# DBMS ACTIVITY 2
# 2024

## PROJECT REPORT

*Submitted by*

| | |
|---|---|
| 23BCAR0252 | BHARATH K |
| 23BCAR0263 | LOCHAN S |
| 23BCAR0606 | HARSHA K |
| 23BCAR0607 | VIPLAV KUMAR REDDY |
| 23BCAR0302 | SIDDANTH REDDY |

**Semester: 3**
**Specialization: Data analytics**

*in partial fulfillment for the award of the degree of*

## BACHELOR OF COMPUTER APPLICATIONS WITH SPECIALIZATION IN DATA ANALYTICS

**DEPARTMENT OF COMPUTER SCIENCE & IT**

**JAIN KNOWLEDGE CAMPUSJAYANAGAR 9TH BLOCK BANGALORE -560069**

**OCTOBER - 2024**

JGi JAIN DEEMED-TO-BE UNIVERSITY

School Of
Computer
Science and IT

**DEPARTMENT OF COMPUTER SCIENCE &** IT

**Jain Knowledge Campus**
**Jayanagar 9th Block Bangalore, 560069**

This is to certify that the project entitled

## ONLINE VOTING SYSTEM (OVS)

***is the Bonafide record of project work done by***

| | |
|---|---|
| 23BCAR0252 | BHARATH K |
| 23BCAR0263 | LOCHAN S |
| 23BCAR0606 | HARSHA K |
| 23BCAR0607 | VIPLAV KUMAR REDDY |
| 23BCAR0302 | SIDDANTH REDDY |

**OCTOBER-2024**

**Dr. M Santhalakshmi**
Faculty In-charge
JAIN (Deemed-to-be University)

# DECLARATION

I, Bharath K (23BCAR0252), Lochan S (23BCAR0263), Harsha K (23BCAR0606), Siddanth Reddy (23BCAR0302) and Viplav Kumar Reddy (23BCAR0607), affirm that the project work titled "Online Voting System," being submitted in partial fulfillment for the award of the BACHELOR OF COMPUTER APPLICATIONS WITH SPECIALIZATION IN DATA ANALYTICS, is the original work carried out by us. It has not been a part of any other project work submitted for the award of any degree or diploma, either in this or any other university or institution.

Bharath: ________________________

Lochan: ________________________

Harsha: ________________________

Viplav: ________________________

Siddanth Reddy: ________________________

JGi JAIN DEEMED-TO-BE UNIVERSITY
School Of
Computer
Science and IT

## CERTIFICATE

This is to certify that Bharath K has satisfactorily completed activity prescribed by JAIN(Deemed to be University) for the second semester degree course in the year 2024-2025.

Assignment topic Online Voting System(OVS) for Activity-2.

| SI.NO | CRITERIA | MARKS | MARKS OBTAINED |
|---|---|---|---|
| 1 | ON-TIME SUBMISSION | 5 | |
| 2 | INTRODUCTION/ABSTRACT | 10 | |
| 3 | IMPLEMENTATIONS | 20 | |
| 4 | REPORT AND VIVA | 15 | |
| | TOTAL | 50 | |
| | CONVERT | 15 | |

| MARKS | |
|---|---|
| MAX | OBTAINED |
| | |

**Date of Submission: 23 -10-2024**

**Signature of student** **Signature of the Faculty**

JAIN
DEEMED-TO-BE UNIVERSITY
School Of
Computer
Science and IT

## CERTIFICATE

This is to certify that Lochan S has satisfactorily completed activity prescribed by JAIN(Deemed to be University) for the second semester degree course in the year 2024-2025.

Assignment topic Online Voting System(OVS) for Activity-2.

| SI.NO | CRITERIA | MARKS | MARKS OBTAINED |
|---|---|---|---|
| 1 | ON-TIME SUBMISSION | 5 | |
| 2 | INTRODUCTION/ABSTRACT | 10 | |
| 3 | IMPLEMENTATIONS | 20 | |
| 4 | REPORT AND VIVA | 15 | |
| | TOTAL | 50 | |
| | CONVERT | 15 | |

| MARKS | |
|---|---|
| MAX | OBTAINED |
| | |

**Date of Submission: 23 -10-2024**

**Signature of student** **Signature of the Faculty**

JGi JAIN DEEMED-TO-BE UNIVERSITY | School Of Computer Science and IT

## CERTIFICATE

This is to certify that Harsha K satisfactorily completed activity prescribed by JAIN(Deemed to be University) for the second semester degree course in the year 2024-2025.

Assignment topic Online Voting System(OVS) for Activity-2.

| SI.NO | CRITERIA | MARKS | MARKS OBTAINED |
|---|---|---|---|
| 1 | ON-TIME SUBMISSION | 5 | |
| 2 | INTRODUCTION/ABSTRACT | 10 | |
| 3 | IMPLEMENTATIONS | 20 | |
| 4 | REPORT AND VIVA | 15 | |
| | TOTAL | 50 | |
| | CONVERT | 15 | |

| MARKS | |
|---|---|
| MAX | OBTAINED |
| | |

**Date of Submission: 23 -10-2024**

**Signature of student** **Signature of the Faculty**

JGi JAIN
DEEMED-TO-BE UNIVERSITY
School Of
Computer
Science and IT

## CERTIFICATE

This is to certify that Viplav Kumar Reddy has satisfactorily completed activity prescribed by JAIN(Deemed to be University) for the second semester degree course in the year 2024-2025.

Assignment topic Online Voting System(OVS) for Activity-2.

| SI.NO | CRITERIA | MARKS | MARKS OBTAINED |
|---|---|---|---|
| 1 | ON-TIME SUBMISSION | 5 | |
| 2 | INTRODUCTION/ABSTRACT | 10 | |
| 3 | IMPLEMENTATIONS | 20 | |
| 4 | REPORT AND VIVA | 15 | |
| | TOTAL | 50 | |
| | CONVERT | 15 | |

| MARKS | |
|---|---|
| MAX | OBTAINED |
| | |

**Date of Submission: 23 -10-2024**

**Signature of student**

**Signature of the Faculty**

JGI JAIN DEEMED-TO-BE UNIVERSITY

School Of
Computer
Science and IT

## CERTIFICATE

This is to certify that Siddanth Reddy has satisfactorily completed activity prescribed by JAIN(Deemed to be University) for the second semester degree course in the year 2024-2025.

Assignment topic Online Voting System(OVS) for Activity-2.

| SI.NO | CRITERIA | MARKS | MARKS OBTAINED |
|---|---|---|---|
| 1 | ON-TIME SUBMISSION | 5 | |
| 2 | INTRODUCTION/ABSTRACT | 10 | |
| 3 | IMPLEMENTATIONS | 20 | |
| 4 | REPORT AND VIVA | 15 | |
| | TOTAL | 50 | |
| | CONVERT | 15 | |

| MARKS | |
|---|---|
| MAX | OBTAINED |
| | |

**Date of Submission: 23 -10-2024**

**Signature of student** **Signature of the Faculty**

# TABLE OF CONTENTs

# Abstract

In an era where digital transformation is pivotal, electronic voting systems offer a significant leap from traditional paper-based systems. This project presents the design and implementation of a Database Management System (DBMS) for a secure and efficient online voting platform, aimed at institutions or organizations conducting elections at a local scale. The system is developed with SQLite as its backend database and Python (Flask framework) for backend processing, ensuring data security and integrity while providing a user-friendly interface for voters.

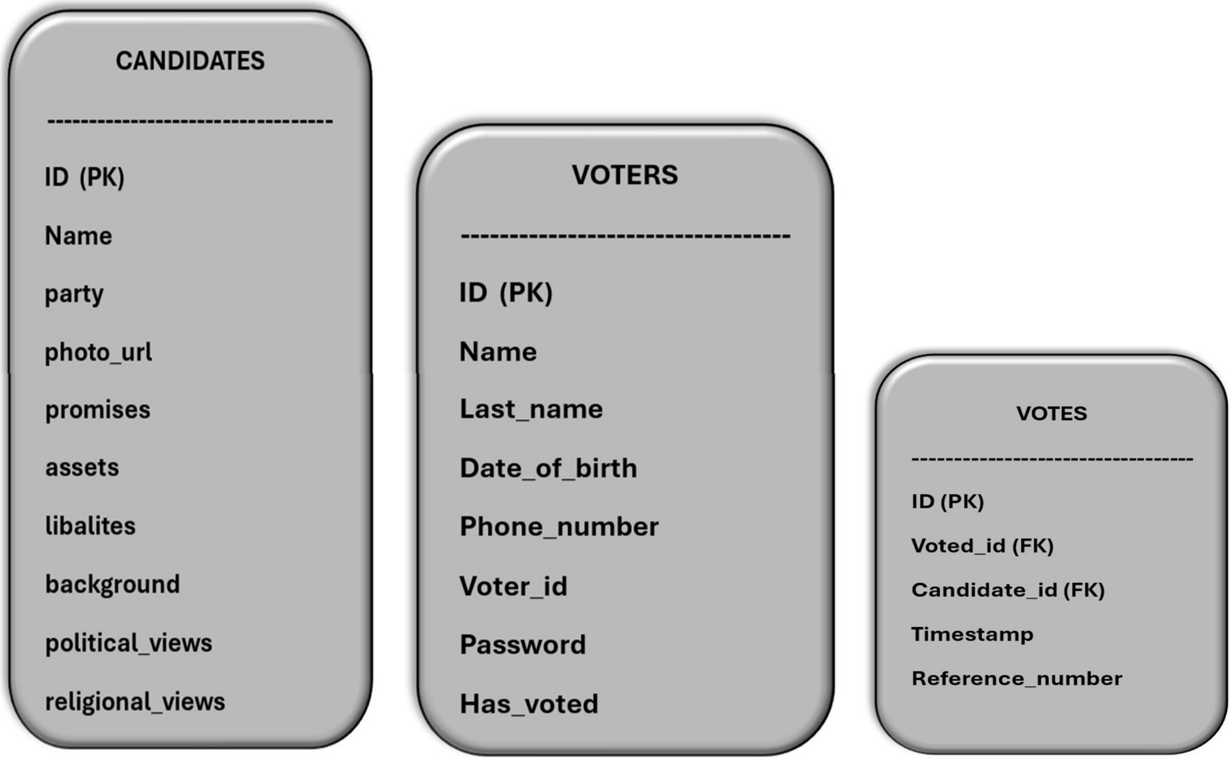
This project addresses several key challenges in the current voting process, such as the inefficiencies of manual counting, risks of fraud, and issues related to voter accessibility. The system is modular, with distinct components for user registration, authentication, candidate display, vote casting, and data management. Voters register through a secure process, log in with encrypted credentials, view detailed candidate profiles, and cast their vote. The system ensures that each voter can vote only once, recording every vote securely and providing a unique confirmation reference for transparency.

From a technical perspective, the project incorporates industry best practices in database design, data security, and system scalability. The database comprises three central tables—voters, candidates, and votes—each of which is designed to optimize performance, prevent data redundancy, and ensure referential integrity. The system is equipped with robust validation mechanisms, ensuring secure password storage (through hashing) and protection against common vulnerabilities such as SQL injection. Data transactions, particularly during the voting process, are atomic, ensuring that votes are accurately recorded and cannot be altered post-submission.

The frontend interface, designed using HTML, CSS, and JavaScript, ensures ease of navigation for users, with responsive design features that cater to different devices. The modular design allows for scalability and easy future enhancements, including the migration to more complex databases if needed. The use of real-time data retrieval and interactive elements enhances the user experience, providing voters with immediate feedback on their interactions with the system, such as successful vote submissions or login attempts.

In conclusion, this project represents a secure, reliable, and user-friendly solution to modernizing the electoral process in small-scale settings. Its design ensures transparency, voter privacy, and election integrity, making it a valuable tool for educational institutions, corporations, or community organizations seeking to digitize their voting processes.

# List of Database Tables



CANDIDATES

--------------------------------

ID (PK)

Name

party

photo_url

promises

assets

libalites

background

political_views

religional_views

VOTERS

----------------------------------

ID (PK)

Name

Last_name

Date_of_birth

Phone_number

Voter_id

Password

Has_voted

VOTES

---------------------------------

ID (PK)

Voted_id (FK)

Candidate_id (FK)

Timestamp

Reference_number

## 1. Voters Table (voters)

- **Purpose**: This table stores the personal and authentication details of all users (voters) who register in the system. It ensures that only registered voters can log in and vote. It also tracks whether a user has already voted.

- **Structure**:
    - **id**: INTEGER (Primary Key)
      A unique identifier for each voter.
    - **name**: TEXT (NOT NULL)
      Stores the first name of the voter.
    - **last_name**: TEXT (NOT NULL)
      Stores the last name of the voter.

  - **date_of_birth**: TEXT (NOT NULL)
    Records the date of birth of the voter (used for identification purposes).
  - **phone_number**: TEXT (NOT NULL)
    Stores the voter's phone number for contact information.
  - **voter_id**: TEXT (NOT NULL, UNIQUE)
    A unique identifier (e.g., national ID or any voter-specific ID) for each user to prevent duplicate registrations.
  - **password**: TEXT (NOT NULL)
    Stores the voter's encrypted password for authentication.
  - **has_voted**: INTEGER (DEFAULT 0)
    A flag indicating whether the voter has already cast a vote (0 = not voted, 1 = voted). This helps enforce the rule that each voter can vote only once.

- **Summary**:
  This table acts as the backbone for user authentication and tracks whether users have already voted, preventing multiple votes by the same individual.

## 2. Candidates Table (candidates)

- **Purpose**: This table holds detailed information about the candidates running in the election. It provides voters with candidate profiles, helping them make informed decisions during voting.
- **Structure**:
  - **id**: INTEGER (Primary Key)
    A unique identifier for each candidate.
  - **name**: TEXT (NOT NULL)
    The name of the candidate.
  - **party**: TEXT (NOT NULL)
    The political party the candidate is associated with.
  - **photo_url**: TEXT (NOT NULL)
    A URL or path to the candidate’s photo, which is displayed in the UI.
  - **promises**: TEXT (NOT NULL)
    A summary of the candidate's campaign promises.
  - **assets**: TEXT (NOT NULL)
    Information about the candidate's financial assets (for transparency).

- **liabilities**: TEXT (NOT NULL)
  Information about the candidate's financial liabilities (for transparency).
- **background**: TEXT (NOT NULL)
  A brief background of the candidate, including professional and political experience.
- **political_views**: TEXT (NOT NULL)
  The candidate’s key political views on major issues.
- **regional_views**: TEXT (NOT NULL)
  The candidate’s views on regional issues that may affect their constituency.

