|  |
| --- |
| **Title: Prototype Implementation for the MiniProject.** |

**Expected Outcome of Experiment:**

**CO3: Implement and prototype creation for the specified application.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

<https://www.researchgate.net/figure/Blockchain-based-e-voting-system-architecture_fig2_357827345>

<https://www.investopedia.com/terms/b/blockchain.asp>

<https://core.ac.uk/download/pdf/155779036.pdf>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Introduction:**

[System Implementation](https://www.sebokwiki.org/wiki/Implementation_(glossary)) uses the structure created during [architectural design](https://www.sebokwiki.org/wiki/Architecting_(glossary)) and the results of [system analysis](https://www.sebokwiki.org/wiki/System_Analysis) to construct [system elements](https://www.sebokwiki.org/wiki/System_Element_(glossary)) that meet the [stakeholder requirements](https://www.sebokwiki.org/wiki/Stakeholder_Requirement_(glossary)) and [system requirements](https://www.sebokwiki.org/wiki/System_Requirement_(glossary)) developed in the early [life cycle](https://www.sebokwiki.org/wiki/Life_Cycle_(glossary)) phases. These system elements are then integrated to form intermediate [aggregates](https://www.sebokwiki.org/wiki/Aggregate_(glossary)) and finally the complete [system](https://www.sebokwiki.org/wiki/System-of-Interest_(glossary))

**DIGIPOLLS**

**An e voting system using blockchain technology.**

1. **Modules Description:**

|  |  |
| --- | --- |
| **Module** | **Name, Definition, purpose** |
| Voter | Name: Voter Definition: The Voter module is a software application designed for individuals eligible to participate in the voting process. It facilitates secure authentication, access to candidate information, and verification of vote status. Shape: Software application Purpose:   * Identifier: Voter * Name: Voter Module * Description: This module allows eligible voters to securely authenticate themselves, access information about candidates, and verify their vote status. * Type: Software application |
| Admin | Name: Admin Definition: The Admin module is a software application designed for administrators or election officials responsible for managing and overseeing the voting system. It provides functionalities for system configuration and monitoring. Shape: Software application Purpose:   * Identifier: Admin * Name: Admin Module * Description: This module empowers administrators to configure system parameters and monitor the voting process. * Type: Software application |

**Activity: Voter Module**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Voting Page</title>

<link rel="stylesheet" type="text/css" href="../css/index.css">

</head>

<body>

<div id="head" class="text-center">

<h1 align="center">Decentralized Voting Using Ethereum Blockchain</h1>

<h2 align="center">Welcome for Voting</h2>

<h3 align="center"><p>Voting Dates: <span id="dates"></span></p></h3>

<br/>

</div>

<div id="candidate" class="container">

<table class="table text-center">

<thead>

<tr>

<th>Name</th>

<th>Party</th>

<th>Total Vote</th>

</tr>

</thead>

<tbody id="boxCandidate"></tbody>

</table>

<br/>

<div id="vote">

<p>Please select one of the candidates and click the vote button.</p>

<button id="voteButton" class="btn btn-primary" onclick="App.vote()" disabled>Vote</button>

<div id="msg"></div>

<div id="vote-box"></div>

</div>

</div>

<div id="account">

<p id="accountAddress" class="text-center"></p>

</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"></script>

<script src="../dist/app.bundle.js" type="module"></script>

</body>

</html>

Input:

* Unique credentials for voter authentication
* Candidate information request (e.g., candidate names, parties)
* Vote verification request

Output:

* Authentication confirmation
* Candidate information (names, parties, etc.)
* Vote status verification confirmation

**Activity: Admin module**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Admin portal</title>

<link rel="stylesheet" type="text/css" href="../css/admin.css">

</head>

<body>

<div id="head" class="text-center">

<h1 align="center" style="color: azure;">Decentralized Voting Using Ethereum Blockchain</h1>

