**Full Stack Development with MERN**

**BATCH 9(CSE-C) Project Documentation format**

1. Introduction

2. Project Overview

3. Architecture

4. Setup Instructions

5. Folder Structure

6. Running the Application

7. API Documentation

8. Authentication

9. User Interface

10. Testing

11. Screenshots or Demo

12. Known Issues

13. Future Enhancements

1.Introduction:

**1.1 PROJECT TITLE**: iVote - Your Gateway to Transparent Decision-Making!

iVote is the ultimate online voting application designed to streamline the democratic process organizations. This user-friendly platform allows organizations to create customized voting polls, empowering registered users to cast their votes and even participate as candidates, fostering a culture of engagement and transparency.

Effortlessly create and manage voting polls tailored to your organization's needs with iVote. The intuitive interface within ensures a seamless experience for both administrators and voters. Define voting parameters, set deadlines, and configure options with ease, putting the power of decision-making in your hands.

Registered users can access iVote to cast their votes securely and conveniently from anywhere. With the option to apply as a candidate, individuals can actively participate in the decision-making process, fostering a sense of community involvement and empowerment.

iVote's robust security features ensure the integrity of the voting process. Built-in authentication mechanisms and encryption protocols safeguard the confidentiality and accuracy of every vote, providing peace of mind for both administrators and voters.

Stay informed with iVote's comprehensive reporting tools. Gain valuable insights into voting patterns, participation rates, and candidate performance. Real-time analytics empower organizations to make data-driven decisions and enhance the overall democratic experience.

As an administrator, easily manage candidate applications, monitor voting progress, and generate detailed reports. iVote's flexibility allows organizations to adapt the platform to their unique needs, creating a tailored and effective voting environment.

Join the iVote community today and revolutionize the way your organization makes decisions. Download iVote now to unlock a transparent and efficient voting experience, empowering your members to actively contribute to the future of your organization. Take the lead in democratic decision-making with iVote!

**1.2 TEAM MEMBERS:**

Bandi. Likhitha

Thota. Leela Durga

Tiyyagurra. Sadvika

Bulla. Dhanush

Posa. Leela Krishna

**Roles of Team Members:**

**Voter (User):**

* Registration: Complete the registration process to become a verified user of iVote.
* Profile Maintenance: Keep personal information and preferences up-to-date for accurate identification and communication.
* Participation: Actively engage in voting processes by casting informed and responsible votes.
* Candidate Application: If interested, apply as a candidate for relevant polls by providing necessary details and qualifications.
* Secure Voting: Ensure the confidentiality and security of the voting process by adhering to the guidelines provided by iVote.

**Organization:**

* Account Creation: Create and manage an organizational account on iVote.
* Voting Poll Setup: Configure and customize voting polls based on the organization's requirements, specifying options, deadlines, and other relevant parameters.
* User Management: Administer user accounts, ensuring that only eligible and verified individuals are registered participants.
* Candidate Management: Oversee candidate applications, validate qualifications, and approve or reject candidates accordingly.
* Security Oversight: Implement and monitor security measures to safeguard the integrity of the voting process.
* Data Analysis: Utilize iVote's reporting tools to analyze voting patterns, participation rates, and candidate performance for informed decision-making.

**Admin:**

* System Administration: Manage the overall functionality, security, and performance of the iVote platform.
* User Support: Provide support to users, addressing queries, technical issues, and ensuring a positive user experience.
* Security Protocols: Implement and update security protocols to protect the platform from potential threats and ensure the confidentiality of votes.
* Poll Monitoring: Keep track of ongoing voting polls, ensuring they adhere to set deadlines and resolving any issues that may arise.

**2. PROJECT OVERVIEW**

The iVote project aims to revolutionize the voting process by creating a secure and user-friendly digital platform. This platform will allow voters to cast their ballots from anywhere, promoting accessibility for individuals with disabilities and those living outside their home country. By providing a straightforward voting interface, the project ensures that all users can navigate the voting process easily.

Security is a top priority for iVote, incorporating measures like multi-factor authentication and data encryption to protect voter information and maintain the integrity of the election. Additionally, the platform will offer real-time ballot tracking, allowing voters to monitor their votes from submission to counting. This level of transparency is designed to build public trust and confidence in the electoral process.

