PROJECT REPORT

DEPARTMENT OF COMPUTER SCIENCE

SREE NARAYANA COLLEGE CHERTHALA

Affiliated to University of Kerala
NAAC Re-Accredited with A+ Grade

TrueVoice

ADARSH B	**32022131001**
AKSHAY PT	**32022131006**
GOURY SANTHOSH	**32022131020**
MUHAMMED MUHASIN K	**32022131028**

SUBMITTED IN PARTIAL FULFILMENT OF REQUIREMENT FOR THE AWARD OF BSC COMPUTER SCIENCE DEGREE OF UNIVERSITY OF KERALA
2025

SREE NARAYANA COLLEGE, CHERTHALA
Affiliated to University of Kerala
NAAC Re-Accredited with 'A+' Grade

BSC Computer Science
2022 – 2025

CERTIFICATE

This is to certify that the project work entitled "**TrueVoice**" is Bonafied report of the work done by **Mr. Adarsh B, Mr. Akshay PT, Ms. Goury Santhosh,** and **Mr. Muhammed Muhasin K** in partial fulfilment of the requirements for the award of the Bachelor's degree in B.Sc. Computer Science of the institution during the year 2022-2025.

Staff in charge Head of the Department Principal

Presented for the viva-voice examination conducted by the University of Kerala held on ...../...../2025 at Sree Narayana College, Cherthala & verified by

Date: Internal Examiner External Examiner

DECLARATION

We hereby declare that the project report entitled **"TrueVoice"** is a record of project work done under the guidance of Mrs. Aswathy K, Assistant Professor on Contract, and Mrs. Chinnu K Deepu, Assistant Professor on Contract, Department of Computer Science, Sree Narayana College,Cherthala.

We also declare that this report has not been submitted to any other University or Institute for the award of any fellowship or Degree or Diploma.

Place: Cherthala
Date:

Names: Adarsh B
Akshay PT
Goury Santhosh
Muhammed Muhasin K

ACKNOWLEDGMENT

This project marks our inaugural endeavour in developing a web application, and we are pleased to report that the resultant software aligns closely with our initial objectives. Throughout the project's duration, we received invaluable support and guidance from our peers, family members, and faculty advisors, to whom we extend our sincere gratitude. Their contributions have been instrumental in facilitating the successful completion of this project.

We are esteemed to thank, The Principal **Dr. T.P. BINDU**, Sree Narayana College,Cherthala for granting us permission to do the project.

We convey our sincere thanks to **Mr.ANSU A T (Head of the Department of Computer Science), Mrs. Aswathy K (Assistant Professor on Contract, Department of Computer Science), Mrs. Chinnu K Deepu K (Assistant Professor on Contract, Department of Computer Science)** for their valuable instructions and guidance to the project and their apt encouragement towards the successful completion of our project.

Adarsh B

Akshay PT

Goury Santhosh

Muhammed Muhasin K

ABSTRACT

The growing need for secure, transparent, and efficient election systems has led to the exploration of blockchain technology as a viable solution for addressing the challenges associated with traditional voting methods. This project aims to develop a Blockchain Based College Election Voting System tailored to the specific requirements of conducting elections in a college environment. By leveraging blockchain's decentralized and immutable nature, the proposed system ensures data integrity, enhances voter anonymity, and provides a transparent voting process.

The system is built using a three-layered architecture: frontend, backend, and blockchain network. The frontend provides a cross-platform, user-friendly interface for voter registration, candidate information display, and secure voting. The backend, powered by Flask, manages authentication, voter and candidate data, and API integration with the blockchain. A custom blockchain, implemented using Python, records votes as transactions, ensuring that each vote is immutable and verifiable.

The project is designed to address common challenges, such as vote duplication, tampering, and lack of trust in the election process. It also emphasizes scalability and usability, allowing the system to adapt to elections with varying numbers of participants. The use of blockchain technology fosters trust and transparency, while the intuitive user interface ensures accessibility for all voters. In conclusion, the Blockchain-Based College Election Voting System not only modernizes the election process but also sets a benchmark for adopting secure and transparent technologies in institutional decision-making. By implementing this system, colleges can conduct elections efficiently, with improved accuracy and fairness, thereby promoting confidence in the democratic process.

CONTENTS

LIST OF FIGURES

Gantt Chart

LIST OF TABLES

1. INTRODUCTION

1.1 EXISTING SYSTEM AND DISADVANTAGES

- **Ballot voting** : It is a traditional method of casting votes in an election, where voters mark their choices on a paper or electronic ballot. While it ensures anonymity and is widely used in democratic processes, it has several disadvantages. Paper-based ballot voting is prone to errors such as miscounting, ballot tampering, and invalid votes due to improper marking. It also involves high costs for printing, transportation, and storage. Additionally, manual counting can be time-consuming and susceptible to human bias or fraud. Electronic voting systems, while faster, raise concerns about security vulnerabilities, hacking, and lack of transparency. Moreover, in large-scale elections, logistical challenges such as long waiting times and accessibility issues can hinder voter participation.
- **Electronic Voting Machine (EVM)** : It is a device used to record and count votes digitally, replacing traditional paper ballots. It enhances efficiency by reducing manual errors and speeding up the counting process. However, EVMs have several disadvantages. One major concern is security, as they can be vulnerable to hacking, tampering, or software manipulation, potentially compromising election integrity. Unlike paper ballots, EVMs lack a verifiable physical trail, making it difficult to audit or recount votes in case of disputes. Technical failures, such as software glitches or hardware malfunctions, can lead to vote loss or misrepresentation.
- **Other online voting systems** : Internet and mobile voting offer convenience but have major drawbacks. They are vulnerable to hacking, fraud, and technical failures, risking election integrity. Voter authentication is challenging, increasing impersonation risks, while the lack of a verifiable paper trail makes audits difficult. The digital divide excludes those without internet access, and voter coercion is easier outside polling stations. Trust issues also arise due to transparency concerns. These risks make many countries prefer hybrid systems with paper-based verification for security.

1.2 PROPOSED SYSTEM AND ADVATAGES

- **TrueVoice** is blockchain based college election voting system.It ensures security, transparency, and efficiency by recording votes on a decentralized ledger, making them tamper-proof and verifiable. Each vote is encrypted, ensuring anonymity and privacy while preventing fraud, multiple voting, and unauthorized access through

cryptographic verification and smart contracts. Blockchain eliminates the risk of vote manipulation and allows real-time verification without compromising voter identities. It also automates vote counting, reducing human errors and speeding up result declaration. Additionally, students can vote from anywhere, improving accessibility and participation, while the system remains cost-effective by eliminating paper ballots and manual processes. This enhances trust and fairness in college elections, making the voting process more secure and reliable.

1.3 PROJECT PROFILE

Title	:	TrueVoice
Type	:	Blockchain based
Objective	:	Create a system to conduct college elections.
Duration	:	3 Months
Internal Guide	:	Mrs. Aswathy K
	:	Mrs. Chinnu K Deepu
Project Team	:	Adarsh B
		Akshay PT
		Goury Santhosh
		Muhammed Muhasin K

1.4 ORGANISATION PROFILE

Sree Narayana College, Cherthala is affiliated to University of Kerala and approved by NAAC and UGC. Sree Narayana College, Cherthala was established By Sree Narayana Trusts, Kollam. The aim of the college at assessing and accrediting institutions of higher learning with an objective of improving quality of education especially of the backward communities of the locality.

1.5 OVERVIEW

TrueVoice leverages decentralized technology to enhance the security, transparency, and efficiency of student elections. By recording votes on an immutable ledger, TrueVoice ensures tamper-proof and fraud-resistant voting, eliminating risks like vote manipulation and multiple voting. TrueVoice maintains voter anonymity and privacy through encryption while allowing real-time verification of results. Features like smart contracts and cryptographic authentication enhance reliability and prevent unauthorized access. Additionally, it offers convenience and accessibility, enabling students to vote remotely, while also being cost-

effective by reducing reliance on paper ballots and manual counting. TrueVoice fosters trust and fairness, making college elections more secure and efficient.

2. REQUIREMENT ENGINEERING

Requirement engineering provides the appropriate mechanism for understanding what the user needs, analysing type of services, assessing the feasibility, negotiating reasonable solutions, unambiguously validating the specification and managing the requirements as they are transformed into an operational system.

2.1 REQUIREMENT ELICITATION

Requirements elicitation of requirements gathering focuses on the objectives of the systems .

2.1.1 Feasibility Study

The main objective of this study is to determine whether the proposed system is feasible or not. System analysis is the most important phase in the lifecycle of the system development. The investigation points to questioning whether the project is feasible. After conducting a thorough feasibility study, we have determined that our project “TrueVoice” is feasible. The system analysis phase of the system development life cycle was crucial in identifying and evaluating various options to determine the best system that meets all requirements.. As a result, we have selected the best system for the job, and we are confident that it will meet all the necessary requirements.

2.1.1.1 Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. We have determined that ”TrueVoice” is operationally feasible. Through the scope definition and requirements analysis phases of system development, we have identified the problems and opportunities that our proposed system must address. Based on our evaluation, we are confident that our proposed system satisfies the identified requirements and effectively solves the identified problems while taking advantage of the identified opportunities. Therefore, we believe that our system will be operationally feasible.

2.1.1.2 Technical Feasibility

The technical feasibility of TrueVoice is supported by the availability of advanced blockchain and web technologies that can power its platform effectively. The development of a secure

and user-friendly web application, incorporating decentralized vote storage and a robust authentication system, is within the scope of current technological capabilities. Additionally, the integration of blockchain ensures transparency and security in the voting process, making vote tampering nearly impossible. TrueVoice can leverage established technology stacks and frameworks to provide a seamless user experience while ensuring the integrity and anonymity of votes. These factors make the project technically viable for implementation and scalability.

2.1.1.3 Economical Feasibility

The economic feasibility of TrueVoice is supported by the cost-effectiveness of web-based and blockchain technologies used for its implementation. Since TrueVoice is designed as a digital voting system, the primary costs involve web development, blockchain integration, and hosting services, which are manageable within an academic or institutional setting. The decentralized nature of blockchain eliminates the need for expensive physical infrastructure, reducing long-term operational costs. Additionally, open-source frameworks and cloud-based services can be leveraged to optimize expenses while ensuring security and scalability. Given these factors, the project is economically viable within its intended scope.

2.1.2 Requirement Definition

TrueVoice is a blockchain-based college election voting system designed to provide a secure, transparent, and tamper-proof digital voting platform. The system enables students to cast their votes online while ensuring the integrity of the election process through decentralized vote storage. TrueVoice aims to simplify and modernize college elections by replacing traditional paper-based voting with a streamlined, digital alternative that minimizes human intervention and enhances security. The system will feature an admin module for managing elections, a candidate module for registration, and a student module for voting.

2.1.2.1 Functional Requirement

Functional requirements of the system should clearly describe each of the functions that the system needs to perform along with the corresponding input and output datasets.

ADMIN

- Login
- Course Management
- Student Management
- Post Management
- Election Management

- View Candidates and verify
- View Verified Candidates
- View Campaigns and Remove
- View Complaints and send Reply
- View Result
- View Feedback
- Change Password

CANDIDATES

- Login
- View Profile
- View Students
- View all Candidates
- Manage Campaigns
- View Result
- Change Password

STUDENT

- Login
- View Elections
- View Posts
- Apply as Candidates
- View all Candidates
- View Campaigns
- Caste Voting in Blockchain
- Send Complaint and Feedback
- View Complaint and Reply
- View Result
- Change Password

2.1.2.2 Non-Functional Requirement

Non-functional requirements describe how the software system should behave, i.e., performance, security, reliability, and usability requirements.

The web application should meet the following non-functional requirements:

1. Usability: The system must offer an intuitive and user-friendly interface that simplifies navigation and interaction. This includes well-organized layouts, clear instructions, and consistent design patterns that cater to both novice and advanced users, ensuring that every feature is easily accessible and straightforward to use.

2. Reliability: The platform should maintain high availability and operational stability, even during periods of peak usage. It must incorporate robust error handling, automatic recovery procedures, and redundant systems to minimize downtime and ensure that users can depend on the service at all times.

3. Security: Strong security measures are essential to protect user data and ensure privacy. The application must implement advanced authentication protocols, encryption of sensitive data, regular security audits, and proactive monitoring to prevent unauthorized access and mitigate potential threats.

4. Scalability: The architecture should be designed to handle growth in user numbers and data volume without performance degradation. This includes the ability to scale both horizontally and vertically, allowing the platform to expand its resources and capabilities in response to increasing demand and additional feature requirements.

5. Maintainability: Code quality, clear documentation, and modular design practices must be prioritized to facilitate future updates and modifications. This ensures that the platform can be easily maintained, debugged, and enhanced over time, reducing long-term costs and effort required for continuous improvement.

6. Performance: The system must deliver fast response times and efficient processing, particularly for real-time features like code execution, debugging, and data updates. Optimized algorithms, resource management, and performance monitoring tools should be in place to ensure a smooth and responsive user experience.

7. Accessibility: Ensure the platform is accessible to users with diverse abilities, following best practices and standards for inclusive design.

8. Data Integrity: The platform must ensure data accuracy and reliability through validation, error-checking, and backups.

3. DEVELOPMENT ENVIRONMENT

3.1 HARDWARE REQUIREMENTS

Input Devices	:	Mouse, Keyboard
Output Devices	:	Monitor
Processor	:	Intel core i3 or above
Memory	:	4 Gb Ram(Minimum)

3.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows 8/10 for Better Performance
Front End	:	HTML, CSS-Bootstrap, JavaScript
Back End	:	Python(Flask), MySQL
Blockchain	:	Etherium, Ganache
Browser	:	Internet Explorer/Google Chrome/Brave
Web Server	:	Apache, MySQL
Text Editor	:	Visual Studio Code

3.2.1 About Windows

Microsoft Windows (or simply Windows) is a met family of graphical operating systems developed, marketed, and sold by Microsoft. It consists of several families of operating systems, each of which cater to a certain sector of the computing industry.90% market share, overtaking Mac OS, Microsoft introduced an operating environment named Windows on November 20, 1985, as a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces (GUIs). soft Windows came to dominate the world's personal computer market with over. As of July 2015, the most recent version of Windows for personal computers, tablets and smartphones is Windows 10. The most recent versions for server computers and embedded are respectively Windows Server 2012 R2 and Windows Embedded 8. A specialised version of Windows runs on the Xbox One game console. All Windows versions from Windows NT 3 have been based on a file system permission system referred to as AGLP (Accounts, Global, Local, Permissions) AGDLP which in essence where file permissions are applied to the file/folder in the form of a 'local

group' which then has other 'global groups' as members. These global groups then hold other groups or users depending on different Windows versions used.

3.2.2 MySQL

MySQL is an open-source relational database management system (RDBMS) that is widely used for storing and managing structured data. It allows you to create, read, update, and delete data in a structured manner through SQL (Structured Query Language) queries. MySQL is known for its performance, reliability, and scalability, making it a popular choice for web applications, content management systems, and various data-driven software applications. It supports multiple programming languages and platforms and can be used for both small-scale and large-scale data storage and retrieval.

What is MySQL?

- *Open Source: MySQL is open-source and free to use, which makes it cost-effective and widely adopted.*
- *High Performance: It's known for its fast data processing and efficient data retrieval, making it suitable for high traffic websites and applications.*
- *Scalability: MySQL can handle both small-scale and large-scale data needs. It supports various storage engines to optimize performance and scalability.*
- *Reliability: MySQL is known for its data integrity and reliability. It's often used in mission-critical applications.*
- *Security: It provides robust security features to protect the database from unauthorized access and attacks.*

MySQL is widely used in web applications, content management systems, e-commerce platforms, and a wide range of data-driven software applications. It's a versatile and powerful database system with a large and active user community.

What is the advantage of using MySQL?

MySQL offers several advantages : it's open source and free, highly scalable and boasts strong performance and reliability. With features like replication and transactions, MySQL is a trusted choice for a wide range of application and industries.

Why use MySQL?

Using MySQL for TrueVoice, a blockchain-based college election voting system, offers several benefits. MySQL's open-source nature and cost-effectiveness help keep operational expenses low. Its scalability and performance ensure that the system can handle increasing numbers of

voters and election data efficiently. MySQL's reliability and robust security features protect voter information and election records, ensuring data integrity. Its support for ACID compliance makes it ideal for handling sensitive transactions such as vote storage and retrieval. Additionally, MySQL's efficient data management capabilities, ease of integration with web applications, and widespread industry adoption make it a suitable database choice for a secure and transparent voting system like TrueVoice.

3.2.3 Ethereum

Ethereum is a decentralized, open-source blockchain platform that enables the execution of smart contracts, making it ideal for secure and transparent voting systems like TrueVoice. Unlike traditional databases, Ethereum operates on a distributed network of nodes, ensuring that election data is immutable and protected from tampering. Smart contracts, written in Solidity, govern the voting process by automatically recording and verifying votes, eliminating the need for intermediaries. This decentralized approach enhances trust and transparency, ensuring that votes are securely stored on the blockchain and cannot be altered or manipulated. Additionally, Ethereum's robust security features and cryptographic mechanisms provide a high level of protection against fraud and unauthorized access, making it a reliable choice for digital voting systems.

3.2.4 Ganache

Ganache is a personal Ethereum blockchain used for testing and development, providing a controlled environment where developers can deploy and debug smart contracts before launching them on the live Ethereum network. In TrueVoice, Ganache is essential for simulating blockchain transactions, allowing developers to test the voting process, identify issues, and fine-tune smart contracts without the risks or costs associated with deploying on the mainnet. It offers an easy-to-use graphical interface and a command-line tool, enabling real-time monitoring of transactions, gas usage, and contract execution. By using Ganache, TrueVoice ensures that its blockchain-based voting system is thoroughly tested and optimized for security, accuracy, and efficiency before being deployed in a real-world election scenario.

3.2.5 Python(Flask)

Python is used for the backend development of TrueVoice, handling essential tasks required for a secure and efficient voting system. Python's versatility, extensive libraries, and frameworks make it an ideal choice for building web applications. TrueVoice utilizes Flask, a lightweight and flexible web framework, to create the core structure of the application, route user requests, and manage database interactions.

The backend is responsible for handling user authentication, election management, and vote processing while ensuring data integrity. It manages different user roles, including Admins for overseeing elections, Candidates for participating in the election, and Students for casting votes. Python facilitates secure vote storage by interacting with MySQL, ensuring the voting

process remains transparent and tamper-proof. Additionally, Flask's ability to integrate with blockchain technology enhances the security and reliability of the system.

Overall, Python's flexibility, security features, and robust ecosystem contribute to TrueVoice's efficient and seamless operation, ensuring a fair and trustworthy digital election process.

3.2.6 Visual Studio Code

Visual Studio Code (VS Code) is a versatile and powerful code editor used for the development of TrueVoice, primarily for building and maintaining the backend components of the application. Backend development in VS Code involves creating the logic, database interactions, and API endpoints that power TrueVoice's functionality. Developers use this code editor to write Flask-based server-side scripts, set up MySQL databases to store election-related data, and create RESTful APIs for communication between the frontend and backend.

VS Code provides a rich ecosystem of extensions and plugins that enhance productivity, making it easier to debug code, manage version control with Git, and integrate seamlessly with Flask and MySQL. Its lightweight nature, strong performance, and support for multiple programming languages make it an ideal choice for developing a robust, secure, and scalable backend system for TrueVoice, ensuring smooth and efficient election management.

Features

1. **Ease of Learning**: Python, used in TrueVoice, has a simple and readable syntax, making it accessible for developers of all skill levels.
2. **Wide Range of Libraries**: Python offers a rich ecosystem of libraries and frameworks, such as Flask, for web development and data handling.
3. **Cross-Platform Compatibility**: Python runs on multiple operating systems, ensuring the web application works consistently across different platforms.
4. **Scalability**: Flask and MySQL provide a scalable foundation that allows TrueVoice to handle growing numbers of voters and elections.
5. **Web Development Frameworks**: Flask, a lightweight yet powerful framework, expedites the development of secure and efficient web applications.
6. **Data Processing**: Python enables efficient handling of election data, including candidate details, voter registrations, and vote storage.
7. **Community Support**: Python has a large and active community that provides extensive resources and support for developers working on TrueVoice.
8. **Security**: Flask and MySQL provide built-in security features to protect sensitive election data from unauthorized access and tampering.
9. **Integration Capabilities**: Python allows seamless integration with blockchain technologies and other APIs for enhanced security and functionality.
10. **Efficiency and Performance**: Flask's lightweight nature ensures a responsive and efficient election system with minimal latency.

3.2.7 HTML

HTML (Hypertext Markup Language) is crucial for the front-end development of the TrueVoice voting system. It serves as the backbone of the web application, structuring and presenting content to users. HTML defines the layout, text, and media elements, including election details, candidate profiles, and user interfaces for voters, candidates, and administrators. It provides the foundation for user interaction, allowing students to view election-related information, cast their votes, and track election progress. Through HTML, TrueVoice ensures that information is well-organized, accessible, and easy to navigate, offering a seamless and transparent user experience for all participants in the voting process.

3.2.8 Cascading Style Sheets (CSS)

CSS (Cascading Style Sheets) is essential for the front-end design of TrueVoice, defining the visual presentation and layout of the web application. CSS is used to control colors, fonts, spacing, and responsiveness, ensuring a uniform and professional appearance across different devices. In TrueVoice, CSS ensures that the voting system is visually appealing, easy to navigate, and accessible to students, candidates, and administrators. A well-designed interface improves user engagement, making the voting process intuitive and transparent while enhancing overall usability and accessibility.

3.2.9 JavaScript

JavaScript (JS) plays a vital role in the front-end development of TrueVoice, adding interactivity and dynamic functionalities. It enhances the user experience by enabling real-time features such as live election updates, form validations, and interactive candidate profiles. JavaScript ensures smooth navigation and responsiveness, making the voting process seamless. Additionally, Bootstrap, a front-end framework, is used to ensure a responsive design, pre-built components, and a structured layout that adapts to different screen sizes. Together, JavaScript and Bootstrap enable TrueVoice to deliver a user-friendly, efficient, and visually engaging voting experience for students, candidates, and administrators.

4. SOFTWARE DESIGN

4.1 SYSTEM DESIGN

System design is the process of defining the architecture, components, interface and data for a system to satisfy specified requirements. Object-oriented analysis and design methods are the most widely used methods for computer system design. It translates the system requirement into ways of making them operational. The design phase focuses on the detailed implementation of the system recommended in the feasibility study. Design goes through logical and physical stages of development. The characters of well-defined system are:

1. Security
2. Practicality
3. Efficiency
4. Acceptability
5. Flexibility
6. Economy
7. Reliability
8. Simplicity

First, it addresses functional design specifying how the system will perform the required functions, the algorithms involved, and the data flows. This is essential to ensure that the system aligns with the expected functionality Second, it delves into user interface design where the look, feel, and usability of the system are considered. Designing an intuitive, user-friendly interface is paramount for user satisfaction and efficiency.

