Blockchain Explorer

An Web-application to query block data from ethereum chain. User can fetch block data using both block hash and block number and can fetch transaction data using transaction hash. User can connect to metamask and fetch data from the metamask provider else can fetch data via backend.

Links:
Deployed-app : (blockchain-explorer-021.onrender.com)
GitHub-repository : (https://github.com/Akshat021/BlockchainExplorer)

Technologies Used:

- MetaMask chrome extension
- Backend : NodeJs, ExpressJs, Web3Js
- Frontend : ReactJs, EtherJs

Platform and Implementation:
The web application is built using Node.js and Express.js on the backend, and React.js on the frontend. It uses the Web3.js library to interact with the Ethereum blockchain, and the Ether.js library to connect to MetaMask. The user can fetch block data using both block hash and block number, and transaction data using transaction hash.

The backend server uses the Web3.js library to interact with the Ethereum blockchain. It exposes two endpoints: "/block/:block" and "/tx/:tx_hash". When the user makes a request to these endpoints, the server fetches the requested data from the blockchain using Web3.js and returns it in JSON format.

The frontend application is built using React.js and Ether.js. It provides a user interface for the user to input the block number/hash or

transaction hash, and displays the corresponding data fetched from the backend.
If the user has MetaMask installed, the frontend will use Ether.js to connect to MetaMask and fetch data directly from the blockchain. If MetaMask is not installed, the frontend will send a request to the backend to fetch the data.

Why these technologies were used:

- Node.js and Express.js were used to build the backend server because they provide a lightweight and efficient platform for building web applications.
- Web3.js was used to interact with the Ethereum blockchain because it provides a simple and easy-to-use API for fetching data from the blockchain.
- React.js was used for the frontend because it provides a fast and responsive user interface.
- Ether.js was used to connect to MetaMask because it provides a simple and easy-to-use API for interacting with MetaMask.

Source Code :

Backend

```
const express = require('express')
const app = express();
const Web3 = require("web3")
const cors = require('cors')

require('dotenv').config()
const port = process.env.PORT || 5000;

// web3 package
let provider = process.env.ETHEREUM_NODE_URL;
let web3Provider = new Web3.providers.HttpProvider(provider);
let web3 = new Web3(web3Provider);

app.use(cors())
```

```
app.get('/', (req, res) =>{
    res.send("Hello web3")
})

// same endpoint for both block number and block hash.
app.get('/block/:block', async (req, res) => {
    try {
        const data = await web3.eth.getBlock(req.params.block);
        res.json(data);
    } catch (error) {
        console.log(error);
        res.send("something went wrong..");
    }
})

app.get('/tx/:tx_hash', async (req, res) => {
    try {
        const data = await web3.eth.getTransaction(req.params.tx_hash);
        res.json(data);
    } catch (error) {
        console.log(error);
        res.send("something went wrong..");
    }
})

app.listen(port, () => {
    console.log(`server is listening at port ${port}`);
})
```

Frontend

```
import React, { useState, useEffect } from "react";
import { ethers } from "ethers";
import "./App.css"

function App() {
  const [provider, setProvider] = useState(null);
  const [connectedAddress, setConnectedAddress] = useState("");
  const [blockData, setBlockData] = useState(null);
  const [txdata, setTxdata] = useState(null);

  useEffect(() => {
    const loadProvider = async () => {
      if (window.ethereum) {       // checking for metamask wallet
        const provider = new ethers.providers.Web3Provider(window.ethereum);
        console.log(provider);
        try {
          await window.ethereum.enable();
          setProvider(provider);
          const signer = provider.getSigner();
          const address = await signer.getAddress();
          setConnectedAddress(address);
        } catch (error) {
          console.error(error);
        }
      }
    }
    loadProvider();
  }, []);

  const handleBlockNumberSubmit = async (event) => {
    event.preventDefault();
    const blockNumber = parseInt(event.target.elements.blockNumber.value);
    if (!blockNumber) return;
    let block = null;
    if (provider) block = await provider.getBlock(blockNumber);
    else {
      block = await fetch(
        `https://blockchain-explorer-api.onrender.com/block/${event.target.elements.blockNumber.value}`
      );
      block = await block.json();
    }

    setBlockData(block);
  };
```

```
  const handleBlockHashSubmit = async (event) => {
    event.preventDefault();
    const blockHash = event.target.elements.blockHash.value;
    if (!blockHash) return;
    let block = null;
    if (provider) block = await provider.getBlock(blockHash);
    else {
      block = await fetch(
        `https://blockchain-explorer-api.onrender.com/block/${event.target.elements.blockHash.value}`
      );
      block = await block.json();
    }
    setBlockData(block);
  };
```