The implementation of eVote will involve several phases, including research, design, development, and testing, ensuring a comprehensive approach to create a reliable platform. With an emphasis on technology and user experience, iVote seeks to modernize voting, making it more efficient while ensuring that every voice is heard and counted in a secure environment.

**2.1 Purpose**

The purpose of a digital voting platform is to enhance the electoral process by making it more accessible, secure, and efficient. Here are the key goals:

1. **Accessibility**: To enable voters to cast their votes easily from any location, including those with disabilities or those residing outside their home country, thereby increasing voter participation.
2. **Security**: To protect voter data and ensure the integrity of the voting process through robust security measures, such as encryption and multi-factor authentication.
3. **Transparency**: To provide real-time tracking and reporting of votes, helping to build public trust and confidence in the electoral system.
4. **Efficiency**: To streamline the voting process, reducing the time and resources needed for traditional voting methods, which can lead to faster election results.
5. **User Experience**: To create an intuitive and straightforward interface that guides users through the voting process, making it accessible to all demographics.

**ii) Goals of a Digital Voting Platform**

1.**Increase Voter Participation**:

Make voting more accessible to a wider audience, including individuals with disabilities, those living abroad, and younger voters who may prefer digital solutions.

2.**Enhance Security**:

Implement advanced security measures to protect voter information, prevent fraud, and ensure the integrity of the voting process.

3. **Ensure Transparency**:

Provide real-time tracking of ballots and transparent reporting of results to build public trust in the electoral system.

4**. Streamline the Voting Process**:

Reduce the time and resources required for traditional voting methods, enabling quicker and more efficient election outcomes**.**

5. **Improve User Experience:**

Create an intuitive and user-friendly interface that guides voters through the process, making it straightforward and accessible for everyone.

6. **Facilitate Remote Voting:**

Allow voters to cast their ballots securely from any location, especially beneficial during emergencies or for those unable to attend polling places.

7. **Support Multiple Languages**:

Cater to diverse populations by offering the platform in multiple languages, ensuring inclusivity for all voters.

8. **Enable Auditability:**

Maintain a clear and traceable record of all voting activities for auditing purposes, enhancing accountability in the electoral process.

9**. Adapt to Different Election Types:**

Provide flexibility to accommodate various types of elections, from local to national, as well as different voting methods (e.g., ranked choice, proportional representation).

10.**Promote Education and Awareness:**

Educate voters about the digital voting process to minimize confusion and encourage informed participation in elections.

**Features of a Digital Voting Platform**

1.User Registration

2. Intuitive Voting Interface

3. Ballot Selection

4. Real-Time Ballot Tracking

5. Multi-Factor Authentication

6. Accessibility Options

7. Data Encryption

8. Audit Trail

9.Secure Results Reporting

10.Mobile Compatibility

11.Educational Resources:

12. Support and Helpdesk:

13.Feedback Mechanism:

**Functionalities:**

1.Voter Registration: Allow users to create accounts and verify their identities through various methods, such as ID uploads or biometric verification.

2.Voting Process: Provide an easy-to-follow voting interface where users can select candidates or measures and submit their ballots securely.

3.Ballot Review: Enable voters to review their selections before final submission to prevent mistakes.

4.Real-Time Status Tracking: Offer users the ability to track their ballot status, from submission to confirmation of receipt and counting.

5.Multi-Language Support: Provide the platform in multiple languages to accommodate diverse populations.

6.Accessibility Features: Include options for text-to-speech, screen reader compatibility, and other assistive technologies to ensure all voters can participate.

7.Secure Authentication: Implement multi-factor authentication (MFA) to enhance security during the login and voting process.

8.Data Encryption: Ensure all data, including personal information and votes, is encrypted both in transit and at rest to protect against unauthorized access.

9.Results Compilation and Reporting: Automatically compile and report election results in real-time, providing transparency and timely information to the public.

10.Audit and Verification: Maintain a secure audit trail for all voting activities, allowing for post-election audits and verification processes.

11.User Support System: Offer a helpdesk or chat support feature for voters needing assistance or having questions about the voting process.

12.Feedback Collection: Include mechanisms for voters to provide feedback on their experience, which can be used for future improvements.

