PowerLedger

BLOCKCHAIN-BASED ELECTRICITY BILL GENERATOR

B.Tech. III Year I Semester - Project Report

Roll No. :	2200030876
Name of The Student :	P SATYA TEJA
Course Code :	**22DLA3203**
Course Instructor Name :	Dr. Mohan Kumar Chandol

K L Deemed to be University
Department of Computer Science and Engineering
2024-2025, Odd Sem

Abstract

This project aims to create a decentralized application (DApp) for securely and transparently calculating electricity bills. The DApp enables users to input their previous and current electricity meter readings, calculate power consumption, and generate a corresponding bill. Key technologies utilized include **Solidity** (smart contracts), **Ganache** (blockchain simulation), **MetaMask** (user authentication and transaction signing), **Ethereum IDE (Remix)** for smart contract development and testing, and a user-friendly web interface built with **HTML, CSS, and JavaScript**.

The project demonstrates blockchain's potential in utility management, ensuring data immutability, decentralization, and user autonomy. It holds immense potential for real-world applications, especially in creating tamper-proof, automated billing systems.

Literature Review

- **Traditional Billing Systems**

  Traditional electricity billing systems rely heavily on centralized infrastructure, where data is collected, processed, and managed by utility providers. These systems are prone to inaccuracies, delays, and fraud, often leaving users dissatisfied with transparency and accountability.

- **Blockchain in Utility Billing**

  Blockchain technology provides an immutable ledger, enhancing transparency and security in billing processes. Unlike centralized systems, blockchain-based solutions enable users to access real-time, verifiable data, ensuring trust between consumers and providers.

- **Similar DApps**

  Existing DApps for utility billing often lack user-friendly interfaces or are overly complex for practical use. This project bridges the gap by offering a simple, locally deployable solution, focusing on usability and transparency while addressing common limitations of traditional systems.

Introduction

Problem Background

Accurate and secure billing systems are critical for electricity providers and consumers. However, centralized systems are vulnerable to human error, data tampering, and inefficiencies. A decentralized approach ensures trust, transparency, and security in billing processes.

Objective

To design and implement a DApp that calculates electricity usage and generates a corresponding bill using blockchain technology. The solution should be accessible, transparent, and tamper-proof.

Scope

The project is limited to local testing using Ganache for blockchain simulation. MetaMask is used for wallet integration and transaction signing. While the solution is currently deployed on a private blockchain, future scalability to public networks is possible

Brief Overview of The Project, Its Objectives, And Main Findings.

Project Objectives

- To develop a secure and decentralized billing system for electricity consumption.
- To enhance user trust by leveraging blockchain's transparency.
- To demonstrate the practical use of blockchain in utility management.

Main Findings

- The DApp successfully calculates electricity bills based on user inputs.
- It ensures tamper-proof records through blockchain immutability.

- User interaction is simplified with an intuitive front-end interface.

Architectural design of the DApp

System Architecture

The architecture comprises:

1. **Smart Contract Backend:** Handles meter readings, power usage calculations, and bill generation.
2. **Ganache:** Simulates a local blockchain for testing.
3. **MetaMask:** Facilitates wallet integration and transaction signing.
4. **Ethereum IDE (Remix):** Used for developing and testing smart contracts.
5. **Front-End Interface:** Provides a simple UI for user interaction.

Workflow

1. **User Input:** Users enter their previous and current meter readings.
2. **Smart Contract Execution:** The contract calculates the units used and the bill amount.
3. **Blockchain Record:** The bill is stored immutably on the blockchain.

Modules and Components

Modules

1. **Meter Readings Input Module:** Accepts user inputs for meter readings.
2. **Bill Calculation Module:** Computes electricity usage and the corresponding cost.
3. **Blockchain Storage Module:** Records billing data immutably.
4. **User Interface Module:** Displays calculated bills to the user and handles interactions.

Components

- **Smart Contract (Solidity):** Implements the logic for calculating electricity bills.
- **Ethereum IDE (Remix):** A browser-based IDE for smart contract testing and deployment.
- **Ganache:** Provides a local blockchain environment.
- **MetaMask:** Ensures user authentication and transaction approvals.
- **Web3.js:** Connects the front-end interface to the blockchain.
- **HTML, CSS, JavaScript:** Powers the front-end user experience.