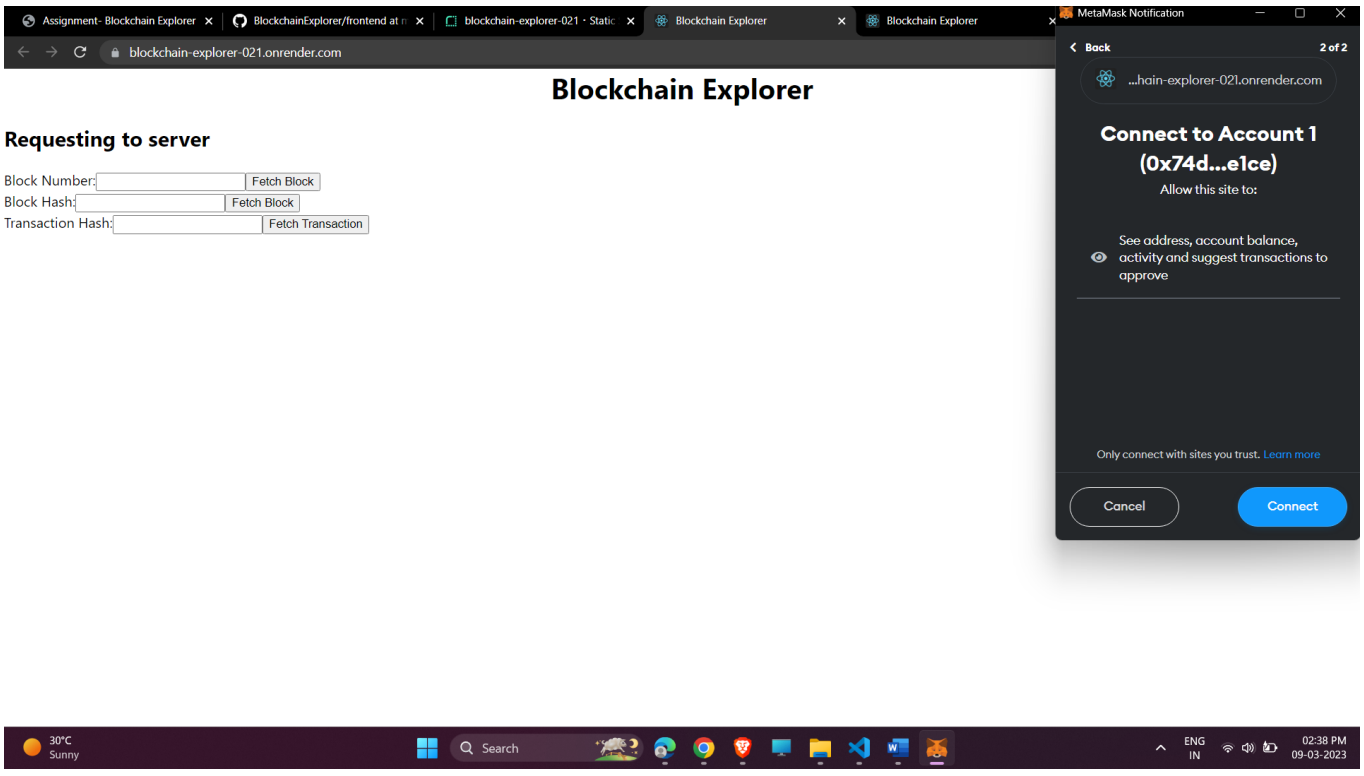
```
const handleTransactionSubmit = async (event) => {
  event.preventDefault();
  const txHash = event.target.elements.transactionHash.value;
  if (!txHash) return;
  let tx = null;

  if (provider) tx = await provider.getTransaction(txHash);
  else {
    tx = await fetch(
      `https://blockchain-explorer-api.onrender.com/tx/${event.target.elements.transactionHash.value}`
    );
    tx = await tx.json();
  }
  console.log(tx);
  setTxdata(tx);
};

return (
  <div className="container">
    <h1 className="heading">Blockchain Explorer</h1>
    <div>
      {provider ? (
        <div>
          <h2>MetaMask Connected</h2>
          <p>Connected Address: {connectedAddress}</p>
          <form onSubmit={handleBlockNumberSubmit}>
            <label htmlFor="blockNumber">Block Number:</label>
            <input type="number" id="blockNumber" name="blockNumber" />
            <button type="submit">Fetch Block</button>
          </form>
          <form onSubmit={handleBlockHashSubmit}>
            <label htmlFor="blockHash">Block Hash:</label>
            <input type="text" id="blockHash" name="blockHash" />
            <button type="submit">Fetch Block</button>
          </form>
          <form onSubmit={handleTransactionSubmit}>
            <label htmlFor="transactionHash">Transaction Hash:</label>
            <input type="text" id="transactionHash" name="transactionHash" />
            <button type="submit">Fetch Transaction</button>
          </form>
        </div>
      ) : (
```

```
        <div className="heading">
          <h2>Requesting to server</h2>
          <form onSubmit={handleBlockNumberSubmit}>
            <label htmlFor="blockNumber">Block Number:</label>
            <input type="number" id="blockNumber" name="blockNumber" />
            <button type="submit">Fetch Block</button>
          </form>
          <form onSubmit={handleBlockHashSubmit}>
            <label htmlFor="blockHash">Block Hash:</label>
            <input type="text" id="blockHash" name="blockHash" />
            <button type="submit">Fetch Block</button>
          </form>
          <form onSubmit={handleTransactionSubmit}>
            <label htmlFor="transactionHash">Transaction Hash:</label>
            <input type="text" id="transactionHash" name="transactionHash" />
            <button type="submit">Fetch Transaction</button>
          </form>
        </div>
      )}
    </div>
```

