A PROJECT REPORT ON

SECURE JOB RECRUITMENT THROUGH BLOCKCHAIN DOCUMENT VERIFICATION

ABSTRACT

This project presents a decentralized document verification platform tailored for job recruitment workflows. It uses blockchain's immutable ledger to store cryptographic hashes of candidate documents (e.g., resumes, certificates). Candidates upload their document hashes through a user-friendly interface integrated with MetaMask wallet, ensuring ownership and authentication. Employers can instantly verify the integrity of these documents by matching document hashes against the blockchain record. The system also maintains a transaction ledger logging all uploads and verifications for transparency and auditing. The solution combines Ethereum smart contracts, IPFS for decentralized document storage, and MySQL-backed backend for efficient log management. The user experience is streamlined via a web-based frontend, which integrates with MetaMask, enabling candidates to upload document hashes and prove ownership. Employers or recruiters can subsequently verify documents by uploading them to the same interface; the system hashes the uploaded document and compares it with the blockchain record. A match confirms the integrity and originality of the document, while any discrepancy points to tampering or forgery. All interactions, uploads and verifications are logged in a dedicated transaction ledger, enhancing auditability and supporting transparent hiring practices. This blockchain-based approach offers a secure, fast, and scalable alternative to conventional verification methods, significantly reducing fraud while enhancing user trust.

TABLE OF CONTENTS

LIST OF DIAGRAMS

1. INTRODUCTION

In the digital age, verifying the authenticity of documents is vital, especially in sensitive processes like job recruitment. Traditional manual verification methods are prone to fraud, delays, and high costs. This project leverages blockchain technology to create a secure, transparent, and tamper-proof document verification system that enhances trust between job applicants and employers. By integrating Ethereum smart contracts and decentralized storage, the system enables users to upload document hashes securely and employers to verify document authenticity instantly, reducing fraud and administrative overhead.

2. OBJECTIVES

- Develop a blockchain-backed system for secure document hash registration to ensure integrity and non-repudiation.
- Enable document verification by employers or authorized entities based on blockchain-stored hashes.
- Implement seamless MetaMask wallet integration for authentication and transaction signing.
- Build a decentralized and tamper-proof ledger that logs all document upload and verification events.
- Provide a responsive user interface to facilitate user-friendly upload and verification workflows.
- Create a backend API for handling logging, data storage, and ledger retrieval with MySQL.
- Foster transparency and trust in job recruitment by preventing document forgery and simplifying verification.

3. TECHNOLOGIES USED

FRONTEND

- **JavaScript:** Core programming language for client-side logic and UI interaction.
- **Web3.js**: Ethereum JavaScript API for communicating with smart contracts and blockchain nodes.
- **Crypto JS:** Library for cryptographic hashing (SHA-256) to generate document hashes securely on the client side.
- **MetaMask:** Browser extension wallet for Ethereum, enabling account management and transaction signing by users.
- **HTML&CSS:** Markup and styling for the user interface, including responsive and interactive elements.

BACKEND

- **Node.js:** JavaScript runtime for running server-side.
- **Express.js:** Web framework for building RESTful API endpoints to receive logs and serve data.
- **MySQL:** Relational database management system used to store transaction metadata such as uploads and verifications.
- **Truffle Suite:** Development environment and testing framework for compiling, deploying, and managing smart contracts.
- **Alchemy:** Blockchain infrastructure platform for hosting and interacting with Ethereum nodes (public blockchain access).

4. PROJECT DIRECTORY

JOB-RECRUITMENT-SYSTEM

├── .dist

├── .vscode

├── backend

```
│   ├── node_modules
│   ├── package-lock.json
│   ├── package.json
│   └── server.js
├── build
│   └── contracts
│       └── DocumentVerification.json
├── contracts
│   └── DocumentVerification.sol
├── frontend
│   ├── app.js
│   ├── index.html
│   └── style.css
├── migrations
│   └── 1_deploy_contracts.js
├── node_modules
├── package-lock.json
├── package.json
└── truffle-config.js
└── .env
```

5. FEATURES AND OUTCOMES

- Immutable record of document hashes confirms authenticity.
- Quick verification reduces hiring cycle time.
- Transparent ledger for audit and dispute resolution.
- User-friendly interface powered by real-time blockchain interactions.
- Secure transaction logging with cryptographic verification.
- Scalable architecture combining blockchain and centralized backend.

6. SYSTEM ARCHITECTURE OVERVIEW

1. User connects MetaMask wallet for identity and transaction signing.
2. User uploads a document file, which is hashed client-side using SHA-256.
3. Document hash (with metadata) is submitted to a smart contract deployed on Ethereum blockchain.
4. Smart contract records the hash immutably on blockchain.
5. Backend logs the upload transaction metadata (user address, transaction hash, block number, timestamp) in MySQL.
6. Employers verify documents by uploading the file, hashing it, and querying the smart contract.
7. Verification transaction data is also logged in the backend.

7. STEPS TO DOWNLOAD AND INITIALIZE

7.1 PREREQUISITES

- Node.js (v14+) and npm installed
- MySQL server configured and running locally or remotely
- MetaMask browser extension installed
- Alchemy public blockchain network utilization
- Truffle suite installed for contract compilation and deployment

7.2 THINGS TO DOWNLOAD

- **VS CODE -** **Download Visual Studio Code - Mac, Linux, Windows**
- **NODE JS -** **Node.js — Download Node.js®**
- **METAMASK ENTENSION**

- Go To Chrome Web Store
- Search For MetaMask
- Click Add to Chrome
- Create An Account
- Save The Password for After Usage