Technologies Used

- **Solidity:** Smart contract development language.
- **Ganache:** Local Ethereum blockchain simulator.
- **MetaMask:** Wallet for user authentication and transaction signing.
- **Ethereum IDE (Remix):** A web-based tool for developing and deploying smart contracts.
- **Web3.js:** JavaScript library for interacting with the blockchain.
- **HTML, CSS, JavaScript:** Tools for building the front-end interface.

System Design

Smart Contract Design

Key functions in the Solidity smart contract:

1. setReadings(uint previous, uint current): Accepts meter readings from the user.
2. calculateBill(): Calculates the electricity usage and bill amount based on a predefined rate per unit.
3. getBill(): Retrieves the generated bill details.

Front-End Design

The interface includes:

- **Input Fields:** For entering previous and current meter readings.
- **Button:** To trigger bill generation.
- **Display Section:** Shows the calculated bill details.

Implementation

Key Files

1. **ElectricityBill.sol:** The Solidity smart contract developed and tested using Ethereum IDE (Remix).
2. **index.html:** The front-end structure of the application.
3. **app.js:** Connects the smart contract with the front-end using Web3.js.
4. **truffle-config.js:** Configures the deployment of smart contracts on Ganache.

Challenges and Limitations

Challenges

- **MetaMask Integration:** Ensuring smooth connection between the wallet and the application.
- **Testing in Remix and Ganache:** Debugging contracts to ensure accurate results.

Limitations

- Currently limited to local testing with Ganache.
- Deployment to public blockchains (Ethereum or Polygon) would require additional optimizations.

FUTURE SCOPE

- Deploying the DApp on live networks such as Ethereum or Polygon.
- Adding historical usage and billing records for user reference.
- Expanding support for additional utilities like water and gas billing.
- Enhancing the user interface for improved accessibility and experience.

Conclusion

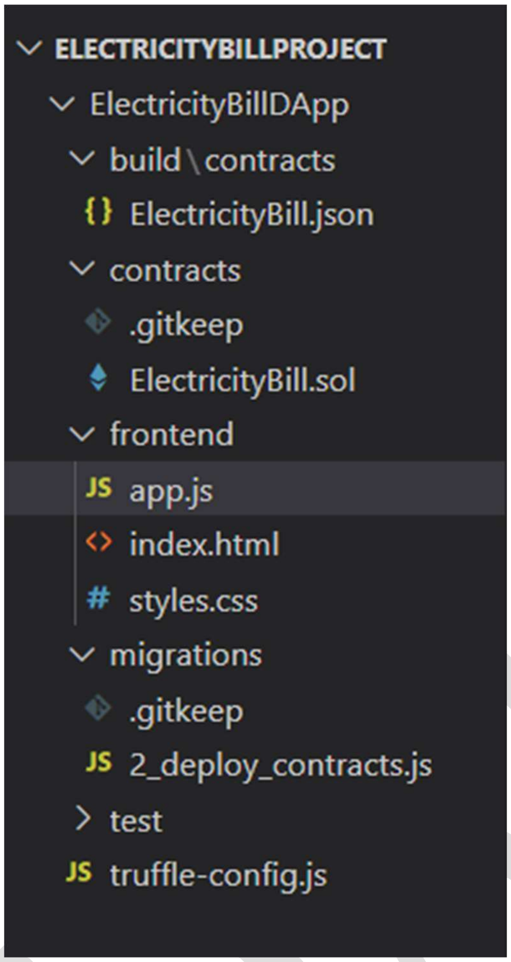
This project demonstrates the real-world potential of blockchain in transforming utility billing systems. By building a decentralized application (DApp) for calculating electricity bills, we've successfully addressed the need for a more secure and transparent billing process. The use of blockchain ensures that all transactions are immutable and reliable, making the system not only efficient but also trustable. With future improvements, such as scaling to public blockchains and additional features, this system could significantly enhance how electricity billing works in the real world.