13.Emergency Protocols: Implement contingency plans for technical failures or security breaches, including backup voting methods if necessary.

14.Customizable Voting Methods: Allow for different voting methods, such as ranked choice or proportional representation, depending on the election type.

15.Educational Resources: Provide access to materials that explain the voting process, candidates, and issues to help inform voters before they cast their ballots.

**ARCHITECTURE:**

A diagram of a login

Description automatically generated

**3.1 FRONTEND**

Here’s a detailed overview of the front-end architecture for a digital voting platform using React:

Overall Structure

1. **Single Page Application (SPA):**

Utilize React to create a SPA for a seamless user experience, allowing users to navigate without full page reloads.

2. **Folder Structure**

3.**Routing**

- React Router:

- Set up routes in `App.js` to navigate between different pages (e.g., Home, Login, Register, Vote).

- Example:

```jsx

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/register" element={<RegisterForm />} />

<Route path="/verify-otp" element={<VerifyOTP />} />

<Route path="/login" element={<LoginForm />} />

<Route path="/homepage" element={<ViewParticipants />} />

<Route path='/admin' element={<AdminPanel />}>

<Route path='add-participant' element={<AddParticipant />} />

<Route path='edit-participant/:id' element={<AddParticipant />} />

<Route path='view-participants' element={<ParticipantsView />} />

<Route path='user-table' element={<UserTable />} />

<Route path='live-view-leaderboard' element={<LiveViewLeaderboard />} />

<Route path='dashboard' element={<Dashboard />} />

</Route>

</Router>

**4**. **State Management**

Context API:

- Create context files (e.g., `AuthContext.js`) for global state management like user authentication and voting status.

Local State:

- Use React’s built-in state management for form handling and local component state.

5. **Form Handling**

- Controlled Components:

- Implement controlled components for user inputs in forms (registration, voting).

-Validation:

- Use libraries like Formik and Yup for form validation

6. **API Integration**

Axios or Fetch:

- Create a service file (`api.js`) to manage API calls for user authentication, voting, and results retrieval.

- Error Handling:

- Handle API errors gracefully, providing feedback to users.

7. **Styling**

CSS Modules or Styled Components:

- Use CSS Modules for scoped styles or styled-components for dynamic styling.

-Responsive Design:

- Implement responsive design using Flexbox and Grid for a consistent experience across devices.

8. **Accessibility**

ARIA Roles:

- Use ARIA attributes to ensure the application is accessible to users with disabilities.

Keyboard Navigation:

Ensure all interactive elements can be navigated using a keyboard.

9. **Performance Optimization**

Code Splitting:

- Implement lazy loading of components using `React.lazy` and `Suspense` for improved load times.

10**. Testing**

- Unit Testing

- End-to-End Testing

11. **Deployment**

Static Site Hosting:

- Deploy the application on platforms like Vercel, Netlify, or AWS S3 for scalable access and fast loading.

**Backend Architecture**

The back-end architecture of a digital voting platform is essential for managing data, ensuring security, and providing reliable services. Here’s a detailed overview of how this architecture can be structured:

**Overall Architecture**

**1. Microservices or Monolithic Architecture**: Depending on the scale, either a microservices architecture (for better scalability and maintainability) or a monolithic approach (for simpler applications) can be used.

**2. Technology Stack**

* **Programming Language**: Node.js, Python (Django/Flask), or Java (Spring Boot).
* **Database**:
  + Relational Database (PostgreSQL, MySQL) for structured data.
  + NoSQL Database (MongoDB) for flexibility and scalability.
* **API**: RESTful API or GraphQL for communication between the front-end and back-end.

3. **Folder Structure**

**4. Authentication and Authorization**

* **JWT (JSON Web Tokens)**: Use JWT for user authentication, providing secure access to the API.
* **OAuth2**: Implement OAuth2 for third-party integrations if needed (e.g., for social logins).

**5. Database Management**

* **User Management**: Store user information securely, including hashed passwords and roles (voter, admin).
* **Voting Data**: Record ballots and election data, ensuring integrity and security.
* **Audit Logs**: Maintain logs for actions taken (e.g., vote submissions) for transparency.

**6. Business Logic**

* **Vote Submission**: Handle logic for submitting votes, ensuring users can only vote once per election.
* **Ballot Tracking**: Provide functionality to track the status of ballots, from submission to counting.