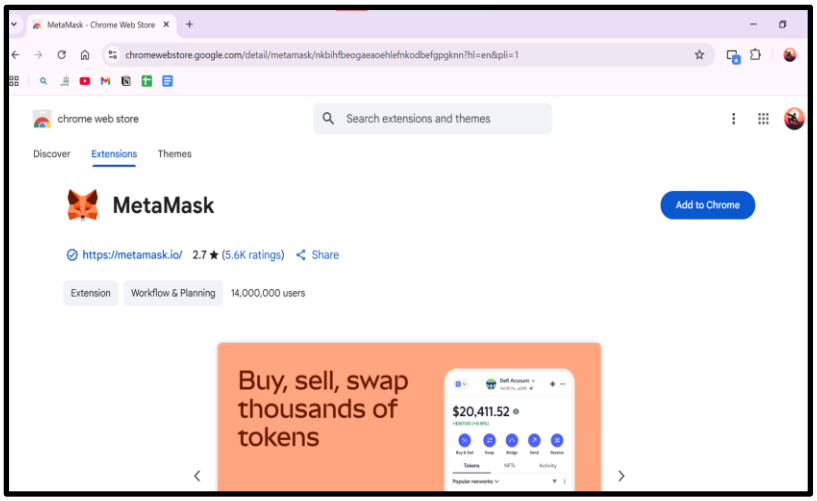


Fig1: MetaMask Extension

- **THINGS TO BE CONFIGURED IN METAMASK**

1. Click on the 3 Bar Icon on the Top Right Corner
2. Click on Settings
3. Click on Advanced

- **TURN ON SHOW TEST NETWORKS**

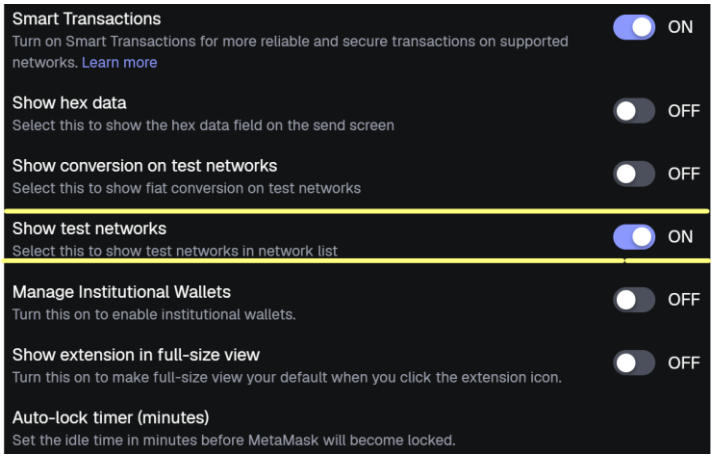


Fig2: Turning on test Network

- **NOW CLICK ON ETHEREUM MAINNET**

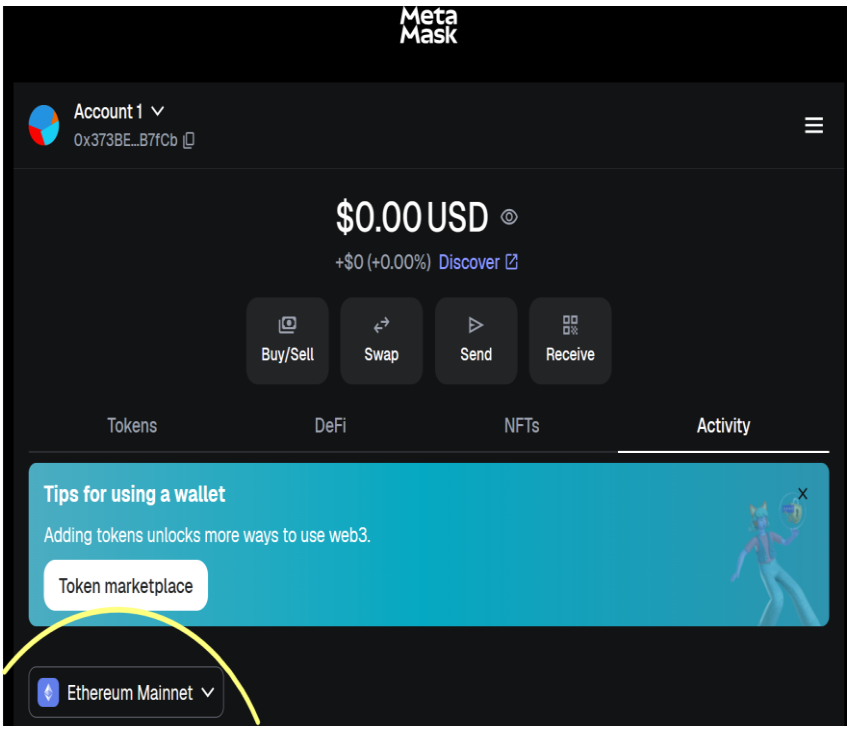


Fig3: Clicking on Ethereum Mainnet

- **CLICK ON CUSTOM AND CHOOSE SEPOLIA**

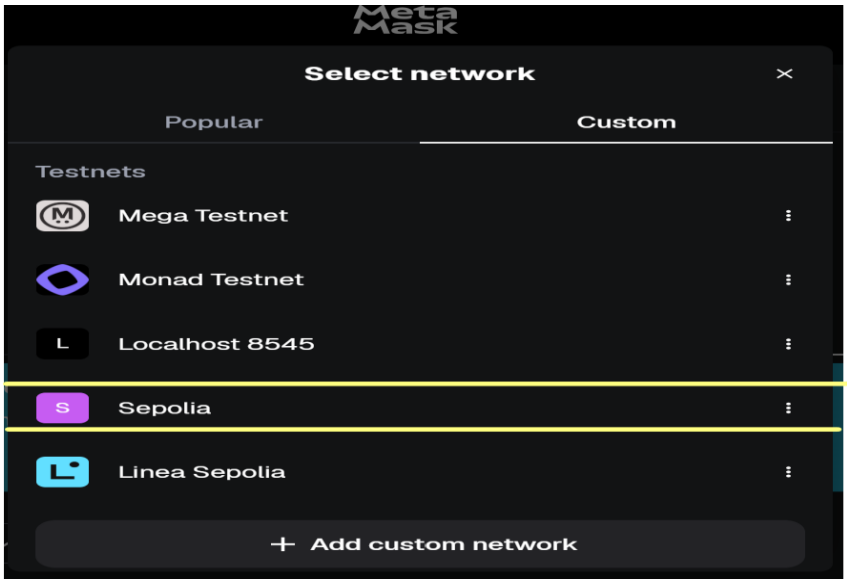


Fig4: Switching to Sepolia

- **NOW THERE IS NO ETHER SO WE HAVE TO ADD ETHEREUM TO IT SO GO TO GHOST FAUCET**

  [GHOST Faucet - Get TestNet Tokens]

- **CLICK ON CLAIM 0.01 SEPOLIA ETH**

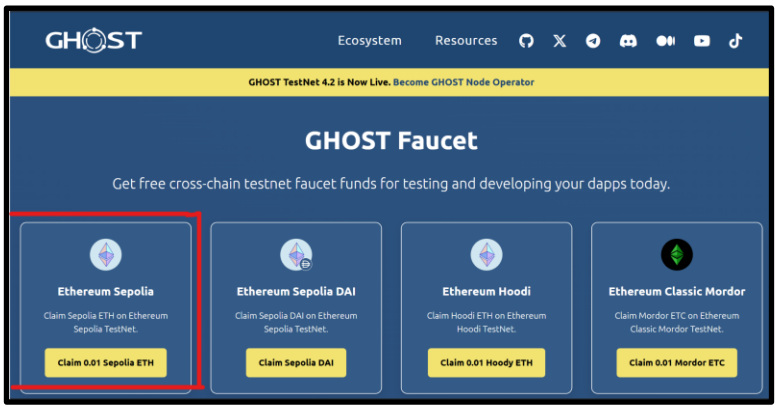


Fig5: Claiming 0.01 Sepolia ETH

- **PASTE THE WALLET ADDRESS FROM METAMASK TO GHOST FAUCET [THE WALLET ADDRESS WILL BE BELOW THE PROFILE PHOTO IN METAMASK]**