Third, security design is a vital facet, focusing on safeguarding the system against threats and vulnerabilities. This entails defining access controls, encryption measures, and authentication mechanisms. Fourth, performance design seeks to optimize the system's speed and resource usage. This involves selecting appropriate data structures and algorithms and planning for scalability to accommodate growing demands. Fifth, error handling and recovery design addresses how the system should handle unexpected errors, ensuring robustness and minimizing data loss. Lastly, maintenance and documentation design includes planning for long-term support, providing comprehensive technical documentation, and guidelines for future enhancements or modifications.

A well-defined system not only satisfies requirements but also excels in terms of functionality, user experience, security, efficiency, and adaptability being thoroughly documented and prepared for ongoing maintenance.

4.2 DATA FLOW DIAGRAM (DFD)

A DFD is a graphical representation of the flow of data through a system, showing the inputs, processing, storage, and outputs.

LEVEL 0 : CONTEXT DIAGRAM

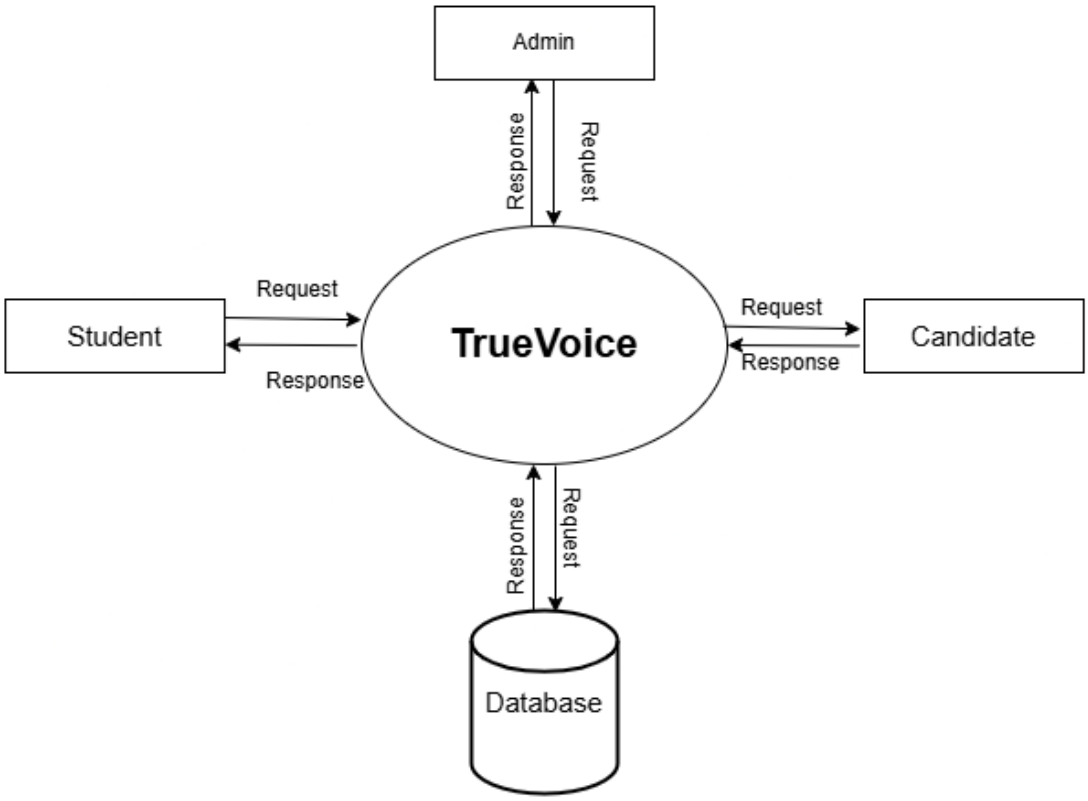


Figure 1: Level 0 DFD

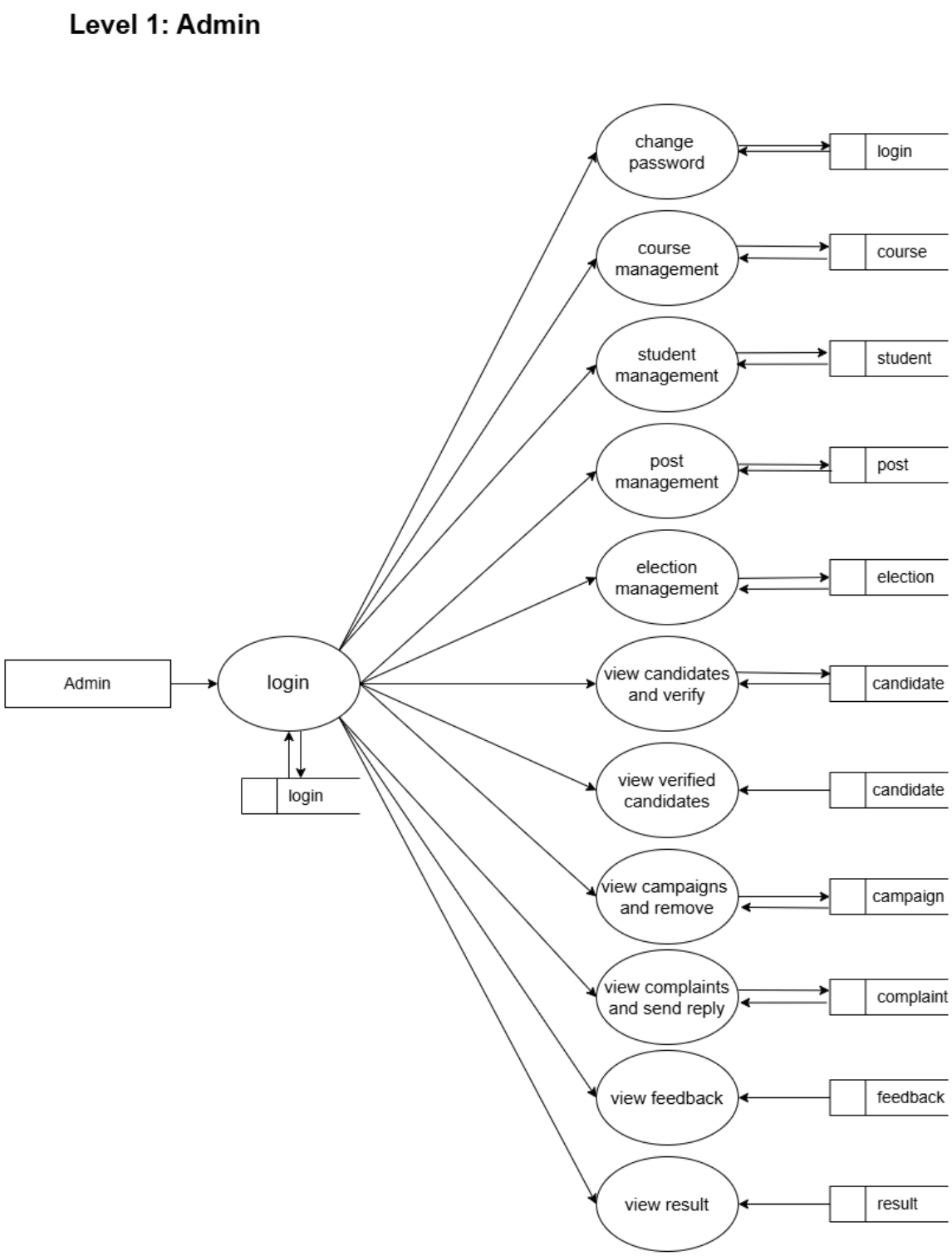


Figure 2: Level 1 DFD Admin

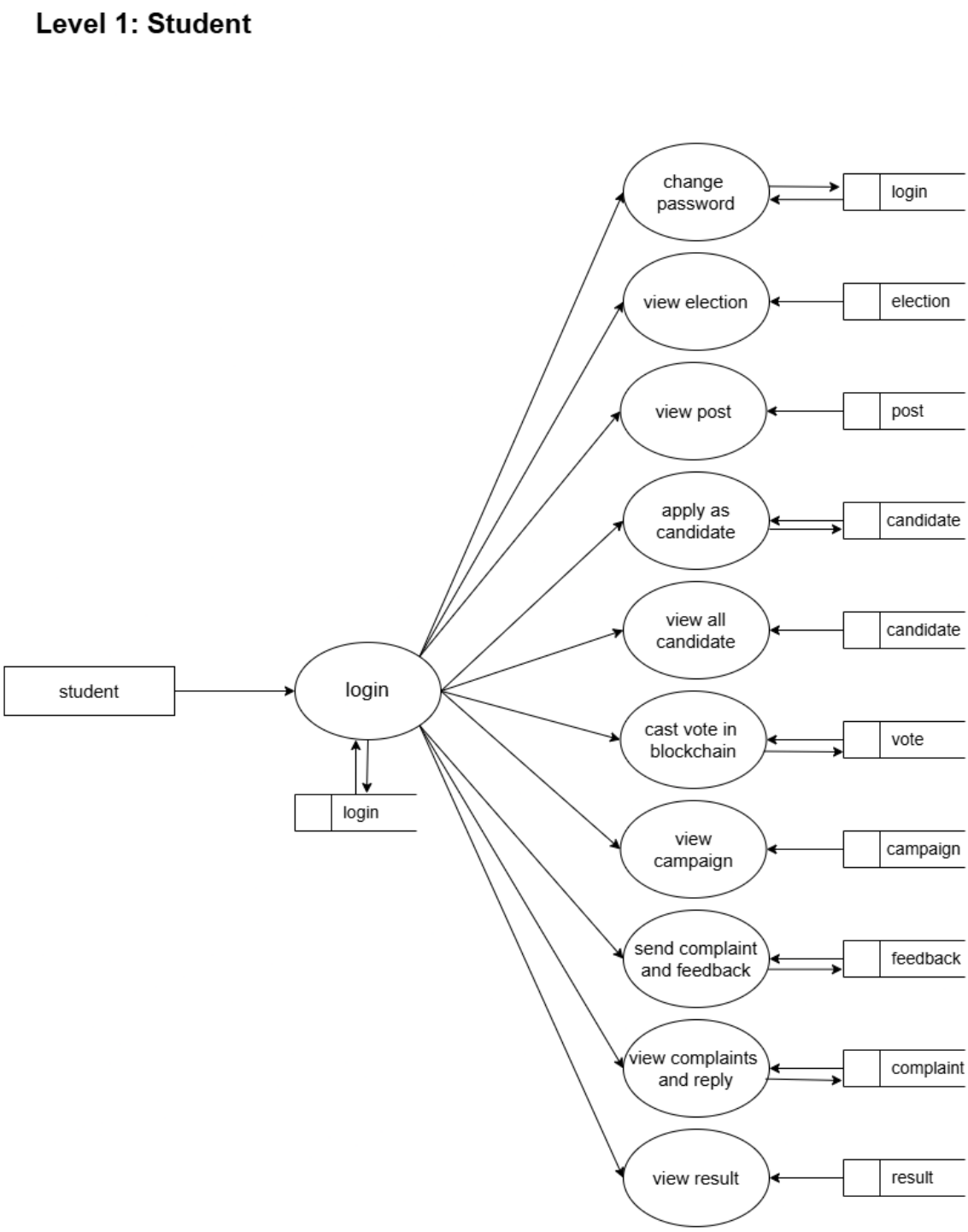


Figure 3: Level 1 DFD Student

Level 1: Candidate

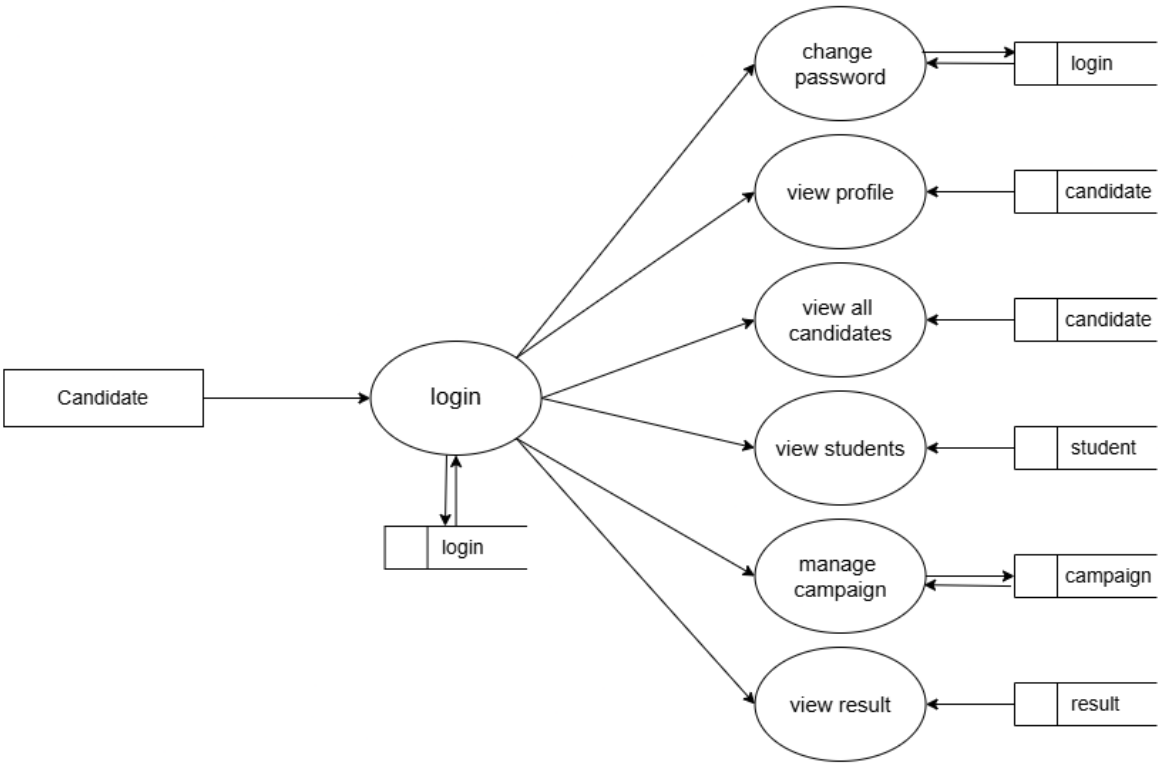


Figure 4: Level 1 DFD Candidate

4.3 ENTITY RELATIONSHIP DIAGRAM (E-R DIGRAM)

An Entity-Relationship (ER) diagram is a visual representation of the structure and relationships in a database. It's a tool used in database design to graphically illustrate the relationships between entities (tables) and their attributes (columns).

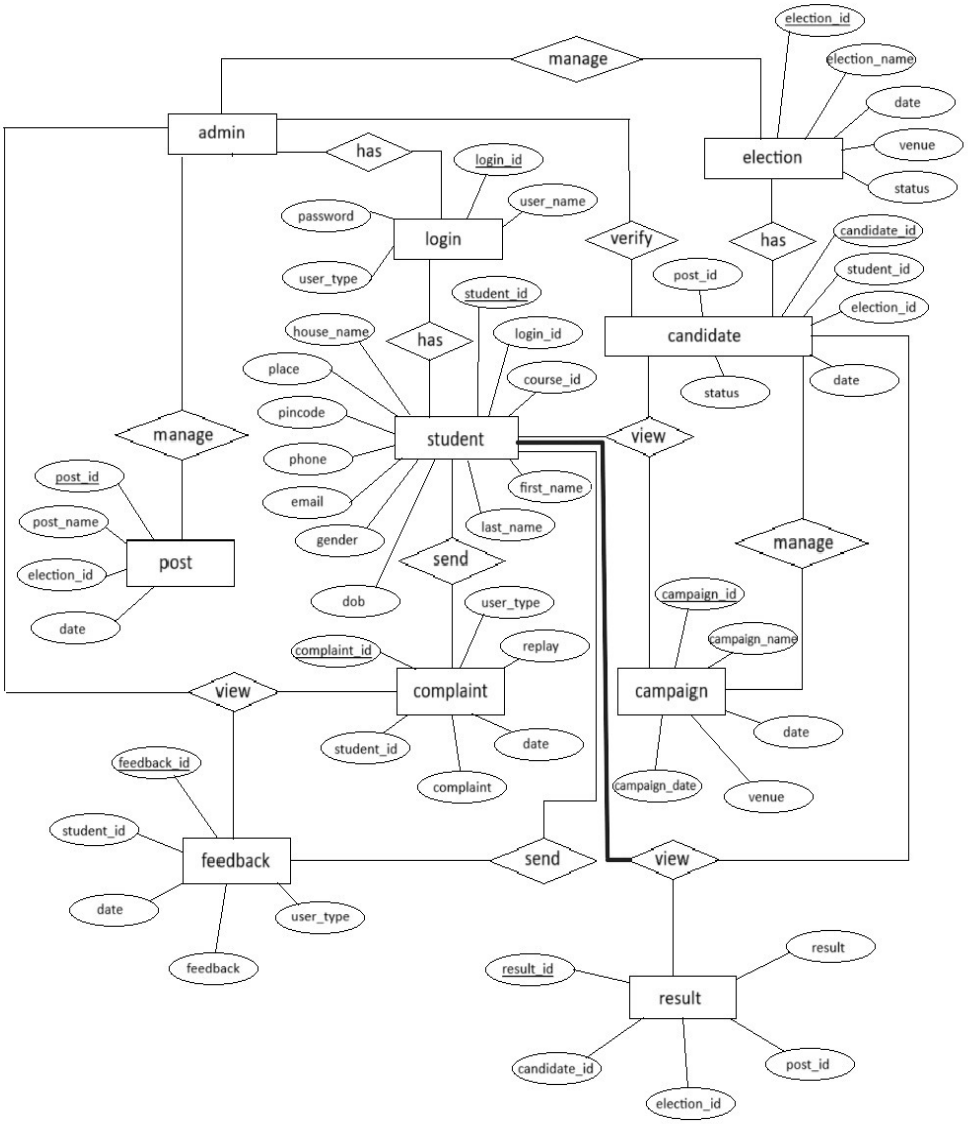
ER diagrams typically consist of:

1. Entities: Represented by rectangles, entities are the tables in the database.

2. Attributes: Represented by columns, attributes are the individual pieces of data stored in each entity.

3. Relationships: Represented by lines, relationships show how entities interact with each other.

Figure 4: E-R Diagram



4.4 USE-CASE DIGRAM

A use case diagram is a way of visualizing the interactions and relationships between the users and the system. It shows the different types of users, the use cases they can perform, and how they are connected. A use case diagram can help to understand the functional requirements of a system, and to design a system from the user's perspective.

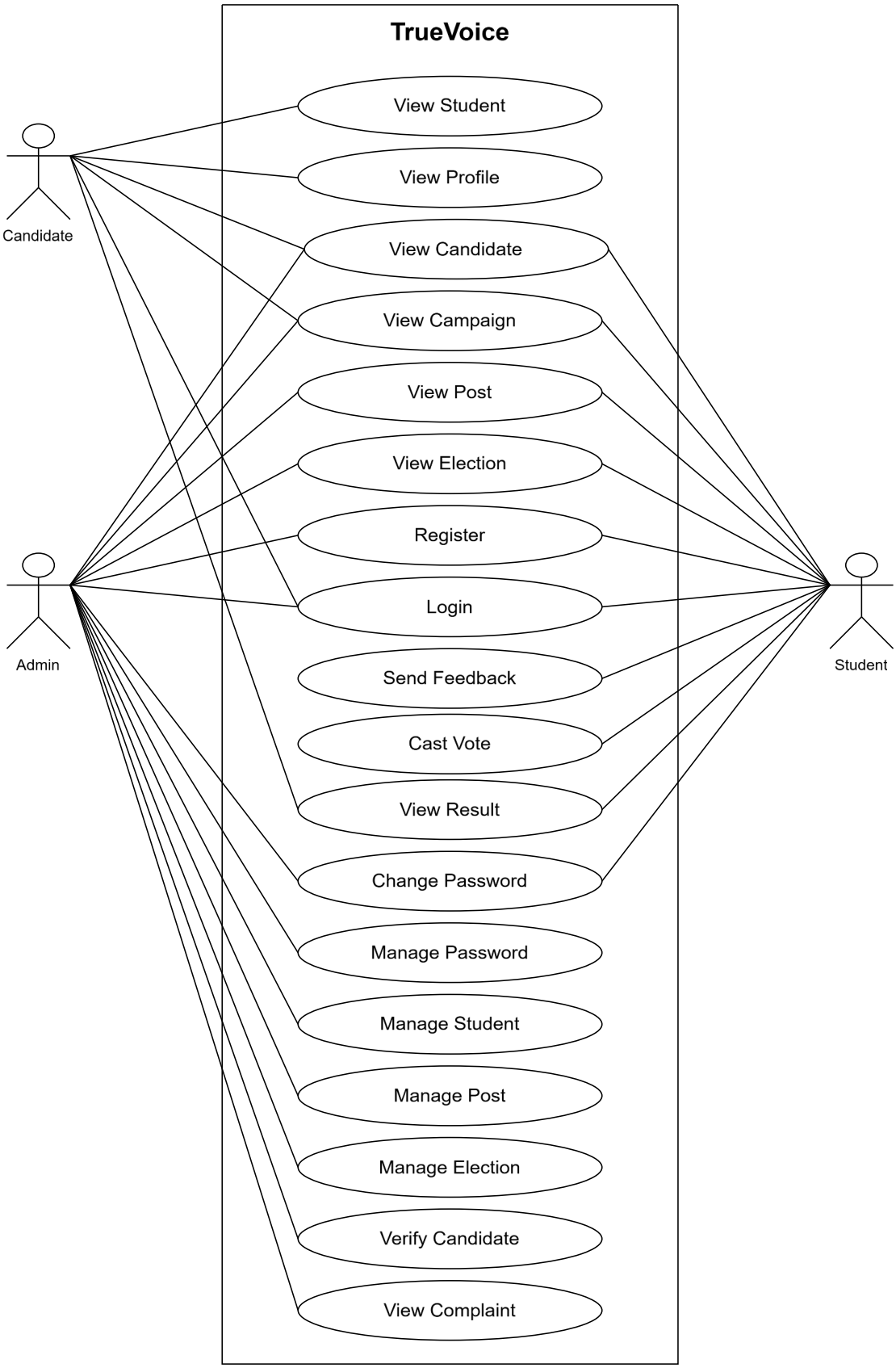


Figure 5: Use-Case Diagram

4.5 ACTIVITY DIAGRAM

An activity diagram is a way of visualizing the dynamic behaviour of a system or a process. It shows the flow of control and data from one activity to another, as well as the choices, iterations, and concurrency that may occur. An activity diagram can help you understand how a system works, what are the inputs and outputs, and what are the possible outcomes.