* **Results Calculation**: Implement algorithms for counting votes and generating results based on different voting methods.

**API Development**

* **RESTful API Endpoints**: Create endpoints for user registration, authentication, voting, and retrieving results.
  + Example endpoints:
    - POST /api/auth/register
    - POST /api/auth/login
    - POST /api/vote
    - GET /api/results
* **Data Validation**: Use middleware for validating incoming data to ensure integrity.

**8. Security Measures**

* **Data Encryption**: Encrypt sensitive data both in transit (using HTTPS) and at rest (in the database).
* **Rate Limiting**: Implement rate limiting to prevent abuse of the API.
* **Input Sanitization**: Sanitize user inputs to prevent SQL injection and XSS attacks.

**9. Performance Optimization**

* **Caching**: Use caching (e.g., Redis) for frequently accessed data, like election results, to reduce database load.
* **Load Balancing**: Implement load balancing to distribute traffic across multiple instances of the service.

**10. Monitoring and Logging**

* **Logging**: Use logging libraries (e.g., Winston, Morgan) to track requests and errors for troubleshooting.
* **Monitoring**: Utilize monitoring tools (e.g., Prometheus, Grafana) to track the health and performance of the application.

**11. Testing**

* **Unit Testing**: Implement unit tests for controllers and models to ensure functionality.
* **Integration Testing**: Test interactions between different components of the system.

**12. Deployment**

* **Cloud Hosting**: Deploy the application on cloud platforms like AWS, Azure, or Google Cloud for scalability.
* **CI/CD Pipeline**: Set up a continuous integration/continuous deployment pipeline for automated testing and deployment.

**DataBase**

Here’s a comprehensive design for a MongoDB database tailored for a digital voting platform, including collections, schemas, and relationships:

**1. Collections Overview**

1. **Admin**
2. **Config**
3. **Local**
4. **Voting**
5. **Audit Logs**

**2. Voting**

Customers:

**id**:ObjectId('6719c07322ee12d64fc76a51')

**first\_name**: "XXXXXXX"

**middle\_name**: "X"

**last\_name**: "XXX"

**email**: "XXXXXXXXXXXX123@gmail.com"

**phone**: "XXXXXXXXXX"

**photo**: "1729740915923.jpg"

**aadhar\_no**: XXXXXXXXXXXX

**voter\_id**: "XXXXXXXXX"

**age**: XX

**otp**: XXXX

**role**: "customer"

**status**: "active"

**createdAt**: 2024-10-24T03:35:15.930+00:00

**updatedAt**: 2024-10-24T03:35:15.930+00:00

**\_\_v**: 0

**Participants:**

"party\_name": "BJP",

"party\_leader": "Narendra Modi",

"party\_slogan": "Swatcha Bharath",

"party\_symbol": "1729680455584.png",

"participant\_photo": "1729680455586.jpeg",

"voting\_count": 1,

"role": "participant",

"createdAt":2024-10-23T07:02:24.387Z"

"updatedAt" : 2024-10-23T10:47:35.607Z"

"vote" :"6718a57a9bff78f6560ee0c3”

"\_\_v": 0

4. **Setup Installation**

Here are the key prerequisites for developing a full-stack application using Node JS, Express JS, MongoDB, React.js:

✔**Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

To get started with Node.js and npm, follow these installation steps:

1. Download Node.js:

Visit the official Node.js website at [nodejs.org](https://nodejs.org/) and download the latest version of Node.js for your operating system. Choose the LTS (Long-Term Support) version for stability.

2. Installation:

Run the Node.js installer and follow the on-screen instructions to install Node.js on your machine. The installer will also automatically install npm (Node Package Manager), which is used to install and manage Node.js packages.

3. Verify the Installation:

 Open a command prompt or terminal and run the following commands to verify that Node.js and npm are installed correctly:

node -v

   npm -v

These commands should display the versions of Node.js and npm installed on your machine without any errors.

✔**Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

To get started with Express.js, follow these installation steps:

1. Create a Project Directory:

Create a new directory for your Express.js project. Open a command prompt or terminal and navigate to the desired location where you want to create the project directory.

2. Initialize the Project:

Run the following command to initialize a new Node.js project and create a package.json file that will keep track of your project's dependencies:

 npm init

The `npm init` command will prompt you to provide project information such as the package name, version, and entry point file. You can press Enter to accept the default values.

3. Install Express.js:

Run the following command to install Express.js as a dependency for your project:

   npm install express

This command will download and install the latest version of Express.js and its dependencies into your project's `node\_modules` directory.