We would like to take a moment to sincerely thank our mentor, **Dr. Mohan Kumar Chandol**, for his constant support and guidance. His expertise has been a guiding light throughout this project, helping us navigate the complexities of blockchain technology. His constructive feedback and encouragement have played a vital role in shaping this project into what it is today. Thank you for believing in us and inspiring us to push the boundaries of what’s possible.

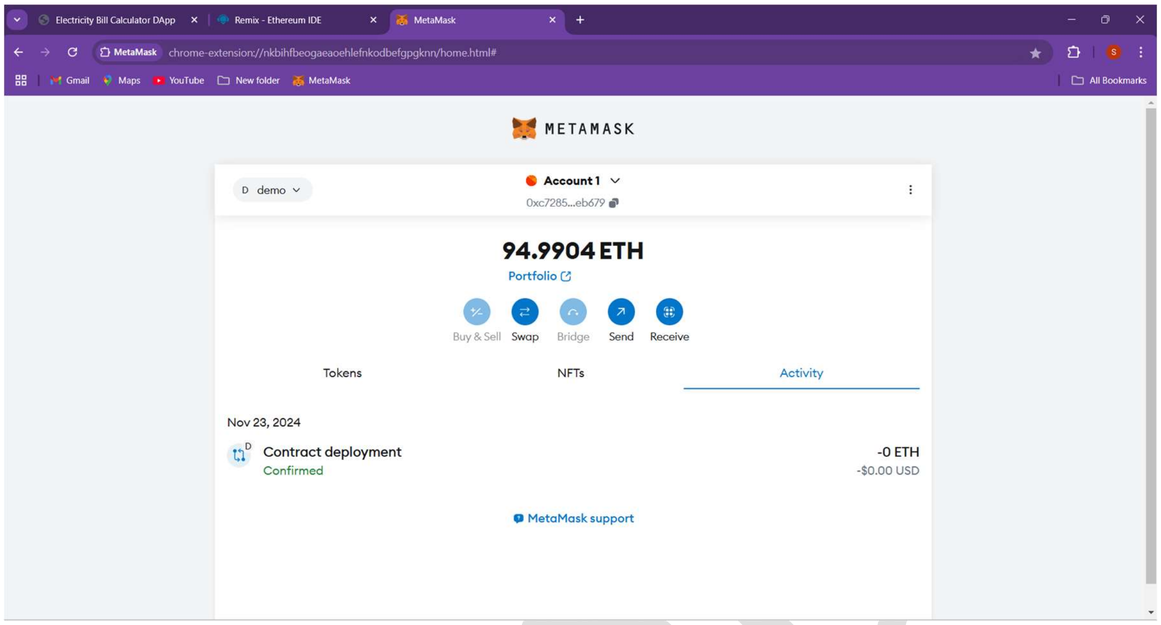
References

- Ethereum Documentation: https://ethereum.org
- Solidity Documentation: https://soliditylang.org
- Remix IDE Documentation: https://remix.ethereum.org
- MetaMask Documentation: https://metamask.io
- Truffle Suite: https://trufflesuite.com

Output Screenshots:

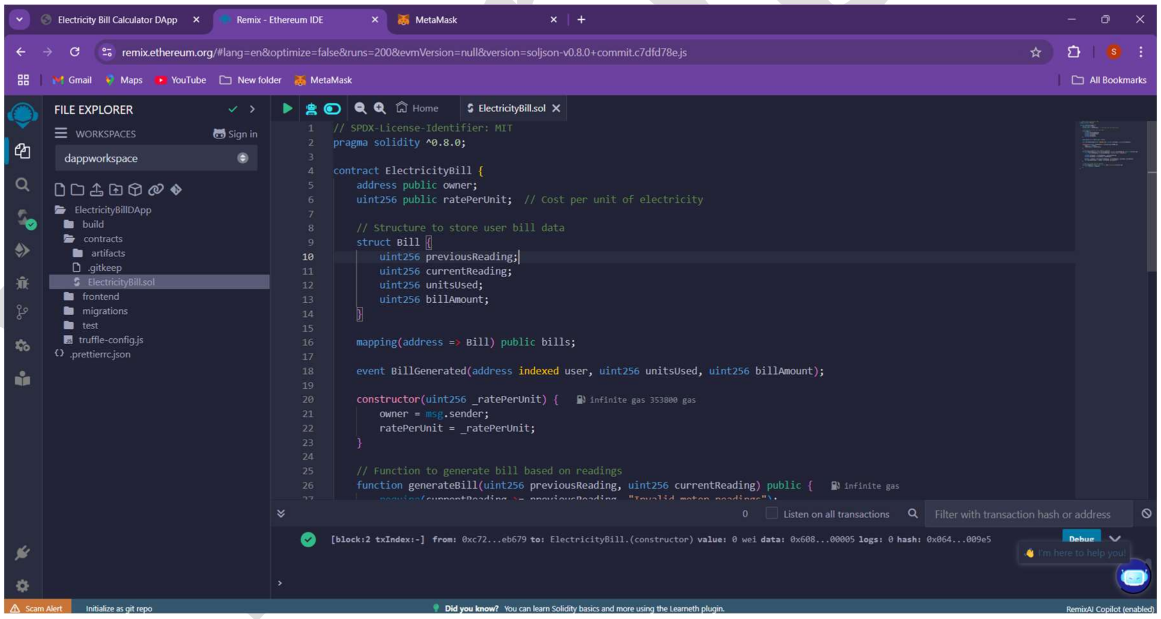
- 

```
ELECTRICITYBILLPROJECT
  ElectricityBillDApp
    build\contracts
      ElectricityBill.json
    contracts
      .gitkeep
      ElectricityBill.sol
    frontend
      app.js
      index.html
      styles.css
    migrations
      .gitkeep
      2_deploy_contracts.js
    test
    truffle-config.js
```



Nov 23, 2024

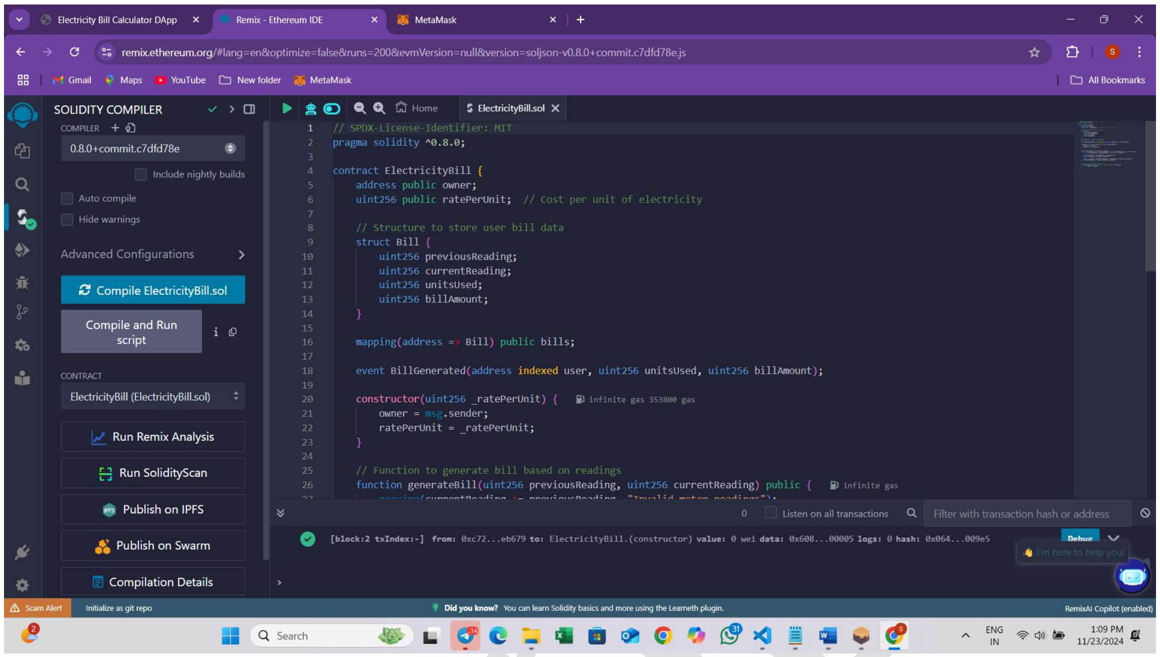
Contract deployment
Confirmed
-0 ETH
-$0.00 USD