<br/>

</div>

<div class="container">

<legend>Add Candidate</legend>

<table class="table text-center">

<tr>

<th>Name </th><td> <input id="name" type="text" name="name" placeholder ="Candidates name" autocomplete="off"></td>

<th>Party</th><td><input id="party" type="text" name="party" placeholder ="Candidates party"></td>

</tr>

</table>

<input id="addCandidate" type="submit" name="submit" value="Add Candidate">

<p id="Aday"></p>

</div>

<div class="container">

<legend>Define Voting Dates</legend>

<table class="table text-center">

<tr>

<th>Start date</th><td><input id="startDate" type="date" name=""></td>

<th>End date</th><td> <input id="endDate" type="date" name="" ></td>

</tr>

</table>

<input id="addDate" type="submit" name="submit" value="Define Dates">

<p id="Aday"></p>

</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"></script>

<script src="../dist/app.bundle.js" type="module"></script>

</body>

</html>

Input:

* System parameters (voting period, candidate registration, etc.)
* Candidate verification request
* Voting process initiation request

Output:

* System parameter configuration confirmation
* Candidate verification confirmation
* Voting process initiation confirmation

1. **Integration:**

Integration is a critical step within software implementation and it involves migrating data, compiling all modules in one system. With proper integrations, one can expect the project to be producing expected output.

Integration Strategy:

* Data Migration:
  + Ensure smooth migration of voter data, including voter credentials and vote status, from existing systems to the new voting system.
  + Verify data integrity during migration to prevent any loss or corruption of voter information.
* Module Compilation:
  + Compile all individual modules, including Voter and Admin, into a unified system.
  + Ensure compatibility between different modules to avoid conflicts or errors during integration.

1. **Implementation details**
2. MIGRATIONS.SOL

pragma solidity ^0.5.15;

contract Migrations {

address public owner;

uint public last\_completed\_migration;

modifier restricted() {

require(msg.sender == owner, "Access restricted to owner");

\_;

}

constructor() public {

owner = msg.sender;

}

function setCompleted(uint completed) public restricted {

last\_completed\_migration = completed;

}

function upgrade(address new\_address) public restricted {

Migrations upgraded = Migrations(new\_address);

upgraded.setCompleted(last\_completed\_migration);

}

}

1. Voting.sol

pragma solidity ^0.5.15;

contract Voting {

struct Candidate {

uint id;

string name;

string party;

uint voteCount;

}

mapping (uint => Candidate) public candidates;

mapping (address => bool) public voters;

uint public countCandidates;

uint256 public votingEnd;

uint256 public votingStart;

function addCandidate(string memory name, string memory party) public returns(uint) {

countCandidates ++;

candidates[countCandidates] = Candidate(countCandidates, name, party, 0);

return countCandidates;

}

function vote(uint candidateID) public {

require((votingStart <= now) && (votingEnd > now));

require(candidateID > 0 && candidateID <= countCandidates);

//daha önce oy kullanmamıs olmalı

require(!voters[msg.sender]);

voters[msg.sender] = true;

candidates[candidateID].voteCount ++;

}

function checkVote() public view returns(bool){

return voters[msg.sender];

}

function getCountCandidates() public view returns(uint) {

return countCandidates;

}

function getCandidate(uint candidateID) public view returns (uint,string memory, string memory,uint) {

return (candidateID,candidates[candidateID].name,candidates[candidateID].party,candidates[candidateID].voteCount);

}

function setDates(uint256 \_startDate, uint256 \_endDate) public{

require((votingEnd == 0) && (votingStart == 0) && (\_startDate + 1000000 > now) && (\_endDate > \_startDate));

votingEnd = \_endDate;

votingStart = \_startDate;

}

function getDates() public view returns (uint256,uint256) {

return (votingStart,votingEnd);