ACTIVITY DIAGRAM: ADMIN

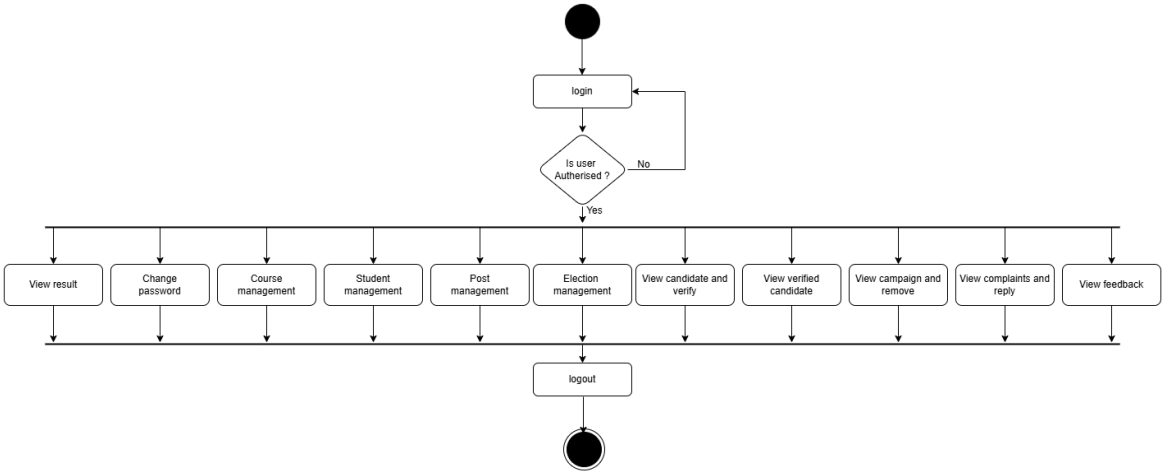


Figure 6: Activity Diagram-Admin

ACTIVITY DIAGRAM: STUDENT

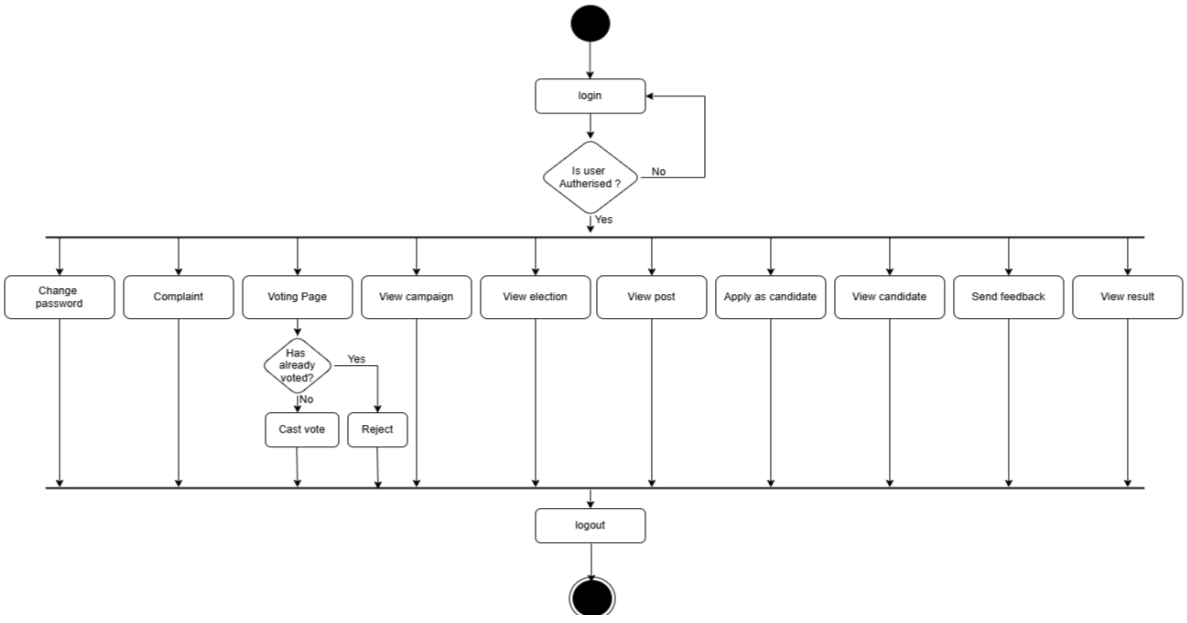


Figure 7: Activity Diagram-Student

ACTIVITY DIAGRAM: CANDIDATE

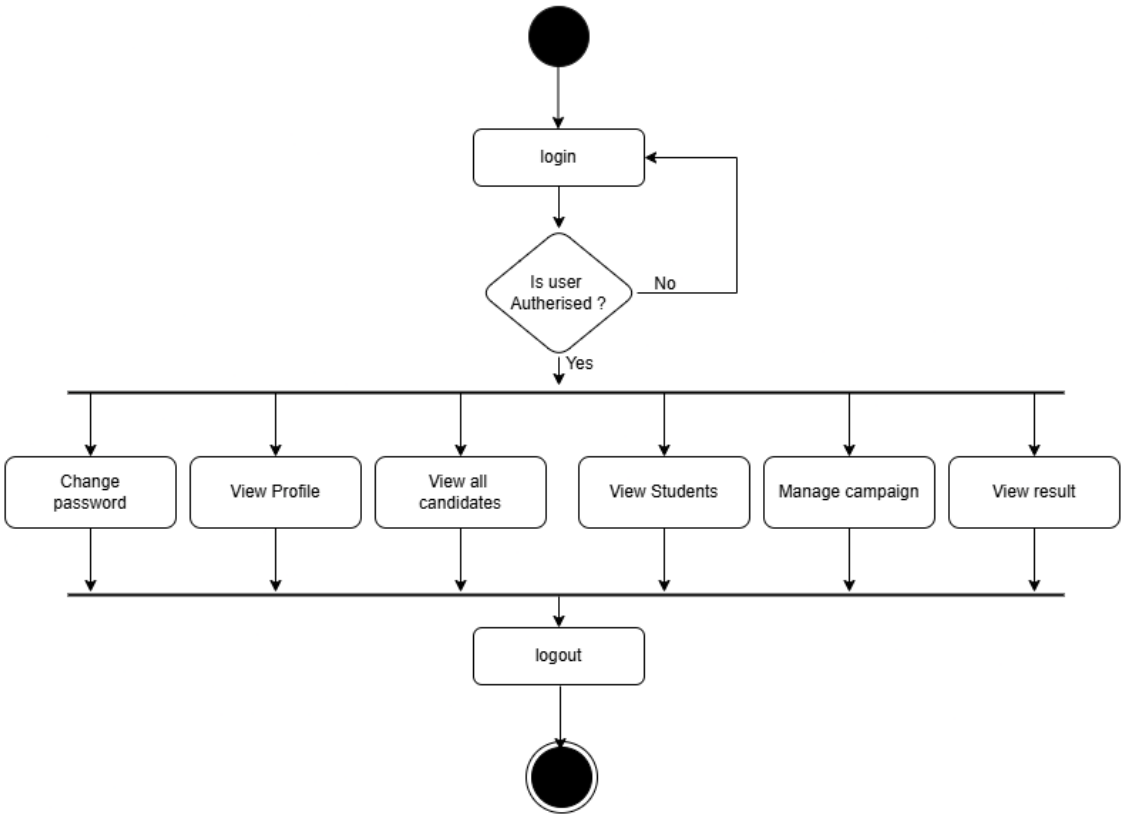


Figure 8: Activity Digram-Candidate

4.6 SEQUENCE DIAGRAM

A sequence diagram is a UML diagram that visualizes interactions between system components over time, emphasizing the order of messages exchanged. It uses lifelines (vertical lines representing objects or actors), activation bars (showing active processing periods), and messages (arrows for communication like method calls or data transfers). Messages can be synchronous (solid arrows, waiting for a response) or asynchronous (dashed, non-blocking). These diagrams simplify workflows, such as user authentication or API requests, by mapping step-by-step interactions. They bridge technical specifications and implementation, helping validate system design and clarify dependencies for developers and stakeholders.

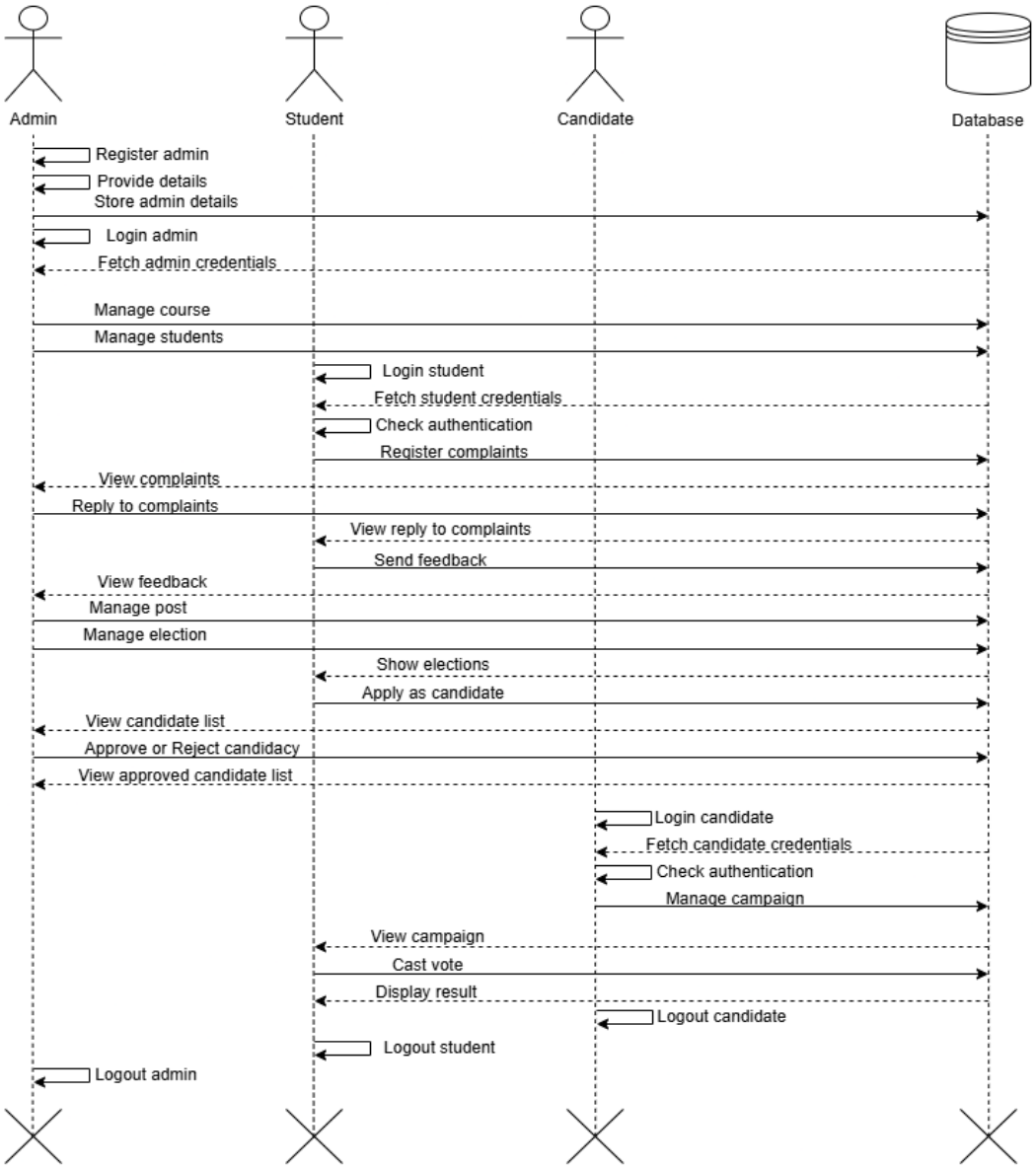


Figure 9: Sequence Diagram

4.7 PROGRAM LIST

S.L. NO.	FIELD	DESCRIPTION
1	**main.py**	**Runnable executable file**
2	**admin.py**	**Responsive administrative file**
2.1	index	Administrative home view
2.2	ad_change_password	Administrative change password view
2.3	manage_course	Administrative manage course view
2.4	manage_student	Administrative manage student view
2.5	manage_post	Administrative manage post view
2.6	manage_election	Administrative manage election view
2.7	view_candidates	Administrative candidates view
2.8	view_complaints	Administrative complaints view
2.9	sent_reply	Administrative sent reply to complaints view
2.10	view_campaigns	Administrative campaign view
2.11	a_view_election	Administrative election view
2.12	a_view_post	Administrative post view
2.13	view_result	Administrative election result view
3	**students.py**	**Responsive students file**
3.1	generate_otps	Function to generate OTP
3.2	a_view_all_candidates	Student’s candidate view functionality
3.3	sotp	Student’s OTP verification view functionality
3.4	index	Student’s home view functionality
3.5	st_change_password	Student’s change password functionality

3.6	s_view_election	Student’s election view functionality
3.7	s_view_post	Student’s post view functionality
3.8	st_view_campaigns	Student’s campaign view functionality
3.9	apply_as_candidate	Student’s apply for candidate functionality
3.10	feedback	Student’s feedback functionality
3.11	cd_sent_complaint	Student’s send complaint functionality
3.12	st_view_result	Student’s result view functionality
4	**candidate.py**	**Responsive candidate file**
4.1	generate_otps	Candidate OTP generation
4.2	view_all_candidates	Candidate view candidates functionality
4.3	verify_otp	Candidate OTP verification
4.4	candidate_home	Candidate home page
4.5	cd_change_password	Candidate password change functionality
4.6	view_profile	Candidate view profile
4.7	manage_student	Candidate student management view
4.8	manage_campaigns	Candidate campaign management view
4.9	cd_view_election	Candidate election view
4.10	cd_view_post	Candidate post view
4.11	cd_feedback	Candidate feedback functionality
4.12	cd_sent_complaints	Candidate send complaints
4.13	cd_view_result	Candidate result view
5	**database.py**	**MySQL Database query file**
5.1	select	MySQL Database select query
5.2	delete	MySQL Database delete query

5.3	update	MySQL Database update query
5.4	insert	MySQL Database update query
6	**public.py**	**Responsive public file**
6.1	index	Public index view
6.2	login	Public login view
7	**voting.sql**	**Database schema file**

Table 1: Program List

4.8 DATABASE DESIGN

Database design involves creating a comprehensive data model that outlines the logical and physical structure of a database. This process encompasses making key design decisions and defining storage parameters, which are then used to generate a data definition language. This language is used to bring the design to life and create a functional database.

TABLE NAME : login

PRIMARY KEY : login_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
login_id	INT	Primary Key	Login ID
user_name	VARCHAR(100)	NOT NULL	User Name
password	VARCHAR(15)	NOT NULL	Password
user_type	VARCHAR(20)	NOT NULL	User Type

Table 2: Login

TABLE NAME : course

PRIMARY KEY : course_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
course_id	INT	Primary Key	Course ID
course_name	VARCHAR(100)	NOT NULL	Course Name

Table 3: Course

TABLE NAME : election

PRIMARY KEY : election_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
election_id	INT	Primary Key	Election ID
election_name	VARCHAR(20)	NOT NULL	Election Name
date	VARCHAR(20)	NOT NULL	Date
venue	VARCHAR(100)	NOT NULL	Venue
status	VARCHAR(50)	NOT NULL	Status

Table 4: Election

TABLE NAME : student

PRIMARY KEY : student_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
student_id	INT	Primary Key	Student ID
login_id	INT	Foreign Key, NOT NULL	Login ID
course_id	INT	Foreign Key, NOT NULL	Course ID
first_name	VARCHAR(50)	NOT NULL	First Name
last_name	VARCHAR(50)	NOT NULL	Last Name
house_name	VARCHAR(50)	NOT NULL	House Name
place	VARCHAR(50)	NOT NULL	Place
pincode	INT	NOT NULL	Pincode
phone	INT	NOT NULL	phone
email	VARCHAR(100)	NOT NULL	Email
gender	VARCHAR(100)	NOT NULL	Gender
DoB	VARCHAR(50)	NOT NULL	Date of Birth

Table 5: Student

TABLE NAME : candidate

PRIMARY KEY : candidate_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
candidate_id	INT	Primary Key	Candidate ID
student_id	INT	Foreign Key, NOT NULL	Student ID
election_id	INT	Foreign Key, NOT NULL	Election ID
date	VARCHAR(20)	NOT NULL	Date
status	VARCHAR(20)	NOT NULL	Status
post_id	INT	Foreign Key, NOT NULL	Post ID

Table 6: Candidate

TABLE NAME : post

PRIMARY KEY : post_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
post_id	INT	Primary Key	Post ID
post_name	VARCHAR(50)	NOT NULL	Post Name
election_id	INT	Foreign Key, NOT NULL	Election ID
date	VARCHAR(20)	NOT NULL	Date

Table 7: Post

TABLE NAME : feedback

PRIMARY KEY : feedabck_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
feedback_id	INT	Primary Key	Feedback ID
student_id	INT	Foreign Key, NOT NULL	Student ID
date	VARCHAR(20)	NOT NULL	Date
feedback	VARCHAR(255)	NOT NULL	Feedback
user_type	VARCHAR(20)	Foreign Key, NOT NULL	User Type

Table 8: Feedback

TABLE NAME : result

PRIMARY KEY : result_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
result_id	INT	Primary Key	Result ID
candidate_id	INT	Foreign Key, NOT NULL	Candidate ID
election_id	INT	Foreign Key, NOT NULL	Election ID
post_id	INT	Foreign Key, NOT NULL	Post ID
result	VARCHAR(20)	NOT NULL	Result

Table 9: Result

TABLE NAME : campaign

PRIMARY KEY : campaign_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
campaign_id	INT	Primary Key	Campaign ID
campaign_name	VARCHAR(200)	NOT NULL	Campaign Name
date	VARCHAR(20)	NOT NULL	Date
venue	VARCHAR(200)	NOT NULL	Venue
campaign_date	VARCHAR(20)	NOT NULL	Campaign Date

Table 10: Campaign

TABLE NAME : vote

PRIMARY KEY : vote_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
vote_id	INT	Primary Key	Vote ID
student_id	INT	Foreign Key, NOT NULL	Student ID
election_id	INT	Foreign Key, NOT NULL	Election ID
post_id	INT	Foreign Key, NOT NULL	Post ID
status	VARCHAR(20)	NOT NULL	Status

Table 11: vote

TABLE NAME : complaint

PRIMARY KEY : complaint_id

NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
complaint_id	INT	Primary Key	Complaint ID
student_id	INT	Foreign Key, NOT NULL	Student ID
complaint	VARCHAR(255)	NOT NULL	Complaint
date	VARCHAR(20)	NOT NULL	Date
reply	VARCHAR(255)	NOT NULL	Reply
user_type	VARCHAR(20)	Foreign Key, NOT NULL	User Type

Table 12: Complaint

4.9 INPUT DESIGN

TrueVoice's input system is designed to ensure secure, seamless, and efficient voting operations. Users interact with the platform primarily through authentication and vote submission. Authentication requires users to enter their registered credentials (such as email and password), with robust validation mechanisms ensuring secure access. The system also records administrator inputs for election setup, including candidate registration, voter list management, and election scheduling.

The core interaction happens through the voting interface, where authenticated students can securely cast their votes. Each vote is recorded as an immutable transaction on the blockchain, ensuring transparency and preventing tampering. Users input their choice by selecting a candidate, and once confirmed, the vote is cryptographically signed and stored. Administrators also input data for monitoring election progress, viewing candidate details, and managing the election lifecycle.

The blockchain component of TrueVoice processes input data in real time. When a user submits a vote, the system validates the transaction, ensures eligibility, and securely records it on the blockchain. The system also tracks election results, counting votes as they are stored in the decentralized ledger. All user interactions, including login attempts, successful votes, and election-related actions, are logged for security and auditing purposes, ensuring a transparent and verifiable electoral process.

4.10 OUTPUT DESIGN

When users interact with TrueVoice, they receive clear and secure feedback on their actions. Upon successful registration, voters and candidates gain access to their respective dashboards. Students see election details, including voting status (whether they have voted or not) and candidate profiles. Admins can monitor ongoing elections, track voter turnout, and view candidate registrations.

Once a user casts a vote, they receive immediate confirmation that their vote has been securely recorded on the blockchain. The system provides real-time election updates, displaying the number of votes cast but ensuring voter anonymity. After the election concludes, results are displayed transparently, showing the final vote count for each candidate.

For security and auditability, TrueVoice logs key actions such as login attempts, successful votes, and admin interventions. Administrators can generate reports on voter participation, ensuring the integrity of the election process. The system's output ensures that all stakeholders —voters, candidates, and admins—receive relevant, real-time information while maintaining the election’s security and transparency.

4.10 INTERFACE DESIGN

TrueVoice prioritizes a user-friendly interface to ensure a seamless and secure voting experience for students, candidates, and administrators. The design emphasizes clarity, accessibility, and efficiency, making it easy for users to navigate the platform, cast votes, and manage elections.

The voting interface is intuitive, guiding students through the election process with clear instructions. Candidates have a dedicated section where they can view election details, while administrators have an organized dashboard to manage election settings, approve candidates, and monitor real-time voting statistics.

The system uses a clean and responsive layout to enhance usability across different devices. A minimalist and structured design ensures that voters can quickly select their preferred candidate without distractions. Real-time updates provide instant election insights, while secure authentication ensures a safe and transparent voting process.