Fig6: Pasting the Metamask Address

- **PASTE YOUR WALLET ADDRESS AND NOW YOU WILL SEE 0.01 IN YOUR METAMASK**

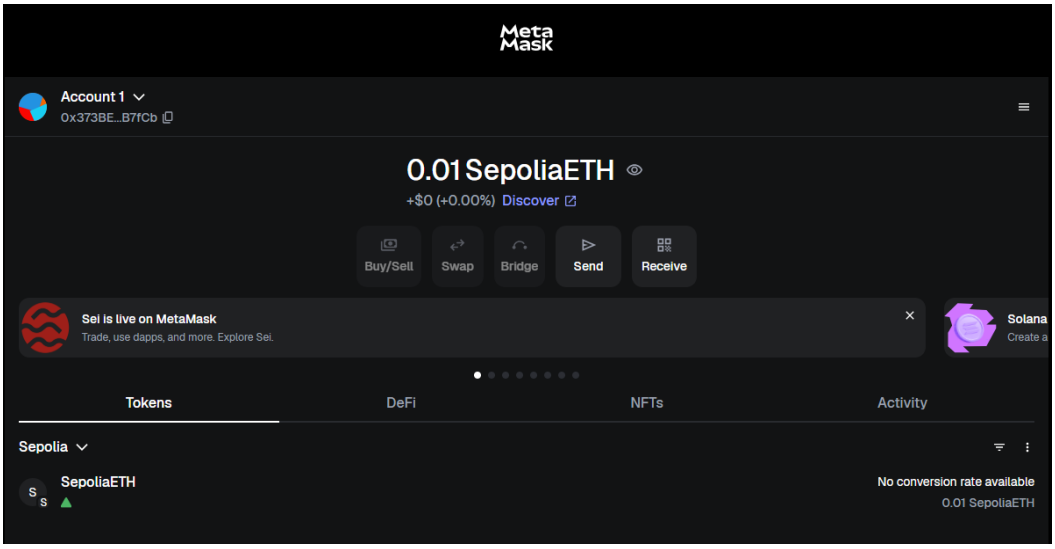


Fig7: Attaining 0.01 in Metamask Wallet

- **MYSQL SETUP GUIDE**
  1. Go to the official MySQL download page: https://dev.mysql.com/downloads/
  2. Download and install MySQL Community Server for your operating system.
  3. During installation, set a root username and password.
  4. Open your terminal (or command prompt).
  5. Connect to MySQL with: **mysql -u root -p**
  6. Enter the root password you created during installation.

```
PS D:\job-recruitment-system final> mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.4.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Fig8:Connection to MySQL

- **ACCESS TO PUBLIC BLOCKCHAIN VIA ALCHEMY RPC URL**

1. **Sign up / Log in**
   - Go to Alchemy https://www.alchemy.com/
   - Sign up or log in with your account.

2. Create a New App / Project

1. Click **"Create App"** (usually a button on your dashboard).
2. Fill in the details:
   - **Name:** Your project name
   - **Network:** Choose the blockchain network you want (Ethereum, Polygon, etc.)
   - **Environment:** Development / Production
3. Click **"Create App"**.

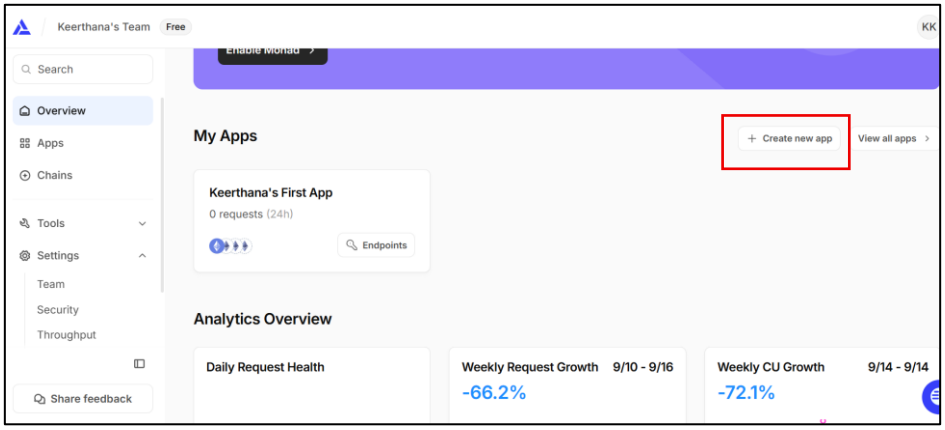


Fig9: Create new app

4. Access your RPC URL

1. Open your newly created app/project.
2. Navigate to the **"View Key" / "API Key" / "Endpoints"** section.
3. You'll see **HTTP / WebSocket URLs** (RPC URLs) for your network.
4. Copy the **HTTP RPC URL** — you'll use this in your Web3 or smart contract code.
   - Example:

     https://eth-goerli.g.alchemy.com/v2/yourAlchemyApiKey

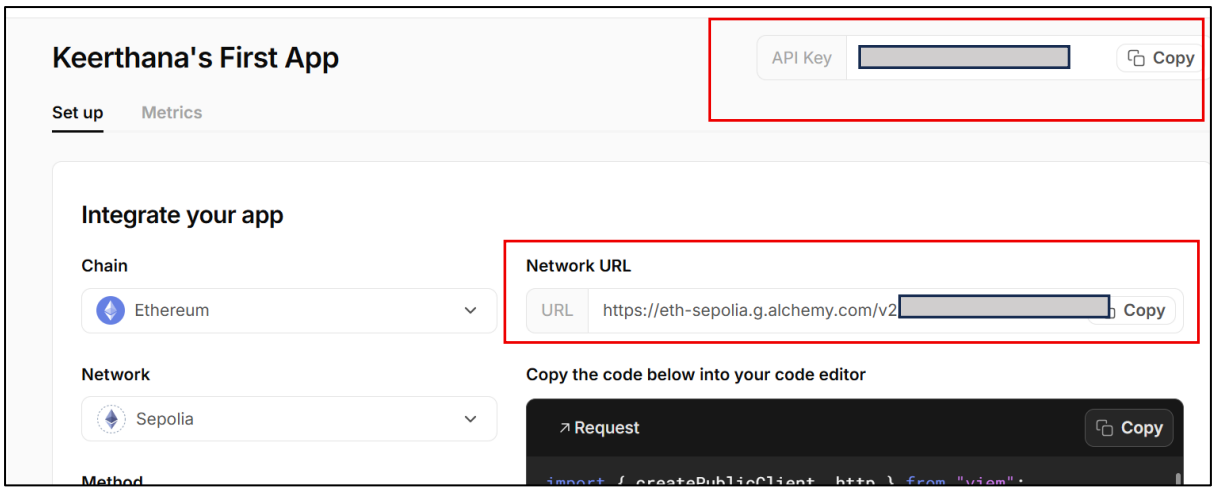


Fig10: Get your RPC URL from you created project

8. CODES

Create a folder – FRONTEND

App.js

```
// app.js (client-side)
let web3;
let contract;
let userAccount;
let contractArtifact;

const backendURL = 'http://localhost:5000';

async function loadContractArtifact() {
  try {
    const response = await fetch('/build/contracts/DocumentVerification.json');
    if (!response.ok) throw new Error('Failed to load contract artifact');
    contractArtifact = await response.json();
  } catch (err) {
    console.error('Error loading contract artifact:', err);
    alert('Failed to load smart contract information. Check console.');
  }
}

window.onload = async () => {
  await loadContractArtifact();

  document.getElementById("connectWalletBtn").onclick = connectWallet;
  document.getElementById("uploadBtn").onclick = uploadDocument;
```