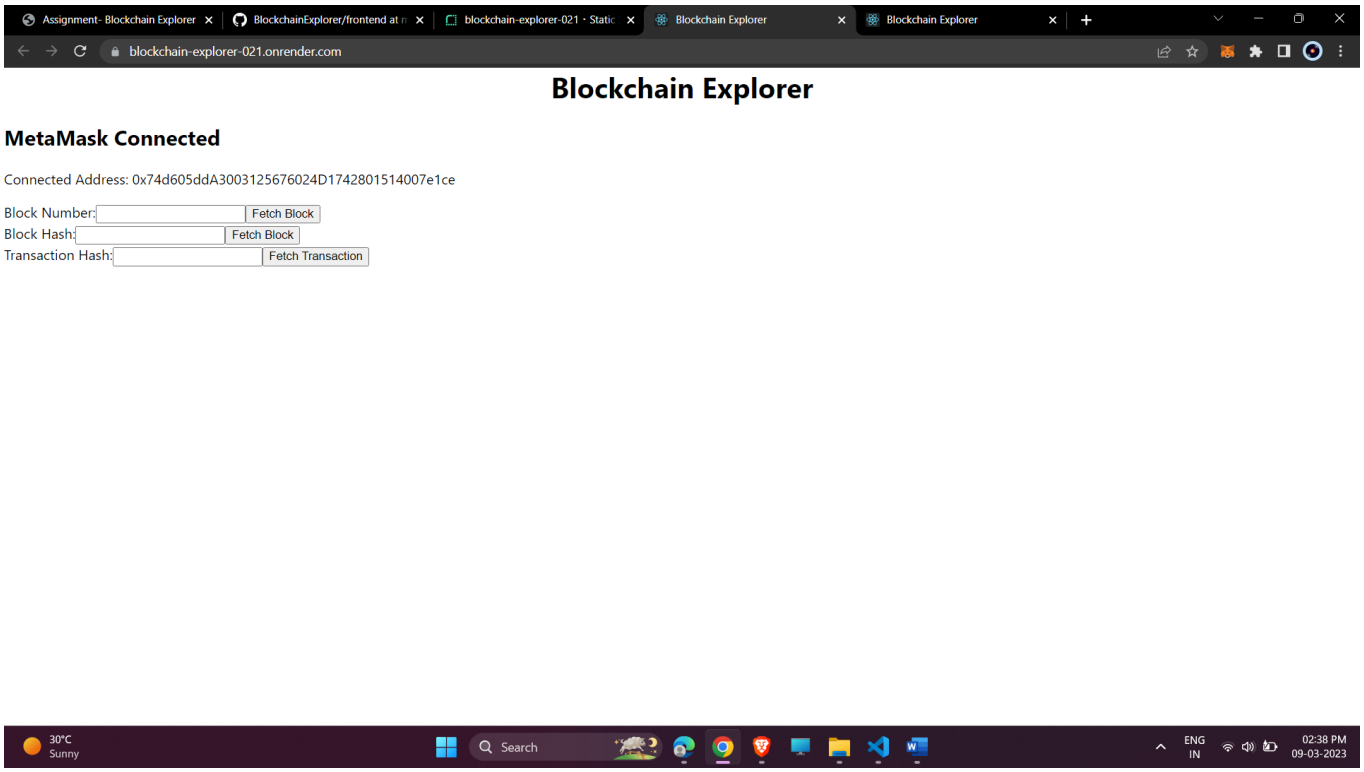
```
      {blockData && (
        <div>
          <h2>Block Data:</h2>
          <p>Number: {blockData.number}</p>
          <p>Hash: {blockData.hash}</p>
          <p>
            Timestamp: {new Date(blockData.timestamp * 1000).toLocaleString()}
          </p>
          <p>Transactions: {blockData.transactions.length}</p>
        </div>
      )}

      {txdata && (
        <div>
          <h2>Transaction Data:</h2>
          <p>blockNumber: {txdata.blockNumber}</p>
          <p>transactionIndex: {txdata.transactionIndex}</p>
          <p>Fee Recipient: {txdata.to}</p>
          <p>Nonce: {txdata.nonce}</p>
        </div>
      )}
    </div>
  );
}

export default App;
```

