Centurion University

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement : Talk to the World – Backend and Oracle Integration

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

 Introduction:

Gas optimization in smart contracts focuses on reducing transaction costs and improving execution efficiency without compromising security or functionality. Each operation in Ethereum consumes gas, and optimizing code ensures cost-effective deployment and execution. Techniques like using efficient data types, minimizing storage writes, reusing variables, and reducing loop iterations help enhance contract performance and scalability.

 Algorithm / Steps:

1. **Start** the Solidity smart contract in Remix IDE.
2. **Write the initial version** of the contract with basic logic (e.g., storing and updating data).
3. **Compile and deploy** the contract to observe the initial gas consumption.
4. **Analyze gas usage** using Remix's "Gas Analysis" tool after each function execution.
5. **Apply optimization techniques**, such as: o Using memory instead of storage when possible.
   - o Declaring variables with the smallest suitable data type.
   - o Combining operations and reducing function calls.
6. **Recompile and redeploy** the optimized contract.
7. **Compare** gas usage before and after optimization.
8. **Stop** after verifying reduced gas consumption and correct functionality.

* Software Used:

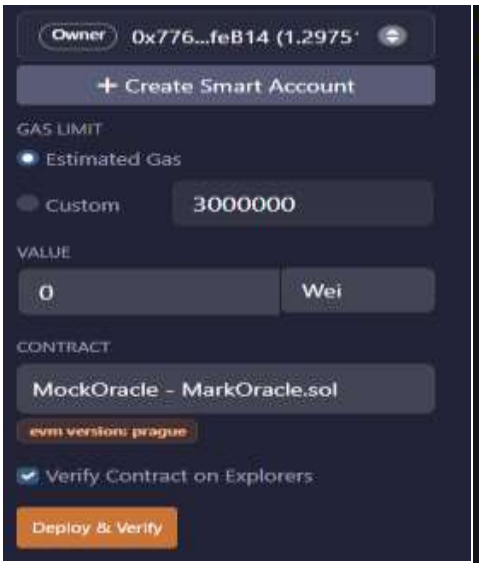
- **Solidity (v0.8.x)** – Smart contract programming language.
- **Remix IDE / Hardhat / Truffle** – for writing and deploying contracts.
- **Node.js & Express.js** – Backend service integration.
- **MetaMask** – for blockchain wallet and transactions.
- **Web3.js / Ethers.js** – For interacting with the contract and analyzing gas costs programmatically.

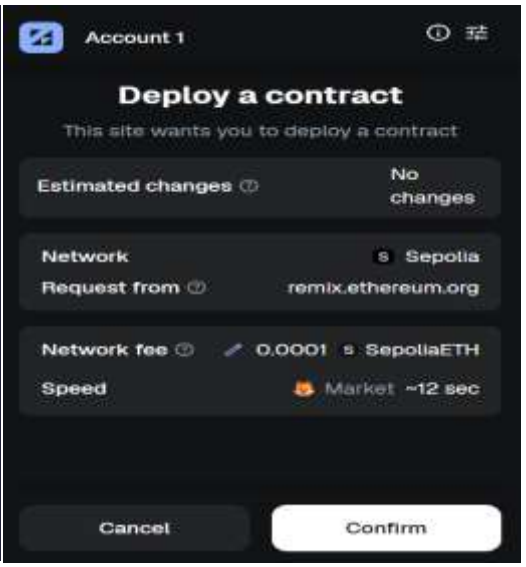
Procedure:

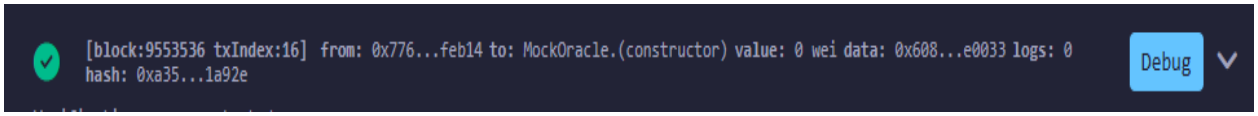
1. Open **Remix IDE** and create a new file MockOracle.sol.

