Amount: 14796 wei
🥇 Bill to pay: 14796 wei
✅ Bill paid successfully! TX Hash: 0xde007f02f9e790b17417e8041cfd4c1c3a59cbd66301e8cbde79dd2622b832ec
> |
```
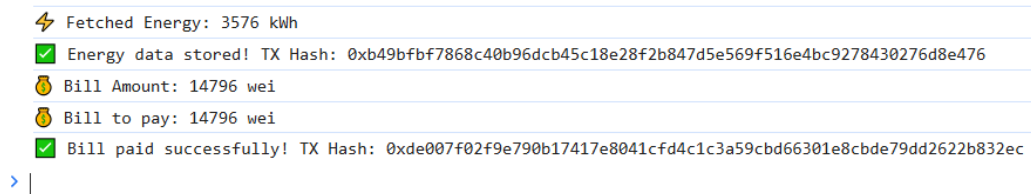
Figure 7.2: Transaction log from frontend interactions with the smart contract

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

7.8 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 8

Dockerizing the Complete System

8.1 Introduction

Docker is an open-source platform that provides lightweight virtualization via Linux Containers (LXC). Docker containers, unlike typical virtual machines, share the host operating system kernel, which improves efficiency and portability. Namespaces for process separation, cgroups for resource management, and union file systems such as OverlayFS for efficient storage are all important technologies.

Docker provides consistency across environments by packaging applications alongside their dependencies[38], making it perfect for microservices, CI/CD pipelines, and cloud deployments [39]. Its scalability and low overhead have transformed modern DevOps procedures[40].

8.2 Dockerization Process

Dockerization is the process of packaging an application and its dependencies into a container. It simplifies deployment, ensures consistency across environments, and facilitates scaling. In this project, we use Docker to containerize three main components: Ganache, ThingsBoard, and a React application.

8.2.1 Docker Compose

To orchestrate multiple containers, we use Docker Compose. The following YAML configuration defines the services required for this setup:

```
version: '3.8'

```

```yaml
services:
  ganache:
    image: trufflesuite/ganache
    container_name: ganache
    restart: always
    ports:
      - "8545:8545"
    command: ["--db", "/ganache-data", "--mnemonic", "candy maple cake sugar
      pudding cream honey rich smooth crumble sweet treat"]
    volumes:
      - ganache_data:/ganache-data
    networks:
      - app_network

  thingsboard:
    image: thingsboard/tb-postgres
    container_name: thingsboard
    restart: always
    ports:
      - "9090:9090"
      - "1883:1883" # MQTT
      - "5683:5683/udp" # CoAP
    environment:
      - DATABASE_TS_TYPE=sql
      - HTTP_CORS_ALLOW_ORIGIN=*  # Enable CORS for ThingsBoard
    volumes:
      - tb-data:/data
      - tb-logs:/var/log/thingsboard
    networks:
      - app_network

  react-app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: react-app
    restart: always
    ports:
      - "80:80"
    depends_on:
```

```
42        - ganache
43        - thingsboard
44     env_file:
45        - .env
46     networks:
47        - app_network
48
49 networks:
50   app_network:
51     driver: bridge
52
53 volumes:
54   ganache_data:
55   tb-data:
56   tb-logs:
```

This configuration defines the following services:

- **Ganache:** A blockchain development tool for Ethereum.

- **ThingsBoard:** An IoT platform for data management.

- **React App:** A front-end web application served using Nginx.

All services are connected using a shared network, `app_network`, ensuring seamless inter-container communication.

8.2.2   Dockerfile

The front-end React application is built using a multi-stage process. The following `Dockerfile` specifies how the React app is built and served with Nginx:

```
1 # Stage 1: Build React App with Vite
2 FROM node:18 AS build
3 WORKDIR /app
4 COPY package.json package-lock.json ./
5 RUN npm install
6 COPY . .
7 RUN npm run build
8
9 # Stage 2: Serve with Nginx
10 FROM nginx:alpine
```

```
11  COPY --from=build /app/dist /usr/share/nginx/html
12  EXPOSE 80
13  CMD ["nginx", "-g", "daemon off;"]
```

The build process follows two stages:

1. In the first stage, Node.js is used to install dependencies and build the React application.

2. In the second stage, the built application is copied into an Nginx container to serve it efficiently.

8.2.3 Running the Containers

Once the configuration is complete, the following command is used to start all containers:

```
1  docker-compose up -d
```

To verify that the containers are running, use:

```
1  docker ps
```

If there are updates to the React application or other services, rebuild and restart the containers using:

```
1  docker-compose up --build -d
```
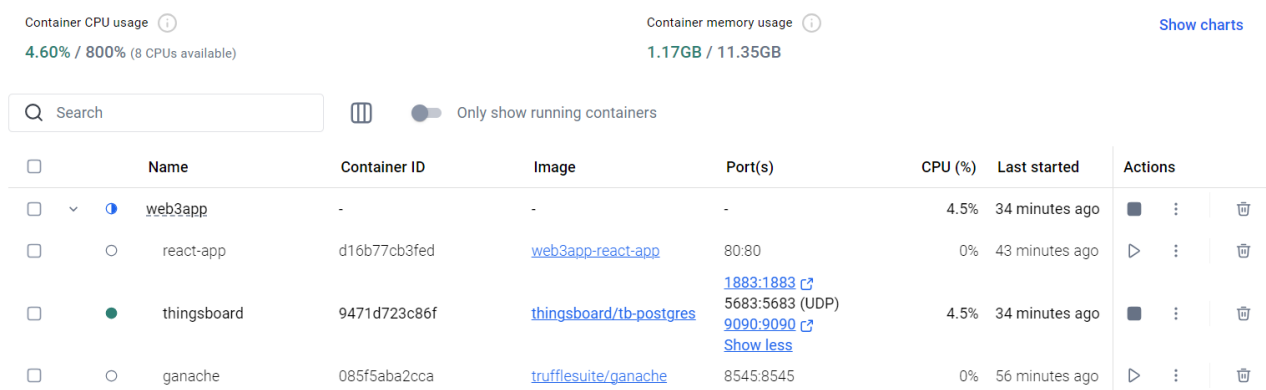

Figure 8.1: Docker Containers

8.2.4 Stopping and Removing Containers

To stop all running containers, use:

```
1  docker-compose down
```

To remove all containers, networks, and volumes, use:

```
docker-compose down -v
```

8.3 Conclusion

Dockerization provides a scalable and efficient way to manage applications by encapsulating dependencies in containers. It simplifies deployment, ensures consistency across environments, and allows for seamless integration between components. By using Docker Compose, we streamline the management of multiple services, making the development and deployment process more efficient.

Chapter 9

Future Scope

The proposed smart home automation system can be improved by incorporating advanced technologies and improving its capabilities. One important enhancement is the incorporation of smart grid technology, which enables homeowners to share excess solar energy with neighbors via blockchain-based smart contracts. This peer-to-peer energy trading model improves energy efficiency and encourages sustainable living. Furthermore, demand-response systems can be used to optimize energy consumption based on real-time grid circumstances, resulting in improved load management and less energy waste. Another area for improvement is the combination of solar energy with water pumping systems. By adding solar panel monitoring, the system can effectively measure energy generation and automate energy distribution to various home appliances. Smart water pumping systems can also be incorporated to efficiently regulate water usage, resulting in optimal operation based on energy supply and demand. This not only minimizes reliance on traditional power sources, but also fosters a self-sustaining smart home ecosystem.

Furthermore, upgrading to a sophisticated microcontroller with built-in Wi-Fi, such as the ESP32 or other IoT-enabled hardware, can considerably boost system performance. These microcontrollers outperform the Raspberry Pi in terms of processing power, connectivity, and power consumption, making them perfect for real-time IoT applications. The use of such hardware would improve the scalability, efficiency, and ease of deployment of future smart home automation solutions.

Chapter 10

Conclusion

The integration of IoT, blockchain, and smart energy management into home automation has the potential to transform how energy is consumed and billed. This project uses a decentralized approach to assure openness, security, and efficiency in energy transactions while eliminating reliance on intermediaries. Smart contracts simplify billing processes, making energy management more efficient and cost-effective.

Looking ahead, adding community energy sharing, solar and water management, and better microcontrollers will improve the system's capabilities and practicality. As smart homes evolve, integrating more sustainable and intelligent solutions will result in a more efficient and environmentally friendly future. This project is a first step toward a completely automated and decentralized smart energy management system, paving the door for future advancements in home automation and energy distribution.

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] M. M. Rahman, M. O. Islam, M. S. Salakin *et al.*, "Arduino and gsm based smart energy meter for advanced metering and billing system," in *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. IEEE, 2015, pp. 1–6.

[4] 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.

[5] 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

[6] 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

[7] 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

[8] 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.

[9] 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.

[10] U. Khair, A. J. Lubis, I. Agustha, Dharmawati, and M. Zulfin, "Modeling and simulation of electrical prevenion 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

[11] C. Dout, "2.7v 4-channel/8-channel 12-bit a/d converters with spi ® serial interface," 1999.

[12] 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

[13] 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.

[14] 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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Z. Kegenbekov and A. Saparova, "Using the mqtt protocol to transmit vehicle telemetry data," *Transportation Research Procedia*, vol. 61, pp. 410–417, 2022.

[20] Z. Shelby, K. Hartke, and C. Bormann, "Rfc 7252: The constrained application protocol (coap)," 2014.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] I. Brightwell, J. Cucurull, D. Galindo, and S. Guasch, "An overview of the ivote 2015 voting system," *New South Wales Electoral Commission, Australia, Scytl Secure Electronic Voting, Spain*, pp. 1–25, 2015.

[28] J. P. Gibson, R. Krimmer, V. Teague, and J. Pomares, "A review of e-voting: the past, present and future," *Annals of Telecommunications*, vol. 71, pp. 279–286, 2016.

[29] K.-H. Wang, S. K. Mondal, K. Chan, and X. Xie, "A review of contemporary e-voting: Requirements, technology, systems and usability," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 31–47, 2017.

[30] 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.

[31] K. Nabben, "Web3 as 'self-infrastructuring': The challenge is how," *Big Data & Society*, vol. 10, no. 1, p. 20539517231159002, 2023.

[32] 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.

[33] 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.

[34] J. Sadowski and K. Beegle, "Expansive and extractive networks of web3," *Big Data & Society*, vol. 10, no. 1, p. 20539517231159629, 2023.

[35] 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.

[36] 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.

[37] 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.

[38] P. Dana, "Interoperability and portability between clouds: Challenges and case study," in *Proceedings of the 4th European Conference ServiceWave*, vol. 6994, Poznan, 2011, pp. 62–74.

[39] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," in *World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2011, pp. 1–6.

[40] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011, pp. 203–216.