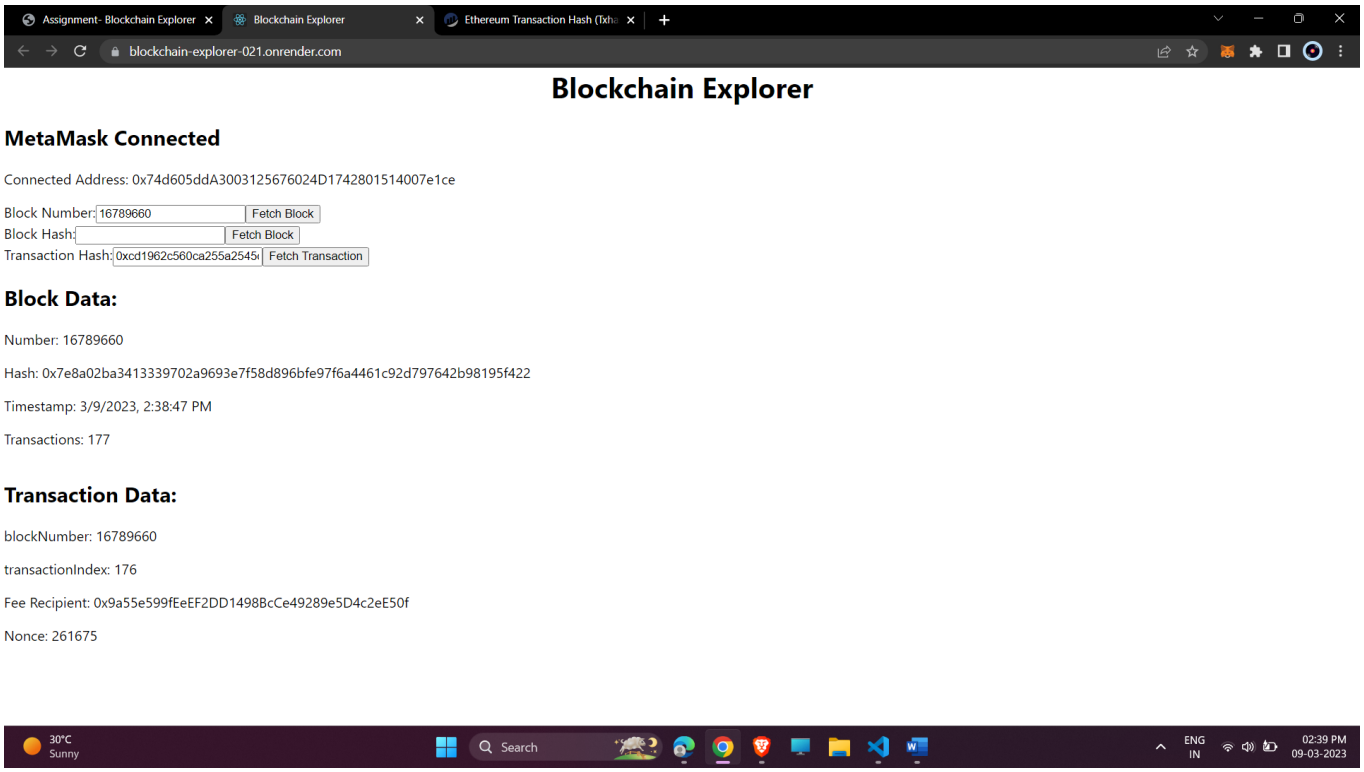
User Interface:

When MetaMask is installed in Browser.