```
  document.getElementById("verifyBtn").onclick = verifyDocument;

  refreshLedger();
  setInterval(refreshLedger, 10000); // Refresh ledger every 10s
};
async function connectWallet() {
  if (!window.ethereum) {
    alert("MetaMask not detected!");
    return;
  }
  try {
    const accounts = await ethereum.request({ method: 'eth_requestAccounts' });
    userAccount = accounts[0];
    web3 = new Web3(window.ethereum);

    const balanceWei = await web3.eth.getBalance(userAccount);
    const balanceEth = web3.utils.fromWei(balanceWei, "ether");
    document.getElementById("walletInfo").innerText = `Connected: ${userAccount} |
Balance: ${balanceEth} ETH`;

    const networkId = await web3.eth.net.getId();
    const deployedNetwork = contractArtifact.networks[networkId];
    if (!deployedNetwork) {
      alert('Smart contract not deployed on the detected network.');
      return;
    }
    contract = new web3.eth.Contract(contractArtifact.abi, deployedNetwork.address);
    document.getElementById("uploadBtn").disabled = false;
    document.getElementById("verifyBtn").disabled = false;
  } catch (error) {
    alert("Failed to connect wallet: " + (error.message || error));
  }
}
// Convert arrayBuffer -> CryptoJS WordArray
function arrayBufferToWordArray(ab) {
  const u8 = new Uint8Array(ab);
  const words = [];
  for (let i = 0; i < u8.length; i += 4) {
    words.push(
      ((u8[i] << 24) | ((u8[i + 1] || 0) << 16) | ((u8[i + 2] || 0) << 8) | ((u8[i + 3] || 0))) >>> 0
    );
  }
  return CryptoJS.lib.WordArray.create(words, u8.length);
}
```

```
function getFileHash(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onload = function () {
      try {
        const wordArray = arrayBufferToWordArray(reader.result);
        const hash = CryptoJS.SHA256(wordArray).toString(CryptoJS.enc.Hex);
        resolve(hash);
      } catch (err) {
        reject(err);
      }
    };
    reader.onerror = (err) => reject(err);
    reader.readAsArrayBuffer(file);
  });
}
async function logTransaction(log) {
  try {
    const payload = {
      type: log.type || null,
      user_address: log.from || log.user_address || null,
      document_hash: log.documentHash || log.document_hash || null,
      transaction_hash: log.transactionHash || log.transaction_hash || null,
      block_number: log.blockNumber || log.block_number || null,
      verified: typeof log.verified !== "undefined" ? (log.verified ? 1 : 0) : null,
      error_msg: log.error || log.error_msg || null,
      timestamp: log.timestamp || new Date().toISOString(),
    };
    console.log("👉 Sending to backend:", payload);
    const resp = await fetch(`${backendURL}/logTransaction`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload),
    });
    if (!resp.ok) {
      const errText = await resp.text();
      console.error("❌ Backend error:", resp.status, errText);
    } else {
      const j = await resp.json();
      console.log("✅ Backend response:", j);
    }
  } catch (err) {
    console.error("❌ Logging failed:", err);
```

```
  }
}
async function uploadDocument() {
  if (!contract) {
    alert("Please connect MetaMask wallet first.");
    return;
  }
  const fileInput = document.getElementById("uploadFile");
  if (!fileInput.files.length) {
    alert("Please choose a file to upload");
    return;
  }
  const file = fileInput.files[0];
  try {
    const hash = await getFileHash(file);
    const hashHex = "0x" + hash;

    const receipt = await contract.methods
      .uploadDocument(hashHex)
      .send({ from: userAccount, gas: 300000 });

    document.getElementById("uploadStatus").innerText =
      "✅ Document hash uploaded successfully!";
    // 🔥 log upload with verified=1
    await logTransaction({
      type: "upload",
      from: userAccount,
      documentHash: hashHex,
      transactionHash: receipt.transactionHash,
      verified: 1,
      timestamp: new Date().toISOString(),
    });
  } catch (error) {
    console.error("❌ Upload error:", error);
    document.getElementById("uploadStatus").innerText =
      "Upload failed or hash exists: " + (error.message || error);
    await logTransaction({
      type: "upload_failed",
      from: userAccount,
      error: error.message || String(error),
      verified: 0,
      timestamp: new Date().toISOString(),
    });
```

```
  }
}
async function verifyDocument() {
  if (!contract) {
    alert("Please connect MetaMask wallet first.");
    return;
  }
  const fileInput = document.getElementById("verifyFile");
  if (!fileInput.files.length) {
    alert("Please choose a file to verify");
    return;
  }
  const file = fileInput.files[0];
  try {
    const hash = await getFileHash(file);
    const hashHex = "0x" + hash;
    const exists = await contract.methods.verifyDocument(hashHex).call();

    document.getElementById("verifyStatus").innerText = exists ? "Verified ✅" : "Not
Verified ❌";

    await logTransaction({
      type: 'verify',
      from: userAccount,
      documentHash: hashHex,
      verified: exists ? 1 : 0,
      timestamp: new Date().toISOString(),
    });
  } catch (error) {
    console.error("Verify error:", error);
    document.getElementById("verifyStatus").innerText =
      "Verification failed: " + (error.message || error);
    await logTransaction({
      type: 'verify_failed',
      from: userAccount,
      error: error.message || String(error),
      verified: 0,
      timestamp: new Date().toISOString(),
    });
  }
}

async function refreshLedger() {
```

```
try {
  const response = await fetch(`${backendURL}/logs`);
  if (!response.ok) {
    console.error('Failed to fetch logs:', response.status);
    return;
  }
  const logs = await response.json();
  const tbody = document.getElementById('ledgerBody');
  if (!tbody) return;
  tbody.innerHTML = '';

  if (!Array.isArray(logs) || logs.length === 0) {
    tbody.innerHTML =
      `<tr><td colspan="5" style="text-align:center;">No transactions logged yet.</td></tr>`;
    return;
  }

  logs.forEach(log => {
    const tr = document.createElement('tr');
    tr.style.borderBottom = '1px solid #d3126b';

    let verifiedDisplay = '-';
    if (log.type === 'verify') {
      verifiedDisplay = log.verified ? 'Yes' : 'No';
    } else if (log.type === 'upload') {
      verifiedDisplay = 'Uploaded';
    } else if (log.type === 'upload_failed') {
      verifiedDisplay = 'Upload Failed';
    } else if (log.type === 'verify_failed') {
      verifiedDisplay = 'Verify Failed';
    }

    const txHashDisplay = log.transaction_hash
      ? log.transaction_hash.slice(0, 10) + '...'
      : '-';
    const fromShort = log.user_address
      ? log.user_address.slice(0, 6) + '...' + log.user_address.slice(-4)
      : '-';
    const timestampStr = log.timestamp
      ? new Date(log.timestamp).toLocaleString()
      : '-';

    tr.innerHTML = `
      <td style="padding:8px;">${log.type}</td>
```

```
      <td style="padding:8px;">${fromShort}</td>
      <td style="padding:8px;">${txHashDisplay}</td>
      <td style="padding:8px;">${verifiedDisplay}</td>
      <td style="padding:8px;">${timestampStr}</td>
    `;
    tbody.appendChild(tr);
  });
 } catch (err) {
  console.error("Failed to load ledger:", err);
 }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Job Recruitment Verification</title>
<link rel="stylesheet" href="style.css" />
</head>
<body>
 <div class="container">
  <h1 class="section-title">JOB RECRUITMENT VERIFICATION</h1>
  <div class="flex-center">
   <div class="card-section">
    <h2>Upload Document</h2>
    <div class="file-label">Select Document to Upload</div>
    <input type="file" id="uploadFile" />
    <button class="main-action-btn" id="uploadBtn" disabled>Upload</button>
    <div id="uploadStatus" class="status"></div>
   </div>
   <div class="card-section">
    <h2>Verify Document</h2>
    <div class="file-label">Select Document to Verify</div>
    <input type="file" id="verifyFile" />
    <button class="main-action-btn" id="verifyBtn" disabled>Verify</button>
    <div id="verifyStatus" class="status"></div>
   </div>
  </div>
  <button id="connectWalletBtn" class="main-action-btn">Connect Wallet</button>
  <div id="walletInfo" class="status"></div>
```