5. CODING

main.py

```
from flask import *
from public import public
from admin import admin
from candidate import candidate
from student import student
import smtplib
from email.mime.text import MIMEText
# from flask_mail import Mail
app=Flask(__name__)

app.config['MAIL_SERVER']='smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USERNAME'] = 'projectblockchain2025@gmail.com'
app.config['MAIL_PASSWORD'] = 'icrv cqay bqsb clsl'

app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
# mail=Mail(app)



app.register_blueprint(public)
app.register_blueprint(admin)
app.register_blueprint(candidate)
app.register_blueprint(student)

app.secret_key="qweert"
app.run(debug=True,port=5679)
```

admin.py

```
from flask import *
from database import *
import json
from web3 import Web3, HTTPProvider

# truffle development blockchain address
blockchain_address = 'http://127.0.0.1:7545'
# Client instance to interact with the blockchain
web3 = Web3(HTTPProvider(blockchain_address))
# Set the default account (so we don't need to set the "from" for every transaction call)
web3.eth.defaultAccount = web3.eth.accounts[0]
compiled_contract_path =
'C:/Users/rajes/Desktop/final/final/Blockchainvoting/voting/node_modules/.bin/build/
contracts/votin.json'
# Deployed contract address (see `migrate` command output: `contract address`)
deployed_contract_address = '0xE04D34D9F2d50b982D5AB1Da64fa4E2A5eF397A8'

admin=Blueprint(__name__,'admin')



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

#change password
@admin.route("/ad_change_password",methods=['get','post'])
def ad_change_password():
    login_id=session['admin_id']
```

```
    if "submit" in request.form:
        password=request.form['password']
        q1="UPDATE login SET password='%s' WHERE login_id='%s'"%(password,login_id)
        update(q1)
        return "<script>alert('password
changed');window.location='/ad_change_password';</script>"
    return render_template("admin/ad_change_password.html")


#manage course
@admin.route('/manage_course', methods=['GET', 'POST'])
def manage_course():
    # Display data
    data = {}
    q0 = "SELECT * FROM course"
    data['view'] = select(q0)

    # Insert data
    if 'submit' in request.form:
        course_name = request.form['course_name']

        # Check for duplicate course name
        q_check = "SELECT * FROM course WHERE course_name='%s'" % (course_name)
        existing_course = select(q_check)

        if existing_course:
            return "<script>alert('Course  already
added!');window.location='/manage_course'</script>"
        else:
            q2 = "INSERT INTO course VALUES (NULL, '%s')" % (course_name)
            insert(q2)
```

```
        return "<script>alert('Course added
successfully');window.location='/manage_course'</script>"


    # Update data
    if 'action' in request.args:
        action = request.args['action']
        course_id = request.args['course_id']
    else:
        action = None

    if action == 'update':
        q1 = "SELECT * FROM course WHERE course_id='%s'" % (course_id)
        res = select(q1)
        data['up'] = res

    if 'updates' in request.form:
        course_name = request.form['course_name']
        q3 = "UPDATE course SET course_name='%s' WHERE course_id='%s'" %
(course_name, course_id)
        update(q3)
        return "<script>alert('Course updated
successfully');window.location='/manage_course'</script>"
    #delete data
    if action=='delete':
        q4="delete from course where course_id='%s'"%(course_id)
        delete(q4)
        return"<script>alert('delete sucessfully');window.location='manage_course'</script>"

    return render_template('admin/manage_course.html',data=data)

#delete data
```

```
    # if action=='delete':
    #     q4="delete from course where course_id='%s'"%(course_id)
    #     delete(q4)
    #     return"<script>alert('delete
sucessfully');window.location='manage_course'</script>"

    # return render_template('admin/manage_course.html',data=data)

#manage student
@admin.route('/manage_student',methods=['get','post'])
def manage_student():
    data={}
    q="""select * from student inner  join course on student.course_id=course.course_id
      inner join login on student.login_id=login.login_id"""
    q1="select * from course "
    data ['stud']=select(q)
    data ['cour']=select(q1)
#insert into database
    if 'submit' in request.form:
        username=request.form['username']
        password=request.form['password']
        course_id=request.form['course_id']
        first_name=request.form['first_name']
        last_name=request.form['last_name']
        email=request.form['email']
        phone=request.form['phone']
        gender=request.form['gender']
        dob=request.form['dob']
        house_name=request.form['house_name']
        place=request.form['place']
```

```
        pincode=request.form['pincode']


        q2="insert into login values(null,'%s','%s','student')"%(username,password)
        log_details=insert(q2)



        q3="insert into student
values(null,'%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s')"%
(log_details,course_id,first_name,last_name,email,phone,gender,dob,house_name,place,pinc
ode)
        insert(q3)
        return"<script>alert('student added
sucessfully');window.location='manage_student'</script>"


    if 'action' in request.args:
        action=request.args['action']
        student_id=request.args['student_id']
    else:
        action=None



    if action=="update":
        q2="select * from student inner join course on student.course_id = course.course_id
where  student_id='%s'"%(student_id)
        data['up']=select(q2)


    #update data
    if 'updates' in request.form:
        course_id=request.form['course_id']
        first_name=request.form['first_name']
        last_name=request.form['last_name']
```

```
        email=request.form['email']
        phone=request.form['phone']
        gender=request.form['gender']
        dob=request.form['dob']
        house_name=request.form['house_name']
        place=request.form['place']
        pincode=request.form['pincode']

        q3="update student set
course_id='%s',first_name='%s',last_name='%s',email='%s',phone='%s',gender='%s',dob='
%s',house_name='%s',place='%s',pincode='%s' where student_id='%s'"%
(course_id,first_name,last_name,email,phone,gender,dob,house_name,place,pincode,student_
id)
        update(q3)
        return "<script>alert('student update
sucessfully');window.location='manage_student'</script>"

    # delete data
     if action=='delete':
        q4="delete from student where student_id='%s'"%(student_id)
        delete(q4)
        return"<script>alert('delete sucessfully');window.location='manage_student'</script>"

    return render_template('admin/manage_student.html',data=data)

# manage post

@admin.route('/manage_post',methods=['get','post'])
def manage_post():
#display data
    data={}
```

```
    q0="select * from post inner join election where post.election_id=election.election_id"
    q1="select *  from election "

    data['view']=select(q0)
    data ['election']=select(q1)



    #insert  data
    if 'submit' in request.form:
        election_id=request.form['election']
        post_name=request.form['post_name']

        print(election_id)
        print(post_name)

        q2="insert into post values(null,'%s',curdate(),'%s')"%(post_name,election_id)
        insert(q2)
        return "<script>alert('post added
sucessfully');window.location='/manage_post'</script>"
#update data
    if 'action' in request.args:
        action=request.args['action']
        post_id=request.args['post_id']
    else:
        action=None

    if action=='update':
        q1="select * from post inner join election on post.election_id=election.election_id where
post_id='%s'"%(post_id)
        res=select(q1)
```

```
        data['up']=res


    if 'updates' in request.form:
        election_id=request.form['election_id']
        post_name=request.form['post_name']


        q3="update post set election_id='%s', post_name='%s' where post_id='%s'"%
(election_id,post_name,post_id)
        update(q3)
        return "<script>alert('post update
sucessfully');window.location='manage_post'</script>"


   # delete data
    if action=='delete':
        q4="delete from post where post_id='%s'"%(post_id)
        delete(q4)
        return"<script>alert('delete sucessfully');window.location='manage_post'</script>"



    return render_template('admin/manage_post.html',data=data)



# manage election


@admin.route('/manage_election',methods=['get','post'])
def manage_election():
#display data
    data={}
    q0="select * from election"
    data['view']=select(q0)
```

```
    #insert  data
    if 'submit' in request.form:
        election_name=request.form['election_name']
        date=request.form['date']
        venue=request.form['venue']

        q2="insert into election values(null,'%s','%s','%s','pending')"%
(election_name,date,venue)
        insert(q2)
        return "<script>alert('election added
sucessfully');window.location='/manage_election'</script>"
    #update data
    if 'action' in request.args:
        action=request.args['action']
        election_id=request.args['election_id']
    else:
        action=None

    if action=='update':
        q1="select * from election where election_id='%s'"%(election_id)
        res=select(q1)
        data['up']=res
    if 'updates' in request.form:
        election_name=request.form['election_name']
        date=request.form['date']
        venue=request.form['venue']

        q3="update election set election_name='%s',date='%s',venue='%s' where
election_id='%s'"%(election_name,date,venue,election_id)
```

```
        update(q3)
        return "<script>alert('election update
sucessfully');window.location='manage_election'</script>"

    if 'action' in request.args:
        action=request.args['action']
        election_id=request.args['election_id']
    else:
        action=None
    if action=="start":
        q7="update election set status='start' where election_id='%s'"%(election_id)
        update(q7)
        return "<script>alert('Election Start
Successfully');window.location='/manage_election'</script>"

    if action=="completed":
        q7="update election set status='completed' where election_id='%s'"%(election_id)
        update(q7)
        return "<script>alert('Election Stop
Successfully');window.location='/manage_election'</script>"

    # delete data
    if action=='delete':
        q4="delete from election where election_id='%s'"%(election_id)
        delete(q4)
        return"<script>alert('delete sucessfully');window.location='manage_election'</script>"


    return render_template('admin/manage_election.html',data=data)
@admin.route('/view_candidates',methods=['get','post'])
def view_candidates():
```

```
    # student_id=session['student_id']


#display data
    data={}
    q0='''select candidate.*,election.election_name ,post.*,student.*,course.* from candidate
inner join election on candidate.election_id=election.election_id
     inner join post on candidate.post_id=post.post_id
     inner join student on candidate.student_id=student.student_id
     inner join course on student.course_id=course.course_id'''

    data['view']=select(q0)
#delete data
    if 'action' in request.args:
        action=request.args['action']
        student_id=request.args['student_id']
    else:
        action=None

    if action=='approve':
        qq1="select * from student where student_id='%s'"%(student_id)
        res=select(qq1)
        std_log_id=res[0]['login_id']
        q6="update login set usertype='candidate' where login_id='%s'"%(std_log_id)
        update(q6)
        q7="update candidate set status='approved' where student_id='%s'"%(student_id)
        update(q7)
        return "<script>alert('Approved
Successfully');window.location='/view_candidates'</script>"
    if action=="reject":
        qq1="select * from student where student_id='%s'"%(student_id)
```

```
        res=select(qq1)
        stds_log_id=res[0]['login_id']
        q10="update login set usertype='student' where login_id='%s'"%(stds_log_id)
        rr=update(q10)
        print(rr)

        q9="update candidate set status='rejected' where student_id='%s'"%(student_id)
        rrr=update(q9)
        print(rrr)
        return "<script>alert('Rejected');window.location='/view_candidates'</script>"

    return render_template('admin/view_candidate.html',data=data)


@admin.route('/view_verified_candidate',methods=['get','post'])

def view_verified_candidate():
    data={}
    q0='''select candidate.*,election.election_name ,post.*,student.*,course.* from candidate
inner join election on candidate.election_id=election.election_id
     inner join post on candidate.post_id=post.post_id
     inner join student on candidate.student_id=student.student_id
     inner join course on student.course_id=course.course_id where
candidate.status='approved' '''
    data['view']=select(q0)
    return render_template('admin/view_verified_candidate.html',data=data)

#view complaints
@admin.route('/view_complaints',methods=['GET','POST'])
def view_complaints():
```

```
    data={}


    q = """
    SELECT complaints.*,
        CASE
            WHEN complaints.usertype = 'student' THEN student.first_name
            WHEN complaints.usertype = 'candidate' THEN s2.first_name
        END AS name,
        complaints.usertype
    FROM complaints
    LEFT JOIN student ON complaints.id = student.student_id
    LEFT JOIN candidate ON complaints.id = candidate.candidate_id
    LEFT JOIN student AS s2 ON candidate.student_id = s2.student_id
    """
    data['view']=select(q)
    return render_template('admin/view_complaints.html',data=data)


@admin.route('/send_reply',methods=['GET','POST'])
def sent_reply():
    complaint_id=request.args['id']
    if 'submit' in request.form:
        reply=request.form['reply']
        q="update complaints set reply='%s' where complaint_id='%s'"%(reply,complaint_id)
        update(q)
        return "<script>alert('Replied
successfully');window.location='/view_complaints'</script>"
    return render_template('admin/send_reply.html')
#feedback
@admin.route('/view_feedback',methods=['GET','POST'])
def view_feedback():
```

```
    data={}


    q = """
    SELECT feedback.*,
        CASE
            WHEN feedback.usertype = 'student' THEN student.first_name
            WHEN feedback.usertype = 'candidate' THEN s2.first_name
        END AS name,
        feedback.usertype
    FROM feedback
    LEFT JOIN student ON feedback.id = student.student_id
    LEFT JOIN candidate ON feedback.id = candidate.candidate_id
    LEFT JOIN student AS s2 ON candidate.student_id = s2.student_id
    """
    data['view']=select(q)
    return render_template('admin/view_feedback.html',data=data)



 #campaigns
@admin.route('/view_campaigns',methods=['GET','POST'])
def view_campaigns():
    data={}
    q="select * from campaigns"
    data['view']=select(q)
    if 'action' in request.args:
        action=request.args['action']
        campaign_id=request.args['campaign_id']
    else:
        action=None
```

```
    if action=='delete':
        q4="delete from campaigns where campaign_id='%s'"%(campaign_id)
        delete(q4)
        return"<script>alert('delete sucessfully');window.location='view_campaigns'</script>"


    return render_template('admin/view_campaigns.html',data=data)
#RESULT
@admin.route('/a_view_election',methods=['GET','POST'])
def a_view_election():
    data={}
    q="select * from election"
    data['view']=select(q)
    return render_template('admin/a_view_election.html',data=data)


@admin.route('/a_view_post',methods=['GET','POST'])
def a_view_post():
    p_id= request.args.get('id')


    data={}
    q="select * from post inner join election on post.election_id=election.election_id where
election.election_id='%s' """%(p_id)
    data['view']=select(q)
    return render_template('admin/a_view_post.html',data=data)
@admin.route('/view_result', methods=['GET', 'POST'])
def view_result():
    election_id = request.args.get('election_id')
    post_id = request.args.get('post_id')
    data = {}
```

```
    with open(compiled_contract_path) as file:
        contract_json = json.load(file)
        contract_abi = contract_json['abi']
    contract = web3.eth.contract(address=deployed_contract_address, abi=contract_abi)
    blocknumber = web3.eth.get_block_number()
    candidate_votes = {}

    # Initialize variables to prevent UnboundLocalError
    max_candidate = None
    max_votes = 0

    try:
        # Loop through all blocks to collect votes for each candidate
        for i in range(blocknumber, 0, -1):
            a = web3.eth.get_transaction_by_block(i, 0)
            decoded_input = contract.decode_function_input(a['input'])

            # Debug: print decoded input
            print("Decoded input:", decoded_input)

            # Check if the transaction matches the voting function
            if str(decoded_input[0]) == "<Function
add_vote(uint256,uint256,uint256,uint256,uint256,string,string)>":
                print("Found matching function call in transaction.")

                # Check if it matches the specified post and election
                if int(decoded_input[1]['post_id']) == int(post_id) and int(decoded_input[1]
['election_id']) == int(election_id):
                    candidate_id = decoded_input[1].get('candidate_id')
```

```
            print("candidate_id:", candidate_id)

            # Count votes for each candidate_id
            if candidate_id:
                candidate_votes[candidate_id] = candidate_votes.get(candidate_id, 0) + 1
                print("Updated candidate_votes:", candidate_votes)

    except Exception as e:
        print("Error:", e)

    # Determine the candidate with the maximum votes after populating candidate_votes
    if candidate_votes:
        print("++++++++++++++++++++++++++")
        max_candidate = max(candidate_votes, key=candidate_votes.get)
        max_votes = candidate_votes[max_candidate]
        print("Winner candidate_id:", max_candidate, "with votes:", max_votes)
    else:
        print("No votes found for the specified election and post.")

    # Debug: print the final candidate_votes dictionary
    print("Final candidate_votes dictionary:", candidate_votes)

    # Secure insert using parameterized query
    q = "SELECT * FROM result WHERE result.post_id='%s' AND result.election_id='%s'" %
(post_id, election_id)
    res = select(q)

    if res:
        result_id = res[0]['result_id']
```

```
        qq = "SELECT * FROM `student` INNER JOIN candidate USING (student_id) INNER JOIN `result` USING(candidate_id) INNER JOIN course USING(course_id) WHERE result_id='%s'" % (result_id)
        data['viewsss'] = select(qq)
    else:
        query = "INSERT INTO result (candidate_id, election_id, post_id, result) VALUES (%s, %s, %s, %s)" % (max_candidate, election_id, post_id, max_votes)
        insert(query)
        print("Inserted result into database:", query)

    # Store the winner and all vote counts in the data dictionary
    data['winner'] = max_candidate
    data['max_votes'] = max_votes
    data['candidate_votes'] = candidate_votes
    print("Final winner:", max_candidate)
    print("Vote counts:", candidate_votes)

    return render_template('admin/view_result.html', data=data)
```

student.py

```
from flask import *
from database import *

# from flask_mail import Mail, Message

import random
import string
import smtplib
from email.mime.text import MIMEText
```

```
import datetime
import json
from web3 import Web3, HTTPProvider

# truffle development blockchain address
blockchain_address = 'http://127.0.0.1:7545'
# Client instance to interact with the blockchain
web3 = Web3(HTTPProvider(blockchain_address))
# Set the default account (so we don't need to set the "from" for every transaction call)
web3.eth.defaultAccount = web3.eth.accounts[0]
compiled_contract_path =
'C:/Users/rajes/Desktop/final/final/Blockchainvoting/voting/node_modules/.bin/build/
contracts/votin.json'
# Deployed contract address (see `migrate` command output: `contract address`)
deployed_contract_address = '0xE04D34D9F2d50b982D5AB1Da64fa4E2A5eF397A8'

import random
student=Blueprint(__name__,'student')

def generate_otps():
    return ''.join(random.choices(string.digits, k=6))

#view all cadidate
@student.route('/a_view_all_candidates',methods=['get','post'])

def a_view_all_candidates():
    p_id= request.args.get('id')
    student_id=session['student_id']

    data={}
```

```
    q0="""select candidate.*,election.election_name ,post.*,student.*,course.* from candidate inner join election on candidate.election_id=election.election_id
     inner join post on candidate.post_id=post.post_id
     inner join student on candidate.student_id=student.student_id
     inner join course on student.course_id=course.course_id where candidate.status='approved'and candidate.post_id='%s' """%(p_id)
    data['view']=select(q0)
    q1="select * from vote where student_id='%s' and post_id='%s' "%(student_id,p_id)
    data['views']=select(q1)

    if 'action' in request.args:
        action=request.args['action']

        candidate_id = request.args["candidate_id"]
        # email = request.args["email"]
        election_id=request.args['election_id']
        post_id=request.args['post_id']
    else:
        action=None
    if action=='sotp':
# Generate OTP
        otp = generate_otps()
        print("----------",otp)

# Send OTP to the user's email
        try:
            gmail = smtplib.SMTP('smtp.gmail.com', 587)
            gmail.ehlo()
            gmail.starttls()
            gmail.login('projectblockchain2025@gmail.com', 'icrv cqay bqsb clsl')
```

```
        msg = MIMEText(f'Your OTP for voting is {otp}')
        msg['Subject'] = 'Your OTP for Voting'
        msg['To'] = session['email']
        print("----------------------",session['email'])
        msg['From'] = 'projectblockchain2025@gmail.com'

        gmail.send_message(msg)
        gmail.quit()

        flash('An OTP has been sent to your email. Please enter it to complete your registration.')
    except smtplib.SMTPException as e:
        print("Couldn't send email: " + str(e))
        flash("Failed to send OTP. Please try again.")
        return redirect(url_for('student.student_home'))


    # # Temporarily store user details and OTP in session
    session['user_details'] = {
        'candidate_id': candidate_id,

        'email': session['email'],
        # 'student_id': student_id,
        'election_id': election_id,
        'post_id': post_id,

    }
    session['otp'] = otp
```

```
        return redirect(url_for('student.sotp',))

    return render_template('student/a_view_all_candidates.html',data=data)


@student.route('/sotp', methods=['GET', 'POST'])
def sotp():

    student_id=session['student_id']
    if 'verify' in request.form:
        entered_otp = request.form['otp']
        if entered_otp == session.get('otp'):
            shop_details = session.get('user_details')
            if shop_details:
                candidate_id=shop_details['candidate_id']
                election_id=shop_details['election_id']
                post_id=shop_details['post_id']
                print("----------",post_id)
                d=datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                with open(compiled_contract_path) as file:
                    contract_json = json.load(file)  # load contract info as JSON
                    contract_abi = contract_json['abi']  # fetch contract's abi - necessary to call its
functions
                contract = web3.eth.contract(address=deployed_contract_address,
abi=contract_abi)
                id=web3.eth.get_block_number()
                message =
contract.functions.add_vote(int(id),int(election_id),int(post_id),int(student_id),int(candidate_
id),d,'voted').transact()
```

```
            q2 = "insert into vote values(null, '%s', '%s', '%s', 'completed')" % (student_id,
election_id, post_id)
            insert(q2)



            flash("verified successfully")
          return "<script>alert(' THANK YOU FOR YOUR
VOTING .');window.location='student_home'</script>"
       else:
          flash('Invalid OTP. Please try again.')


   return render_template('student/otps.html')
#Home page
@student.route('/student_home')
def index():
   return render_template('student/student_home.html')
#change password
@student.route("/st_change_password",methods=['get','post'])
def st_change_password():
   login_id=session['login_id']
   if "submit" in request.form:
      password=request.form['password']
      q1="UPDATE login SET password='%s' WHERE login_id='%s'"%(password,login_id)
      update(q1)
      return "<script>alert('password changed
sucessfully');window.location='/st_change_password';</script>"
   return render_template("student/st_change_password.html")



#view election
@student.route('/view_election',methods=['GET','POST'])
```

```
def view_election():
    data={}
    q="select * from election"
    data['view']=select(q)
    return render_template('student/view_election.html',data=data)


#view post
@student.route('/view_post',methods=['GET','POST'])
def view_post():
    p_id= request.args.get('id')


    data={}
    q="select * from post inner join election on post.election_id=election.election_id where election.election_id='%s' """%(p_id)
    data['view']=select(q)
    return render_template('student/view_post.html',data=data)



#view campaigns
@student.route('/st_view_campaigns',methods=['GET','POST'])
def st_view_campaigns():
    data={}
    q="select * from campaigns"
    data['view']=select(q)


    return render_template('student/st_view_campaigns.html',data=data)


#apply as candidate
@student.route('/apply_as_candidate', methods=['GET', 'POST'])
def apply_as_candidate():
```

```
    student_id = session['student_id']
    data = {}

    # Fetch available posts and elections
    q0 = "SELECT * FROM post INNER JOIN election WHERE post.election_id = election.election_id"
    q1 = "SELECT * FROM election"

    data['post'] = select(q0)
    data['election'] = select(q1)

    # Insert into the database
    if 'submit' in request.form:
        post_id = request.form['post_id']
        election_id = request.form['election_id']

        # Check if the student has already applied for a post in the selected election
        q_check = "SELECT * FROM candidate WHERE student_id='%s' AND election_id='%s'" % (student_id, election_id)
        existing_application = select(q_check)

        if existing_application:
            # If the student has already applied for a post in this election, show an error
            return "<script>alert('You have already applied for a post in this election. You can only apply for one post per election.');window.location='apply_as_candidate'</script>"

        # Proceed with the application if no existing record is found
        q2 = "UPDATE login SET usertype='pending' WHERE login_id='%s'" % (session['login_id'])
        log_details = insert(q2)
```

```
        q3 = "INSERT INTO candidate VALUES (null, '%s', '%s', CURDATE(), 'pending', '%s')" % (student_id, election_id, post_id)
        insert(q3)

        return "<script>alert('Candidate application submitted successfully. Please wait for admin approval.');window.location='apply_as_candidate'</script>"

    if 'action' in request.args:
        action = request.args['action']
        candidate_id = request.args['candidate_id']
    else:
        action = None

    return render_template('student/apply_as_candidate.html', data=data)


#feedback


@student.route('/feedback',methods=['GET','POST'])
def feedback():

    student_id= session['student_id']
    data={}
    q1="select * from feedback where id='%s'"%(student_id)
    data['view']=select(q1)
    print(data)


    if 'submit' in request.form:
```

```
    feedback=request.form['feedback']
    q1="insert into feedback values(NULL,'%s',curdate(),'%s','student')"%
(student_id,feedback)
    insert(q1)
    return "<script>alert(' Feedback Send Successfully
');window.location='feedback'</script>"


  return render_template('student/feedback.html',data=data)


#complaints
@student.route('/send_complaint',methods=['GET','POST'])
def cd_sent_complaint():
  student_id= session['student_id']
  data={}
  q1="select * from complaints inner join student on student.student_id=complaints.id
where id='%s'"%(student_id)

  data['view']=select(q1)
  if 'submit' in request.form:
    complaint=request.form['complaint']
    q="insert into complaints values(NULL,'%s','%s',curdate(),'pending','student')"%
(student_id,complaint)
    insert(q)
    return "<script>alert(' Complaints Send Successfully
');window.location='send_complaint'</script>"

  return render_template('student/send_complaint.html',data=data)

#RESULT
```

```
@student.route('/s_view_election',methods=['GET','POST'])
def s_view_election():
    data={}
    q="select * from election"
    data['view']=select(q)
    return render_template('student/s_view_election.html',data=data)



@student.route('/s_view_post',methods=['GET','POST'])
def s_view_post():
    p_id= request.args.get('id')


    data={}
    q="select * from post inner join election on post.election_id=election.election_id where election.election_id='%s' """%(p_id)
    data['view']=select(q)
    return render_template('student/s_view_post.html',data=data)
@student.route('/st_view_result', methods=['GET', 'POST'])
def st_view_result():
    election_id = request.args.get('election_id')
    post_id = request.args.get('post_id')
    data = {}


    with open(compiled_contract_path) as file:
        contract_json = json.load(file)
        contract_abi = contract_json['abi']
    contract = web3.eth.contract(address=deployed_contract_address, abi=contract_abi)
    blocknumber = web3.eth.get_block_number()
    candidate_votes = {}
```

```
    # Initialize variables to prevent UnboundLocalError
    max_candidate = None
    max_votes = 0

    try:
        # Loop through all blocks to collect votes for each candidate
        for i in range(blocknumber, 0, -1):
            a = web3.eth.get_transaction_by_block(i, 0)
            decoded_input = contract.decode_function_input(a['input'])

            # Debug: print decoded input
            print("Decoded input:", decoded_input)

            # Check if the transaction matches the voting function
            if str(decoded_input[0]) == "<Function
add_vote(uint256,uint256,uint256,uint256,uint256,string,string)>":
                print("Found matching function call in transaction.")

                # Check if it matches the specified post and election
                if int(decoded_input[1]['post_id']) == int(post_id) and int(decoded_input[1]
['election_id']) == int(election_id):
                    candidate_id = decoded_input[1].get('candidate_id')
                    print("candidate_id:", candidate_id)

                    # Count votes for each candidate_id
                    if candidate_id:
                        candidate_votes[candidate_id] = candidate_votes.get(candidate_id, 0) + 1
                        print("Updated candidate_votes:", candidate_votes)

    except Exception as e:
```

```
        print("Error:", e)

    # Determine the candidate with the maximum votes after populating candidate_votes
    if candidate_votes:
        print("++++++++++++++++++++++++++")
        max_candidate = max(candidate_votes, key=candidate_votes.get)
        max_votes = candidate_votes[max_candidate]
        print("Winner candidate_id:", max_candidate, "with votes:", max_votes)
    else:
        print("No votes found for the specified election and post.")

    # Debug: print the final candidate_votes dictionary
    print("Final candidate_votes dictionary:", candidate_votes)

    # Secure insert using parameterized query
    q = "SELECT * FROM result WHERE result.post_id='%s' AND result.election_id='%s'" % (post_id, election_id)
    res = select(q)

    if res:
        result_id = res[0]['result_id']
        qq = "SELECT * FROM `student` INNER JOIN candidate USING (student_id) INNER JOIN `result` USING(candidate_id) INNER JOIN course USING(course_id) WHERE result_id='%s'" % (result_id)
        data['viewsss'] = select(qq)
    else:
        query = "INSERT INTO result (candidate_id, election_id, post_id, result) VALUES (%s, %s, %s, %s)" % (max_candidate, election_id, post_id, max_votes)
        insert(query)
        print("Inserted result into database:", query)
```

```
    # Store the winner and all vote counts in the data dictionary
    data['winner'] = max_candidate
    data['max_votes'] = max_votes
    data['candidate_votes'] = candidate_votes
    print("Final winner:", max_candidate)
    print("Vote counts:", candidate_votes)
    return render_template('student/st_view_result.html', data=data)
```