- **Summary**:
  This table helps voters get to know the candidates. The detailed information here is displayed in the user interface to give voters insights into each candidate’s profile, promises, and background.

## 3. Votes Table (votes)

- **Purpose**: This table records each vote cast by a voter. It links the voter to a candidate and ensures that all votes are stored with a timestamp for tracking and accountability.
- **Structure**:
  - **id**: INTEGER (Primary Key)
    A unique identifier for each vote.
  - **voter_id**: INTEGER (Foreign Key from voters.id, NOT NULL)
    References the unique ID of the voter casting the vote.
  - **candidate_id**: INTEGER (Foreign Key from candidates.id, NOT NULL)
    References the unique ID of the candidate receiving the vote.
  - **timestamp**: TEXT (NOT NULL)
    The date and time when the vote was cast, used for record-keeping and auditing.
  - **reference_number**: TEXT (NOT NULL, UNIQUE)
    A unique reference number for each vote, ensuring traceability and uniqueness for each voting transaction.
- **Summary**:
  This is the central table for recording the results of the election. It creates a link between voters and candidates, ensuring that each vote is properly accounted for and securely stored.

## Note on the Database:

- **Database Used**: The project uses SQLite, which is a lightweight, file-based database system ideal for local applications like this voting system.
- **Purpose of SQLite**: Since the project is run locally, SQLite is a perfect fit because it doesn't require a separate database server, and all data is stored in a single .db file (voting_system.db). The tables are initialized when the system starts, and SQLite allows for fast and efficient querying of user and vote data.

# List of Figures

# Chapter -1

# Introduction to the Project

In recent years, the demand for digital transformation has reshaped the way various processes are carried out, from education to governance. Among these transformations, the shift from traditional, paper-based voting systems to electronic and online voting platforms stands out as a critical step toward modernizing democratic participation. Voting, one of the most fundamental rights in any democratic institution, requires a system that guarantees security, transparency, and accessibility. However, the adoption of electronic voting systems has been slow, primarily due to concerns about security vulnerabilities, fraud, accessibility challenges, and data management complexities.

This project addresses these issues by developing a robust online voting system, particularly suited for small-scale elections in local institutions, educational settings, or organizations. By leveraging the power of modern database management systems, secure data handling practices, and intuitive user interfaces, this project aims to provide an end-to-end solution for conducting elections in a digital format. From voter registration and authentication to candidate profiling and vote casting, the system offers a seamless experience for both voters and election administrators.

The project is structured around several core components, each designed to address a specific aspect of the voting process while ensuring security, integrity, and ease of use. One of the primary challenges in building any online voting system is ensuring that the system is secure enough to prevent fraudulent activities, such as multiple votes by the same individual or tampering with vote data. In this system, security is given paramount importance, with features such as encrypted passwords, session management, and vote transaction verification to ensure that each vote is counted accurately and securely. Additionally, the system uses well-established techniques for safeguarding against common threats like SQL injection, cross-site scripting (XSS), and session hijacking.

The heart of the system is its database, built using SQLite, a lightweight and highly efficient database engine. SQLite’s ease of use, combined with its ability to store the entire database in a single file, makes it an ideal choice for this project, allowing for quick setup and deployment in local environments. The database is designed with three key tables—voters, candidates, and votes—that serve as the foundation for managing election data. Each table is structured to ensure data integrity and optimized to handle large volumes of data without compromising performance. The voter registration process ensures that only legitimate users can access the system, while the voting process prevents any user from casting multiple votes, thus preserving the integrity of the election.

In terms of backend processing, the system is powered by Python and the Flask framework, a lightweight web application framework that allows for easy development of the application's logic and interaction with the database. The backend handles everything from processing user requests and managing data transactions to ensuring that each vote is accurately recorded in the database. The system also includes features for managing user sessions, ensuring that once a voter logs in, they remain authenticated throughout the session, and any attempts to bypass authentication are blocked.

On the frontend, the system provides an intuitive and user-friendly interface built using HTML, CSS, and JavaScript. The design prioritizes simplicity, ensuring that even users with minimal technical skills can navigate the system with ease. Voters can quickly register, log in, view a list of candidates, and cast their votes without any unnecessary complexity. The frontend also includes dynamic elements powered by JavaScript, such as real-time form validation, which enhances the overall user experience by providing instant feedback and preventing user errors. Additionally, the system’s responsive design ensures compatibility across a range of devices, allowing users to access the platform from desktops, tablets, or smartphones.

One of the standout features of this project is its modular design. Each component of the system—the database, backend logic, and frontend interface—has been developed as a distinct module, allowing for greater flexibility in development, testing, and future enhancements. This modularity also allows the system to be easily scaled up or modified in the future, depending on the specific needs of the institution using it. For instance, the system could be expanded to handle larger elections or integrated with more sophisticated database systems like PostgreSQL or MySQL to support additional features or higher traffic volumes.

Beyond technical functionalities, the project places significant emphasis on the user experience. A well-designed voting system should not only be secure and efficient but also accessible and easy to use. This system incorporates feedback mechanisms to guide users throughout their interaction with the platform. From error messages during the registration process to vote confirmation messages after a successful vote submission, every step of the process has been carefully designed to ensure transparency and user satisfaction.

Furthermore, the system has been designed with compliance in mind. As digital voting continues to evolve, various legal and regulatory frameworks govern the process to ensure fairness and data protection. The system adheres to these guidelines by incorporating features like secure password storage, data encryption, and detailed audit trails that ensure every action taken within the system can be traced and verified. These audit trails also provide election administrators with the tools they need to conduct post-election analysis and ensure that the voting process was free from any manipulation or errors.

From a pedagogical perspective, this project offers valuable lessons in database management, security, and software development. For students, the project provides hands-on experience in working with databases like SQLite, developing secure web applications using Flask, and designing user-friendly interfaces with HTML, CSS, and JavaScript. The modular nature of the project also allows students to focus on individual components of the system—whether it’s designing an efficient database schema, implementing secure authentication mechanisms, or creating an intuitive user interface. Moreover, by working on a real-world application like an online voting system, students gain insights into the practical challenges of building secure and scalable systems that meet the needs of users in a real-world context.

In conclusion, this project is a comprehensive solution for conducting small-scale elections in a digital format, addressing the core challenges of security, accessibility, and data management. By leveraging the strengths of modern database systems, secure backend processing, and intuitive frontend design, this system ensures a seamless and transparent voting process. Its flexibility, scalability, and ease of use make it an ideal platform for institutions looking to modernize their electoral processes. As digital voting becomes more prevalent, systems like this will play an increasingly important role in ensuring that democratic processes are both secure and accessible to all.

# Chapter 2

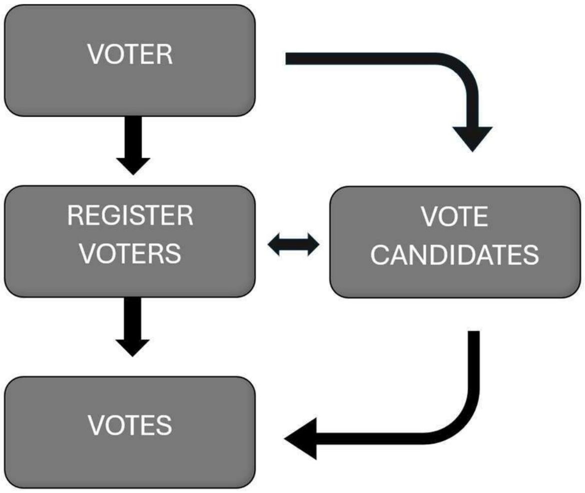
# Problem Definition

In today's fast-paced world, conducting elections efficiently and securely is a critical challenge faced by governments, organizations, and institutions. Traditional paper-based voting systems are often time-consuming, costly, and prone to errors, including miscounts and fraudulent activities such as double voting. Additionally, with the growing reliance on technology, there is a strong demand for digitized solutions that can enhance the voting experience by ensuring convenience, transparency, and security.

The problem we seek to address through this project is the lack of accessible, efficient, and secure voting systems that can be used in a local or institutional context. Specifically, many institutions such as colleges, corporations, or local communities require a reliable means of conducting elections in a controlled environment but may not have the resources to deploy large-scale electronic voting solutions. These institutions need a simple, cost-effective solution that guarantees the integrity of the voting process while providing ease of use for voters.

In this project, we propose an online voting system that provides a secure platform for conducting elections in a digital format. The system enables voters to log in, select a candidate, and cast their vote, all while ensuring that each voter can only vote once. The database ensures that votes are securely recorded and that the voting process is transparent, preventing fraudulent activity like double voting. By leveraging a lightweight database such as SQLite, this system is designed to be implemented in a local environment, making it ideal for small-scale elections.

Our project addresses the common challenges of traditional voting systems, such as inefficiency, lack of transparency, and potential for fraud, by providing a streamlined, digitized solution that can be deployed in smaller settings. This system is particularly beneficial for organizations looking to modernize their election processes while maintaining a secure and straightforward user experience



# Chapter 3

# Project Design

The online voting system is designed as a modular web application with several key components, each responsible for a specific aspect of the system. Dividing the project into modules ensures that each part of the system is manageable and allows for easier development, testing, and debugging. Below are the major modules of the project:

### 1. User Interface (UI) Module

- **Description**: This module handles the presentation of the web application. It is responsible for the layout, navigation, and interactive elements that users interact with while using the system.
- **Components**:
  - **HTML Files**: These provide the structure and content of the web pages.
    - index.html: The landing page of the application.
    - login.html: The login page where users can enter their credentials.
    - register.html: The registration page for new users.
    - dashboard.html: The main page after login, showing the election status.
    - candidate_list.html: Displays a list of candidates available for voting.
    - candidate_detail.html: Shows detailed information about a specific candidate.
    - vote_confirmation.html: Confirms the vote has been cast successfully.
  - **CSS**: The styles.css file within the static/css/ directory provides styling for the web pages, making the UI visually appealing and responsive.
  - **JavaScript**: The script.js file in the static/js/ directory handles any client-side interactivity, like form validation or dynamic content updates.
- **Purpose**: This module ensures a clean, simple, and user-friendly interface that guides users through the voting process, from registration to voting confirmation.

## 2. User Authentication Module

- **Description**: This module handles the login and registration functionalities. It ensures that only authorized users (registered voters) can access the voting system.
- **Components**:
    - **Login Functionality**: Processes login requests by verifying the user’s email (or voter ID) and password against the records in the voters table.
    - **Registration Functionality**: Allows new users to create accounts by filling in personal details such as name, voter ID, and password. This information is stored in the voters table.
- **Purpose**: The authentication module secures the system by ensuring only registered users can vote. It also prevents unauthorized access and ensures data integrity by using unique voter IDs.

## 3. Voting Module

- **Description**: This is the core functionality of the system, allowing users to cast their vote for a candidate.
- **Components**:
    - **Candidate List**: The candidate_list.html file displays the list of candidates fetched from the candidates table in the database.
    - **Candidate Detail**: The candidate_detail.html page shows additional information about each candidate, such as their party affiliation, promises, background, etc., pulled from the database.
    - **Vote Casting**: After selecting a candidate, the system records the vote by creating a new entry in the votes table with the voter's ID, candidate’s ID, and a timestamp.
    - **Vote Confirmation**: The vote_confirmation.html page provides feedback to the voter that their vote has been successfully cast.
- **Purpose**: This module ensures that users can cast a vote and records the voting transaction securely, preventing multiple votes by the same user through the has_voted field in the voters table.

## 4. Database Module

- **Description**: This module manages the entire database functionality of the system, including user, candidate, and vote management.
- **Components**:

  - **SQLite Database (voting_system.db)**: The SQLite database is located in the database/ directory and stores the data for voters, candidates, and votes.
  - **Tables**:
    - voters: Stores voter details like names, voter ID, passwords, and a flag indicating whether they've voted.
    - candidates: Stores candidate details, including their names, party affiliations, background, and other attributes.
    - votes: Records each vote, linking a voter to a candidate with a timestamp.
  - **SQL Queries**: Queries are used to interact with the database—such as inserting new users, fetching candidate lists, recording votes, and checking whether a voter has already voted.
- **Purpose**: This module is crucial for managing persistent data storage and ensuring that all operations (registration, voting, etc.) are securely recorded in the database.

## 5. Backend (Business Logic) Module

- **Description**: This module serves as the brain of the application, connecting the UI with the database. It processes user requests, retrieves necessary data from the database, and updates the database as required.
- **Components**:
  - **Python Scripts**:
    - app.py: This is the main file responsible for handling HTTP requests and responses. It routes users to the appropriate pages (e.g., login, registration, candidate list) and interacts with the database.
    - database.py: This file handles database connections, table creation, and query execution.
    - models.py: This file may define database models for interacting with the database in a structured way.
    - config.py: Contains configuration settings such as database paths or application-specific configurations.
- **Purpose**: The backend module ensures smooth communication between the user interface and the database. It implements business logic such as login verification, vote recording, and data fetching.

**Purpose of Modular Design**

Each module serves a distinct function in the system, allowing for separation of concerns. This modular approach simplifies development, testing, and maintenance, as each component can be worked on independently. For instance, the front-end development team can focus on the UI while the back-end team works on database integration and business logic.

# Detailed Explanation of Modules

## Module 1: User Interface (UI) Module

The **User Interface (UI) Module** is the front-facing part of the online voting system. It is the part of the system that users directly interact with, and thus plays a critical role in ensuring the application is user-friendly, intuitive, and responsive. A well-designed UI not only enhances user experience but also ensures users can navigate through the system with minimal errors, making the voting process simple and efficient. This module consists of several components, including HTML, CSS, and JavaScript files, which work together to present a cohesive interface.

**1.1 Components of the UI Module**

1. **HTML Files (HyperText Markup Language)**
    - HTML is the backbone of any web interface. In this project, HTML files define the structure of the web pages and serve as the skeleton of the voting system. Each HTML page is responsible for a different function of the voting system, such as user registration, login, viewing candidates, and casting votes.