```
  </div>

  <!-- Ledger Container OUTSIDE main .container, will be below -->
  <h2 class="section-title" style="margin-top: 40px;">TRANSACTION LEDGER</h2>
  <div id="ledgerContainer">
   <table id="ledgerTable">
    <thead>
     <tr>
      <th>Type</th>
      <th>From</th>
      <th>Tx Hash</th>
      <th>Verified</th>
      <th>Timestamp</th>
     </tr>
    </thead>
    <tbody id="ledgerBody">
     <!-- Rows populated by app.js -->
    </tbody>
   </table>
  </div>
  <!-- Scripts -->
  <script src="https://cdn.jsdelivr.net/npm/web3/dist/web3.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-
js.min.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

style.css

```
html, body {
  height: 100%;
  margin: 0;
  padding: 0;
  background:
url('https://i.pinimg.com/736x/49/eb/fd/49ebfd9e96bcd243a3926a210a2ff848.jpg') no-repeat
center center fixed;
  background-size: cover;
}

body {
  min-height: 100vh;
  margin: 0;
  font-family: 'Poppins', sans-serif;
```

```
}

/* Center both containers and table nicely */
.container {
  width: 90%;
  max-width: 1000px;
  padding: 40px;
  margin: 50px auto 0 auto;
  background: rgba(197, 15, 15, 0.05);
  backdrop-filter: blur(15px);
  border-radius: 20px;
  box-shadow: 0 0 15px 5px rgba(255, 12, 12, 0.6),
              0 10px 40px rgba(255, 20, 20, 0.5);
  display: flex;
  flex-direction: column;
  align-items: center;
}

.section-title {
  font-size: 2.1em;
  font-weight: 700;
  color: #fff;
  font-family: cursive;
  text-align: center;
  margin-bottom: 40px;
  letter-spacing: 1px;
  text-shadow: 0 0 15px #00fff4;
}

.flex-center {
  display: flex;
  flex-wrap: wrap;
  gap: 30px;
  justify-content: center;
  width: 100%;
}

.card-section {
  flex: 1 1 320px;
  max-width: 350px;
  background: rgba(255, 255, 255, 0.08);
  padding: 30px;
  border-radius: 20px;
  backdrop-filter: blur(12px);
```

```
  box-shadow: 0 8px 25px rgba(0, 255, 255, 0.2);
  display: flex;
  flex-direction: column;
  align-items: center;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}
.card-section:hover {
  transform: translateY(-5px);
  box-shadow: 0 15px 35px rgba(0, 255, 255, 0.3);
}

.card-section h2 {
  color: #00fff4;
  margin-bottom: 25px;
  font-size: 1.5em;
  text-align: center;
  text-shadow: 0 0 10px #00fff480;
}

.file-label {
  color: #80ffe0;
  font-weight: 600;
  margin-bottom: 12px;
  align-self: flex-start;
}
input[type="file"] {
  width: 100%;
  padding: 10px 12px;
  border-radius: 10px;
  border: 1.5px solid rgba(0,255,255,0.5);
  background: rgba(255,255,255,0.03);
  color: #fff;
  margin-bottom: 20px;
  cursor: pointer;
  transition: all 0.2s ease;
}
input[type="file"]:hover {
  background: rgba(255,255,255,0.06);
}
.main-action-btn {
  width: 100%;
  padding: 14px 0;
  border: none;
  border-radius: 12px;
```

```
  background: linear-gradient(90deg, #00fff4, #00c0ff);
  color: #0d1b1f;
  font-weight: 700;
  font-size: 1.1em;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 5px 15px rgba(0,255,255,0.3);
}
.main-action-btn:disabled {
  background: rgba(0,255,255,0.2);
  color: #99b7c9;
  cursor: not-allowed;
  box-shadow: none;
}
.main-action-btn:hover:not(:disabled) {
  transform: translateY(-2px);
  box-shadow: 0 8px 25px rgba(0,255,255,0.5);
}
.status {
  margin-top: 12px;
  color: #fff;
  text-align: center;
  font-weight: 600;
  font-family: 'century Gothic';
  min-height: 18px;
}
#connectWalletBtn {
  margin-top: 40px;
  max-width: 380px;
}

/* -------- TRANSACTION LEDGER TABLE STYLING ---------- */

h2[style*="Transaction Ledger"] {
  margin-top: 40px;
  color: #5e0000;
  text-align: center;
  font-size: 2.1em;
  font-weight: 700;
  font-family: cursive;
  letter-spacing: 1px;
  text-shadow: 0 0 15px #00fff4;
}
```

```
#ledgerContainer {
  width: 96%;
  max-width: 900px;
  margin: 15px auto 30px auto; /* Top margin separates it from main container */
  border-radius: 12px;
  overflow: hidden;
  box-shadow: 0 2px 18px rgba(0,0,0,0.22);
  background: rgba(245,74,181,0.80);
}

#ledgerTable {
  width: 100%;
  border-collapse: collapse;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  table-layout: fixed;
}

#ledgerTable th {
  background: #d3126b;
  color: #fff;
  font-size: 17px;
  padding: 13px 10px;
  text-align: left;
}
#ledgerTable td {
  font-size: 15px;
  padding: 11px 10px;
  background: rgba(255, 192, 203, 0.55);
  color: #2c003e;
  word-break: break-all;
}
#ledgerTable tr:nth-child(even) td {
  background: rgba(255, 229, 247, 0.7);
}
#ledgerTable tr:hover td {
  background: #f575c5;
  color: #fff;
  transition: all 0.2s;
}

/* Responsive for tablet and mobile */
@media (max-width: 800px) {
  .flex-center {
    flex-direction: column;
```

```
    gap: 25px;
  }
  .card-section {
    width: 90%;
  }
  #ledgerContainer {
    width: 100%;
    padding: 0;
    border-radius: 8px;
  }
}
@media (max-width: 650px) {
  #ledgerTable th, #ledgerTable td {
    font-size: 12px;
    padding: 6px 4px;
  }
  #ledgerContainer {
    max-width: 100%;
  }
}
```