4. Create an Express.js App:

Create a file named `app.js` or `index.js` in your project directory and write the necessary code to create an Express.js application. You can refer to the Express.js documentation and various online resources to learn about building Express.js applications.

With Node.js and Express.js installed and set up, you can now start building powerful and scalable web applications. Feel free to explore the extensive documentation and online tutorials available for both technologies to enhance your development skills.

Happy coding with Node.js and Express.js!

✔**MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Express JS, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: https://www.mongodb.com/try/download/community

Installation instructions: <https://docs.mongodb.com/manual/installation/>

✔**React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

✔**HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✔**Database Connectivity**: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Express JS server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations

✔**Front-end Framework**: Utilize React Js to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard. For making better UI we have also used some libraries like material UI and bootstrap.

✔**Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Git: Download and installation instructions can be found at: https://git-scm.com/downloads

✔**Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from <https://code.visualstudio.com/download>

**Installation:**

• Navigate into the cloned repository directory:

cd  CODE

• Install the required dependencies by running the following commands:

cd ivote

npm install

Start the Development Server:

• To start the development server, execute the following command:

npm run dev

• The iVote app will be accessible at http://localhost:5173/

You have successfully installed and set up the iVote application on your local machine. You can now proceed with further customization, development, and testing as needed.

**5. FOLDER STRUCTURE**

1. **A screenshot of a computer

   Description automatically generated** A screenshot of a computer

   Description automatically generated

**A screenshot of a computer

Description automatically generated**

1. **Running the Application**

Commands to start the frontend and backend severs locally

**Backend:**

**cd backend**

**npm install**

**npm start**

**Frontend**

**cd ivote**

**npm install**

**npm run dev**

1. **API Application:**

#### 1. ****User Authentication****

##### **User Registration**

* **Endpoint**: POST /api/auth/register
* **Description**: Register a new user.
* **Request Body**:

{ "username": "string", "email": "string", "password": "string" }

**Response**:

* **201 Created**:

{

"message": "User registered successfully."

}

* **400 Bad Request**:

{

"error": "Email already exists."

}

##### **1.2. User Login**

* **Endpoint**: POST /api/auth/login
* **Description**: Authenticate a user and return a JWT.
* **Request Body**:

{

"email": "string",

"password": "string"

}

* **Response**:
  + **200 OK**:

{

"token": "JWT\_TOKEN",

"expiresIn": 3600

}

* + **401 Unauthorized**:

{

"error": "Invalid credentials."

}

#### 2. ****Elections Management****

##### **2.1. Create Election**

* **Endpoint**: POST /api/elections
* **Description**: Create a new election (admin only).
* **Request Body**:

{

"title": "string",

"description": "string",

"startDate": "ISODate",

"endDate": "ISODate"

}

* **Response**:
  + **201 Created**:

{

"message": "Election created successfully.",

"electionId": "ObjectId"

}

##### **2.2. Get All Elections**

* **Endpoint**: GET /api/elections
* **Description**: Retrieve a list of all elections.
* **Response**:
  + **200 OK**:

[

{

"\_id": "ObjectId",

"title": "string",

"description": "string",

"startDate": "ISODate",

"endDate": "ISODate",

"status": "string"

}

]

##### **2.3. Get Election by ID**

* **Endpoint**: GET /api/elections/:id
* **Description**: Retrieve details of a specific election.
* **Response**:
  + **200 OK**:

{

"\_id": "ObjectId",

"title": "string",

"description": "string",

"startDate": "ISODate",

"endDate": "ISODate",

"status": "string"

}

* + **404 Not Found**:

{

"error": "Election not found."

}

#### 3. ****Voting Management****

##### **3.1. Cast Vote**

* **Endpoint**: POST /api/vote
* **Description**: Submit a vote for a candidate.
* **Headers**:
  + Authorization: Bearer JWT\_TOKEN
* **Request Body**:

{

"electionId": "ObjectId",

"candidateId": "ObjectId"

}

* **Response**:
  + **201 Created**:

{

"message": "Vote submitted successfully."

}

* + **400 Bad Request**:

{

"error": "You have already voted in this election."