    Here’s a breakdown of each HTML file:

    - **index.html**:
        - **Purpose**: The index.html file serves as the landing page or homepage of the online voting system. This is the first page users see when they visit the system.
        - **Structure**: It typically contains links to the login and registration pages. It may also provide some basic information about the voting system, such as instructions or a brief overview of the candidates.
        - **Details**: This page includes navigation elements like a header with the system’s name, possibly some introductory text, and call-to-action buttons (e.g., “Login” and “Register”).
    - **login.html**:

    - **Purpose**: This page allows registered voters to enter their credentials (such as email/voter ID and password) and log into the system.
    - **Structure**: The form includes fields for the user’s ID and password, along with a submit button to authenticate the user.
    - **Details**: If the login credentials are incorrect or missing, error messages are displayed, ensuring that users understand why they can't log in. Basic client-side validation (using HTML5 form validation or JavaScript) ensures that required fields are filled before submission.
- **register.html**:
    - **Purpose**: This page enables new users (voters) to register for the system by entering personal information such as their name, voter ID, date of birth, phone number, and password.
    - **Structure**: The form includes input fields for the user's details, including their personal information and a password field.
    - **Details**: The page may also include constraints such as password strength validation, checking whether the voter ID is unique, or ensuring that the date of birth meets certain requirements (e.g., voters are over 18 years of age).
- **dashboard.html**:
    - **Purpose**: The dashboard.html page serves as the user’s control panel after they have successfully logged in. It provides access to the voting functionalities and displays useful information such as the list of candidates, the election status, and any previous voting activity.
    - **Structure**: The page may include navigation links to view candidates, cast a vote, or view past votes.
    - **Details**: If the user has already voted, this page will indicate so by showing a message or restricting access to the voting features.
- **candidate_list.html**:
    - **Purpose**: This page provides a list of all the candidates participating in the election, allowing the voter to browse through and learn more about each one before making a decision.
    - **Structure**: The candidate list may display basic information such as the candidate's name, party affiliation, and a small profile picture.
    - **Details**: Each candidate's name may be hyperlinked to the candidate_detail.html page, where more information is available.

- **candidate_detail.html**:
    - **Purpose**: When a voter clicks on a candidate’s name, they are taken to this page, which shows detailed information about the candidate, including their background, political views, promises, assets, liabilities, and other relevant details.
    - **Structure**: The page displays a larger profile picture of the candidate along with a detailed bio and political manifesto.
    - **Details**: This page allows voters to make informed decisions before casting their vote.
- **vote_confirmation.html**:
    - **Purpose**: After a voter selects a candidate and casts their vote, they are directed to this page, which confirms that their vote has been successfully recorded.
    - **Structure**: It may display a simple thank-you message, along with a unique reference number for their vote and possibly a summary of their choice (i.e., the name of the candidate they voted for).
    - **Details**: This page also serves as a final checkpoint, assuring the voter that their participation in the election is complete.

2. **CSS (Cascading Style Sheets)**
    - **Purpose**: CSS is responsible for the styling of the HTML pages. It ensures the system’s web pages are visually appealing, modern, and easy to navigate. The CSS file (styles.css) is located in the static/css/ directory.

Detailed Elements of CSS in the System:

- **Layout and Structure**: CSS helps define the layout of the web pages, ensuring that the interface is clean and well-organized. For example, CSS is used to ensure that form fields are aligned, buttons are prominent, and sections of text are properly spaced.
- **Color Scheme**: CSS is used to define the color palette of the web application. A consistent color scheme is critical in giving the application a professional appearance and making it visually appealing. Colors can also be used to highlight important sections, such as error messages, success messages (e.g., after a vote is cast), or buttons.

- **Typography**: Fonts, font sizes, and font weights are controlled using CSS. The typography is chosen to ensure that text is readable and easy to scan, with a preference for clean, sans-serif fonts that look professional.
- **Responsiveness**: A key feature of the system's CSS is to ensure that the interface is responsive, meaning it can adapt to different screen sizes and resolutions. Whether a user accesses the voting system on a desktop, tablet, or mobile phone, the CSS should adjust the layout to provide a seamless experience.
- **Interactive Elements**: CSS also defines the styles of interactive elements, such as buttons, hover effects, and focus states for form inputs. For example, when a user hovers over a button, it may change color to indicate it is clickable.

3. **JavaScript**

   - **Purpose**: JavaScript is used to handle client-side interactions and dynamic functionality. The script file (script.js) located in the static/js/ directory enhances the user experience by allowing interactive features without requiring a page refresh.

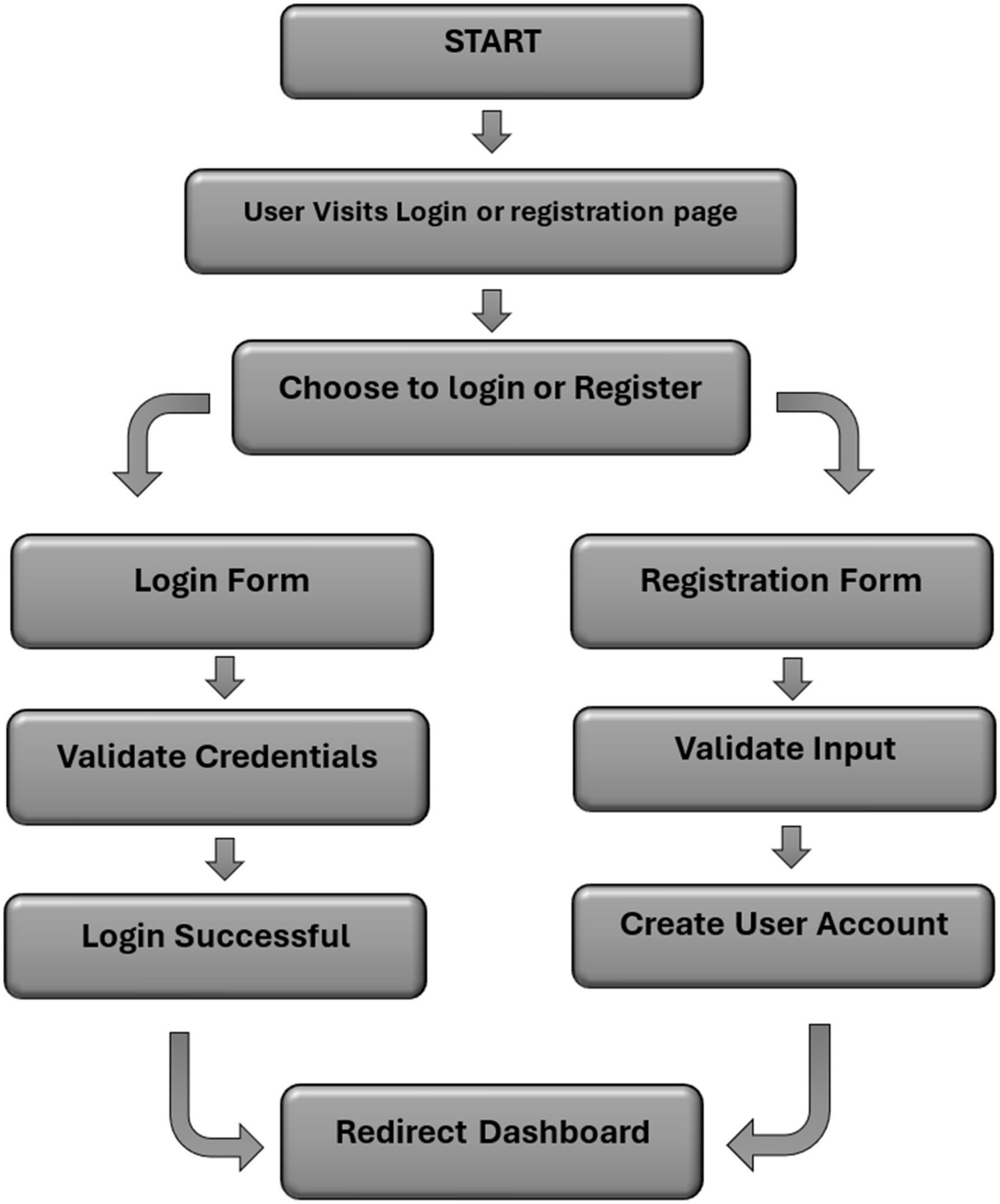
Detailed Features of JavaScript in the System:

- **Form Validation**: JavaScript is used to validate user input in real-time. For example, it ensures that required fields in the registration form (e.g., voter ID, password) are not left empty, that passwords meet certain strength requirements, and that email formats are valid. This helps reduce server-side processing and provides instant feedback to the user.
- **Dynamic Content Updates**: JavaScript can be used to dynamically update certain parts of the page without requiring a full page reload. For instance, after a user casts their vote, JavaScript may display a confirmation message on the same page.
- **UI Enhancements**: Simple JavaScript functions can add subtle enhancements to the UI, such as dropdown menus, modal pop-ups (e.g., confirming actions like vote submission), and interactive animations. These features improve the overall user experience and make the system feel more responsive.
- **Error Handling**: JavaScript can detect and respond to errors in user interaction, such as incorrect login attempts or invalid input formats. When an error occurs, JavaScript can display a clear error message, ensuring users know what went wrong and how to fix it.
- **Asynchronous Requests**: Although optional in this project, JavaScript can be used to make asynchronous requests (using AJAX) to the server without requiring the

page to reload. This could be useful for features like checking if a voter ID is already registered while the user is typing their information during registration.

### 1.2 Integration with Backend

While the UI module primarily deals with the frontend of the voting system, it is closely integrated with the backend (written in Python) through HTTP requests. The form data from the HTML pages is sent to the backend for processing, such as logging in users, registering new voters, and submitting votes. The backend sends responses, which are then displayed on the UI, such as error messages for invalid logins or success messages after voting.


START
User Visits Login or registration page
Choose to login or Register
Login Form
Registration Form
Validate Credentials
Validate Input
Login Successful
Create User Account
Redirect Dashboard

**1.3 User Experience Considerations (continued)**

In designing the User Interface (UI) for the online voting system, several critical factors were considered to ensure a smooth and efficient experience for users. Here's a more in-depth look at these considerations:

1. **Simplicity**:
   - The system's UI is built to be intuitive, following the principle of simplicity. Users need to focus only on essential actions, such as logging in, registering, and casting their vote. There is minimal clutter, and unnecessary features or distractions are avoided.
   - **Visual Hierarchy**: The design employs a clear visual hierarchy where important actions, like the "Vote" button, are emphasized with larger, more prominent design elements. Simpler, secondary actions like "View Candidate Details" are accessible but do not draw as much attention.
   - **Accessibility**: The system's simplicity also enhances accessibility, making it easy for all voters to participate, including those with limited technical skills. All elements, from buttons to form fields, are clearly labeled, ensuring that users know exactly what each component does.
2. **Clarity**:
   - Labels and buttons are named clearly, reducing confusion for the user. Instead of generic terms like "Submit," the system uses more action-oriented buttons like "Vote Now" or "Register" to make the interface more self-explanatory.
   - **Guidance**: Each page provides basic instructions or prompts for users. For example, the login page might include a message like "Please enter your Voter ID and password" to ensure users understand what is required of them.
   - **Error Messaging**: When users input incorrect information (e.g., wrong password or invalid voter ID), clear and specific error messages are displayed. For example, "Incorrect password, please try again" instead of a vague "Error." These messages guide users to resolve issues quickly.
3. **Feedback**:
   - The UI continuously provides feedback to users, reassuring them that their actions are being processed. For instance, once the user clicks the "Vote" button, a loading animation or message (such as "Processing your vote...") may appear to indicate that the system is working on their request.
   - **Confirmation Messages**: After successful interactions, the system acknowledges the action. For example, after submitting their vote, users are taken to a

confirmation page that thanks them for voting and provides a reference number for their vote.

- **Real-Time Validation**: JavaScript ensures that forms are validated in real time. If a required field is missing, users are immediately notified with a message next to the relevant field, such as "This field cannot be left blank," without having to reload the page.

4. **Error Prevention**:

   - The system anticipates potential user errors and prevents them wherever possible. For example, the registration page checks whether the voter ID is unique as the user enters it. This prevents users from submitting invalid data and encountering errors later in the process.
   - **Confirmation Steps**: Before finalizing a vote, the system could display a confirmation modal (pop-up) that asks users to confirm their candidate choice, reducing the likelihood of accidental votes.
   - **Required Fields**: All required fields, such as those for login and registration, are marked with asterisks (*) to ensure users know what information must be provided.

5. **Responsiveness**:

   - The UI is designed to be responsive, meaning it adjusts its layout and presentation based on the device being used. Whether the user accesses the system on a desktop, tablet, or smartphone, the interface remains fully functional and easy to navigate.
   - **Mobile Optimization**: On smaller devices, the layout might shift to a single-column design for easier scrolling, and buttons become larger to accommodate touch inputs.
   - **Cross-Browser Compatibility**: The design is tested to ensure that it works consistently across different web browsers (e.g., Chrome, Firefox, Safari), providing users with a seamless experience regardless of their preferred browser.

6. **Navigation**:

   - The voting system employs a straightforward navigation system that minimizes the number of steps users need to take to perform key actions. A fixed navigation bar or links to essential pages (e.g., "Home," "Vote," "Candidate List") helps users move between sections of the site without getting lost.
   - **Breadcrumbs**: In some cases, breadcrumb navigation might be used to show users where they are within the application's hierarchy (e.g., Home > Candidate List > Candidate Details), allowing them to backtrack easily if needed.

### 1.4 User Flow in the UI Module

The user flow defines the path that users follow from the moment they land on the system's homepage to when they cast their vote. This flow is crucial to understand how each component of the UI module is connected and ensures a seamless transition from one step to the next.

- **Step 1: Home Page (index.html)**
    - Users first arrive at the homepage, where they can either log in if they are a registered voter or navigate to the registration page if they are new.
    - The page includes buttons like "Login" and "Register," allowing users to start their journey through the system.
- **Step 2: Registration (register.html)**
    - If the user selects the "Register" option, they are directed to the registration page. Here, they fill in the required details such as name, date of birth, phone number, voter ID, and password.
    - After completing the registration form, users submit their information, which is validated both on the client side (using JavaScript) and server side (through backend checks).
- **Step 3: Login (login.html)**
    - Once registered, or for returning voters, users must log in using their Voter ID and password. The login form ensures that only authenticated users can access the system's voting features.
    - Successful login takes the user to their personalized dashboard.
- **Step 4: Dashboard (dashboard.html)**
    - The dashboard serves as the voter's control panel, displaying important information about the election and offering navigation to other key pages, like the candidate list and voting section.
    - If the voter has already voted, this page might display a message indicating that they've cast their vote and prevent them from voting again.
- **Step 5: Candidate List (candidate_list.html)**
    - From the dashboard, the user can view the list of candidates participating in the election. Each candidate's name is clickable, leading to a detailed profile page.
- **Step 6: Candidate Details (candidate_detail.html)**
    - When a user selects a candidate, they are taken to the detailed profile page, where they can read about the candidate's promises, background, and political views.

    - After reviewing the candidates, the user is presented with the option to vote for their chosen candidate.