Create a folder –BACKEND

Server.js

```
const express = require('express');

const cors = require('cors');

const mysql = require('mysql2/promise');

const app = express();

app.use(cors());

app.use(express.json());

// Setup MySQL connection pool; adjust credentials and port if needed

const pool = mysql.createPool({

  host: 'localhost',

  user: 'root',

  password: ' ',  //Replace with your password
```

```
  database: 'job_verification',  // create a database in this name and use it or else create new
one
  port: 3306,
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0,
});

// POST /logTransaction - receive and store logs
app.post('/logTransaction', async (req, res) => {
  const {
    type,
    user_address,
    document_hash,
    transaction_hash,
    verified,
    error_msg,
    block_number,
    timestamp,
  } = req.body;
  console.log('Received log:', req.body);
  try {
    const [result] = await pool.execute(
      `INSERT INTO transactions
```

```
      (type, user_address, document_hash, transaction_hash, verified, error_msg,
block_number, timestamp)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
     [
       type || null,
       user_address || null,
       document_hash || null,
       transaction_hash || null,
       typeof verified === 'undefined' ? null : verified,
       error_msg || null,
       block_number || null,
       timestamp ? new Date(timestamp) : new Date(),
     ]
   );
   res.json({ status: 'success', insertedId: result.insertId });
  } catch (err) {
   console.error('DB Insert Error:', err);
   res.status(500).json({ status: 'error', message: err.message });
  }
});
// GET /logs - return recent 100 logs ordered by time desc
app.get('/logs', async (req, res) => {
  try {
   const [rows] = await pool.query(
     'SELECT * FROM transactions ORDER BY timestamp DESC LIMIT 100'
```

```
    );
    res.json(rows);
  } catch (err) {
    console.error('DB Fetch Error:', err);
    res.status(500).json({ status: 'error', message: err.message });
  }
});
const port = 5000;
app.listen(port, () => {
  console.log(`Backend listening on http://localhost:${port}`);
});
```

Create a folder -CONTRACTS

DocumentVerification.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract DocumentVerification {
    mapping(bytes32 => bool) private documentHashes;
    event DocumentUploaded(address indexed uploader, bytes32 documentHash);
    function uploadDocument(bytes32 documentHash) public {
        require(!documentHashes[documentHash], "Document hash already exists");
        documentHashes[documentHash] = true;
        emit DocumentUploaded(msg.sender, documentHash);
    }

    function verifyDocument(bytes32 documentHash) public view returns (bool) {
        return documentHashes[documentHash];
    }
}
```

Separate file

Truffle-config.js [Deploying smart contracts (.sol code) directly]

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
// Replace with your MetaMask wallet mnemonic (keep it secure!)
const mnemonic = '   ';
// Replace with your Sepolia RPC URL (e.g., from Infura or Alchemy)
const sepoliaRpcUrl = '  '; //Replace with RPC URL as mentioned above
module.exports = {
  networks: {
    sepolia: {
      provider: () => new HDWalletProvider(mnemonic, sepoliaRpcUrl),
      network_id: 11155111, // Sepolia network id
      gas: 6000000,
      confirmations: 2,
      timeoutBlocks: 200,
      skipDryRun: true
    }
  },
  compilers: {
    solc: {
      version: "0.8.20"
    }
  }
};
```

NOTE :

- The folder node_modules will be created as you run npm install.
- The file package-lock.json will be created when the command npm run dev is done.
- **Metamask Mnemonic:** Open MetaMask > Account Details > Private Key [Replace it in your code]

To run a project via terminal,

1. Use separate Terminal for each process

[RUN THE COMMANDS AFTER MYSQL IS CONNECTED]

```
CREATE DATABASE job_verification;

USE job_verification;

CREATE TABLE transactions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  type VARCHAR(20),
  user_address VARCHAR(100),
  document_hash VARCHAR(256),
  transaction_hash VARCHAR(256),
  verified BOOLEAN,
  error_msg TEXT,
  block_number INT,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

2. Set up SQL first make it connected and run these queries given below:

In another terminal run the backend with the command-> **node server.js**

- Then go with live server to view the frontend.

For more details, **GITHUB REPOSITORY FOR COMPLETE SET UP AND INSTALLATION** 👇
https://github.com/Keerthana2k24/SECURE_JOB_RECRUITMENT

9. WORKING OUTPUTS

9.1 START THE BACKEND [SERVING TO MYSQL]