candidate.py

```
from flask import *
from database import *
# from flask_mail import Mail, Message
import random
import string
import smtplib
from email.mime.text import MIMEText
import datetime

import json
from web3 import Web3, HTTPProvider


# truffle development blockchain address
blockchain_address = 'http://127.0.0.1:7545'
# Client instance to interact with the blockchain
web3 = Web3(HTTPProvider(blockchain_address))
# Set the default account (so we don't need to set the "from" for every transaction call)
web3.eth.defaultAccount = web3.eth.accounts[0]
```

```
compiled_contract_path =
'C:/Users/rajes/Desktop/final/final/Blockchainvoting/voting/node_modules/.bin/build/
contracts/votin.json'

# Deployed contract address (see `migrate` command output: `contract address`)

deployed_contract_address = '0xE04D34D9F2d50b982D5AB1Da64fa4E2A5eF397A8'


candidate=Blueprint(__name__,'candidate')


def generate_otp():

   return ''.join(random.choices(string.digits, k=6))

#view all cadidate

@candidate.route('/view_all_candidates',methods=['get','post'])


def view_all_candidates():

   p_id= request.args.get('id')

   student_id=session['std_id']


   data={}

   q0="""select candidate.*,election.election_name ,post.*,student.*,course.* from candidate
inner join election on candidate.election_id=election.election_id

    inner join post on candidate.post_id=post.post_id

    inner join student on candidate.student_id=student.student_id

    inner join course on student.course_id=course.course_id where
candidate.status='approved'and candidate.post_id='%s' """%(p_id)

   data['view']=select(q0)

   print("++++++++++",q0)



   q1="select * from vote where student_id='%s' and post_id='%s' "%(student_id,p_id)

   data['views']=select(q1)
```

```
    if 'action' in request.args:
        action=request.args['action']

        candidate_id = request.args["candidate_id"]
        # email = request.args["email"]

        election_id=request.args['election_id']
        post_id=request.args['post_id']
    else:
        action=None
    if action=='cotp':



        q3="select * from student inner join candidate using (student_id) where
login_id='%s'"%(session['login_id'])
        res3=select(q3)
        if res3:
            session['email']=res3[0]['email']
            print("0000000000000000000000000000000000000000000000000",session['email'])
# Generate OTP
        otp = generate_otp()
        print("----------",otp)

# Send OTP to the user's email
        try:
            gmail = smtplib.SMTP('smtp.gmail.com', 587)
            gmail.ehlo()
            gmail.starttls()
            gmail.login('projectblockchain2025@gmail.com', 'icrv cqay bqsb clsl')
```

```
        msg = MIMEText(f'Your OTP for voting is {otp}')
        msg['Subject'] = 'Your OTP for Voting'
        msg['To'] =session['email']
        msg['From'] = 'projectblockchain2025@gmail.com'

        gmail.send_message(msg)
        gmail.quit()

        flash('An OTP has been sent to your email. Please enter it to complete your registration.')
    except smtplib.SMTPException as e:
        print("Couldn't send email: " + str(e))
        flash("Failed to send OTP. Please try again.")
        return redirect(url_for('candidate.candidate_home'))


    # # Temporarily store user details and OTP in session
    session['user_details'] = {
        'candidate_id': candidate_id,

        'email': session['email'],
        # 'student_id': student_id,
        'election_id': election_id,
        'post_id': post_id,

    }
    session['otp'] = otp

    return redirect(url_for('candidate.verify_otp',))
```

```
    return render_template('candidate/view_all_candidates.html',data=data)


@candidate.route('/verify_otp', methods=['GET', 'POST'])
def verify_otp():

    student_id=session['std_id']
    if 'verify' in request.form:
        entered_otp = request.form['otp']
        if entered_otp == session.get('otp'):
            shop_details = session.get('user_details')
            if shop_details:
                candidate_id=shop_details['candidate_id']
                election_id=shop_details['election_id']
                post_id=shop_details['post_id']
                print("----------",post_id)
                d=datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                with open(compiled_contract_path) as file:
                    contract_json = json.load(file)  # load contract info as JSON
                    contract_abi = contract_json['abi']  # fetch contract's abi - necessary to call its functions
                contract = web3.eth.contract(address=deployed_contract_address, abi=contract_abi)
                id=web3.eth.get_block_number()
                message = contract.functions.add_vote(int(id),int(election_id),int(post_id),int(student_id),int(candidate_id),d,'voted').transact()

                q2 = "insert into vote values(null, '%s', '%s', '%s', 'completed')" % (student_id, election_id, post_id)
                insert(q2)
```

```
            flash("verified successfully")
        return "<script>alert(' THANK YOU FOR YOUR
VOTING .');window.location='candidate_home'</script>"
    else:
        flash('Invalid OTP. Please try again.')
  return render_template('candidate/cotp.html')
@candidate.route('/candidate_home')
def candidate_home():
  return render_template('candidate/candidate_home.html')
#change password
@candidate.route("/cd_change_password",methods=['get','post'])
def cd_change_password():
  login_id=session['login_id']
  if "submit" in request.form:
    password=request.form['password']
    q1="UPDATE login SET password='%s' WHERE login_id='%s'"%(password,login_id)
    update(q1)
    return "<script>alert('password
changed');window.location='/cd_change_password';</script>"
  return render_template("candidate/cd_change_password.html")

#view profile
@candidate.route('/view_profile',methods=['GET','POST'])
def view_profile():
  data={}
  q="select * from candidate inner join student on candidate.student_id=student.student_id
inner join course on student.course_id=course.course_id  where candidate_id='%s'"%
(session['candidate_id'])
  data['stud']=select(q)
  return render_template('candidate/view_profile.html',data=data)
```

```
#view students
@candidate.route('/view_students',methods=['get','post'])
def manage_student():
    data={}
    q="select * from student inner join course on student.course_id=course.course_id "
    q1="select * from course "
    data ['stud']=select(q)
    data ['cour']=select(q1)

    return render_template('candidate/view_students.html',data=data)

#manage campaigns
@candidate.route('/manage_campaigns',methods=['get','post'])
def manage_campaigns():
     #view data
    data={}
    q="select * from campaigns "
    data ['view']=select(q)
    #insert  data
    if 'submit' in request.form:
        campaign_name=request.form['campaign_name']
        venue=request.form['venue']
        campaign_date=request.form['campaign_date']
        q2="insert into campaigns values(null,'%s',curdate(),'%s','%s')"%
(campaign_name,venue,campaign_date)
        insert(q2)
        return "<script>alert('campaigns added
sucessfully');window.location='/manage_campaigns'</script>"
```

```
#update data
    if 'action' in request.args:
        action=request.args['action']
        campaign_id=request.args['campaign_id']
    else:
        action=None

    if action=='update':
        q1="select * from campaigns where campaign_id='%s'"%(campaign_id)
        res=select(q1)
        data['up']=res
    if 'updates' in request.form:
        campaign_name=request.form['campaign_name']
        venue=request.form['venue']
        campaign_date=request.form['campaign_date']
        q3="update campaigns set campaign_name='%s',venue='%s',campaign_date='%s'
where campaign_id='%s'"%(campaign_name,venue,campaign_date,campaign_id)
        update(q3)
        return "<script>alert('campaigns update
sucessfully');window.location='manage_campaigns'</script>"



#delete data
    if action=='delete':
        q4="delete from campaigns where campaign_id='%s'"%(campaign_id)
        delete(q4)
        return"<script>alert('delete
sucessfully');window.location='manage_campaigns'</script>"


    return render_template('candidate/manage_campaigns.html',data=data)
```

```
#view election
@candidate.route('/cd_view_election',methods=['GET','POST'])
def cd_view_election():
    data={}
    q="select * from election"
    data['view']=select(q)
    return render_template('candidate/cd_view_election.html',data=data)



#view post
@candidate.route('/cd_view_post',methods=['GET','POST'])
def cd_view_post():
    p_id= request.args.get('id')


    data={}
    q="select * from post inner join election on post.election_id=election.election_id where election.election_id='%s' """%(p_id)
    data['view']=select(q)
    return render_template('candidate/cd_view_post.html',data=data)




#feedback
@candidate.route('/cd_feedback',methods=['GET','POST'])
def cd_feedback():



    candidate_id= session['candidate_id']
    data={}
```

```
    q1="select * from feedback where id='%s'"%(candidate_id)
    data['view']=select(q1)
    print(data)



    if 'submit' in request.form:
        feedback=request.form['feedback']
        q1="insert into feedback values(NULL,'%s',curdate(),'%s','candidate')"%
(candidate_id,feedback)
        insert(q1)
        return "<script>alert(' Feedback Send Successfully
');window.location='cd_feedback'</script>"



    return render_template('candidate/cd_feedback.html',data=data)




#complaints
@candidate.route('/cd_send_complaint',methods=['GET','POST'])
def cd_sent_complaint():
    candidate_id= session['candidate_id']
    data={}
    q1="select * from complaints inner join candidate on
candidate.candidate_id=complaints.id  where id='%s'"%(candidate_id)


    data['view']=select(q1)
    if 'submit' in request.form:
        complaint=request.form['complaint']
```

```
    q="insert into complaints values(NULL,'%s','%s',curdate(),'pending','candidate')"%(candidate_id,complaint)
    insert(q)
    return "<script>alert(' Complaints Send Successfully ');window.location='cd_send_complaint'</script>"


  return render_template('candidate/cd_send_complaint.html',data=data)
#RESULT
 @candidate.route('/r_view_election',methods=['GET','POST'])
def r_view_election():
  data={}
  q="select * from election"
  data['view']=select(q)
  return render_template('candidate/r_view_election.html',data=data)



@candidate.route('/r_view_post',methods=['GET','POST'])
def r_view_post():
  p_id= request.args.get('id')


  data={}
  q="select * from post inner join election on post.election_id=election.election_id where election.election_id='%s' """%(p_id)
  data['view']=select(q)
  return render_template('candidate/r_view_post.html',data=data)
@candidate.route('/cd_view_result', methods=['GET', 'POST'])
def cd_view_result():
  election_id = request.args.get('election_id')
  post_id = request.args.get('post_id')
  data = {}
```

```
    with open(compiled_contract_path) as file:
        contract_json = json.load(file)
        contract_abi = contract_json['abi']
    contract = web3.eth.contract(address=deployed_contract_address, abi=contract_abi)
    blocknumber = web3.eth.get_block_number()
    candidate_votes = {}

    # Initialize variables to prevent UnboundLocalError
    max_candidate = None
    max_votes = 0

    try:
        # Loop through all blocks to collect votes for each candidate
        for i in range(blocknumber, 0, -1):
            a = web3.eth.get_transaction_by_block(i, 0)
            decoded_input = contract.decode_function_input(a['input'])

            # Debug: print decoded input
            print("Decoded input:", decoded_input)

            # Check if the transaction matches the voting function
            if str(decoded_input[0]) == "<Function
add_vote(uint256,uint256,uint256,uint256,uint256,string,string)>":
                print("Found matching function call in transaction.")

                # Check if it matches the specified post and election
                if int(decoded_input[1]['post_id']) == int(post_id) and int(decoded_input[1]
['election_id']) == int(election_id):
                    candidate_id = decoded_input[1].get('candidate_id')
                    print("candidate_id:", candidate_id)
```

```
            # Count votes for each candidate_id
            if candidate_id:
                candidate_votes[candidate_id] = candidate_votes.get(candidate_id, 0) + 1
                print("Updated candidate_votes:", candidate_votes)

    except Exception as e:
        print("Error:", e)

    # Determine the candidate with the maximum votes after populating candidate_votes
    if candidate_votes:
        print("++++++++++++++++++++++++++")
        max_candidate = max(candidate_votes, key=candidate_votes.get)
        max_votes = candidate_votes[max_candidate]
        print("Winner candidate_id:", max_candidate, "with votes:", max_votes)
    else:
        print("No votes found for the specified election and post.")

    # Debug: print the final candidate_votes dictionary
    print("Final candidate_votes dictionary:", candidate_votes)

    # Secure insert using parameterized query
    q = "SELECT * FROM result WHERE result.post_id='%s' AND result.election_id='%s'" %
(post_id, election_id)
    res = select(q)

    if res:
        result_id = res[0]['result_id']
        qq = "SELECT * FROM `student` INNER JOIN candidate USING (student_id) INNER
JOIN `result` USING(candidate_id) INNER JOIN course USING(course_id) WHERE
result_id='%s'" % (result_id)
```

```
        data['viewsss'] = select(qq)
    else:
        query = "INSERT INTO result (candidate_id, election_id, post_id, result) VALUES (%s, %s, %s, %s)" % (max_candidate, election_id, post_id, max_votes)
        insert(query)
        print("Inserted result into database:", query)

    # Store the winner and all vote counts in the data dictionary
    data['winner'] = max_candidate
    data['max_votes'] = max_votes
    data['candidate_votes'] = candidate_votes
    print("Final winner:", max_candidate)
    print("Vote counts:", candidate_votes)


    return render_template('candidate/cd_view_result.html', data=data)
```

database.py

```
import mysql.connector

password=''
database='blockchain_based_voting'

def select(q):

cnx=mysql.connector.connect(user='root',host='localhost',password=password,database=database,port=3306)
        cur=cnx.cursor(dictionary=True)
        cur.execute(q)
        result=cur.fetchall()
```

```
        cnx.close()
        cur.close()
        return result

def delete(q):

cnx=mysql.connector.connect(user='root',host='localhost',password=password,database=dat
abase,port=3306)
        cur=cnx.cursor(dictionary=True)
        cur.execute(q)
        cnx.commit()
        result=cur.rowcount
        cnx.close()
        cur.close()
        return result

def update(q):

cnx=mysql.connector.connect(user='root',host='localhost',password=password,database=dat
abase,port=3306)
        cur=cnx.cursor(dictionary=True)
        cur.execute(q)
        cnx.commit()
        result=cur.rowcount
        cnx.close()
        cur.close()
        return result

def insert(q):

cnx=mysql.connector.connect(user='root',host='localhost',password=password,database=dat
abase,port=3306)
```

```
        cur=cnx.cursor(dictionary=True)
        cur.execute(q)
        cnx.commit()
        result=cur.lastrowid
        cur.close()
        cnx.close()
        return result
```

public.py

```
from flask import *
from database import *
public=Blueprint(__name__,'public')


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

@public.route('/login',methods=['get','post'])
def login():
    if 'submit' in request.form:
        username=request.form['username']
        password=request.form['password']
        q="select * from login where username='%s' and password='%s'"%
(username,password)
        res=select(q)
        if res:
            session['login_id']=res[0]['login_id']
            if res[0]['usertype']=='admin':
                q1="select * from login where login_id='%s'"%(session['login_id'])
```

```
        res1=select(q1)
        if res1:
            session['admin_id']=res1[0]['login_id']
            return "<script>alert('Admin login
success');window.location='/admin_home'</script>"
    elif res[0]['usertype']=='student':
        q2="select * from student where login_id='%s'"%(session['login_id'])
        res2=select(q2)
        if res2:
            session['student_id']=res2[0]['student_id']
            session['email']=res2[0]['email']
            return "<script>alert('Student login
success');window.location='/student_home'</script>"
        else:
            return "<script>alert('invalid student login');window.location='/login'</script>"
    elif res[0]['usertype']=='candidate':

        q3="select * from student where login_id='%s'"%(session['login_id'])
        res3=select(q3)
        student_id=res3[0]['student_id']
        q4="select * from candidate where status='approved' and student_id='%s'"%
(student_id)
        res4=select(q4)
        # session['email']=res4[0]['email']



        if res4[0]['status'] != 'pending':
            session['candidate_id']=res4[0]['candidate_id']

            t=session['std_id']=res4[0]['student_id']
```

```
            # print("----------------------",session['email'])


            return "<script>alert('Candidate login
success');window.location='/candidate_home'</script>"


        else :

            return "<script>alert('Candidate login invalid...you are
student');window.location='/student_home'</script>"


    elif res[0]['usertype']=='pending':

        q5="select * from student where login_id='%s'"%(session['login_id'])

        res3=select(q5)

        if res3:

            session['student_id']=res3[0]['student_id']

            return "<script>alert('u are in wait list for election ... so u are still
student');window.location='/student_home'</script>"

        else:

            return "<script>alert(' student login
invalid');window.location='/login'</script>"



    else:

        return "<script>alert('Invalid username and
password');window.location='/login'</script>"

    return render_template('public/login.html')
```

voting.sql

```
/*

SQLyog Community v13.1.6 (64 bit)

MySQL - 5.7.9 : Database - blockchain_based_voting
```

```
**********************************************************************
*/

/*!40101 SET NAMES utf8 */;

/*!40101 SET SQL_MODE=''*/;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0
*/;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
CREATE DATABASE /*!32312 IF NOT EXISTS*/`blockchain_based_voting` /*!40100
DEFAULT CHARACTER SET latin1 */;

USE `blockchain_based_voting`;

/*Table structure for table `campaigns` */

DROP TABLE IF EXISTS `campaigns`;

CREATE TABLE `campaigns` (
  `campaign_id` int(11) NOT NULL AUTO_INCREMENT,
  `campaign_name` varchar(20) DEFAULT NULL,
  `date` varchar(20) DEFAULT NULL,
  `venue` varchar(20) DEFAULT NULL,
  `campaign_date` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`campaign_id`)
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
```

```
/*Table structure for table `candidate` */

DROP TABLE IF EXISTS `candidate`;

CREATE TABLE `candidate` (
  `candidate_id` int(11) NOT NULL AUTO_INCREMENT,
  `student_id` int(11) DEFAULT NULL,
  `election_id` int(11) DEFAULT NULL,
  `date` varchar(20) DEFAULT NULL,
  `status` varchar(20) DEFAULT NULL,
  `post_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`candidate_id`)
) ENGINE=MyISAM AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;

/*Table structure for table `complaints` */

DROP TABLE IF EXISTS `complaints`;

CREATE TABLE `complaints` (
  `complaint_id` int(11) NOT NULL AUTO_INCREMENT,
  `id` int(11) DEFAULT NULL,
  `complaint` varchar(20) DEFAULT NULL,
  `date` varchar(20) DEFAULT NULL,
  `reply` varchar(20) DEFAULT NULL,
  `usertype` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`complaint_id`)
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

/*Table structure for table `course` */
```

```
DROP TABLE IF EXISTS `course`;

CREATE TABLE `course` (
  `course_id` int(11) NOT NULL AUTO_INCREMENT,
  `course_name` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`course_id`)
) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;


/*Table structure for table `election` */

DROP TABLE IF EXISTS `election`;

CREATE TABLE `election` (
  `election_id` int(11) NOT NULL AUTO_INCREMENT,
  `election_name` varchar(20) DEFAULT NULL,
  `date` varchar(20) DEFAULT NULL,
  `venue` varchar(20) DEFAULT NULL,
  `status` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`election_id`)
) ENGINE=MyISAM AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

/*Table structure for table `feedback` */

DROP TABLE IF EXISTS `feedback`;

CREATE TABLE `feedback` (
  `feedback_id` int(11) NOT NULL AUTO_INCREMENT,
  `id` int(11) DEFAULT NULL,
```

```
  `date` varchar(20) DEFAULT NULL,
  `feedback` varchar(200) DEFAULT NULL,
  `usertype` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`feedback_id`)
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

/*Table structure for table `login` */

DROP TABLE IF EXISTS `login`;

CREATE TABLE `login` (
  `login_id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) DEFAULT NULL,
  `password` varchar(20) DEFAULT NULL,
  `usertype` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`login_id`)
) ENGINE=MyISAM AUTO_INCREMENT=12 DEFAULT CHARSET=latin1;

/*Table structure for table `post` */

DROP TABLE IF EXISTS `post`;

CREATE TABLE `post` (
  `post_id` int(11) NOT NULL AUTO_INCREMENT,
  `post_name` varchar(20) DEFAULT NULL,
  `date` varchar(20) DEFAULT NULL,
  `election_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`post_id`)
) ENGINE=MyISAM AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
```

```
/*Table structure for table `result` */

DROP TABLE IF EXISTS `result`;

CREATE TABLE `result` (
  `result_id` int(11) NOT NULL AUTO_INCREMENT,
  `candidate_id` int(11) DEFAULT NULL,
  `election_id` int(11) DEFAULT NULL,
  `post_id` int(11) DEFAULT NULL,
  `result` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`result_id`)
) ENGINE=MyISAM AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;


/*Table structure for table `student` */
DROP TABLE IF EXISTS `student`;

CREATE TABLE `student` (
  `student_id` int(11) NOT NULL AUTO_INCREMENT,
  `login_id` int(11) DEFAULT NULL,
  `course_id` int(11) DEFAULT NULL,
  `first_name` varchar(20) DEFAULT NULL,
  `last_name` varchar(20) DEFAULT NULL,
  `email` varchar(100) DEFAULT NULL,
  `phone` varchar(20) DEFAULT NULL,
  `gender` varchar(20) DEFAULT NULL,
  `dob` varchar(20) DEFAULT NULL,
  `house_name` varchar(20) DEFAULT NULL,
  `place` varchar(20) DEFAULT NULL,
  `pincode` varchar(20) DEFAULT NULL,
```

```
  PRIMARY KEY (`student_id`)
) ENGINE=MyISAM AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;

/*Table structure for table `vote` */

DROP TABLE IF EXISTS `vote`;

CREATE TABLE `vote` (
  `vote_id` int(11) NOT NULL AUTO_INCREMENT,
  `student_id` int(11) DEFAULT NULL,
  `election_id` int(11) DEFAULT NULL,
  `post_id` int(11) DEFAULT NULL,
  `status` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`vote_id`)
) ENGINE=MyISAM AUTO_INCREMENT=22 DEFAULT CHARSET=latin1;
```