- **Step 7: Voting and Confirmation (vote_confirmation.html)**
    - Once the user selects a candidate and submits their vote, they are directed to a confirmation page. This page thanks the user for voting and displays a reference number, confirming that their vote has been successfully recorded.

### 1.5 Security Considerations in the UI Module

The UI module plays a vital role in ensuring the overall security of the online voting system, especially when it comes to handling sensitive voter data.

- **Input Validation**: All forms (login, registration, voting) are equipped with both client-side (JavaScript) and server-side validation to prevent malicious inputs like SQL injection or XSS (cross-site scripting). JavaScript checks inputs before they are submitted, ensuring they conform to the expected format (e.g., valid email addresses, strong passwords).
- **Password Masking**: During login and registration, the password input fields are masked, showing asterisks (******) instead of plain text to protect users' credentials from being exposed on-screen.
- **Session Handling**: After login, the system tracks the user session to ensure only authenticated users can access certain pages (like the dashboard and voting page). If a user tries to bypass authentication, they are redirected back to the login page.
- **Secure Communication**: Though the project runs locally, in a real-world scenario, the system would enforce HTTPS to encrypt communication between the user's browser and the server, ensuring that sensitive information (such as passwords and votes) is transmitted securely.

### 1.6 Conclusion

The **User Interface (UI) Module** serves as the gateway through which users interact with the online voting system. It combines structure (HTML), design (CSS), and interactivity (JavaScript) to create a user-friendly, secure, and accessible platform for voters. By focusing on simplicity, clarity, and responsiveness, the UI ensures that users can easily navigate the system, register, log in, review candidates, and cast their vote with minimal friction. In addition, the integration with the backend ensures that all interactions are handled securely and efficiently, making this module the cornerstone of the user experience in the voting system.

## Module 2: Backend Processing and Logic Module

The Backend Processing and Logic Module is the core of the system, responsible for handling all the computational logic, data management, and business rules that ensure the seamless operation of the online voting platform. This module is the intermediary between the user interface and the database, managing user requests, interacting with the database, and ensuring data integrity and security. The key components of the backend are built in Python, and they work together to provide a robust, reliable, and scalable foundation for the system.

### 2.1 Overview of the Backend Module

The Backend Processing and Logic Module is responsible for all operations that occur "behind the scenes." It handles requests from users, processes them, and interacts with the SQLite database to retrieve or store the necessary data. The backend is also responsible for managing authentication, voting logic, and the connection between users and the candidates in the system.

The module comprises several important files that work together to provide the necessary functionality:

- **app.py**: The main application file, which manages routing and connects the front-end requests with the necessary backend functions.
- **models.py**: Defines the structure of the data and how it interacts with the database.
- **database.py**: Manages the database connection and initialization, ensuring that the database is set up and available for all interactions.
- **config.py**: Holds configuration details such as database path or secret keys for sessions.

Each of these files plays a crucial role in ensuring that the system functions smoothly and meets the requirements of the voting platform.

### 2.2 File Breakdown and Responsibilities

#### 2.2.1 app.py

- **Request Handling and Routing**:
  - This file serves as the entry point for all user requests. It routes requests to the appropriate functions using a Python web framework (such as Flask). For instance, if a user visits the login page, app.py directs them to the login view. Similarly, if they submit a vote, the request is processed through this file.
  - Each route in app.py corresponds to a specific page or action. For example:
    - /login: Directs users to the login page.

        - /register: Handles user registration requests.
        - /dashboard: Displays the dashboard where users can view candidates or vote.
        - /vote: Processes voting logic and stores the vote in the database.

- **Session Management**:
    - app.py manages user sessions, ensuring that users remain logged in between page visits. Once a user logs in, a session is created, storing relevant information like their voter ID. This session persists until the user logs out, ensuring that only authenticated users can access sensitive pages like the voting page.
    - **Session Security**: The session is encrypted and signed using a secret key stored in config.py, preventing tampering or unauthorized access.
- **Error Handling**:
    - app.py handles errors such as invalid logins or attempts to vote multiple times. It provides meaningful feedback to the user when something goes wrong, such as incorrect login credentials or database connection issues.
    - The routes are designed to catch potential issues and respond gracefully. For example, if a user tries to vote after they've already cast their vote, they are redirected with an error message indicating that they can't vote again.

### 2.2.2 models.py

- **Object-Relational Mapping (ORM)**:
    - models.py defines the structure of the data in terms of Python classes that map to the database tables. This allows the backend to interact with the database using Python objects rather than raw SQL queries, making the code more maintainable and easier to understand.
    - Each class in models.py represents a table in the database. For example:
        - **Voter class**: Maps to the voters table and represents a registered user of the system.
        - **Candidate class**: Maps to the candidates table and represents a candidate in the election.
        - **Vote class**: Maps to the votes table and represents a vote cast by a user.
- **Data Validation**:

    - The models.py file also includes logic to ensure data integrity. For instance, it checks whether a voter ID is unique during registration or ensures that only valid candidate IDs are referenced when a vote is cast.

- **Business Logic**:
    - This file encapsulates some of the key business rules of the system. For instance, it ensures that:
        - A voter can only vote once.
        - A voter can only vote for valid candidates.
        - Votes are recorded with the correct timestamp and reference number.

### 2.2.3 database.py

- **Database Connection and Initialization**:
    - database.py manages the connection to the SQLite database. It ensures that the database file (voting_system.db) is accessible and establishes a connection to allow for queries and updates.
    - **Database Setup**: This file is responsible for initializing the database if it doesn't already exist. It creates the necessary tables (voters, candidates, votes) and ensures that the system is ready to handle data storage and retrieval. This is achieved through the init_db() function, which creates tables and enforces schema constraints.
- **Error Handling**:
    - If there are issues with the database (e.g., if the file can't be found or a query fails), database.py provides detailed error messages and logs, helping developers debug any issues quickly.
    - The connection to the database is closed properly after each interaction, ensuring that there are no lingering open connections that could lead to performance issues or data corruption.

### 2.2.4 config.py

- **Configuration Management**:
    - config.py holds all the configuration variables required by the backend. This includes paths to the database file, secret keys for session encryption, and other environment-specific settings.
    - This modular approach allows the project to be easily reconfigured for different environments (e.g., development or production) by simply changing the configuration settings in one place.

### 2.3 Key Backend Operations

The backend module handles several critical operations, including authentication, voting, and data management. Let's explore these operations in greater detail:

#### 2.3.1 User Authentication

- **Login Process**:
  - When a user submits their voter ID and password, the backend first checks if the voter ID exists in the voters table. If it does, the password provided is hashed and compared to the stored hashed password in the database.
  - If the login is successful, the backend creates a session for the user and redirects them to their dashboard.
  - If login fails (e.g., wrong password or voter ID doesn't exist), the backend returns an error message, and the user is redirected back to the login page.
- **Registration Process**:
  - When a new voter registers, the backend checks to ensure that the provided voter ID is unique. It also validates other data such as the format of the phone number or date of birth.
  - Passwords are hashed before being stored in the database, adding a layer of security and ensuring that sensitive information is protected.
  - After successful registration, the voter is redirected to the login page, where they can log in and cast their vote.

#### 2.3.2 Vote Casting and Storage

- **Voting Logic**:
  - Once a user selects a candidate and submits their vote, the backend checks if the user has already voted. This is done by querying the voters table to check the has_voted flag.
  - If the user has not voted yet, the backend records the vote in the votes table, storing the voter ID, candidate ID, timestamp, and a unique reference number for each vote.
  - The backend then updates the has_voted flag in the voters table to prevent the user from voting again.
- **Vote Confirmation**:

  - After a successful vote, the backend generates a confirmation page displaying the reference number for the vote. This reference number is important for tracking and verifying the vote in the system.
  - The backend also logs the voting activity in the database for audit purposes.

### 2.3.3 Data Retrieval and Display

- **Fetching Candidate Information**:
  - When a user views the list of candidates, the backend queries the candidates table to retrieve information such as the candidate's name, party, promises, and political views. This data is then passed to the frontend, where it is displayed in a user-friendly format.
- **Vote Count**:
  - The backend can also be used to retrieve vote counts from the votes table, allowing for real-time updates on the number of votes each candidate has received. This data could be displayed to users or election officials, depending on the system's requirements.

### 2.4 Security Considerations in the Backend Module (continued)

The Backend Processing and Logic Module is crucial in safeguarding the online voting system from potential threats and ensuring the integrity of the voting process. Below are the key security mechanisms implemented in the backend:

- **Password Hashing**:
  - All user passwords are hashed using a secure hashing algorithm (such as bcrypt or Argon2) before being stored in the database. This means that even if the database is compromised, attackers would not have access to plain-text passwords, significantly reducing the risk of identity theft and unauthorized access.
- **Session Security**:
  - User sessions are managed securely to prevent session hijacking. Each session is assigned a unique identifier that is stored in a secure cookie with attributes like HttpOnly (to prevent access via JavaScript) and Secure (to ensure it is sent over HTTPS).
  - The session data is encrypted using a secret key defined in config.py, preventing unauthorized tampering or access to sensitive information. Additionally, the system can implement session expiration policies to automatically log out users after a period of inactivity, further enhancing security.
- **SQL Injection Protection**:

  - The backend utilizes parameterized queries and Object-Relational Mapping (ORM) techniques to interact with the database. This prevents SQL injection attacks by ensuring that user input is always sanitized and treated as data rather than executable code.
  - By using libraries that abstract SQL queries (such as SQLAlchemy, if applicable), the system can minimize the risk of SQL injection vulnerabilities, ensuring that user inputs cannot alter the structure of database queries.
- **Cross-Site Scripting (XSS) Prevention**:
  - The backend sanitizes and escapes all user-generated content before rendering it in the frontend. This prevents attackers from injecting malicious scripts that could compromise the user's session or capture sensitive data.
  - Input validation checks ensure that only expected data formats are accepted, reducing the risk of XSS attacks.
- **Data Validation and Integrity**:
  - Throughout the backend operations, rigorous data validation processes are in place to ensure the integrity of the data stored in the database. For example, when a voter registers, the system checks that the provided voter ID is unique and that all required fields are completed accurately.
  - The backend also ensures that business rules, such as a voter being able to cast only one vote, are enforced before any data is committed to the database.
- **Logging and Monitoring**:
  - The backend maintains detailed logs of user activities, including registration, logins, and vote submissions. These logs are critical for auditing purposes and can help detect suspicious activities or potential breaches.
  - By implementing monitoring systems, the team can receive alerts for any unusual patterns or failed login attempts, allowing for timely investigation and response to potential threats.

### 2.5 Integration with the Database

The backend module seamlessly integrates with the SQLite database to manage all data-related operations. Here’s how the integration works:

- **Database Initialization**:
  - The backend initializes the database at startup, ensuring that all necessary tables (voters, candidates, votes) are created if they do not already exist. This is handled

in database.py through the init_db() function, which is called when the application starts.

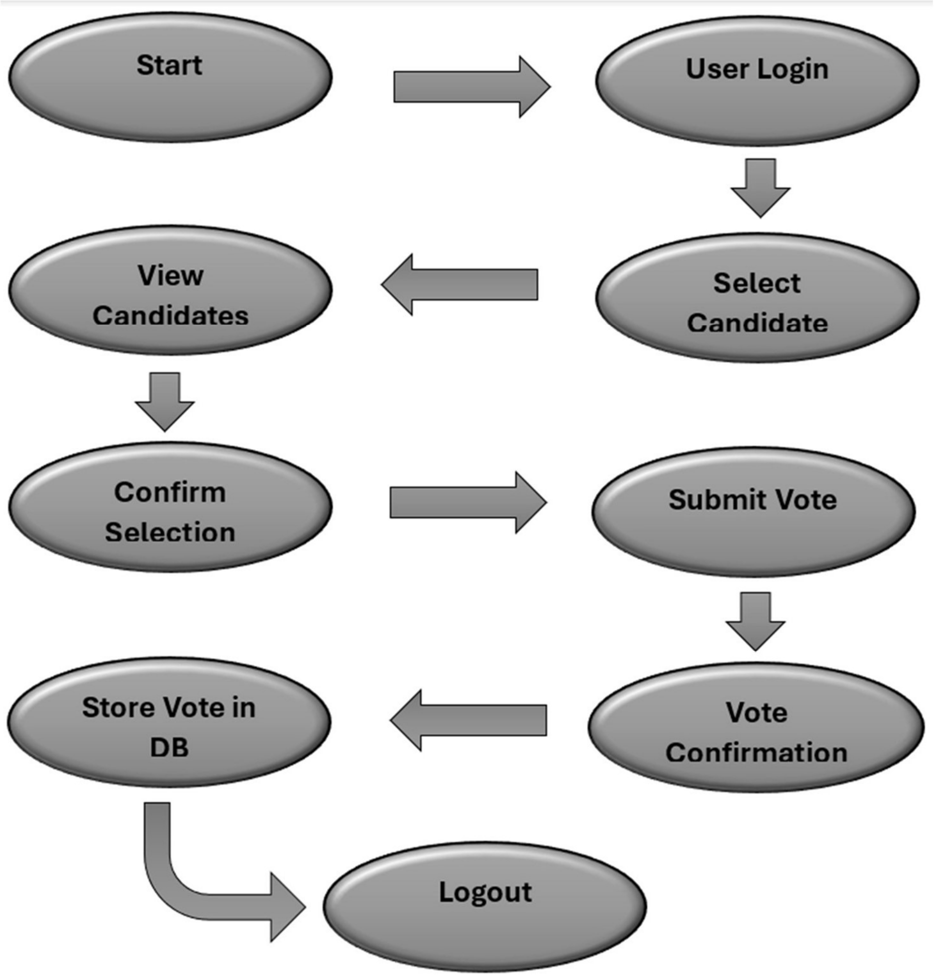
- **CRUD Operations**:
    - The backend provides functions to handle Create, Read, Update, and Delete (CRUD) operations on the database:
        - **Create**: When a new voter registers, the backend inserts a new record into the voters table.
        - **Read**: The system retrieves data from the candidates table to display candidate information on the frontend and checks user credentials during login.
        - **Update**: The backend updates the has_voted flag in the voters table once a user casts their vote, ensuring they cannot vote again.
        - **Delete**: While not typical for voting systems, the backend can handle deletion of user data if necessary, respecting user privacy and regulatory compliance.
- **Transaction Management**:
    - The backend employs transaction management to ensure that database operations are atomic. This means that either all changes are committed, or none are, preventing data inconsistencies in the event of an error during the voting process.
    - For instance, when a vote is cast, both the vote entry and the update to the voter's status (to indicate they have voted) are wrapped in a transaction. If any part of the operation fails, the entire transaction is rolled back.