}

}

1. Main.py

# Import required modules

import dotenv

import os

import mysql.connector

from fastapi import FastAPI, HTTPException, status, Request

from fastapi.middleware.cors import CORSMiddleware

from fastapi.encoders import jsonable\_encoder

from mysql.connector import errorcode

import jwt

# Loading the environment variables

dotenv.load\_dotenv()

# Initialize the todoapi app

app = FastAPI()

# Define the allowed origins for CORS

origins = [

"<http://localhost:8080>",

"<http://127.0.0.1:8080>",

]

# Add CORS middleware

app.add\_middleware(

CORSMiddleware,

allow\_origins=origins,

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# Connect to the MySQL database

try:

cnx = mysql.connector.connect(

user=os.environ['MYSQL\_USER'],

password=os.environ['MYSQL\_PASSWORD'],

host=os.environ['MYSQL\_HOST'],

database=os.environ['MYSQL\_DB'],

)

cursor = cnx.cursor()

except mysql.connector.Error as err:

if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:

print("Something is wrong with your user name or password")

elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:

print("Database does not exist")

else:

print(err)

# Define the authentication middleware

async def authenticate(request: Request):

try:

api\_key = request.headers.get('authorization').replace("Bearer ", "")

cursor.execute("SELECT \* FROM voters WHERE voter\_id = %s", (api\_key,))

if api\_key not in [row[0] for row in cursor.fetchall()]:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Forbidden"

)

except:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Forbidden"

)

# Define the POST endpoint for login

@app.get("/login")

async def login(request: Request, voter\_id: str, password: str):

await authenticate(request)

role = await get\_role(voter\_id, password)

# Assuming authentication is successful, generate a token

token = jwt.encode({'password': password, 'voter\_id': voter\_id, 'role': role}, os.environ['SECRET\_KEY'], algorithm='HS256')

return {'token': token, 'role': role}

# Replace 'admin' with the actual role based on authentication

async def get\_role(voter\_id, password):

try:

cursor.execute("SELECT role FROM voters WHERE voter\_id = %s AND password = %s", (voter\_id, password,))

role = cursor.fetchone()

if role:

return role[0]

else:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Invalid voter id or password"

)

except mysql.connector.Error as err:

print(err)

raise HTTPException(

status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,

detail="Database error"

)

1. Login.js

const loginForm = document.getElementById('loginForm');

loginForm.addEventListener('submit', (event) => {

event.preventDefault();

const voter\_id = document.getElementById('voter-id').value;

const password = document.getElementById('password').value;

const token = voter\_id;

const headers = {

'method': "GET",

'Authorization': `Bearer ${token}`,

};

fetch(`http://127.0.0.1:8000/login?voter\_id=${voter\_id}&password=${password}`, { headers })

.then(response => {

if (response.ok) {

return response.json();

} else {

throw new Error('Login failed');

}

})

.then(data => {

if (data.role === 'admin') {

console.log(data.role)

localStorage.setItem('jwtTokenAdmin', data.token);

window.location.replace(`http://127.0.0.1:8080/admin.html?Authorization=Bearer ${localStorage.getItem('jwtTokenAdmin')}`);

} else if (data.role === 'user'){

localStorage.setItem('jwtTokenVoter', data.token);

window.location.replace(`http://127.0.0.1:8080/index.html?Authorization=Bearer ${localStorage.getItem('jwtTokenVoter')}`);

}

})