6. SOFTWARE TESTING

Testing is the process of evaluating a software application or system to determine whether it meets the required specifications, works as expected, and is free from defects. It involves executing a series of test cases and scenarios to validate the software's functionality, performance, security, and usability.

The primary purpose of testing is to:

1. Ensure the software meets the required specifications and user expectations.
2. Identify and report defects, errors, or inconsistencies.
3. Validate the software's functionality, performance, security, and usability.
4. Provide feedback to the development team to improve the software quality.
5. Reduce the risk of software failures or errors in production.

In our system we performed the following testing:

6.1 TESTING STRATEGY

A test strategy is a comprehensive, high-level document that defines the overall testing approach and outlines the specific types and levels of testing to be conducted for a product. It establishes the testing framework for the software development life cycle, ensuring that the appropriate testing types and levels are executed to validate the product's quality. The test strategy document is a crucial component of testing documentation, encompassing essential elements such as test strategy components, types of test strategies, and various testing activities. In software engineering, common testing strategies include black box testing, white box testing, and unit testing, among others. This document serves as a foundational guide for testing, ensuring that all stakeholders are aligned on the testing approach and objectives.

6.1.1 Unit Testing

Each module is subjected to individual testing, which takes place during the programming stage. This iterative process involves identifying and correcting any logical errors that arise. The ultimate goal is to verify that every module functions as intended. In our system, we have successfully conducted separate testing on all modules, ensuring their independent operation meets expectations.

6.1.2 Integration Testing

In our project, we successfully conducted integration testing to ensure that individual modules worked together seamlessly. We employed an incremental approach, integrating and testing each module separately to identify and debug defects efficiently. Through this process, we validated data integrity and module interactions, guaranteeing cohesive and functional systems. By doing so, we ensured that our project's components interacted correctly, exchanging data accurately and consistently.

6.1.3 System Testing

We successfully completed system testing, thoroughly evaluating our software's functionality, performance, security, usability, and compatibility. Through rigorous end-to-end testing, we ensured our system works seamlessly, identifying and fixing critical defects. Our team's dedication and meticulous testing approach paid off, resulting in a high-quality software system that meets the required standards and exceeds user expectations.

6.2 TEST CASES

Test Case ID	Test Case	Expected Output	Actual Output
1	Admin enters using username and password	Get access to admin dashboard.	Login successful Dashboard displayed
2	Admin tries to change password	Password changed.	Password changed and message displayed
3	Admin manages course	Admin can add, edit, and delete course data.	Admin successfully added, edited and deleted course.
4	Admin manages students.	Admin can add, edit, and delete student data.	Admin successfully added, edited and deleted student.
5	Admin manages post	Admin can add, edit, and delete post.	Admin successfully added, edited and deleted post.
6	Admin manages election	Admin can add, edit, delete, and start/stop elections	Admin successfully added, edited, deleted, and started/stopped elections.
7	Student enters using username and password	Get access to student dashboard	Login Successful Student dashboard

			displayed
8	Student tries to change password	Password changed.	Password changed and message displayed
9	Student submit complaint	Student can submit complaint and display it in a list	Student successfully submitted complaint, and it was displayed in the list
10	Admin views the complaint and replies	Admin can view the complaint and reply	Admin viewed the complaint and replied successfully
11	Student sends feedback	Student can send feedback	Student successfully registered feedback
12	Admin views feedback	Admin can view the feedback	Admin saw the feedback
13	Student view their profile	Student can view their profile	Student dashboard displayed the profile of the student
14	Student applies for candidacy	Student can apply for candidacy for a particular post	Student applied for candidacy and awaits admin's decision
15	Admin views the unrefined candidate list	Admin can view who are all applied for candidacy before accepting them	Admin viewed the list
16	Admin accept/reject the student's nomination	Admin can accept or reject the student	Admin successfully accepted the nomination
17	Admin views list of accepted candidates	Admin can view the list of candidates whose application was accepted	Admin viewed the list
18	Candidate enters using username and password	Get access to candidate dashboard	Successfully logged in
19	Candidate initiates campaign	Candidate can initiate campaign and view it.	Candidate successfully initiated campaign, and viewed it
20	Admin and students view campaigns	Admin and students can view campaigns of candidates	Admin and students could view campaigns of candidates successfully
21	Students views candidate list	Student can view the accepted candidate list	Students could view the list
22	Students and candidates cast their	Students and candidates	Student and candidates

	vote.	cast their vote and verified by OTP	did cast their vote only once and verified by OTP
23	Result display	Result will be displayed in all 3 dashboards	Result was shown on all 3 dashboards

Table 13: Test Cases

7. IMPLEMENTATION

Implementation Overview

Implementation is the process of transforming the system's design into a fully functional and efficient product that meets technical requirements and user expectations. It involves writing robust software components, setting up secure databases, and integrating blockchain functionality to ensure a **tamper-proof and decentralized voting system**. The process ensures seamless interaction between students, candidates, and administrators while maintaining security, transparency, and ease of use.

Implementation Details

- **Flask with MySQL**: We implemented the backend using **Flask**, following modular coding practices to ensure **scalability and maintainability**. The system interacts with a **MySQL database** to store user credentials, voting data, and election results.
- **Blockchain Integration**: TrueVoice leverages **Ethereum blockchain** to record votes in a **decentralized and immutable** manner. We utilized **Ganache** for a local blockchain testing environment and **Web3.py** for smart contract interactions.
- **Database Creation**: The **MySQL database** was structured using the defined schema, ensuring **data integrity, optimized query performance, and scalability**.
- **Front-End Development**: The **user interface** was developed using **HTML, CSS (Bootstrap), and JavaScript**, ensuring a **responsive and user-friendly experience**.
- **Testing**: The system underwent **rigorous testing**, including unit, integration, and blockchain transaction validation, to ensure **accuracy, security, and efficiency**.

Implementation Plan

The implementation of **TrueVoice** followed a structured approach:

1. **Core Development**: We began by developing essential functionalities such as **user authentication, candidate application, and election setup**.
2. **Database Setup**: A **relational database (MySQL)** was designed and integrated with **Flask** to store user data securely.
3. **Blockchain Smart Contracts**: Solidity-based smart contracts were deployed using **Ganache**, ensuring secure vote storage.
4. **Testing & Debugging**: The system was tested using **operational data**, focusing on **security vulnerabilities, performance bottlenecks, and blockchain transaction validation**.

5. **Deployment**: After testing, the system was deployed in a production environment, ensuring smooth operation.
6. **User Training & Documentation**: Detailed **user manuals and training materials** were provided to administrators and students to facilitate easy adoption.

By following this well-structured **implementation plan**, **TrueVoice** delivers a **secure, transparent, and scalable** election system, ensuring a **tamper-proof and democratic** voting experience for colleges.

8. SCREENSHOTS

Home Page

Figure 10: Home Page

Login Page

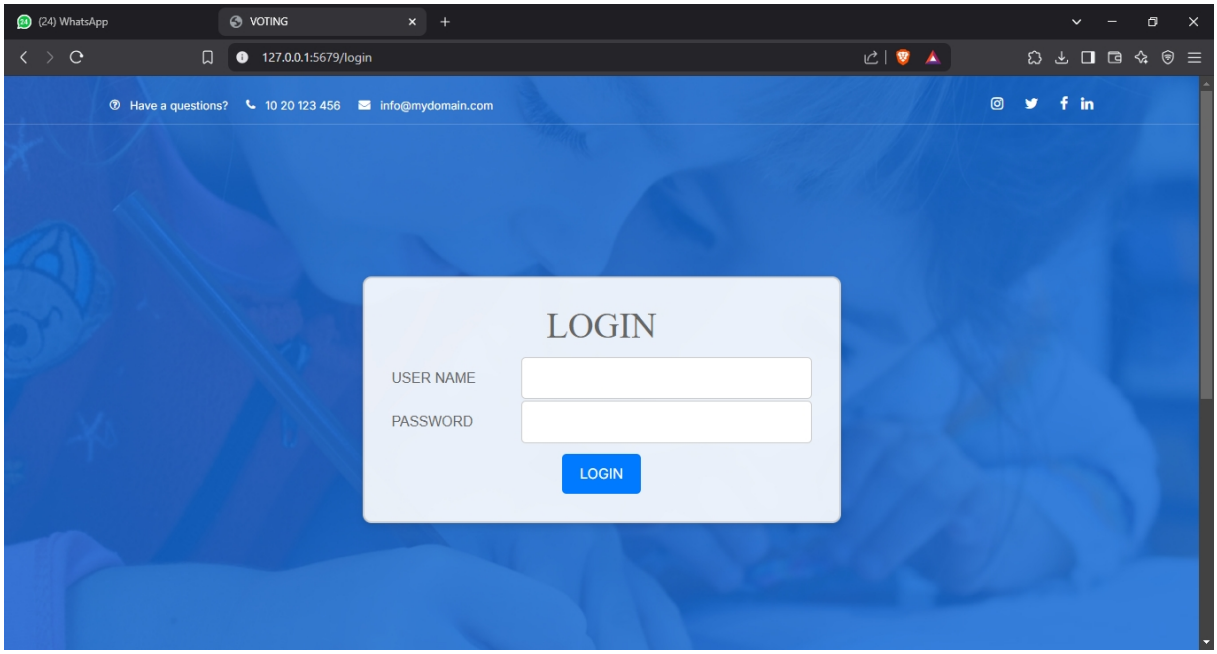


Figure 11: Login Page

Admin Dashboard

Figure 12: Admin Dashboard

Change Password

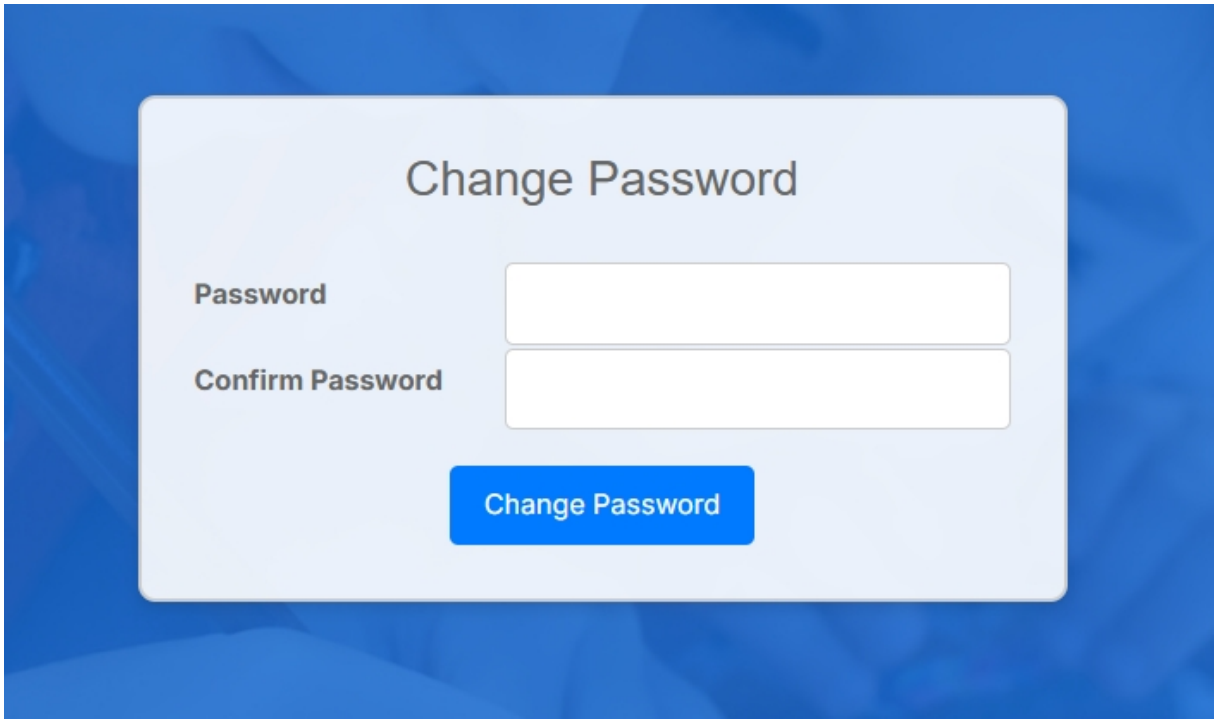


Figure 13: Change Password

Course Management

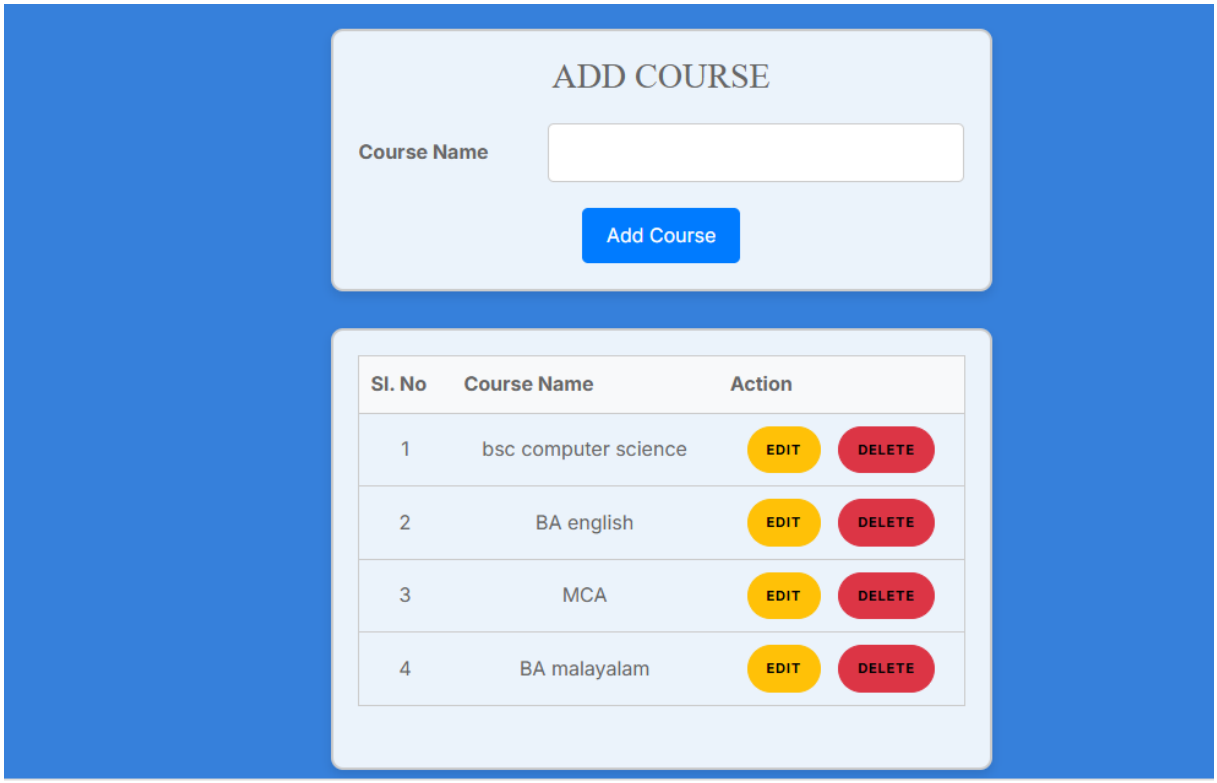


Figure 14: Course Management

Student Registration

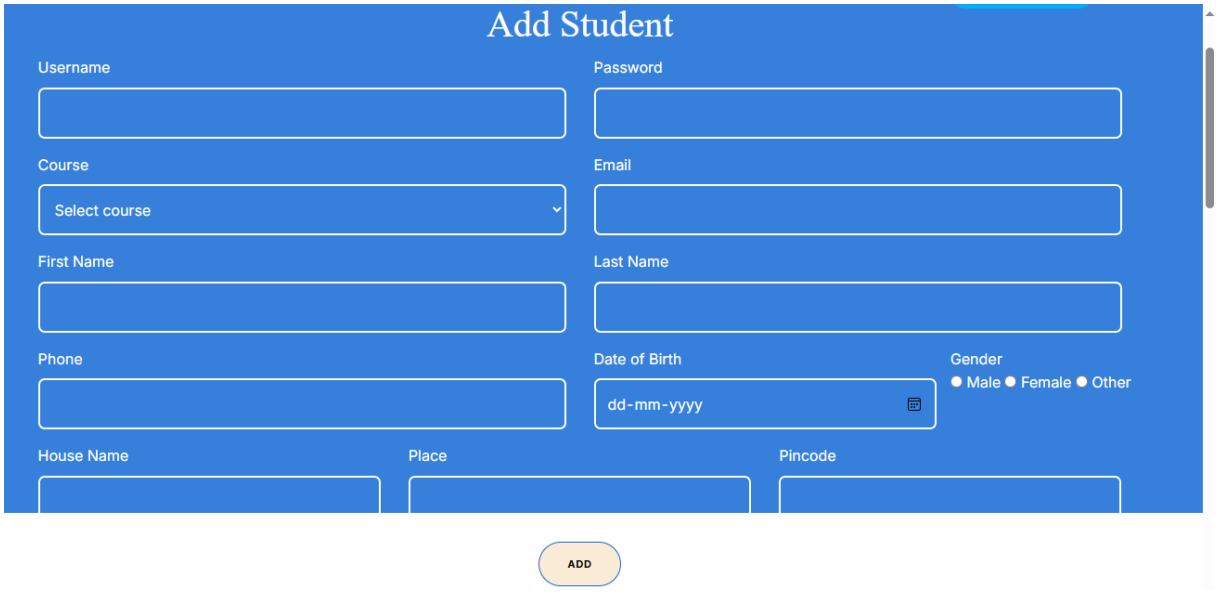


Figure 15: Student Registration

Student Management

STUDENT'S DETAILS

First Name	Last Name	Email	Phone	Gender	DOB	House Name	Place	Pincode	Action
amal	A	shahanavs786@gmail.com	9074331344	male	2002-02-12	house	kaloor	688524	EDIT DELETE
abi	a	shahanavs786@gmail.com	1234567890	male	2002-03-12	ho	kaloor	688524	EDIT DELETE
sree	vs	shahanavs786@gmail.com	9074332345	female	2000-02-15	ALAPPUZHA	cherthala	688524	EDIT DELETE
kiran	kiran	shahanavs786@gmail.com	9123433199	male	2000-03-12	HOUSE	cherthala	688524	EDIT DELETE