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

contract ElectricityBill {
    address public owner;
    uint256 public ratePerUnit;  // Cost per unit of electricity

    // Structure to store user bill data
    struct Bill {
        uint256 previousReading;
        uint256 currentReading;
        uint256 unitsUsed;
        uint256 billAmount;
    }

    mapping(address => Bill) public bills;

    event BillGenerated(address indexed user, uint256 unitsUsed, uint256 billAmount);

    constructor(uint256 _ratePerUnit) {    infinite gas 353800 gas
        owner = msg.sender;
        ratePerUnit = _ratePerUnit;
    }

    // Function to generate bill based on readings
    function generateBill(uint256 previousReading, uint256 currentReading) public {    infinite gas
```

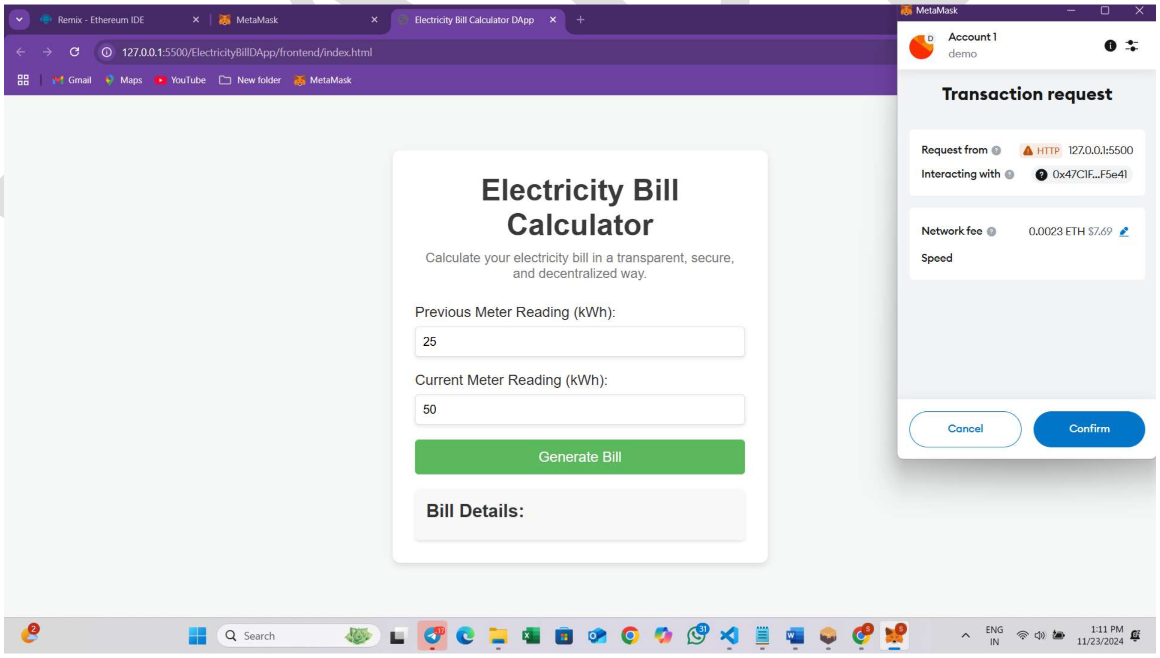
```
[block:2 txIndex:-] from: 0xc72...eb679 to: ElectricityBill.(constructor) value: 0 wei data: 0x608...00005 logs: 0 hash: 0x064...009e5
```

Did you know? You can learn Solidity basics and more using the Learneth plugin.



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

contract ElectricityBill {
    address public owner;
    uint256 public ratePerUnit;  // Cost per unit of electricity

    // Structure to store user bill data
    struct Bill {
        uint256 previousReading;
        uint256 currentReading;
        uint256 unitsUsed;
        uint256 billAmount;
    }

    mapping(address => Bill) public bills;

    event BillGenerated(address indexed user, uint256 unitsUsed, uint256 billAmount);

    constructor(uint256 _ratePerUnit) {
        owner = msg.sender;
        ratePerUnit = _ratePerUnit;
    }

    // Function to generate bill based on readings
    function generateBill(uint256 previousReading, uint256 currentReading) public {
```

```
[block:2 txIndex:-] from: 0xc72...eb679 to: ElectricityBill.(constructor) value: 0 wei data: 0x608...00005 logs: 0 hash: 0x064...009e5
```