}

##### **3.2. Get Vote Status**

* **Endpoint**: GET /api/vote/status
* **Description**: Check if a user has voted in a specific election.
* **Headers**:
  + Authorization: Bearer JWT\_TOKEN
* **Response**:
  + **200 OK**:

{

"hasVoted": true/false

}

#### 4. ****Candidate Management****

##### **4.1. Add Candidate**

* **Endpoint**: POST /api/candidates
* **Description**: Add a candidate to an election (admin only).
* **Request Body**:

{

"name": "string",

"electionId": "ObjectId",

"party": "string",

"bio": "string"

}

* **Response**:
  + **201 Created**:

{

"message": "Candidate added successfully.",

"candidateId": "ObjectId"

}

##### **4.2. Get Candidates by Election**

* **Endpoint**: GET /api/candidates/election/:electionId
* **Description**: Retrieve all candidates for a specific election.
* **Response**:
  + **200 OK**:

[

{

"\_id": "ObjectId",

"name": "string",

"party": "string",

"bio": "string"

}

]

#### 5. ****Audit Logs****

##### **5.1. Get Audit Logs**

* **Endpoint**: GET /api/audit-logs
* **Description**: Retrieve audit logs (admin only).
* **Response**:
  + **200 OK**:

[

{

"\_id": "ObjectId",

"action": "string",

"userId": "ObjectId",

"timestamp": "ISODate",

"details": "string"

}

]

1. **Authentication**

Authentication and authorization are critical components of the digital voting platform, ensuring that users are who they claim to be and that they have the appropriate permissions to perform actions. Here’s how these processes are handled:

#### 1. ****Authentication****

Authentication verifies the identity of users attempting to access the platform. The following methods and mechanisms are employed:

##### **a.** **User Registration**

* Users register by providing essential information such as username, email, and password. The password is securely hashed (using libraries like bcrypt) before being stored in the database to protect user credentials.

##### **b**. **User Login**

* When a user logs in, their credentials are validated against the database.
* Upon successful authentication, a **JSON Web Token (JWT)** is generated. This token encodes user information (like user ID and role) and is signed with a secret key.

##### **c.** **Token-Based Authentication**

* **JWT Generation**: After login, the server sends a JWT back to the client. The token contains claims that represent the user’s identity and permissions.
* **Token Structure**: The JWT typically includes three parts: header, payload, and signature. The payload includes user-specific data and expiration information.
* **Storage**: The client stores this token (usually in local storage or as an HttpOnly cookie) for use in subsequent requests.

##### **d**. **Token Validation**

* For each request to protected routes, the client must include the JWT in the Authorization header as follows:

makefile

Copy code

Authorization: Bearer JWT\_TOKEN

* The server verifies the token's authenticity and expiration. If valid, the user is granted access; if not, the request is denied.

#### 2. ****Authorization****

Authorization determines whether a user has permission to perform a specific action or access certain resources. The following methods are utilized:

##### a. **Role-Based Access Control (RBAC)**

* Users are assigned roles (e.g., "voter," "admin"). The system checks these roles to determine access rights.
* For example, only users with the "admin" role can create or manage elections and candidates, while regular voters can only cast votes and view elections.

##### **b.** **Middleware for Authorization**

* Middleware functions are employed to check user permissions based on their roles before processing requests. If a user lacks the necessary permissions, the middleware sends a 403 Forbidden response.

##### **c.** **Audit Logging**

* Every action taken by users (e.g., vote submission, election creation) is logged in an audit log. This provides accountability and transparency, allowing administrators to monitor user actions.

#### 3. ****Session Management****

While the platform primarily relies on stateless JWTs for authentication, session management can be enhanced using the following methods if needed:

##### **a**. **Session Expiration**

* Tokens are issued with a defined expiration time (e.g., 1 hour). After expiration, users must re-authenticate to receive a new token.

##### **b. Refresh Tokens**

* For extended sessions, refresh tokens can be implemented. After the JWT expires, the client can use a refresh token to obtain a new access token without requiring the user to log in again.

##### **c.** **Logout Functionality**

* Users can explicitly log out, which may involve invalidating the JWT or removing it from client storage, ensuring it cannot be reused.

**4. Security Measures**

* **HTTPS**: All API requests are made over HTTPS to secure data in transit.