```
PS D:\job-recruitment-system final> cd backend
PS D:\job-recruitment-system final\backend> node server.js
Backend listening on http://localhost:5000
```

Fig11: Running the server

9.2 FRONT END VIEW

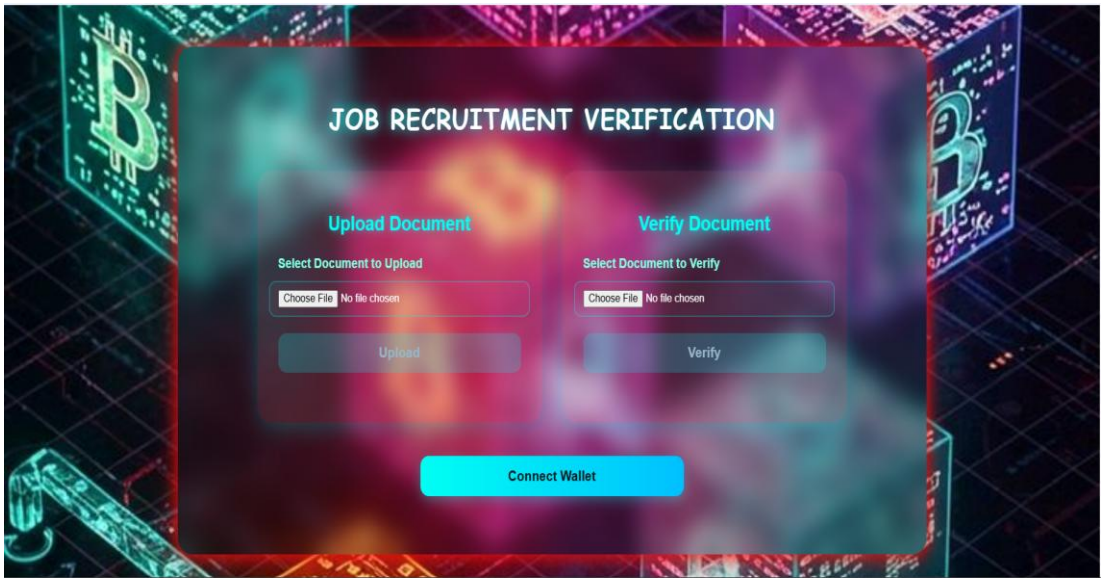


Fig12: Viewing the front end

9.3 ENSURING METAMASK WALLET CONNECTION BEFORE TRANSACTION

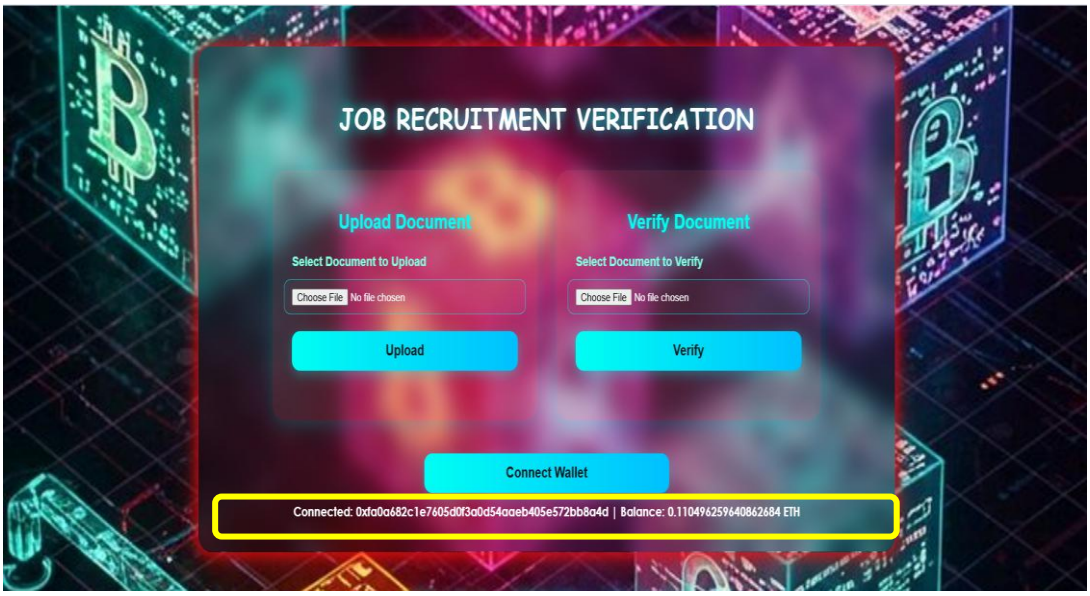


Fig13: Ensuring MetaMask connection

9.4 UPLOAD THE DOCUMENTS

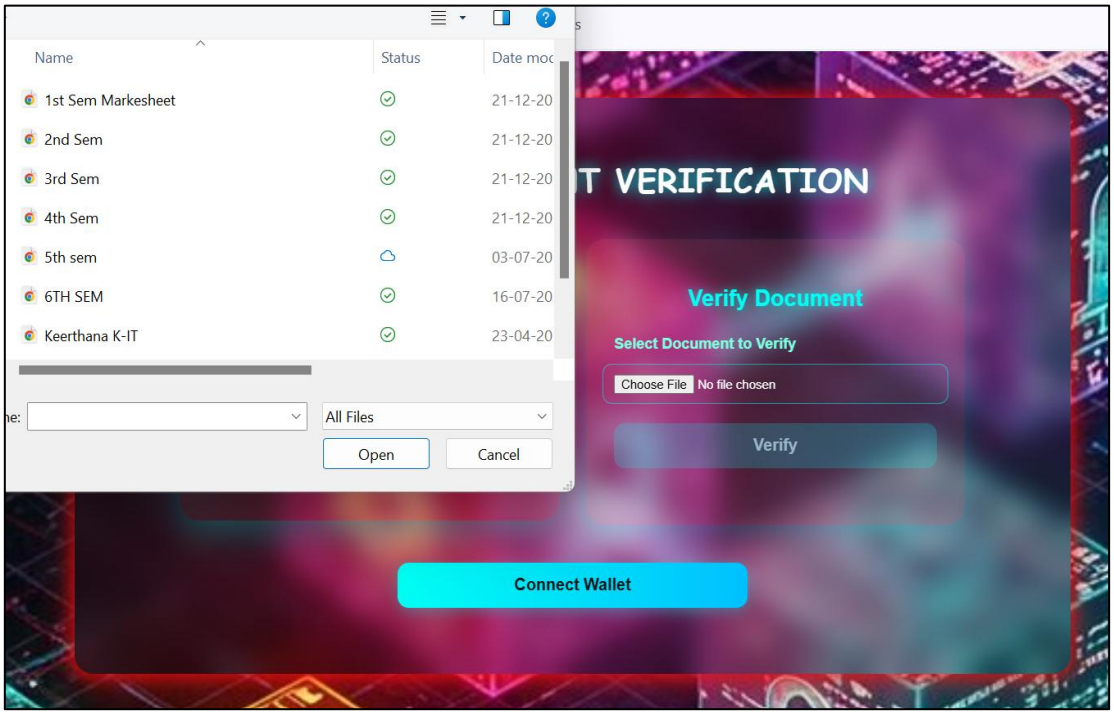


Fig14: Upload the Documents

9.5 SIGNING THE TRANSACTIONS VIA METAMASK

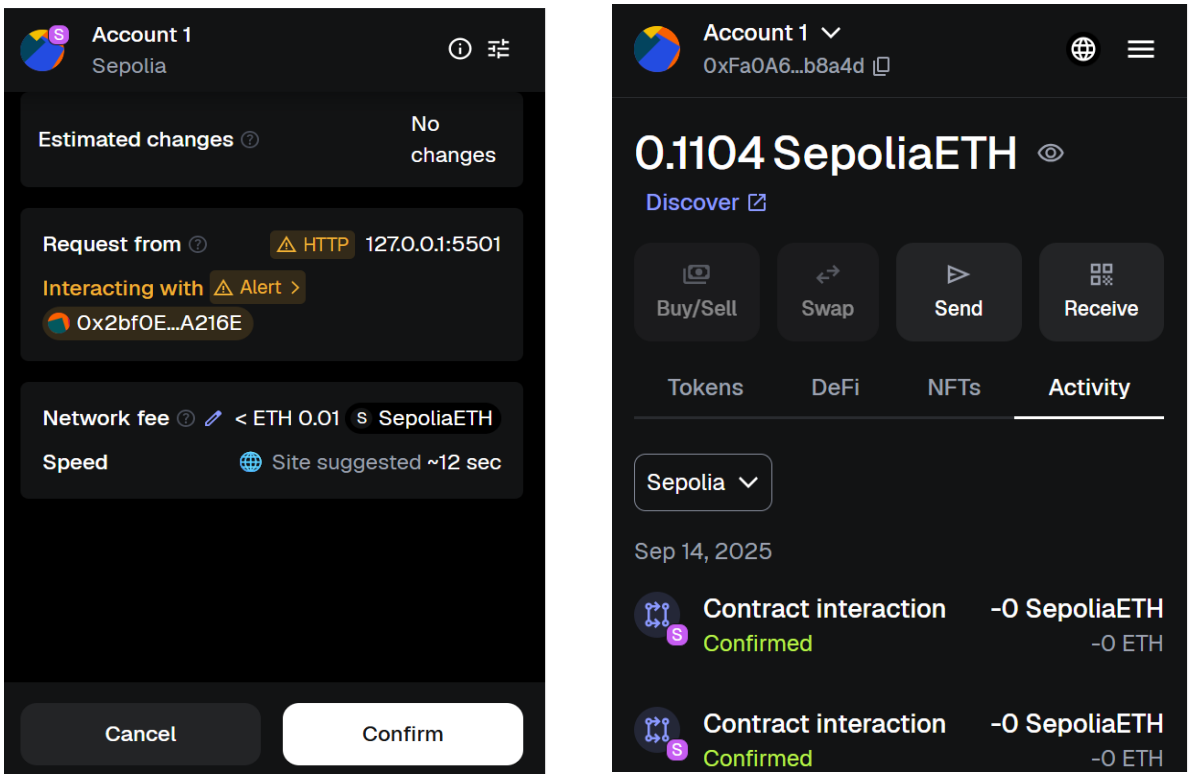


Fig15: MetaMask Transaction Confirmation

9.6 SUCCESSFUL VERIFICATIONS

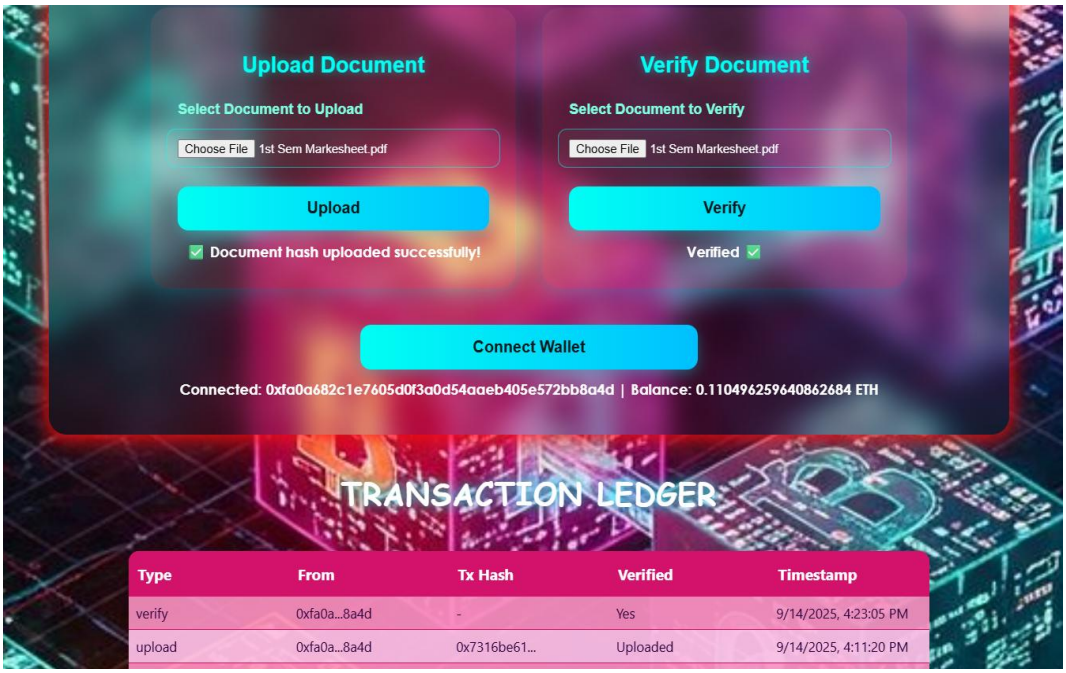
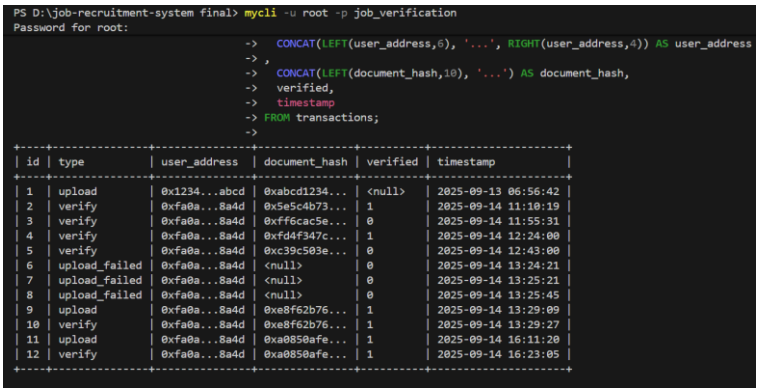


Fig16: Successful Upload and Generation of hash

9.7 TRANSACTION DETAILS IN LEDGER

Fig17: View of Transaction ledger in frontend

9.8 DATABASE STATUS OF THE LEDGER



id	type	user_address	document_hash	verified	timestamp
1	upload	0x1234...abcd	0xabcd1234...	<null>	2025-09-13 06:56:42
2	verify	0xfa0a...8a4d	0x5e5c4b73...	1	2025-09-14 11:10:19
3	verify	0xfa0a...8a4d	0xff6cac5e...	0	2025-09-14 11:55:31
4	verify	0xfa0a...8a4d	0xfd4f347c...	1	2025-09-14 12:24:00
5	verify	0xfa0a...8a4d	0xc39c503e...	0	2025-09-14 12:43:00
6	upload_failed	0xfa0a...8a4d	<null>	0	2025-09-14 13:24:21
7	upload_failed	0xfa0a...8a4d	<null>	0	2025-09-14 13:25:21
8	upload_failed	0xfa0a...8a4d	<null>	0	2025-09-14 13:25:45
9	upload	0xfa0a...8a4d	0xe8f62b76...	1	2025-09-14 13:29:09
10	verify	0xfa0a...8a4d	0xe8f62b76...	1	2025-09-14 13:29:27
11	upload	0xfa0a...8a4d	0xa0850afe...	1	2025-09-14 16:11:20
12	verify	0xfa0a...8a4d	0xa0850afe...	1	2025-09-14 16:23:05

Fig18: View of ledger from database

10. CONCLUSION

The project successfully leverages blockchain technology to provide secure and tamper-proof document verification for job recruitment. It enhances trust by enabling immutable storage of document hashes and seamless verification via smart contracts. The integration of MetaMask and a responsive frontend ensures user-friendly interaction and reliable authentication. Future enhancements can expand functionality with decentralized storage and advanced access control to strengthen recruitment transparency.