Electricity Bill Calculator

Calculate your electricity bill in a transparent, secure, and decentralized way.

Previous Meter Reading (kWh): 25

Current Meter Reading (kWh): 50

Bill Details:

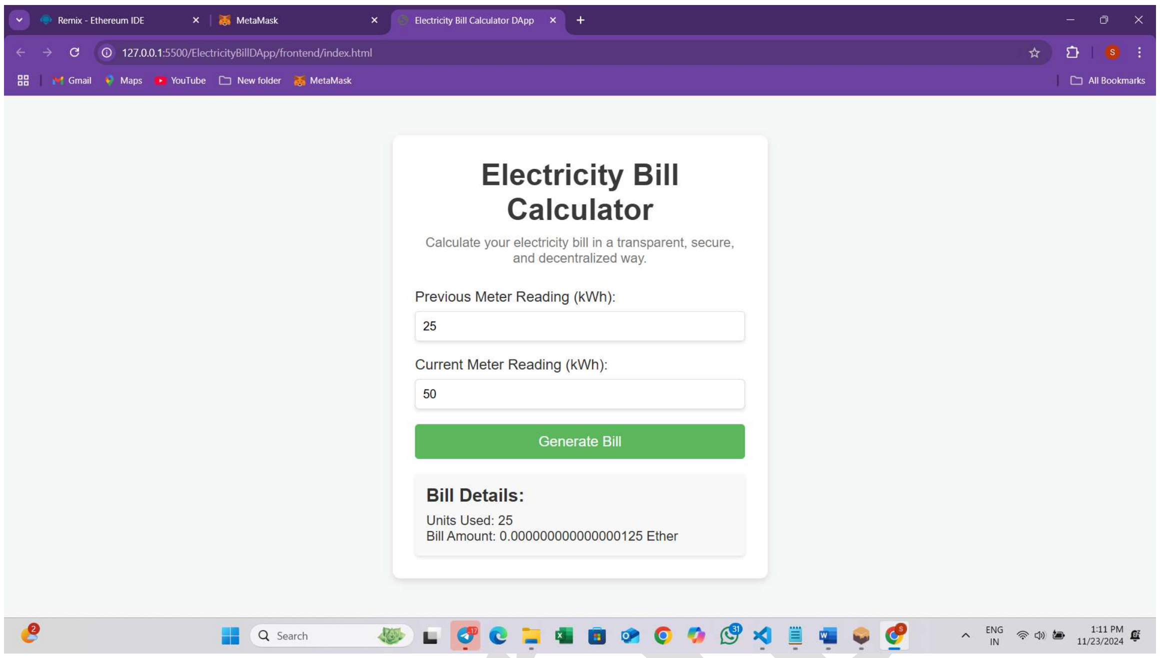
Transaction request

Request from: HTTP 127.0.0.1:5500

Interacting with: 0x47C1F...F5e41

Network fee: 0.0023 ETH $7.69

Speed


Remix - Ethereum IDE
MetaMask
Electricity Bill Calculator DApp
127.0.0.1:5500/ElectricityBillDApp/frontend/index.html
Gmail
Maps
YouTube
New folder
MetaMask
All Bookmarks

Electricity Bill Calculator

Calculate your electricity bill in a transparent, secure, and decentralized way.

Previous Meter Reading (kWh): 25

Current Meter Reading (kWh): 50

Generate Bill

Bill Details:

Units Used: 25
Bill Amount: 0.000000000000000125 Ether