Figure 16: Student Management

Register Election

Figure 17: Register Election

Manage Election

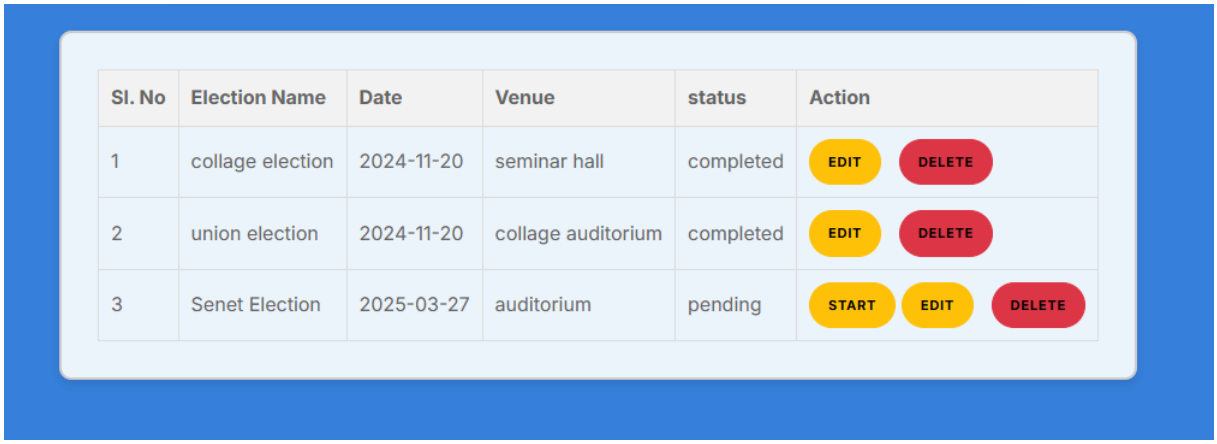


Figure 18: Manage Election

Register Post

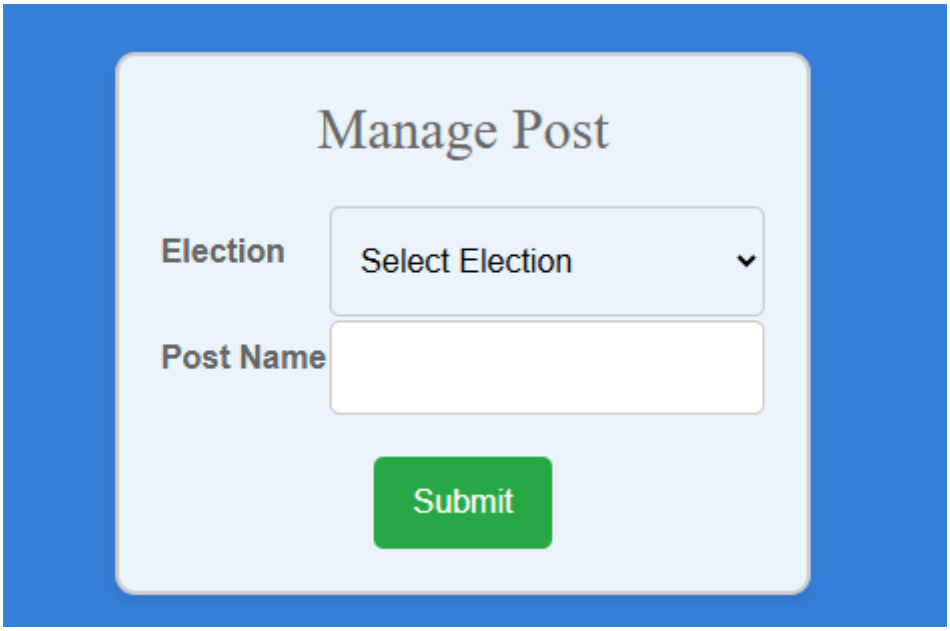
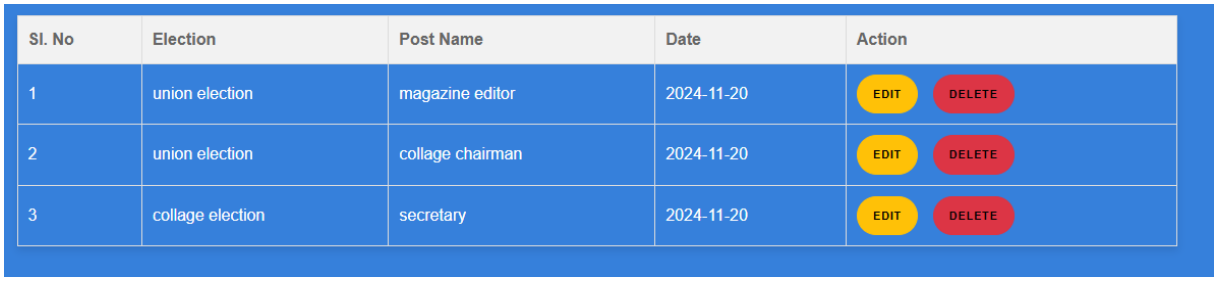


Figure 19: Register Post

Post Management



Sl. No	Election	Post Name	Date	Action
1	union election	magazine editor	2024-11-20	EDIT DELETE
2	union election	collage chairman	2024-11-20	EDIT DELETE
3	collage election	secretary	2024-11-20	EDIT DELETE

Figure 20: Post Management

Student Dashboard

Figure 21: Student Dashboard

Send Feedback

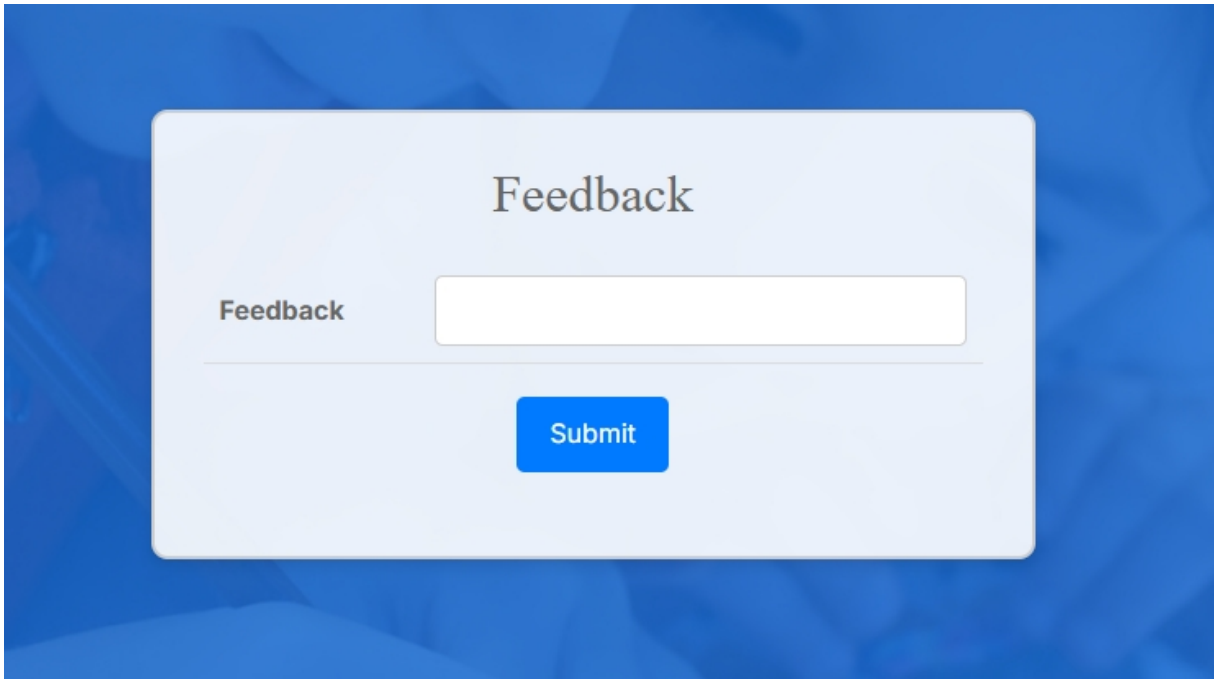


Figure 22: Send Feedback

Register Complaint

Figure 23: Register Complaint

Admin View Feedback

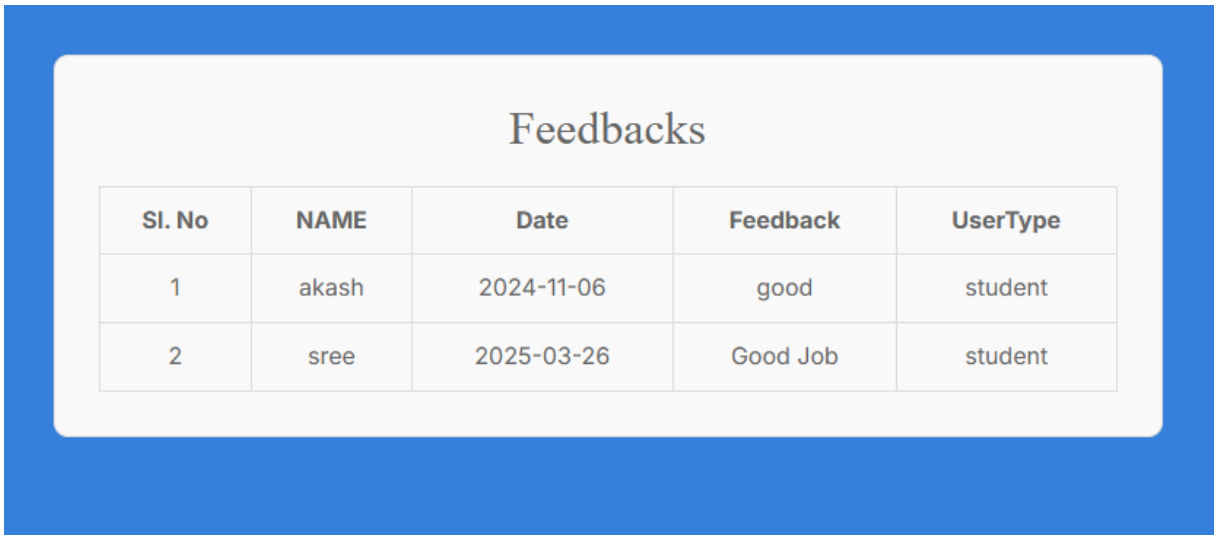


Figure 24: Admin View Feedback

Admin View Complaint

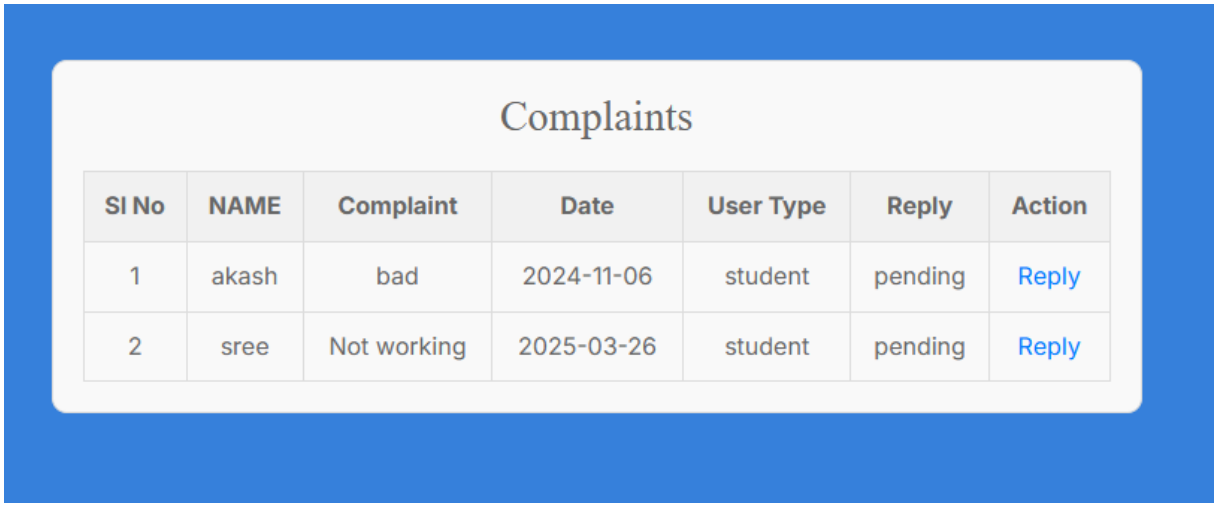


Figure 25: Admin View Complaint

Admin Replying To Complaint

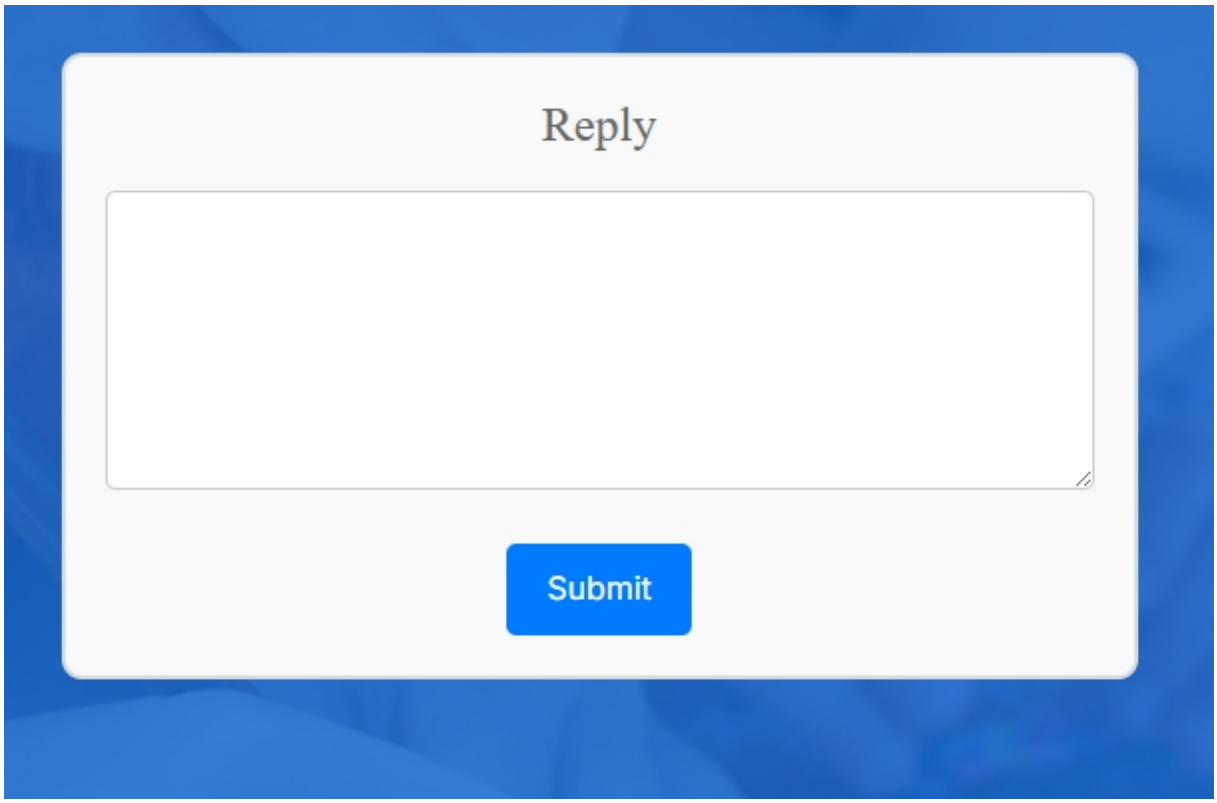


Figure 26: Admin Replying to Complaint

Student Apply as Candidate

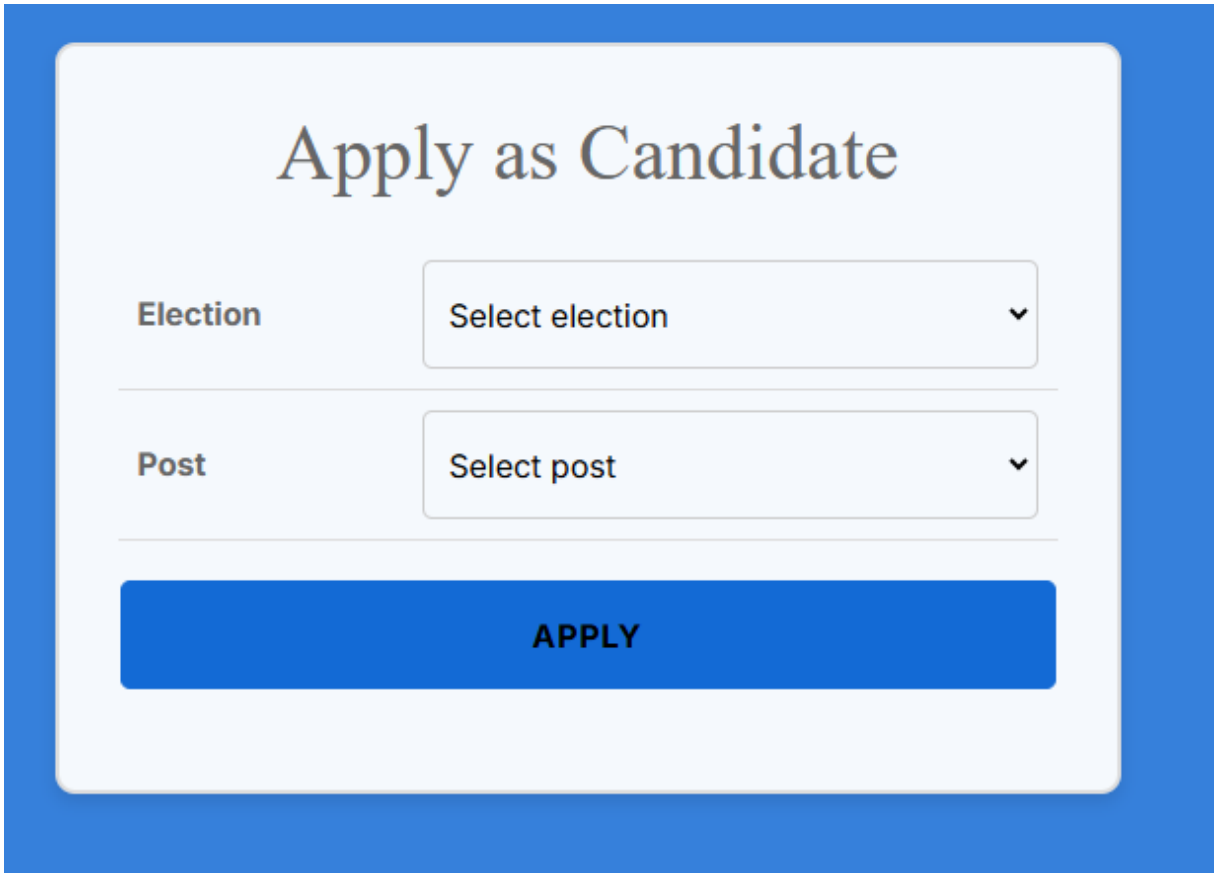


Figure 27: Student Apply as Candidate

List of Applied Candidates

Figure 28: List of Applied Candidates

List of Approved Candidates

Approved Candidates

Sl No	Candidate Name	Course Name	Election Name	Post Name	Status
1	diya	MCA	union election	collage chairman	approved
2	aswin	BA english	collage election	secretary	approved
3	naveen	BA malayalam	collage election	secretary	approved
4	fathima	BA english	union election	collage chairman	approved
5	abi	bsc computer science	union election	magazine editor	approved
6	amal	bsc computer science	union election	magazine editor	approved
7	sree	BA english	Senet Election	Chairman	approved