* **Input Validation**: User inputs are validated and sanitized to protect against injection attacks.
* **Rate Limiting**: Implemented on authentication endpoints to prevent brute-force attacks.
* **Two-Factor Authentication (optional)**: Additional security can be added through optional 2FA for user accounts.

1. **User Interface**

Home page: This is the project home page, where users can participate in voting.

A group of people standing together

Description automatically generated

Admin login page: Here, the admin can enter their email and voter ID to log in to the site.

A screenshot of a computer

Description automatically generated

Voter registration page: This is the voter registration form. By filling it out, the voter will receive an OTP to register their voter details.A screenshot of a computer

Description automatically generated

1. **Testing**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Types of Tests**

**Unit testing :**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing :**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing :**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Functional testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested :**

* Verify that the entries are of the correct format.
* No duplicate entries should be allowed.
* All links should take the user to the correct page**.**

1. **Screenshots**

Admin dashboard: After logging in, the admin can view the dashboard, where they can see voters, participants, vote counts, and participant details.

A screenshot of a chat

Description automatically generated

Add participant form: The admin can add new participants by entering the following details: party name, participant name, party symbol, participant photo, and the party's slogan.

A screenshot of a computer

Description automatically generated

View participants form: Here, the admin can view the details of the added participants as shown below.

A screenshot of a computer

Description automatically generated

Voting results page: The admin can view the voting results as shown below.

A screenshot of a computer

Description automatically generated

OTP verification page: After submitting the registration form, the voter will receive an OTP via email. If the OTP is verified, they will be eligible to log in.

A screenshot of a computer

Description automatically generated

Voting page: Here, voters can choose their preferred party and submit their vote.

A screenshot of a computer

Description automatically generated

1. **Known Issues**

 **User Authentication Problems**

* **Issue**: Users face difficulties logging in due to incorrect password handling or email verification failures.
* **Impact**: This can prevent users from accessing their accounts, leading to frustration and reduced participation.

 **Token Expiration Confusion**

* **Issue**: Users are not clearly informed about token expiration, leading to unexpected logouts or access denials.
* **Impact**: Disruptions during the voting process, especially during peak times, can occur.

 **Mobile Responsiveness**

* **Issue**: The user interface may not render correctly on all mobile devices or screen sizes.
* **Impact**: Users on mobile devices may struggle to navigate the platform, affecting accessibility and usability.

 **Vote Confirmation Notifications**

* **Issue**: Users occasionally do not receive confirmation emails or notifications after casting their votes.
* **Impact**: This creates uncertainty about whether their votes were successfully submitted.

 **Performance Issues During High Traffic**

* **Issue**: The platform may experience slow response times or crashes during peak voting periods.
* **Impact**: Hindered ability for users to cast their votes, risking disenfranchisement

1. **Future Enhancements**

 **Enhanced User Experience:**

* **Improved UI/UX Design**: Conduct user research to refine the interface, making it more intuitive and user-friendly across all devices.

 **Multi-Factor Authentication (MFA):**

* **Implementation of MFA**: Add an extra layer of security by requiring users to verify their identity through a second method (e.g., SMS code or authenticator app) during login.

 **Real-Time Vote Tracking:**

* **Vote Status Dashboard**: Develop a feature that allows users to track the status of their votes in real-time, providing transparency about the voting process.

 **Expanded Accessibility Features:**

* **Comprehensive Accessibility Options**: Enhance features to support users with disabilities, such as screen reader compatibility and customizable text sizes.

 **Mobile Application Development:**

* **Dedicated Mobile App**: Develop a native mobile application for iOS and Android to provide a more seamless voting experience on mobile devices.