### 2.6 Conclusion

The **Backend Processing and Logic Module** serves as the backbone of the online voting system, providing essential functionality that connects the user interface to the underlying database. It handles authentication, manages voting logic, and ensures data integrity while implementing robust security measures to protect sensitive user information. The integration with SQLite allows for efficient data management, enabling the system to provide a seamless experience for users while maintaining the highest standards of security and reliability. This module is designed to be scalable, ensuring that it can handle increased user loads and adapt to future enhancements.

## Module 3: Frontend User Interface Module

The Frontend User Interface Module is responsible for the presentation layer of the online voting system, providing users with an intuitive and engaging experience. This module encompasses all the HTML, CSS, and JavaScript files that create the visual components of the application, allowing users to interact with the system seamlessly. The frontend is designed to be user-friendly, ensuring that even those with limited technical skills can navigate the platform and cast their votes effortlessly.



### 3.1 Overview of the Frontend Module

The Frontend User Interface Module serves as the gateway for users to access the online voting system. It comprises several key files and folders that together create a cohesive and responsive user experience:

- **HTML Files**: These files structure the content of the web application, defining the layout and organization of information presented to the user.

- **CSS Files**: Cascading Style Sheets (CSS) are used to style the application, enhancing the visual appeal and ensuring consistency across different pages.
- **JavaScript Files**: JavaScript adds interactivity to the web application, enabling dynamic content updates and user interactions without requiring a full page reload.
- **Static Assets**: This folder contains images and other static files that are referenced throughout the application, such as candidate photos and icons.

The frontend is designed to work seamlessly with the backend, making AJAX calls to retrieve data and update the user interface dynamically.

### 3.2 File Breakdown and Responsibilities

#### 3.2.1 HTML Files

- **index.html**:
  - This is the landing page of the application, serving as the initial point of contact for users. It provides an overview of the voting process, including a brief description of the candidates and the importance of voting.
  - The page includes links to the login and registration pages, encouraging users to participate in the election.
- **login.html**:
  - This page allows users to enter their voter ID and password to access their accounts. It includes form validation to ensure that users provide valid credentials before submitting the form.
  - Clear feedback messages are displayed for incorrect login attempts, enhancing the user experience by guiding users through the authentication process.
- **register.html**:
  - The registration page enables new users to create an account by entering their personal information, including name, date of birth, phone number, and a unique voter ID.
  - The page incorporates client-side validation to ensure that required fields are filled out correctly and that the voter ID is unique. Users receive real-time feedback as they fill out the form, reducing errors and improving the registration process.
- **dashboard.html**:
  - Once logged in, users are directed to their dashboard, where they can view candidate profiles and cast their votes. The dashboard provides an overview of

available candidates, along with buttons to view detailed candidate information or proceed to vote.

  - This page includes dynamic content that is populated with data retrieved from the backend, ensuring that users have the most up-to-date information available.

- **candidate_list.html**:
  - This page displays a list of candidates participating in the election. Each candidate is presented with key details such as name, party affiliation, and a photo.
  - Users can click on a candidate's name to view more detailed information about their background, promises, and political views, facilitating informed voting decisions.
- **candidate_detail.html**:
  - This page provides an in-depth view of a selected candidate, including detailed descriptions of their political views, background, promises, and any other relevant information.
  - Users can return to the candidate list or navigate to the voting page from this screen, enhancing usability and ensuring easy access to information.
- **vote_confirmation.html**:
  - After a user successfully casts their vote, they are redirected to this confirmation page, which displays a confirmation message along with their unique reference number.
  - This page reassures users that their vote has been recorded and provides information on how to verify their vote if necessary.

### 3.2.2 CSS Files

- **styles.css**:
  - This CSS file defines the overall styling and layout of the web application. It ensures that all pages have a consistent look and feel, enhancing the user experience.
  - Key styles include font choices, color schemes, button styles, and responsive design features that allow the application to adapt to different screen sizes.
  - The CSS file also includes styles for form elements, ensuring they are visually appealing and easy to use. For example, input fields are styled for better accessibility, and error messages are highlighted to draw users' attention.

### 3.2.3 JavaScript Files

- **script.js**:

- This JavaScript file adds interactivity to the web application, enabling features like form validation and dynamic content updates.
- It validates user input on the login and registration pages, providing instant feedback if required fields are not filled or if the user tries to enter an invalid voter ID.
- The script also handles AJAX requests to the backend to fetch candidate data without requiring a full page reload, making the application more responsive and engaging.
- For example, when a user clicks on a candidate's name, JavaScript retrieves detailed candidate information and updates the DOM dynamically, providing a smooth user experience.

### 3.3 User Experience and Accessibility Considerations

The Frontend User Interface Module is designed with user experience (UX) and accessibility in mind. Here are some key features implemented to enhance usability:

- **Responsive Design**:
  - The application is designed to be responsive, meaning it adapts to various screen sizes, including desktops, tablets, and smartphones. This ensures that users can access the platform from any device, making the voting process more accessible.
- **User-Friendly Navigation**:
  - Navigation links are prominently displayed, allowing users to easily move between different pages. Clear call-to-action buttons (e.g., "Vote Now," "Register") guide users through the process, making it straightforward and intuitive.
- **Feedback Mechanisms**:
  - The frontend provides users with immediate feedback for actions taken, such as successful registration, login failures, or successful vote submission. This feedback is crucial for user satisfaction, as it keeps users informed about the status of their interactions with the system.
- **Accessibility Standards**:
  - The application adheres to web accessibility standards (e.g., WCAG) to ensure that all users, including those with disabilities, can navigate and use the system effectively. This includes features like:
    - Alternative text for images (e.g., candidate photos).

        - ARIA (Accessible Rich Internet Applications) attributes for dynamic content.
        - High-contrast color schemes for better visibility.

### 3.4 Integration with the Backend

The Frontend User Interface Module integrates closely with the Backend Processing and Logic Module to provide a seamless experience for users. Key aspects of this integration include:

- **AJAX Calls**:
    - The frontend makes AJAX calls to the backend to retrieve and submit data asynchronously. This allows the application to update parts of the user interface without reloading the entire page.
    - For instance, when users submit their login credentials, the frontend sends an AJAX request to the backend. If the login is successful, the user is redirected to the dashboard without a full page refresh, providing a smoother experience.
- **Dynamic Content Rendering**:
    - The frontend renders dynamic content based on the data received from the backend. For example, when a user visits the candidate list page, the frontend fetches candidate information and populates the page with the latest data from the database.
    - This ensures that users always see the most current information about candidates, enhancing the integrity of the voting process.
- **Error Handling**:
    - The frontend is designed to handle errors gracefully. If an AJAX request fails (e.g., due to a network issue), the frontend provides meaningful error messages to users, guiding them on how to proceed.
    - Similarly, if the backend returns validation errors during login or registration, the frontend displays these errors in real time, helping users correct their inputs immediately.

### 3.5 Security Considerations in the Frontend Module

Security is a critical aspect of the Frontend User Interface Module, especially given the sensitive nature of the voting process. Below are the key security mechanisms implemented:

- **Input Validation**:

  - The frontend performs rigorous input validation before submitting any data to the backend. This includes checking for valid email formats, ensuring that required fields are filled, and verifying that passwords meet complexity requirements.
  - By validating inputs on the client side, the application reduces the risk of malformed data being sent to the backend, which can lead to security vulnerabilities.
- **HTTPS Usage**:
  - The application is served over HTTPS to ensure that all data transmitted between the client and server is encrypted. This protects sensitive information, such as voter credentials and personal data, from interception by malicious actors.
- **Content Security Policy (CSP)**:
  - The application employs a Content Security Policy to mitigate risks associated with Cross-Site Scripting (XSS) attacks. This policy restricts the sources from which content (such as scripts and images) can be loaded, preventing unauthorized code from executing in the user's browser.
- **Cross-Origin Resource Sharing (CORS)**:
  - The backend is configured to handle Cross-Origin Resource Sharing (CORS) requests securely, ensuring that only authorized origins can access resources. This prevents unauthorized domains from making requests to the backend and protects against Cross-Site Request Forgery (CSRF) attacks.

### 3.6 Conclusion

The **Frontend User Interface Module** is a critical component of the online voting system, providing users with a seamless, intuitive, and engaging experience. By combining well-structured HTML, responsive CSS, and dynamic JavaScript, the module ensures that users can easily navigate the application, register, and cast their votes.

The integration with the backend enables real-time data updates, allowing users to view the latest candidate information and voting status without the need for page reloads. This interactivity is crucial in maintaining user engagement and encouraging participation in the electoral process. Furthermore, user-friendly navigation and feedback mechanisms enhance usability, guiding users through each step of the voting experience while ensuring they feel informed and confident in their actions.

Security is a paramount consideration within the Frontend User Interface Module. Through rigorous input validation, the use of HTTPS, and implementation of a Content Security Policy, the application effectively mitigates risks associated with data breaches and attacks, thereby protecting the sensitive information of voters.

Overall, this module not only serves as the visual and interactive face of the online voting system but also embodies the principles of user-centered design, security, and accessibility. By prioritizing these elements, the Frontend User Interface Module plays a vital role in fostering a trustworthy and efficient voting environment, empowering users to exercise their democratic rights with ease and confidence.

## 4. Database Module

The **Database Module** is the core of our online voting system. It manages all the data-related aspects of the application, providing a solid foundation for storing and retrieving information related to voters, candidates, and votes. This module serves as the backbone of the system, ensuring data persistence and integrity. Our project uses **SQLite**, a simple yet powerful database management system, which allows us to create, store, and manage the data in an efficient and structured way.

While the previous module focused on the overall structure and creation of the tables (voters, candidates, and votes), this section will delve into the functionality of the database, how it integrates with the system, and the role it plays in ensuring smooth operations. The database module is essential for handling all user interactions with data, including user registration, login, candidate display, and vote recording.

### 4.1 SQLite Database

SQLite was selected for this project due to its simplicity and the fact that it doesn't require additional server installations. It's a serverless database engine, which makes it ideal for small to medium-sized projects like ours. Since SQLite stores the entire database in a single file (voting_system.db), the deployment and management of the database become more manageable. Its portability means we can easily move or copy the database file between different environments without complications.

One of the major reasons for choosing SQLite is its **ACID (Atomicity, Consistency, Isolation, Durability)** compliance, which ensures the data remains consistent even in cases where the system might experience interruptions or unexpected shutdowns. This is vital for our project, as every vote cast must be accurately recorded and preserved without any risk of corruption.

Additionally, SQLite offers **minimal configuration**. Given that this is a student-led project, focusing on a self-contained system allows us to devote our attention to other aspects, such as user experience and security, rather than complex database configurations. We can easily integrate it into our Python backend using Flask, as SQLite is supported out-of-the-box with Python's sqlite3 module.

### 4.2 SQL Queries and Database Operations

The entire operation of our system is supported by carefully designed SQL queries. These queries form the logic of how data is managed within the system. Below, we break down some of the key operations that interact with the database.

#### 4.2.1 Registration of Voters

The system allows users (voters) to register, and the registration process involves inserting user details into the voters table. The voter's information is validated before being added to the database. This ensures that all user input is correct and avoids duplication of voters with the same credentials.

When a new voter registers, the following information is collected:

- First and last name
- Date of birth
- Phone number
- Voter ID
- Password

The backend handles the insertion of this information securely into the voters table. Additionally, sensitive data such as passwords are stored using hashing techniques to ensure security. The encrypt algorithm is used to hash passwords, which means that even if the database is compromised, the actual passwords remain protected.

### 4.2.2 Voter Login and Authentication

Once a voter has registered, they can log into the system using their voter ID and password. The system checks the credentials by comparing the hashed password stored in the voters table against the password entered by the user during login. This is done using the following steps:

1. **Password Hashing**: The entered password is hashed and compared with the stored hash.
2. **Verification**: If the voter ID and password match, the voter is authenticated and granted access to the voting functionality.
3. **Session Management**: Upon successful login, the system creates a user session. This session helps maintain the user's login state across multiple pages, ensuring that voters do not have to re-enter their credentials every time they navigate between different parts of the system.

The login process also ensures that each voter can only log in once to cast their vote, preventing multiple logins from the same account.

### 4.2.3 Retrieving Candidate Information

The system fetches the list of available candidates from the candidates table. This happens after the voter logs in, ensuring that they are presented with the correct set of candidates to choose from. The candidate information retrieved includes their names, party affiliations, and campaign promises. The system ensures that only valid, registered candidates are shown to the voter.

### 4.2.4 Voting Process and Validation

The voting process is the heart of the system, and the database plays a central role in ensuring that votes are recorded accurately. When a voter selects a candidate, several steps occur in the backend:

1. **Vote Check**: Before proceeding with the vote, the system checks the has_voted flag in the voters table to verify if the voter has already voted. This flag is crucial to maintain the integrity of the voting process, ensuring each voter casts only one vote.
2. **Vote Recording**: Once the voter selects a candidate, the system records the vote in the votes table. It stores the voter's ID, the candidate's ID, and the exact timestamp of the vote.
3. **Vote Confirmation**: After successfully casting a vote, the system generates a unique reference number for the voter as proof of their vote. This number is stored in the votes table alongside the corresponding voter and candidate details.

Each vote is tied directly to a voter, and the system ensures that the same voter cannot vote multiple times by setting the has_voted flag to 1 after they vote.

### 4.2.5 Data Integrity and Consistency

Throughout the process, the database module ensures **data integrity**. This is achieved through several key mechanisms:

- **Foreign Key Constraints**: By linking tables using foreign key relationships (e.g., linking votes to voters and candidates), the system ensures that all data remains consistent. For example, a vote cannot be cast for a non-existent candidate.
- **Transaction Handling**: All voting operations are encapsulated in transactions. This means that if any part of the voting process fails (such as recording the vote or generating a reference number), the transaction is rolled back, ensuring that the database does not end up in an inconsistent state.

Additionally, SQLite provides built-in protection against **data corruption**. By following the ACID principles, the database ensures that even if the system crashes or loses power, the data remains intact and consistent.

### 4.2.6 Data Security

Security is a major concern in any voting system, and our database module is designed to address this in several ways:

- **Password Hashing**: As mentioned, passwords are hashed before being stored in the database, ensuring that sensitive voter data is protected.
- **SQL Injection Prevention**: All SQL queries interacting with the database are parameterized. This prevents SQL injection attacks, where malicious users could otherwise manipulate the system by injecting harmful SQL code.
- **Data Access Controls**: The system has strict access controls. Admin functions, such as viewing vote counts and voter data, are only accessible to authenticated admin users. Regular voters cannot access or alter critical data, ensuring the security of the entire system.