11. FUTURE ENHANCEMENTS

- Full Document Storage on IPFS: Integrate decentralized document storage to store actual files, referencing only hashes on-chain.
- Access Control: Role-based permissions to restrict who can upload or verify documents.
- Multi-Blockchain Support: Add compatibility for other blockchain networks (e.g., Polygon, Binance Smart Chain) for scalability and cost savings.
- Mobile Application: Develop native mobile apps for seamless user experience on smartphones, Enhanced Security: Add multi-factor authentication and encryption for sensitive metadata.
- Smart Contract Upgradability: Implement proxy patterns to allow upgrading and patching smart contracts.

12. REFERENCES

1. Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the Internet of Things. IEEE Access, 4, 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339
2. Al-Turjman, F., & Malekloo, A. (2019). Smart contracts in IoT: Opportunities and challenges. Computer Communications, 136, 146–156. https://doi.org/10.1016/j.comcom.2019.01.014
3. Ethereum Foundation. Ethereum Documentation. https://ethereum.org/en/developers/docs/
4. Ganache. Personal Blockchain for Ethereum Development. Truffle Suite. https://trufflesuite.com/ganache/
5. Mallick, M. & Sengupta, A. & Ingawale, Satyajit & Aljapurkar, Aditi. (2022). Using blockchain technology for recruitment effectiveness in industry 4.0. Prayukti – Journal of Management Applications. 02. 52-57. 10.52814/PJMA.2022.2107.
6. MetaMask. MetaMask Documentation. ConsenSys. https://docs.metamask.io
7. Marella, Venkata & Vijayan, Anoop. (2020). Document Verification using Blockchain for Trusted CV information.
8. Truffle Suite. Truffle Documentation. https://trufflesuite.com/docs/
9. IPFS Project. IPFS Documentation (InterPlanetary File System). https://docs.ipfs.tech/
10. MySQL Documentation. https://dev.mysql.com/doc/