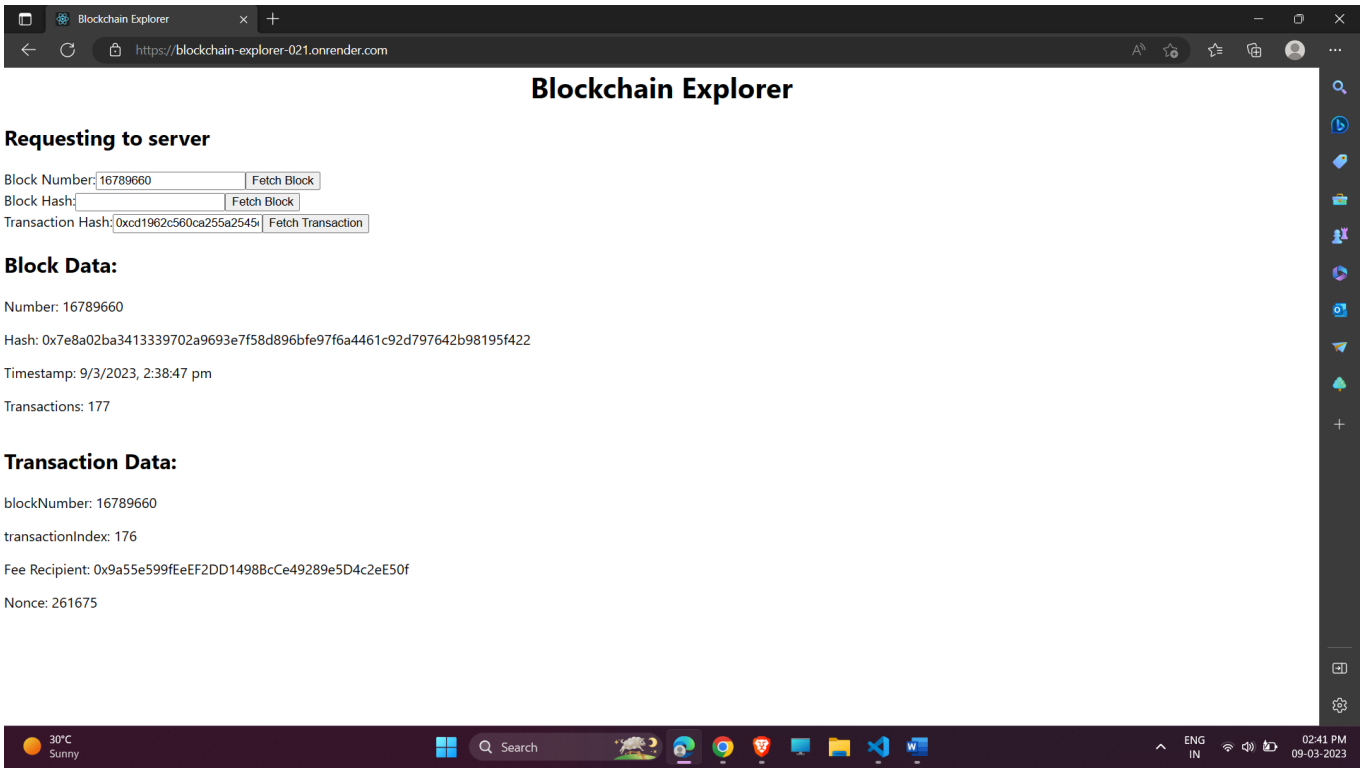


After metamask is connected





When metamask is not connected



Instructions to run the service:

1. Clone the repository from https://github.com/Akshat021/BlockchainExplorer .
2. Navigate to the backend directory and install the dependencies using the command "npm install".
3. Generate Ethereum node url from infura.io and create a .evn file and save that generated url by the name ETHEREUM_NODE_URL in .evn file
4. Start the backend server using the command "node index.js"
5. Navigate to the frontend directory and install the dependencies using the command "npm install".
6. Replace deployed server endpoint with http://localhost:5000/ in App.js.
7. Start the frontend server using the command "npm start"
8. Access the web application in your browser at http://localhost:3000/

Thank you