### 4.2.7 Scalability and Future Enhancements

Although SQLite is used in this version of the project for its simplicity, the database structure has been designed to allow easy migration to more robust database systems like PostgreSQL or MySQL if required in the future. This scalability ensures that the system can grow beyond its current scope to handle larger elections or more complex data operations.

### 4.3 Purpose and Role in the System

The **Database Module** serves multiple purposes within the system:

- **Persistent Data Storage**: The module ensures that all the data entered by users (voters, candidates, votes) is stored permanently, enabling consistent access to that information over time.
- **Reliability**: By using foreign key relationships and transaction handling, the database ensures that all operations—particularly voting—are reliable and consistent.
- **Security and Integrity**: The database ensures that sensitive information, such as passwords and vote records, is stored securely and that data integrity is maintained throughout the voting process.
- **Accountability**: The timestamp and reference number for each vote provide an audit trail, ensuring that each vote can be traced back to a specific voter and candidate, offering transparency and accountability in the voting process.

### Conclusion

The **Database Module** is the backbone of our online voting system, ensuring that all the core functionalities—such as registration, authentication, vote recording, and candidate retrieval—operate smoothly. It ensures the system's integrity by maintaining accurate and consistent data records, leveraging SQLite's simplicity and power. From securely storing user credentials to preventing double voting, the database is vital to the system's success. Moreover, the security measures embedded within the module protect against unauthorized access and data tampering, ensuring a trustworthy and reliable platform for voters. This module not only supports the current requirements but is also scalable for future enhancements, making it an essential component of our project.

## 5. Backend (Business Logic) Module

The **Backend Module** is the central component of our online voting system, handling all the core logic that connects the user interface (UI) with the database. It processes user requests, applies business logic, and ensures the proper functioning of the system by managing how data flows between the frontend and the database. This module can be considered the "brain" of the system, as it is responsible for ensuring that all operations are carried out smoothly and in alignment with the system's objectives.

This module is primarily built using **Python** and leverages the **Flask framework** to handle requests and responses. Flask, being a lightweight web framework, is perfect for this project as it allows us to structure the backend in a way that is both efficient and scalable. Below, we delve into each of the key components of this module and their respective functionalities.

### 5.1 Python Scripts

The backend module is composed of several Python scripts, each responsible for a specific aspect of the system's operations. These scripts are modular, allowing for better code management and scalability. The key files include app.py, database.py, models.py, and config.py.

### 5.1.1 app.py

app.py is the main driver of the backend system. This file handles the majority of the communication between the frontend (user interface) and the database. It is responsible for routing the different HTTP requests (GET, POST) made by users and determining how those requests are processed.

- **HTTP Request Handling**: app.py manages incoming user requests, whether it's for logging in, registering, voting, or fetching candidate details. For instance, when a user submits their login credentials, app.py receives the request and verifies the credentials by interacting with the database.
- **Routing**: Flask uses the concept of routes to define the different endpoints of the web application. For example:
  - /login: Routes the user to the login page.
  - /register: Routes the user to the registration page.
  - /vote: Directs the user to the voting page after successful authentication. These routes are defined in app.py and are linked to corresponding functions that process the request and deliver the appropriate response.
- **Business Logic Implementation**: app.py is also responsible for executing the core business logic. For example, when a user votes, app.py ensures that the vote is properly recorded in the database, and the user cannot vote again. Similarly, it checks if a user is authenticated before allowing access to sensitive parts of the system.

### 5.1.2 database.py

database.py is a crucial component of the backend that handles all interactions with the SQLite database. This file contains functions for establishing and closing database connections, executing SQL queries, and handling the results.

- **Database Connection**: The file provides a seamless connection between the Python backend and the SQLite database. It ensures that whenever app.py needs to access the database (e.g., to retrieve voter or candidate data), the connection is opened, the query is executed, and the connection is properly closed afterward to ensure optimal resource management.
- **Table Creation**: During the initial setup of the system, database.py ensures that all required tables (voters, candidates, votes) are created. This setup ensures the backend has a reliable structure in place to manage all voting data.
- **Query Execution**: All SQL queries are executed via this file. For example, when a user submits their registration form, database.py inserts the user's details into the voters table. Similarly, during voting, this file handles the insertion of vote records into the votes table.

### 5.1.3 models.py

models.py defines the structure and relationships of the data stored in the database. By defining database models, this file helps streamline database interactions by organizing the data in a structured way.

- **ORM (Object-Relational Mapping)**: Although our project primarily uses raw SQL for database queries, models.py could be extended in the future to support an ORM. ORMs allow developers to interact with the database using Python objects rather than raw SQL queries, which can simplify complex database operations.
- **Data Validation**: models.py can be used to validate the structure of data before it is committed to the database. This helps ensure that only valid data is processed, reducing the risk of errors during database interactions.

### 5.1.4 config.py

The **config.py** file is responsible for managing the application's configuration settings. This file contains paths and variables that are crucial for the system to run correctly.

- **Database Path**: One of the primary functions of this file is to store the path to the SQLite database file (voting_system.db). By centralizing this information, config.py makes it easier to update the database path if needed without having to modify multiple files.
- **Other Configuration Settings**: In addition to the database path, config.py may also store other settings such as secret keys for session management, debugging modes, and more. These configurations ensure that the system runs smoothly in different environments (e.g., development vs. production).

## 5.2 Core Functionalities of the Backend Module

The backend module is responsible for implementing the business logic that drives the voting system. Below are some of the key functionalities that the backend handles:

### 5.2.1 User Authentication

One of the most critical functions of the backend is managing user authentication. The system ensures that only registered voters can access the voting functionality, and that they can only vote once.

- **Login Validation**: The backend verifies the voter's credentials (Voter ID and password) by checking the information against the records in the voters table. If the credentials are valid, the user is authenticated and granted access to the voting page.
- **Password Hashing**: The backend uses hashing techniques (e.g., bcrypt) to securely store and validate passwords, ensuring that even if the database is compromised, passwords remain protected.
- **Session Management**: After a successful login, the system creates a user session, which allows the voter to navigate the site without having to re-enter their credentials on each page. Sessions are securely managed to prevent unauthorized access.

### 5.2.2 Candidate Display and Vote Recording

Once a voter logs in, the backend is responsible for displaying the list of candidates. The system fetches the candidates' details (names, party affiliations, campaign promises) from the candidates table and presents them to the voter.

- **Vote Recording**: After the voter selects their preferred candidate, the backend ensures that the vote is recorded in the votes table, along with a timestamp and the voter's ID.
- **Vote Confirmation**: After recording the vote, the system generates a unique reference number for the voter, confirming that their vote has been successfully cast.

### 5.2.3 Admin Functionality

In addition to voter-related functions, the backend module also provides functionality for the system's admin users.

- **Admin Login**: Admins log in using predefined credentials stored in the config.py file. The backend ensures that only admins with the correct username and password can access the administrative dashboard.
- **Vote Analytics**: Once logged in, admins can view analytics related to the votes cast. The backend retrieves data from the votes table and provides a summary of the voting results, including how many votes each candidate has received.

### 5.2.4 Error Handling and Validation

The backend is responsible for handling errors and ensuring that the system runs smoothly. Some of the error handling includes:

- **Form Validation**: The backend checks user input during registration and voting to ensure that all required fields are filled out correctly.
- **Database Errors**: If there are any issues during database interactions (e.g., failed queries), the backend handles those errors gracefully, ensuring that the system does not crash.

### 5.2.5 Security Measures

Given the sensitive nature of a voting system, the backend module includes several security measures to protect against malicious attacks:

- **SQL Injection Prevention**: By using parameterized queries, the backend prevents SQL injection attacks, where malicious users could try to manipulate the system by entering harmful SQL code into form fields.
- **Access Control**: The backend restricts access to certain pages based on user roles (voter vs. admin), ensuring that only authorized users can access sensitive information or functionality.

## 5.3 Purpose of the Backend Module

The backend module plays a crucial role in ensuring that the online voting system functions as intended. It serves as the intermediary between the frontend and the database, processing all user actions and managing the flow of data. Without the backend module, the system would be unable to handle essential tasks such as authentication, voting, and data retrieval.

Additionally, the backend implements the business logic that governs the entire voting process. By validating inputs, processing votes, and maintaining data security, the backend ensures that the system operates with integrity, providing a seamless and secure experience for all users.

## Conclusion

The **Backend (Business Logic) Module** is the engine that drives the entire online voting system. It ensures that all user interactions—whether they be voter registration, login, candidate selection, or vote casting—are processed correctly and securely. By leveraging Python, Flask, and SQLite, this module creates a smooth connection between the user interface and the database, executing critical business logic while maintaining data integrity and security. Through careful routing, authentication, and database interaction, the backend ensures that the system is robust and reliable, providing a streamlined experience for both voters and admins alike. As the core of the project, this module not only supports current functionality but also provides a scalable framework for future enhancements, making it an indispensable part of the system's architecture.

# Chapter 4

## Hardware and Software Used

This section outlines the hardware and software that our group utilized for the development of the **Online Voting System** project. Our aim is to provide a comprehensive overview of all the tools and components necessary to successfully complete this project. This list ensures that we have considered every aspect, no matter how small, required for developing and running the system smoothly.

### Hardware Requirements

1. **Computer/Laptop**:
   As a group of students working on this project, each team member used their personal laptops for development and testing. The minimum required specifications for smooth operation and multitasking are as follows:
   - **Processor**: Intel Core i5 or equivalent (to ensure faster processing and better performance while running the code editor, servers, etc.).
   - **RAM**: 8 GB (16 GB is recommended for multitasking, especially when running multiple tools such as the code editor and web browser simultaneously).
   - **Storage**: At least 256 GB SSD for faster access to files and efficient handling of project data.
   - **Operating System**: We developed on a mix of **Windows 10/11**, **macOS**, and **Ubuntu** systems, ensuring compatibility with all major platforms.
2. **Internet Connection**:
   A stable internet connection was necessary for multiple reasons, including:
   - Downloading required software libraries and dependencies.
   - Collaboration and sharing the project through version control (GitHub).
   - Accessing online resources for troubleshooting during development.
3. **Backup Power Supply (UPS)** (Optional):
   While this is not mandatory, some members used an uninterrupted power supply (UPS) to avoid losing data during development in case of power failures. This ensured continuous progress during coding and testing sessions.

### Software Requirements

The project required several software tools and frameworks to handle both the backend and frontend functionalities, as well as to manage the database and run the server. Here is a detailed breakdown of the software we used:

1. **Operating System**:
   We ensured that our project was cross-platform by developing and testing on multiple operating systems. The project is compatible with the following:
   - **Windows 10/11**
   - **macOS**
   - **Ubuntu/Linux**
2. **Code Editor/IDE**:
   Each member used their preferred code editor or integrated development environment (IDE) to write the code. Some popular tools we used include:
   - **Visual Studio Code (VS Code)**: This was the most widely used editor by our team due to its lightweight design, user-friendly interface, and vast extension library. It allowed us to integrate Flask, SQLite, and Git for streamlined development.
   - **PyCharm**: A few members preferred PyCharm for its powerful Python-specific features, including debugging and code navigation.
   - **Sublime Text**: For those who wanted a minimalist editor with faster performance.
3. **Python Interpreter**:
   The backend of our project was developed in Python, and we used the **Python 3.8+** version to take advantage of its wide range of libraries and modules for web development and database management. Python was crucial for running the backend logic, connecting the database, and rendering templates.
   - Python can be downloaded from python.org.
4. **Flask Framework**:
   For the backend, we chose **Flask**, a lightweight and flexible web framework. Flask helped us build the server-side logic, manage HTTP requests, and render HTML pages with dynamic data. Flask also made it easy to link our database (SQLite) to the user interface.
   - Flask was installed using the Python package manager (pip):

     *pip install flask*
5. **SQLite Database**:
   **SQLite** served as the database engine for our project, storing data such as voter

information, candidate details, and voting records. SQLite was selected for its simplicity and ease of use, as it does not require a separate database server.

- The database file (voting_system.db) is stored locally in the project directory.
- **DB Browser for SQLite**: We used this GUI tool to view and manage the contents of the SQLite database for debugging and testing purposes.

6. **Web Browser**:
   For testing the frontend of the project and interacting with the voting system, each member used a modern web browser. We tested compatibility with:
   - **Google Chrome**
   - **Mozilla Firefox**
   - **Microsoft Edge**
   - **Safari** (for macOS users)
7. **HTML/CSS/JavaScript**:
   The frontend of our project was built using standard web technologies:
   - **HTML**: We used this to create the structure of each webpage.
   - **CSS**: This was used to style the pages, ensuring the voting system had a clean and professional appearance.
   - **JavaScript**: We added interactivity and dynamic content to the pages using JavaScript.

     Additionally, some libraries and frameworks were used to improve the design and responsiveness:
   - **Bootstrap**: A CSS framework that helped in creating responsive layouts.
   - **jQuery**: A lightweight JavaScript library used for simplifying event handling and DOM manipulation.
8. **Version Control (Git/GitHub)**:
   Collaboration among the team was streamlined by using **Git** for version control. Git allowed us to track changes in the code, revert to previous versions if necessary, and collaborate effectively without conflicts.
   - We used **GitHub** as the central repository for our project. Each member was able to contribute code, review changes, and merge pull requests. This was particularly useful when multiple members worked on different modules simultaneously.
9. **Python Libraries**:
   We used several additional Python libraries for specific functionalities:

   - **SQLite3**: For interacting with the SQLite database.
   - **WTForms**: For handling form data and validation (e.g., for registration and voting forms).
   - **Flask-WTF**: A wrapper for integrating WTForms into Flask.
   - **Flask-Session**: For managing user sessions during login and voting processes.
   - **Datetime Module**: To handle timestamps when votes were cast.

10. **Terminal/Command Line**:
    We used the terminal or command prompt to run the Flask server and execute various commands, including managing dependencies, running the development server, and interacting with the database.

```
flask run  # Start the Flask development server
```

**Conclusion**

The combination of the above hardware and software components enabled our team to successfully develop, test, and deploy the **Online Voting System** project. Each software tool played a specific role in ensuring that the system ran efficiently, from writing code and handling the database to ensuring smooth interaction between the frontend and backend. Our hardware specifications ensured that all tasks were performed smoothly, even when multiple processes were running simultaneously.

By using industry-standard tools like Flask, SQLite, and Git, we ensured that our project is scalable and can be extended further if needed. The choice of these tools also aligns well with the objective of creating a lightweight, easy-to-deploy solution.