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

contract MockOracle {
    uint public value = 100;
    function setValue(uint _val) public {    22536 gas
        value = _val;
    }
    function getData() external view returns (uint) {    2409 gas
        return value;
    }
}
```

2. Deploy MockOracle first → copy its address .

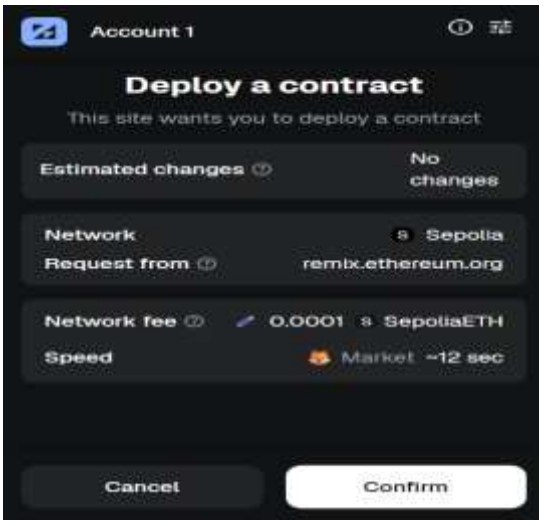




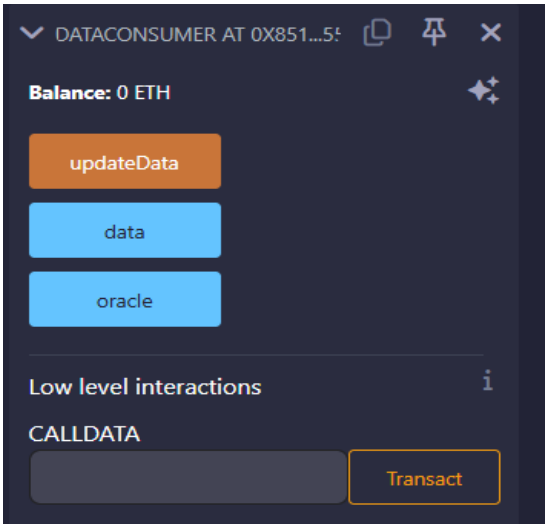


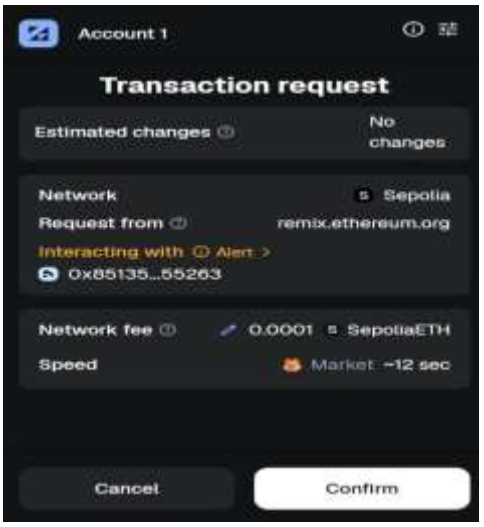
3. Now paste the contract address of the MockOracle in the deploye section of the DataConsumer and deploye the contract.

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

interface IOracle {
    function getData() external view returns (uint);    - gas
}

contract DataConsumer {
    uint public data;
    address public oracle;

    constructor(address _oracle) {    infinite gas 121600 gas
        oracle = _oracle;
    }

    function updateData() public {    infinite gas
        data = IOracle(oracle).getData();
    }
```



4. Now Call updateData() in DataConsumer → fetches data from MockOracle.





Backend Integration (Node.js):

1. Create a folder backend-oracle/.
Initialize a Node.js project:

```
npm init -y
npm install express ethers dotenv
```

2. Create .env file for storing credentials:

```
RPC_URL="https://mainnet.infura.io/v3/143bcff100834977a566b0991bb67475"
PRIVATE_KEY="3273174e57cfec3e11f6da6f5895d9d742f17e3d29652d1837ee47faf3e21ef0"
CONTRACT_ADDRESS="0x851351b777155f3DAa6C08cE5EBDFA4f3FA55263"
```

3. Create server.js file:

```
const express = require('express');
const { ethers } = require('ethers');
require('dotenv').config();

const app = express();
const PORT = 3000;

const provider = new ethers.JsonRpcProvider(process.env.RPC_URL);
const wallet = new ethers.Wallet(process.env.PRIVATE_KEY, provider);

const abi = [
    "function data() public view returns (uint)",
    "function updateData() public"
];
const contract = new ethers.Contract(process.env.CONTRACT_ADDRESS, abi, wallet);

app.get('/update', async (req, res) => {
    const tx = await contract.updateData();
    await tx.wait();
    res.send("Data updated on blockchain");
});
```

```
app.get('/read', async (req, res) => {
    const currentData = await contract.data();
    res.send(`Current Data: ${currentData}`);
});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

4. Run the server:

```
found 0 vulnerabilities
PS C:\Users\shrut\OneDrive\Desktop\backend-oracle> node server.js
[dotenv@17.2.3] injecting env (3) from .env -- tip: sync secrets across teammates & ma
ps
Server running on port 3000
```

5. Now open browser and test:
    - http://localhost:3000/update → updates blockchain data
    - http://localhost:3000/read → reads blockchain data

Observation:

From this experiment, we conclude that:

- Oracles serve as critical bridges between blockchain and the real world.
- Using backend servers (Node.js + Ethers.js), developers can automate off-chain data fetching.
- Chainlink provides a decentralized and secure way to bring external APIs on-chain.
- This integration expands blockchain's potential beyond isolated ledgers — enabling real-world use cases like DeFi price feeds, weather insurance, and supply chain tracking.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Regn. No. :

Signature of the Faculty:

**As applicable according to the experiment. Two sheets per experiment (10-20) to be used.*