.catch(error => {

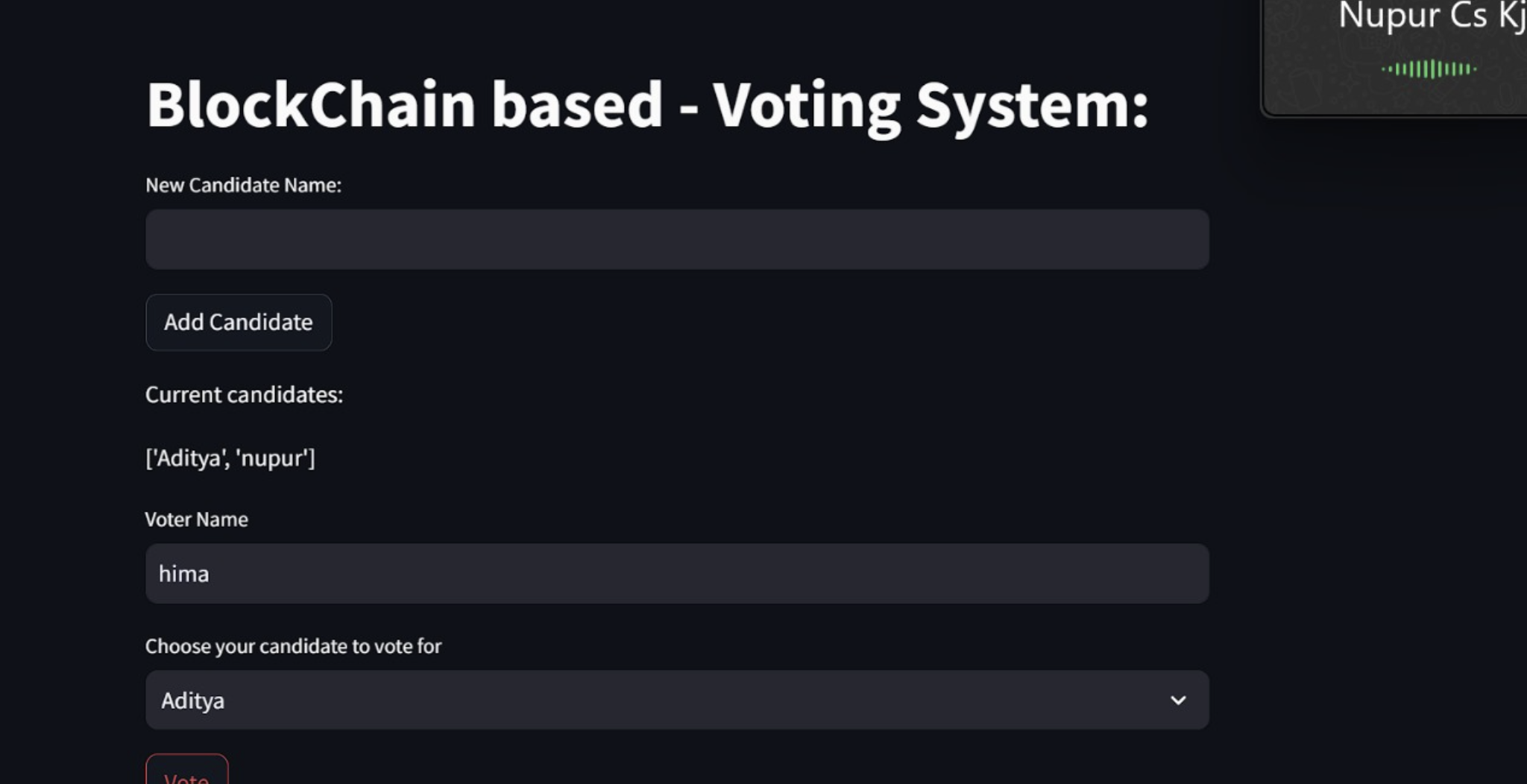
console.error('Login failed:', error.message);

});

});

1. **Implementation Screenshots:**

PROTOTYPE : Mid Term evaluation Implementation



MODULE OUTPUTS

The Voter module is a software application designed for individuals eligible to participate in the voting process. It facilitates secure authentication, access to candidate information, and verification of vote status

The Admin module is a software application designed for administrators or election officials responsible for managing and overseeing the voting system. It provides functionalities for system configuration and monitoring

1. **Git link**

[**https://github.com/NupurItIs/E-voting-system-.git**](https://github.com/NupurItIs/E-voting-system-.git)

**Conclusion:**

Successfully implemented the Voters and the admins modules .