# Chapter 5

# Coding Used

In this section, we will present the code that forms the backbone of our **Online Voting System**. Each of the key files plays an essential role in managing the database, processing user requests, and enabling communication between the frontend and backend. Below, we break down the coding into individual files with a brief explanation of their purpose, followed by space where the relevant code is inserted.

## 1. app.py

**Purpose:** The app.py file is the main entry point of our application. It handles HTTP requests and directs users to the appropriate page based on their actions (login, registration, viewing candidates, voting, etc.). This file manages user sessions and ensures that users are correctly authenticated before accessing voting features. It also interacts with the database to record votes and retrieve data.

**Code:**

```
from flask import Flask, render_template, request, redirect, url_for,
flash, session
from werkzeug.security import generate_password_hash,
check_password_hash
import sqlite3
from database import create_connection
from models import Voter, Candidate, Vote
import uuid
from datetime import datetime

app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Change this to a random secret
key

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        voter_id = request.form['voter_id']
        password = request.form['password']

        conn = create_connection()
        cur = conn.cursor()
        cur.execute("SELECT * FROM voters WHERE voter_id = ?",
(voter_id,))
        user = cur.fetchone()
        conn.close()

        if user and check_password_hash(user[6], password):
            session['user_id'] = user[0]
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid credentials', 'error')

    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        last_name = request.form['last_name']
        date_of_birth = request.form['date_of_birth']
        phone_number = request.form['phone_number']
        voter_id = request.form['voter_id']
        password = request.form['password']

        conn = create_connection()
        cur = conn.cursor()
        cur.execute("SELECT * FROM voters WHERE voter_id = ?",
(voter_id,))
        existing_user = cur.fetchone()

        if existing_user:
            flash('Voter ID already exists', 'error')
        else:
            hashed_password = generate_password_hash(password)
            cur.execute("INSERT INTO voters (name, last_name,
date_of_birth, phone_number, voter_id, password) VALUES (?, ?, ?, ?,
?, ?)",
                        (name, last_name, date_of_birth, phone_number,
voter_id, hashed_password))
            conn.commit()
            flash('Registration successful', 'success')
            return redirect(url_for('login'))
        conn.close()

    return render_template('register.html')

@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    conn = create_connection()
    if conn is None:
        flash('Error connecting to the database', 'error')
        return redirect(url_for('index'))

    try:
        cur = conn.cursor()
        cur.execute("SELECT * FROM voters WHERE id = ?",
(session['user_id'],))
        user = cur.fetchone()

        cur.execute("SELECT * FROM candidates")
        candidates = cur.fetchall()

        if not candidates:
            # If no candidates are found, add so
            # me sample candidates
            sample_candidates = [
                ('John Doe', 'Party 1', 'john_doe.jpg', 'Promise
1\nPromise 2', 'Assets info', 'Liabilities info', 'Background info',
'Political views', 'Regional views'),
                ('Jane Smith', 'Party B', 'jane_smith.jpg', 'Promise
1\nPromise 2', 'Assets info', 'Liabilities info', 'Background info',
'Political views', 'Regional views'),
            ]
            cur.executemany("""
                INSERT INTO candidates (name, party, photo_url,
promises, assets, liabilities, background, political_views,
regional_views)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
            """, sample_candidates)
            conn.commit()

            # Fetch the newly added candidates
            cur.execute("SELECT * FROM candidates")
            candidates = cur.fetchall()
        return render_template('dashboard.html', user=user,
candidates=candidates)
    except sqlite3.Error as e:
        flash(f'An error occurred: {str(e)}', 'error')
        return redirect(url_for('index'))
    finally:
        conn.close()

@app.route('/candidate/<int:candidate_id>')
def candidate_detail(candidate_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    conn = create_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM candidates WHERE id = ?",
(candidate_id,))
    candidate = cur.fetchone()
    conn.close()

    return render_template('candidate_detail.html',
candidate=candidate)

@app.route('/vote/<int:candidate_id>', methods=['POST'])
def vote(candidate_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    conn = create_connection()
    cur = conn.cursor()
    cur.execute("SELECT has_voted FROM voters WHERE id = ?",
(session['user_id'],))
    has_voted = cur.fetchone()[0]

    if has_voted:
        flash('You have already voted', 'error')
        return redirect(url_for('dashboard'))

    reference_number = str(uuid.uuid4())
    timestamp = datetime.now().isoformat()

    cur.execute("INSERT INTO votes (voter_id, candidate_id, timestamp,
reference_number) VALUES (?, ?, ?, ?)",
                (session['user_id'], candidate_id, timestamp,
reference_number))
    cur.execute("UPDATE voters SET has_voted = 1 WHERE id = ?",
(session['user_id'],))
```

```
    conn.commit()
    conn.close()

    flash(f'Thank you for voting! Your reference number is
{reference_number}', 'success')
    return redirect(url_for('vote_confirmation',
reference_number=reference_number))

@app.route('/vote_confirmation/<reference_number>')
def vote_confirmation(reference_number):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    return render_template('vote_confirmation.html',
reference_number=reference_number)

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    return redirect(url_for('index'))

if __name___ == '__main__':
    app.run(debug=True)
```

## 2. database.py

**Purpose:** The database.py file is responsible for managing the connection between the application and the SQLite database. It creates and maintains the database structure, including tables for voters, candidates, and votes. This file handles any SQL queries required by the app, such as inserting new voters or fetching the list of candidates.

**Code:**

```
import sqlite3
from sqlite3 import Error
import os

def create_connection():
    conn = None
    try:
        # Get the absolute path to the database file
        db_path =
os.path.abspath(os.path.join(os.path.dirname(___file__), 'database',
'voting_system.db'))

        # Ensure the directory exists
        os.makedirs(os.path.dirname(db_path), exist_ok=True)

        print(f"Attempting to connect to database at: {db_path}")  #
Debug print
        conn = sqlite3.connect(db_path)
        print(f"Successfully connected to the database at {db_path}")
        return conn
    except Error as e:
        print(f"Error connecting to the database: {e}")
        print(f"Current working directory: {os.getcwd()}") # Debug
print
        print(f"Directory contents: {os.listdir()}")  # Debug print
    return conn

def create_table(conn, create_table_sql):
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(f"Error creating table: {e}")

def init_db():
    database = r"database/voting_system.db"

    sql_create_voters_table = """ CREATE TABLE IF NOT EXISTS voters (
                                    id integer PRIMARY KEY,
                                    name text NOT NULL,
                                    last_name text NOT NULL,
                                    date_of_birth text NOT NULL,
                                    phone_number text NOT NULL,
                                    voter_id text NOT NULL UNIQUE,
                                    password text NOT NULL,
                                    has_voted integer DEFAULT 0
                                ); """

    sql_create_candidates_table = """CREATE TABLE IF NOT EXISTS
candidates (
                                    id integer PRIMARY KEY,
                                    name text NOT NULL,
                                    party text NOT NULL,
                                    photo_url text NOT NULL,
                                    promises text NOT NULL,
                                    assets text NOT NULL,
                                    liabilities text NOT NULL,
                                    background text NOT NULL,
                                    political_views text NOT NULL,
                                    regional_views text NOT NULL
                                );"""

    sql_create_votes_table = """CREATE TABLE IF NOT EXISTS votes (
                                id integer PRIMARY KEY,
                                voter_id integer NOT NULL,
                                candidate_id integer NOT NULL,
                                timestamp text NOT NULL,
                                reference_number text NOT NULL UNIQUE,
                                FOREIGN KEY (voter_id) REFERENCES
voters (id),
                                FOREIGN KEY (candidate_id) REFERENCES
candidates (id)
                            );"""

    conn = create_connection()

    if conn is not None:
        create_table(conn, sql_create_voters_table)
        create_table(conn, sql_create_candidates_table)
        create_table(conn, sql_create_votes_table)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")

if __name__ == '__main__':
    init_db()
```

## 3. models.py

**Purpose:** The models.py file contains the structured representation of the database. Instead of writing raw SQL queries, we use models to abstract the database tables. This file defines how data is stored and retrieved, making it easier to interact with the database in an object-oriented manner.

**Code:**

```
# models.py

from dataclasses import dataclass


@dataclass

class Voter:

    id: int
    name: str
    last_name: str
    date_of_birth: str
    phone_number: str
    voter_id: str
    password: str
    has_voted: bool = False

@dataclass
class Candidate:
    id: int
    name: str
    party: str
    photo_url: str
    promises: str
    assets: str
    liabilities: str
    background: str
    political_views: str
    regional_views: str

@dataclass
class Vote:
    id: int
    voter_id: int
    candidate_id: int
    timestamp: str
    reference_number: str
```

### 4. Templates (HTML Files)

**Purpose:** The frontend of the application is composed of HTML templates stored in the templates/ directory. These files define the structure of the web pages the user interacts with, including the login page, registration page, candidate list, voting page, and confirmation page. These files are rendered dynamically using Flask to display data (such as candidate names or confirmation messages) fetched from the backend.

**Code:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Login - Online Voting System</title>
    <style>
        body {
            font-family: 'Georgia', serif;
            line-height: 1.6;
            color: #333;
            background-color: #f0f0f0;
            margin: 0;
            padding: 0;
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .container {
            background-color: #e6e6e6;
            border-radius: 8px;
            padding: 2rem;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 400px;
            margin: 2rem;
        }

        h1 {
```

```
    font-size: 2.3rem;
    margin-bottom: 1.5rem;
    text-align: center;
    color: #2c3e50;
}

form {
    display: grid;
    gap: 1.5rem;
}

.form-group {
    display: flex;
    flex-direction: column;
}

label {
    margin-bottom: 0.5rem;
    font-weight: bold;
    color: #34495e;
}

input {
    padding: 0.8rem;
    border: 1px solid #bdc3c7;
    border-radius: 4px;
    background-color: #f5f5f5;
    color: #333;
    font-size: 1rem;
    transition: all 0.3s ease;
}

input:focus {
    outline: none;
    border-color: #3498db;
    box-shadow: 0 0 5px rgba(52, 152, 219, 0.5);
}

button {
    padding: 1rem;
    border: none;
    border-radius: 4px;
    background-color: #2c3e50;
    color: #fff;
    font-size: 1.1rem;
    cursor: pointer;
    transition: all 0.3s ease;
```

```
            margin-top: 1rem;
        }

        button:hover {
            background-color: #34495e;
        }

        p {
            text-align: center;
            margin-top: 1rem;
            color: #34495e;
        }

        a {
            color: #2980b9;
            text-decoration: none;
            font-weight: bold;
            transition: all 0.3s ease;
        }

        a:hover {
            text-decoration: underline;
        }

        @media (max-width: 600px) {
            .container {
                margin: 1rem;
                padding: 1.5rem;
            }

            h1 {
                font-size: 2rem;
            }

            input, button {
                font-size: 0.9rem;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Login</h1>
        <form method="POST" action="{{ url_for('login') }}">
            <div class="form-group">
                <label for="voter_id">Voter ID</label>
```

```
                      <input  type="text"  id="voter_id"  name="voter_id"
required>
            </div>
            <div class="form-group">
                <label for="password">Password</label>
                  <input type="password" id="password" name="password"
required>
            </div>
            <button type="submit">Login</button>
        </form>
          <p>Don't  have  an  account?  <a  href="{{  url_for('register')
}}">Register here</a></p>
    </div>
</body>
</html>
```

## Explanation of Integration

Our project follows a **Model-View-Controller (MVC)** structure, where:

- **Model**: The models.py file defines the structure of our database and how data is stored and retrieved.
- **View**: The HTML templates within the templates/ directory define what users see when they visit our web application.
- **Controller**: The app.py file acts as the controller, handling user inputs (HTTP requests) and managing the flow of data between the view and the model (database).

Each component is crucial for ensuring the smooth operation of the voting system, with the backend (Python) and frontend (HTML) communicating seamlessly to offer a streamlined user experience.

## Conclusion

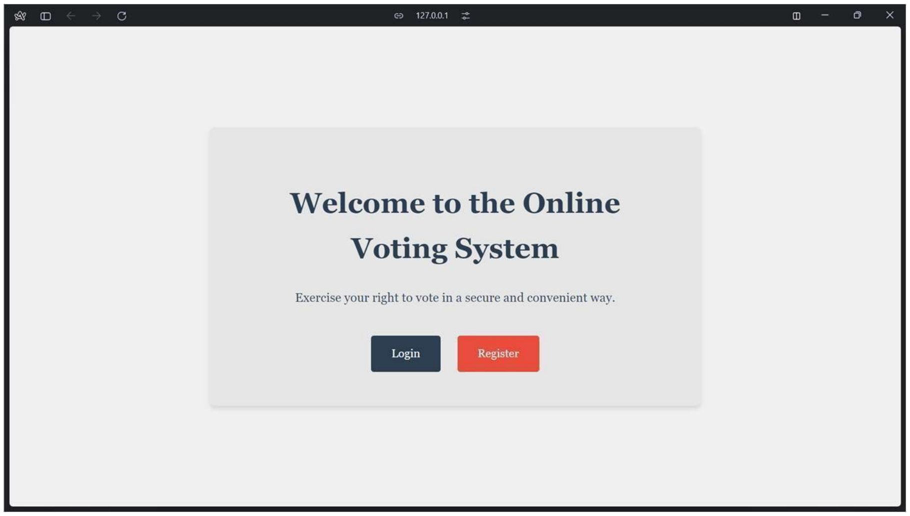
In conclusion, the coding architecture we implemented efficiently bridges the gap between user interactions and database management. Each module—app.py, database.py, models.py, and config.py—plays a vital role in ensuring that data flows smoothly and securely between the user interface and the backend database. The use of clear, structured code makes the system easily maintainable, scalable, and adaptable for future enhancements.

# Chapter 6

# Results and Screenshots

In this section, we present the results of our **Online Voting System** project, demonstrating the functionality and effectiveness of our implementation. We have included screenshots of various pages within the application, along with detailed descriptions to illustrate how the system works. This documentation aims to showcase our understanding of the project and the successful execution of our objectives.

### 1. Homepage (index.html)

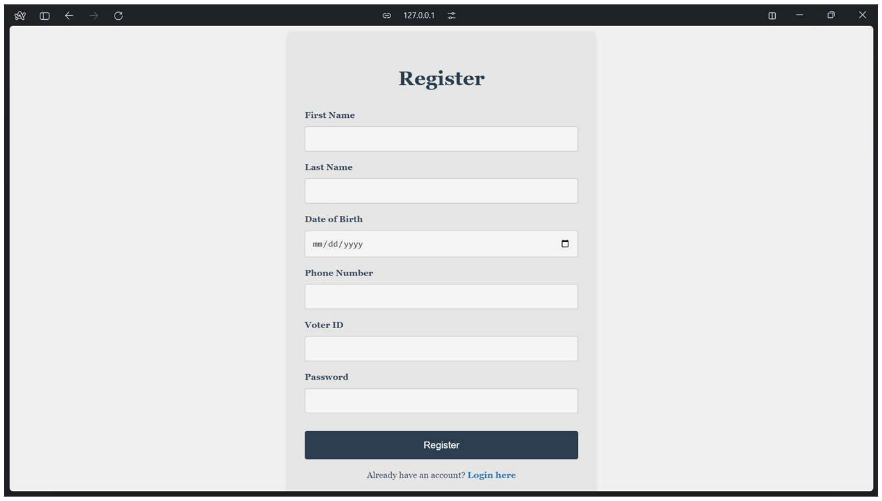


Welcome to the Online Voting System

Exercise your right to vote in a secure and convenient way.

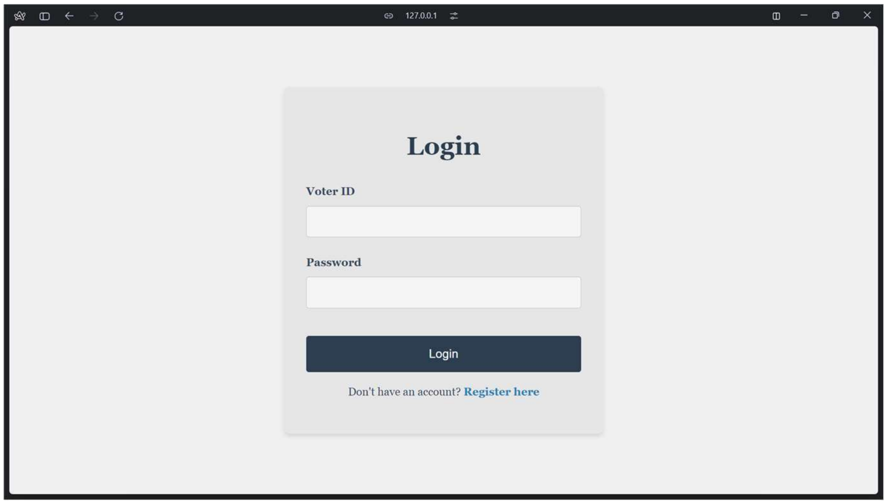
**Description:** The homepage serves as the initial entry point for our application. Here, users can choose between registering as a new voter or logging in if they already have an account. The clean layout and straightforward design are intended to provide an intuitive user experience, making it easy for participants to navigate the voting process. The prominent buttons guide users to the registration and login pages, ensuring a smooth transition to the next steps.

### 2. Voter Registration Page (register.html)



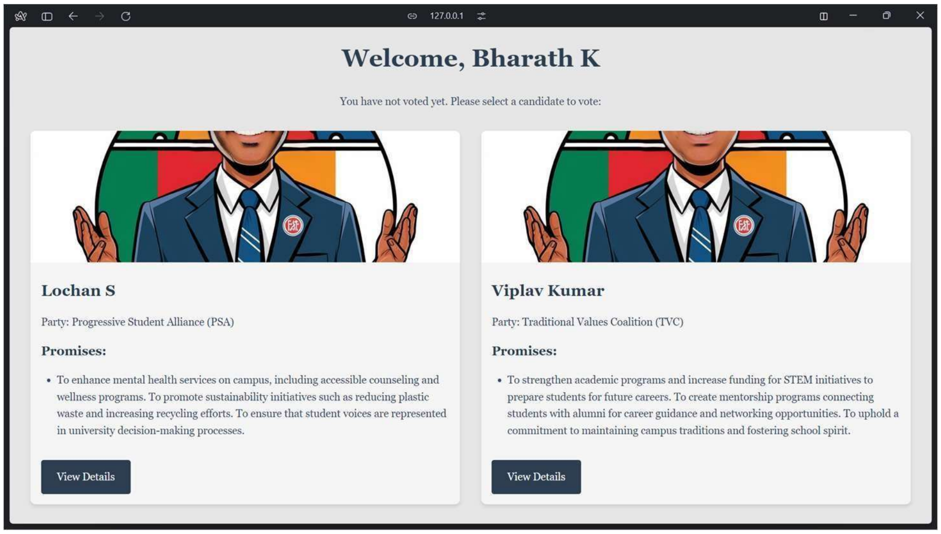
**Description:** The voter registration page allows new users to create an account by entering essential information such as name, voter ID, date of birth, and password. This page was designed with user experience in mind, incorporating clear labels and validation prompts to prevent errors. By focusing on usability, we aimed to minimize barriers for first-time voters and encourage their participation in the electoral process.

### 3. Voter Login Page (login.html)



**Description:** The login page facilitates access for registered voters, who must enter their voter ID and password to gain entry. Security features are emphasized here, with feedback provided for incorrect entries, which reinforces the importance of safeguarding personal information. This design choice was crucial in ensuring that users feel confident about the security of their data.

### 4. Candidate List Page (candidate_list.html)



## Welcome, Bharath K

You have not voted yet. Please select a candidate to vote:

**Lochan S**

Party: Progressive Student Alliance (PSA)

**Promises:**

- To enhance mental health services on campus, including accessible counseling and wellness programs. To promote sustainability initiatives such as reducing plastic waste and increasing recycling efforts. To ensure that student voices are represented in university decision-making processes.

**Viplav Kumar**

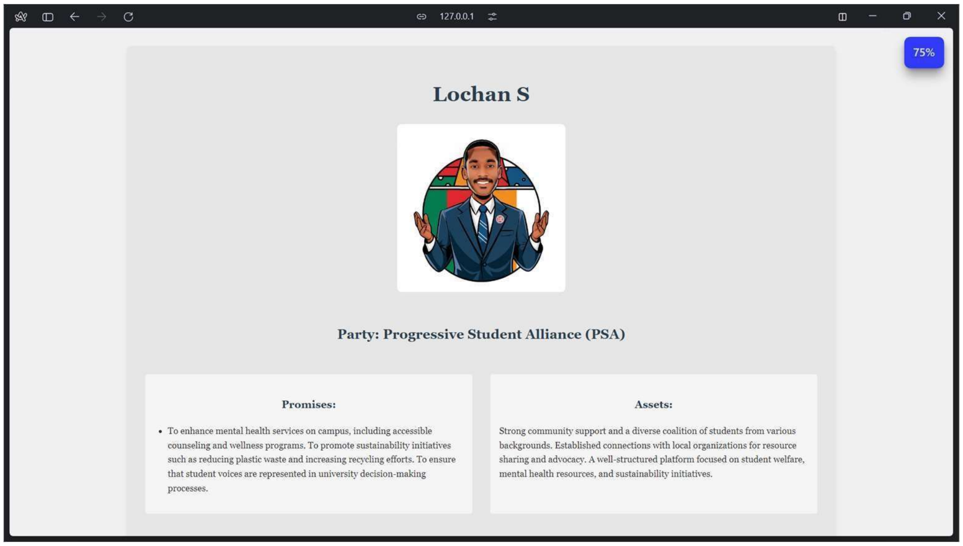
Party: Traditional Values Coalition (TVC)

**Promises:**

- To strengthen academic programs and increase funding for STEM initiatives to prepare students for future careers. To create mentorship programs connecting students with alumni for career guidance and networking opportunities. To uphold a commitment to maintaining campus traditions and fostering school spirit.

**Description:** Upon logging in, users are presented with a list of candidates. This page highlights each candidate's name, party affiliation, and a brief overview of their background and promises. Our goal was to create a visually appealing layout that allows voters to easily access critical information, thus enabling them to make informed choices in the voting process.

### 5. Voting Page (voting.html)



**Lochan S**

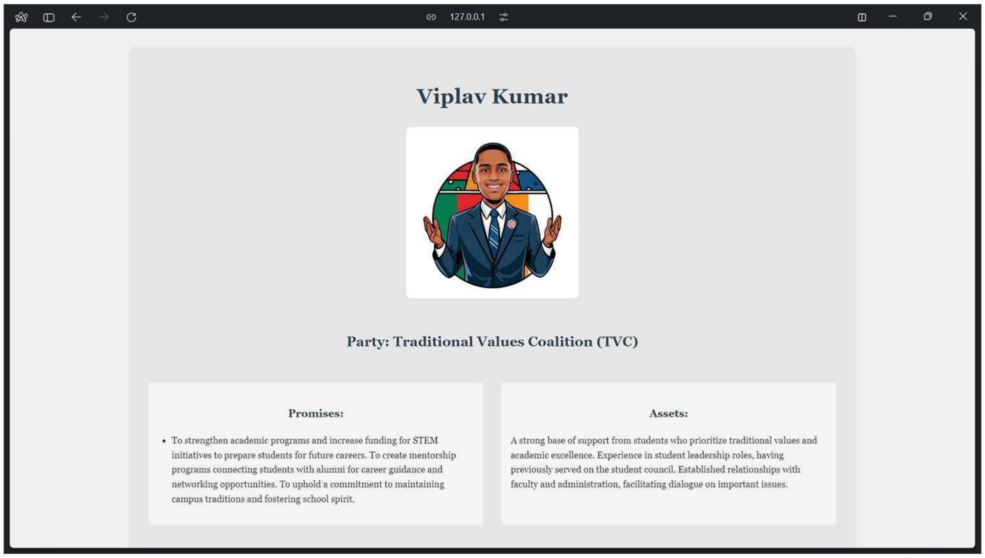
**Party: Progressive Student Alliance (PSA)**

**Promises:**

- To enhance mental health services on campus, including accessible counseling and wellness programs. To promote sustainability initiatives such as reducing plastic waste and increasing recycling efforts. To ensure that student voices are represented in university decision-making processes.

**Assets:**

Strong community support and a diverse coalition of students from various backgrounds. Established connections with local organizations for resource sharing and advocacy. A well-structured platform focused on student welfare, mental health resources, and sustainability initiatives.



127.0.0.1

Viplav Kumar

Party: Traditional Values Coalition (TVC)

Promises:

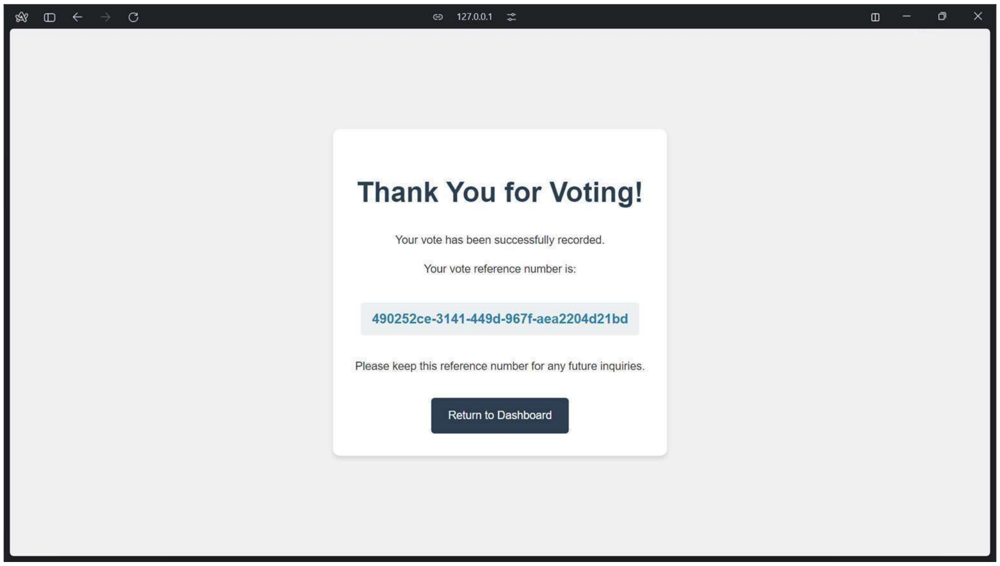
- To strengthen academic programs and increase funding for STEM initiatives to prepare students for future careers. To create mentorship programs connecting students with alumni for career guidance and networking opportunities. To uphold a commitment to maintaining campus traditions and fostering school spirit.

Assets:

A strong base of support from students who prioritize traditional values and academic excellence. Experience in student leadership roles, having previously served on the student council. Established relationships with faculty and administration, facilitating dialogue on important issues.

**Description:** The voting page provides a straightforward interface for users to select their preferred candidate and submit their vote. With clear instructions and an engaging design, this page ensures that voters understand how to cast their vote effectively. We aimed to create an atmosphere that encourages active participation in the democratic process, underscoring the importance of each individual's voice.

### 6. Vote Confirmation Page (vote_confirmation.html)



127.0.0.1

Thank You for Voting!

Your vote has been successfully recorded.

Your vote reference number is:

490252ce-3141-449d-967f-aea2204d21bd

Please keep this reference number for any future inquiries.

**Description:** After casting their vote, users are directed to the vote confirmation page, which confirms that their vote has been successfully recorded. This page also provides a unique reference number, enhancing transparency and allowing voters to verify their voting status. This reassurance was designed to build trust in the system and promote confidence in the integrity of the voting process.

The results of our **Online Voting System** project not only demonstrate the successful integration of multiple components but also reflect our collaborative effort as a team to create an effective platform for voter participation. Each screenshot highlights key functionalities that we designed with the user in mind, ensuring an engaging and intuitive experience. We believe that our project effectively meets its objectives, and we look forward to your feedback as we submit this documentation for evaluation.

# Conclusion

In conclusion, our **Online Voting System** project represents a significant achievement in applying theoretical concepts from our Database Management System course to a practical and functional application. Throughout this project, we have worked collaboratively as a team, leveraging our individual strengths to design and implement a user-friendly platform that facilitates secure and efficient voting.

By integrating an intuitive user interface with a robust backend, we ensured that each step of the voting process—from registration to casting a vote—was seamless and straightforward for users. The use of SQLite for our database allowed us to efficiently manage and store critical information related to voters, candidates, and votes, all while maintaining data integrity and security.

Our design emphasized the importance of usability, ensuring that users could easily navigate the system and access essential information. The feedback and confirmation messages embedded in the application serve to enhance user trust and engagement, reinforcing the significance of their participation in the electoral process.

This project has not only enhanced our technical skills in web development and database management but has also provided us with invaluable experience in teamwork, problem-solving, and project management. As we submit this documentation for evaluation, we hope to convey our commitment to learning and our understanding of how technology can be harnessed to empower democratic processes.

We appreciate the opportunity to work on this project and look forward to your feedback, which will help us further refine our skills and understanding in this field.

**To access our project, Scan the code:**