Figure 29: List of Approved Candidate

Student's Profile

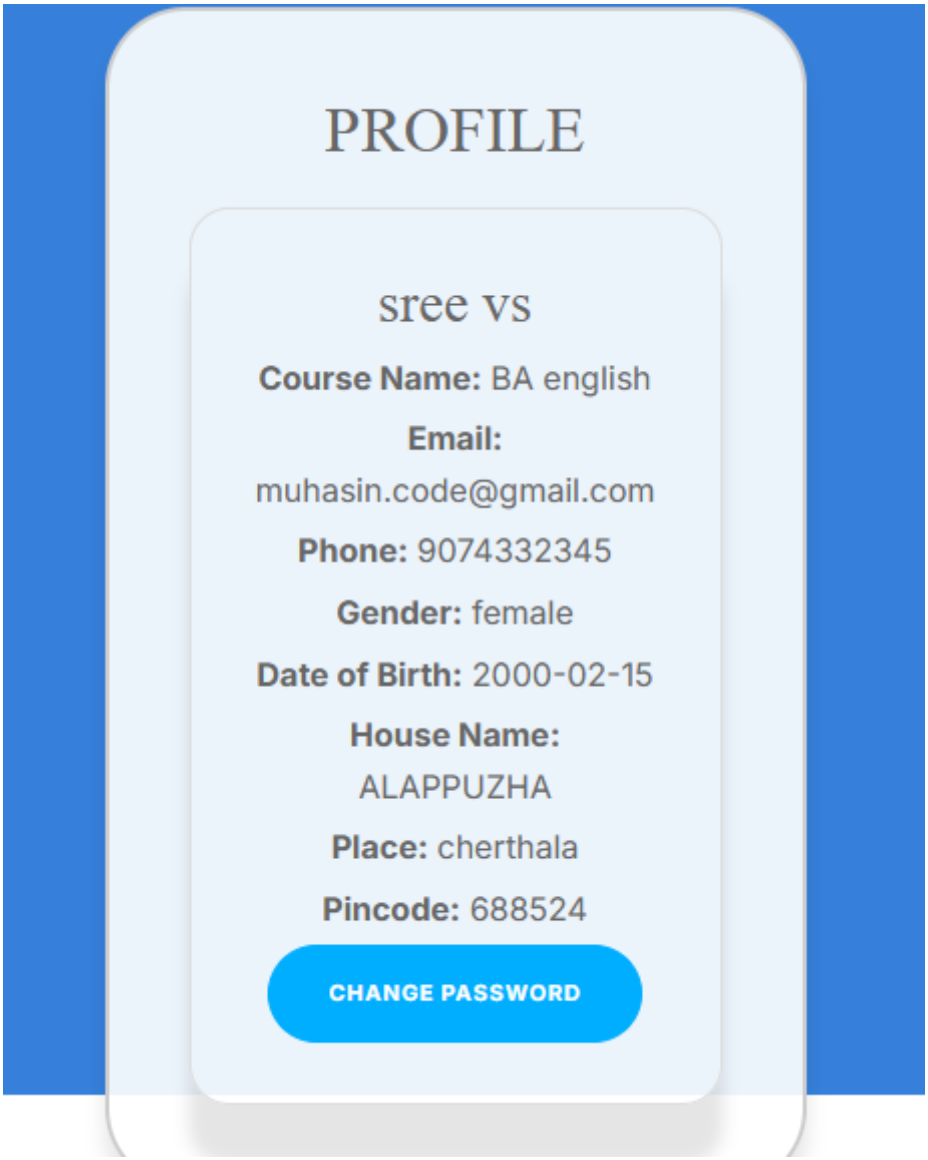


Figure 30: Student's Profile

Campaign Management

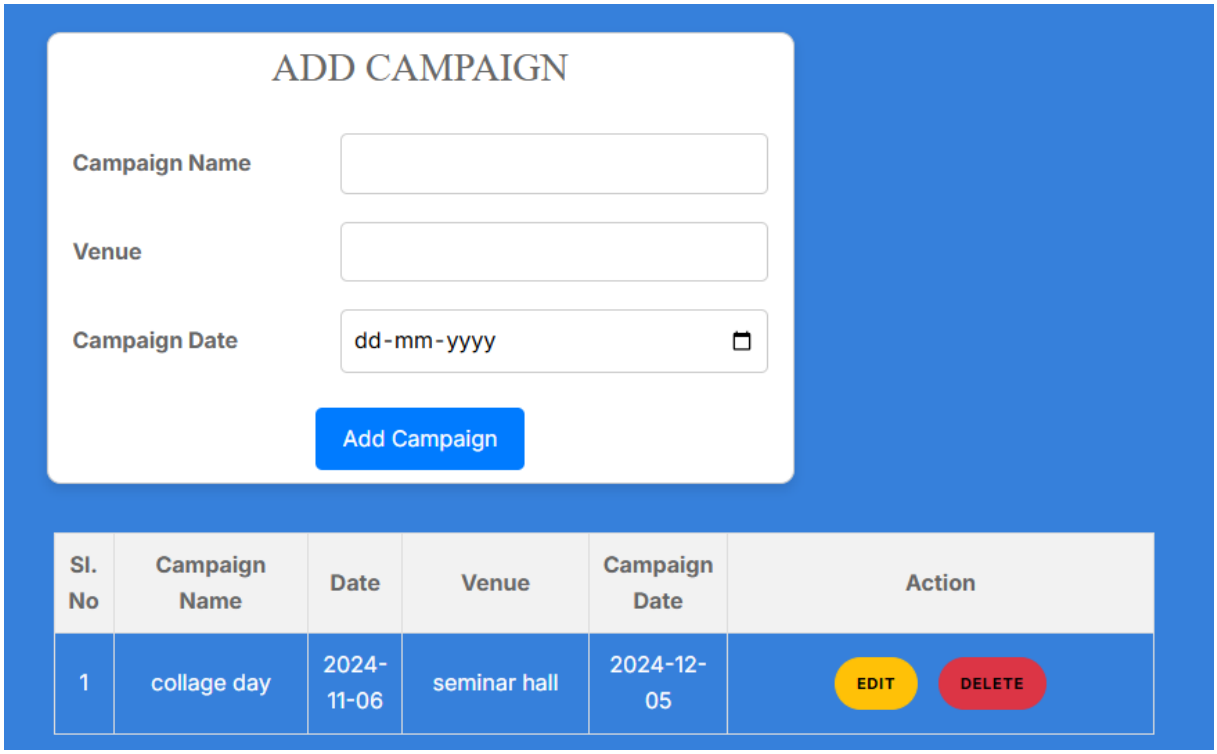


Figure 31: Campaign Management

Election Dashboard

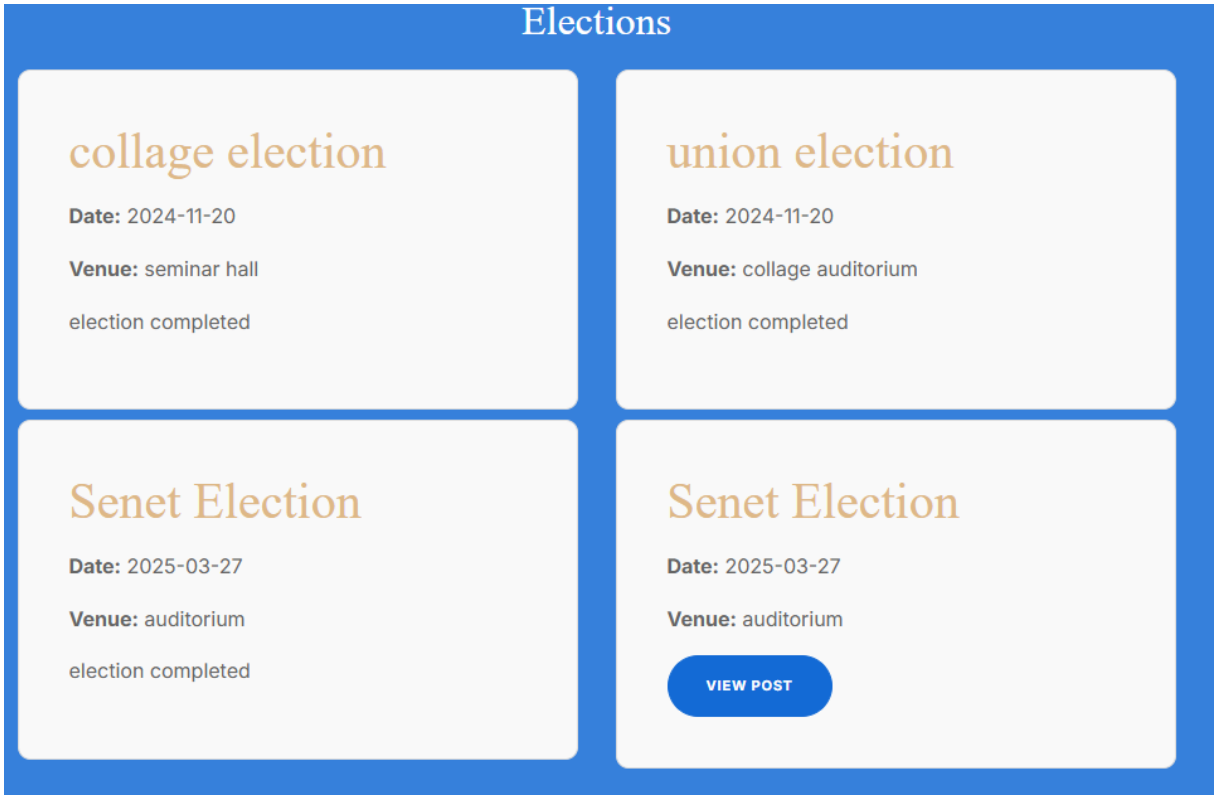


Figure 32: Election Dashboard

Voting

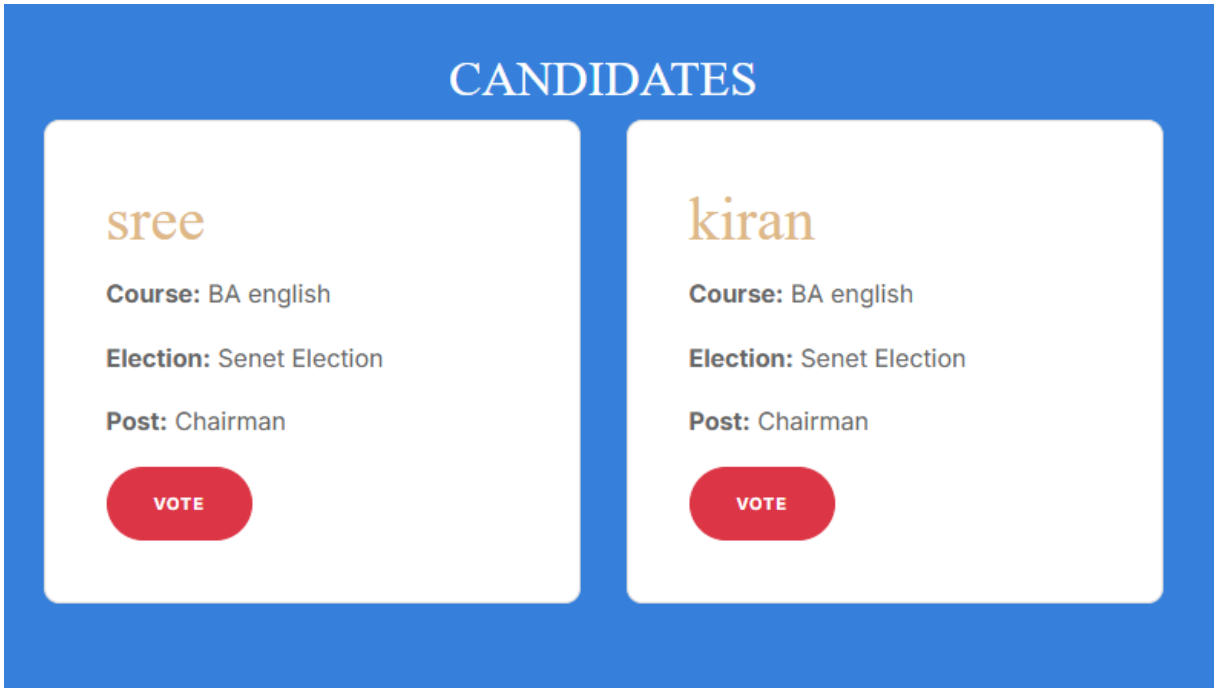


Figure 33: Voting

OTP Verification

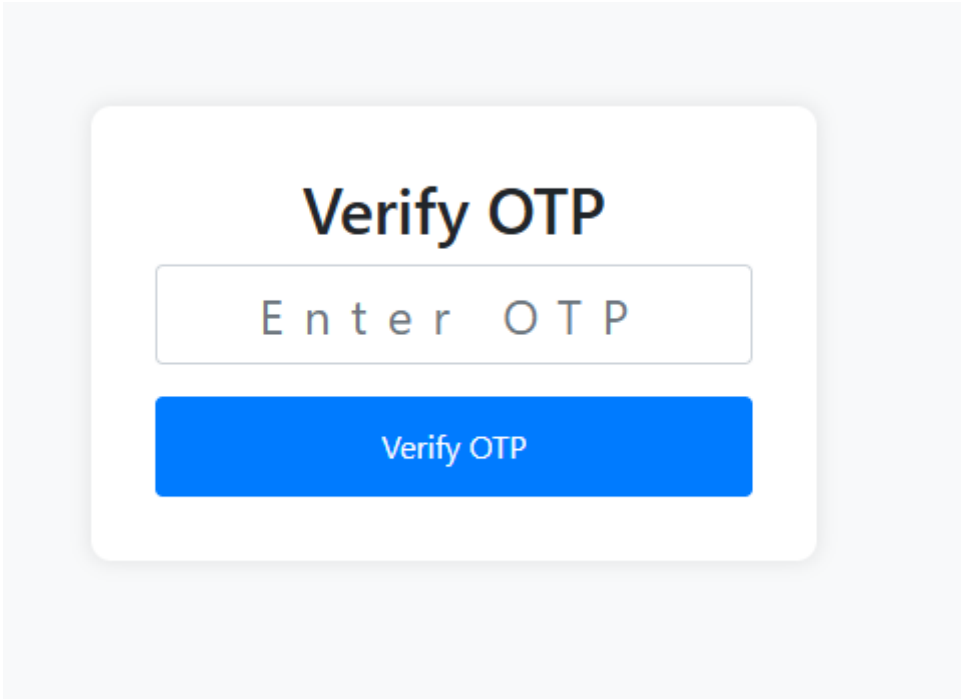


Figure 34: OTP Verification

Result Dashboard

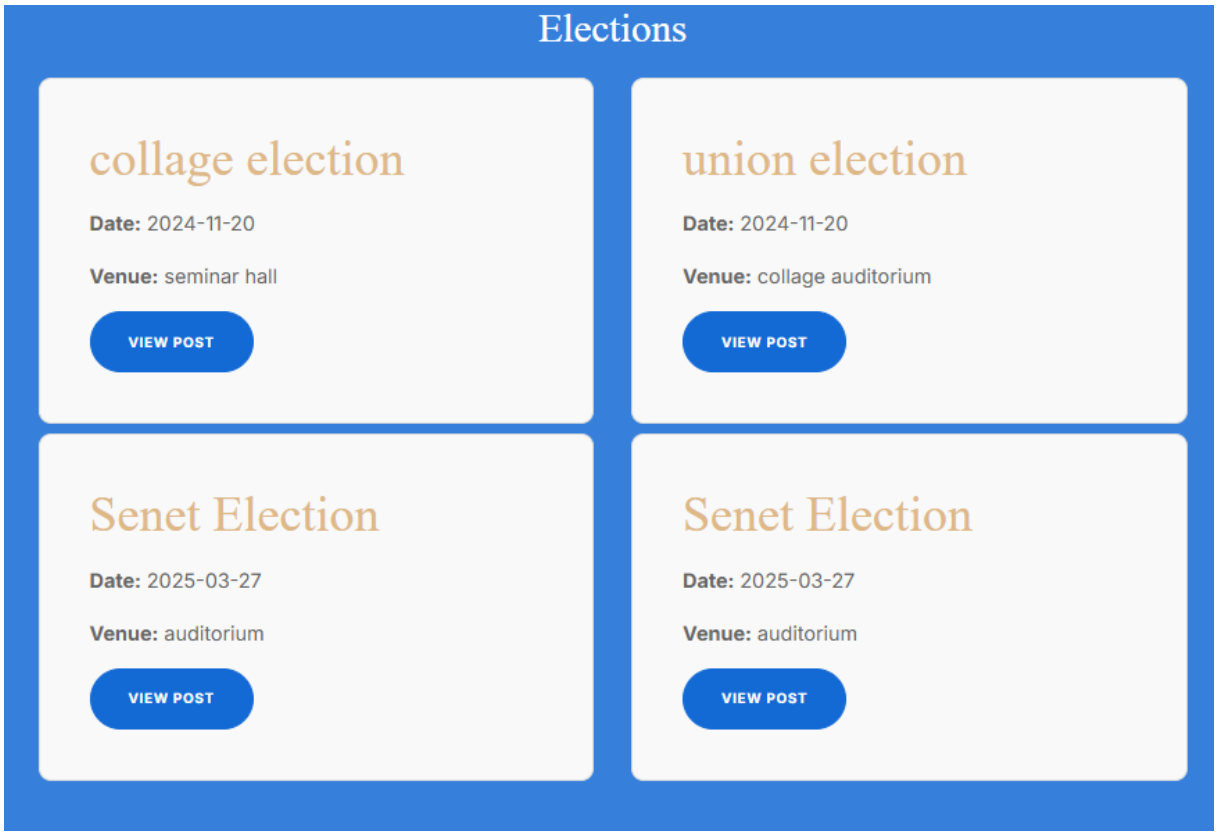


Figure 35: Result Dashboard

Post Wise Result

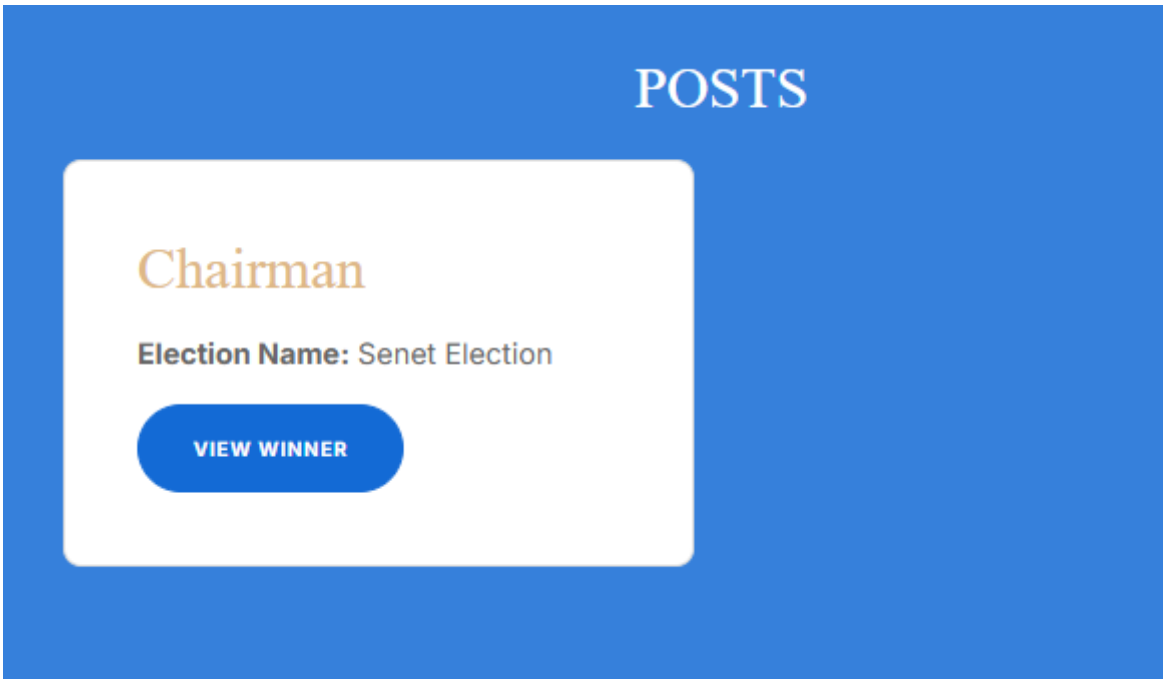


Figure 36: Post Wise Result

Election Result

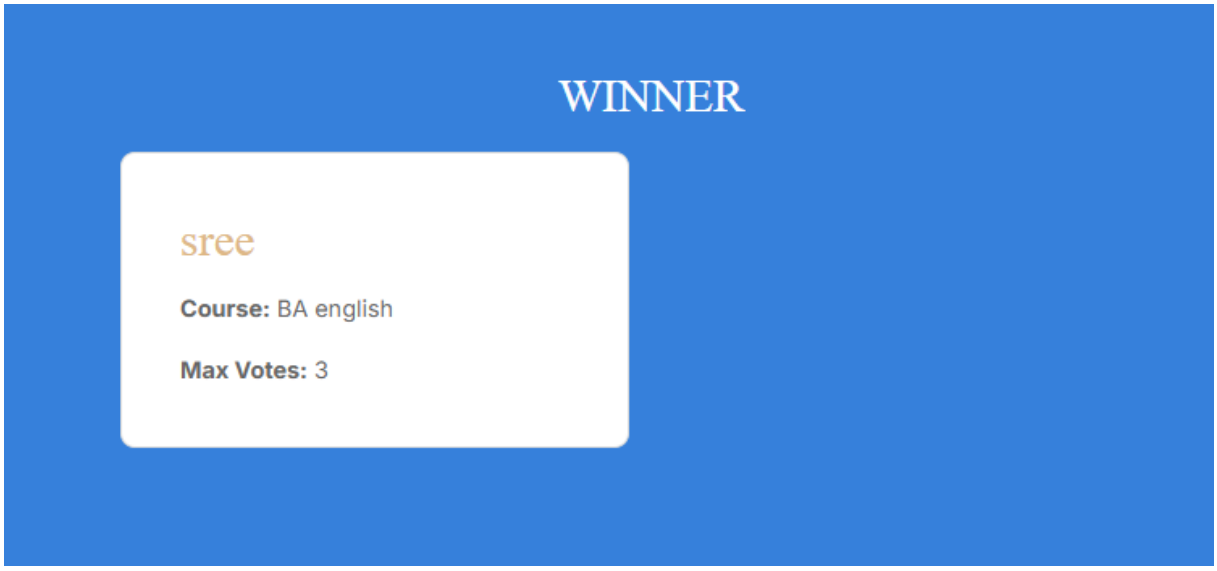


Figure 37: Election Result

9. CONCLUSION

TrueVoice has successfully established a secure, transparent, and efficient platform for conducting college elections using blockchain technology. By leveraging decentralized data storage, cryptographic security, and a tamper-proof voting mechanism, the system ensures fair and fraud-free elections while maintaining voter anonymity. The integration of features such as **email OTP verification, phone IMEI authentication, and live face recognition** further strengthens the security of the election process, ensuring only eligible students can cast their votes.

Our system streamlines the traditional voting process by eliminating manual paperwork and reducing administrative workload. Through a **user-friendly interface**, students can easily apply for candidacy, cast their votes, and track election results in real time, fostering greater participation and engagement. The **admin panel** allows election officials to manage candidates, oversee the election process, and verify results efficiently. This structured and automated approach enhances the overall election experience, making it **more reliable, accessible, and transparent**.

Beyond just conducting elections, TrueVoice contributes to building a **trustworthy and inclusive electoral environment** where every vote matters. The use of blockchain technology eliminates concerns over vote manipulation, ensuring that election results are **immutable and verifiable**. The system not only strengthens digital trust but also cultivates democratic values among students by promoting **fair competition and active participation**.

As we look ahead, TrueVoice is set to evolve further by integrating additional security features, expanding its accessibility through mobile platforms, and optimizing performance to handle larger-scale elections. With continuous improvements in both **blockchain efficiency and user experience**, the platform aims to revolutionize digital voting in educational institutions. By setting a new benchmark in election integrity and accessibility, TrueVoice stands as a **cutting-edge, future-ready solution** for secure and transparent college elections.

10. FUTURE ENHANCEMENTS

As **TrueVoice** continues to evolve, we are committed to **enhancing** and **expanding** the platform to provide an even **more secure, efficient, and user-friendly election system**. Some of the planned future enhancements include:

Security and Authentication Enhancements

- **Biometric Authentication**: Implementing **fingerprint or facial recognition** to further secure the voting process.
- **Blockchain Enhancements**: Exploring more **advanced consensus mechanisms** and **multi-chain interoperability** to enhance performance and security.

User Experience Improvements

- **Enhanced UI/UX**: Refining the platform's interface to provide a **seamless and intuitive** voting experience.
- **Mobile App Integration**: Expanding TrueVoice with a **mobile-friendly** version for convenient access across devices.

Election Features Expansion

- **Live Vote Tracking**: Allowing real-time **vote count visualization** while maintaining voter anonymity.
- **Multi-Election Support**: Enabling colleges to conduct **multiple elections simultaneously** for different student organizations.

Advanced Analytics & Reporting

- **Voter Participation Insights**: Providing election authorities with **real-time statistics** on voter turnout and engagement.
- **Detailed Post-Election Reports**: Generating **comprehensive audit reports** to ensure full transparency and legitimacy.

Community & Administration Features

- **Candidate Profiles & Campaigning**: Introducing **candidate manifestos, debates, and Q&A sessions** within the platform.
- **AI-Powered Anomaly Detection**: Using **AI-driven analysis** to detect any **suspicious voting patterns** and prevent potential manipulation.

These enhancements will **strengthen TrueVoice's role as a pioneering digital election platform**, ensuring **fair, secure, and transparent** voting for student communities. We remain

committed to **continuous improvement**, adapting to the latest **technological advancements** and **user needs** to make college elections **simpler, safer, and more reliable**.

11. BIBLIOGRAPHY

11.1 TEXT REFERENCES

1. S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.
2. M. Swan, *Blockchain: Blueprint for a New Economy*, 1st ed., O'Reilly Media, 2015.
3. A. Bahga and V. Madisetti, *Blockchain Applications: A Hands-On Approach*, 1st ed., VPT, 2017.
4. A. Antonopoulos, *Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications*, 2nd ed., O'Reilly Media, 2020.
5. A. Silberschatz, H. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed., McGraw Hill, 2021.
6. R. Mall, *Fundamentals of Software Engineering*, 5th ed., Prentice Hall of India Private Limited, 2018.

11.2 ONLINE REFERENCES

1. Ethereum, Official Documentation. https://ethereum.org/en/developers/docs/
2. Ganache, Official Documentation. https://trufflesuite.com/ganache/
3. Flask, Official Documentation. https://flask.palletsprojects.com/en/2.0.x/
4. MySQL, Official Documentation. https://dev.mysql.com/doc/
5. W3C, Web Accessibility Guidelines. https://www.w3.org/WAI/standards-guidelines/

12. APPENDIX

12.1 GANTT CHART

A Gantt chart, invented by Henry Gantt, is a specialized bar chart that visually represents a project's schedule. It showcases the commencement and completion dates of both terminal and summary elements, which collectively form the project's work breakdown structure. This chart provides a clear and concise overview of the project's timeline, enabling efficient tracking and management of tasks.

Meeting Minute

Date :	Jan 03,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

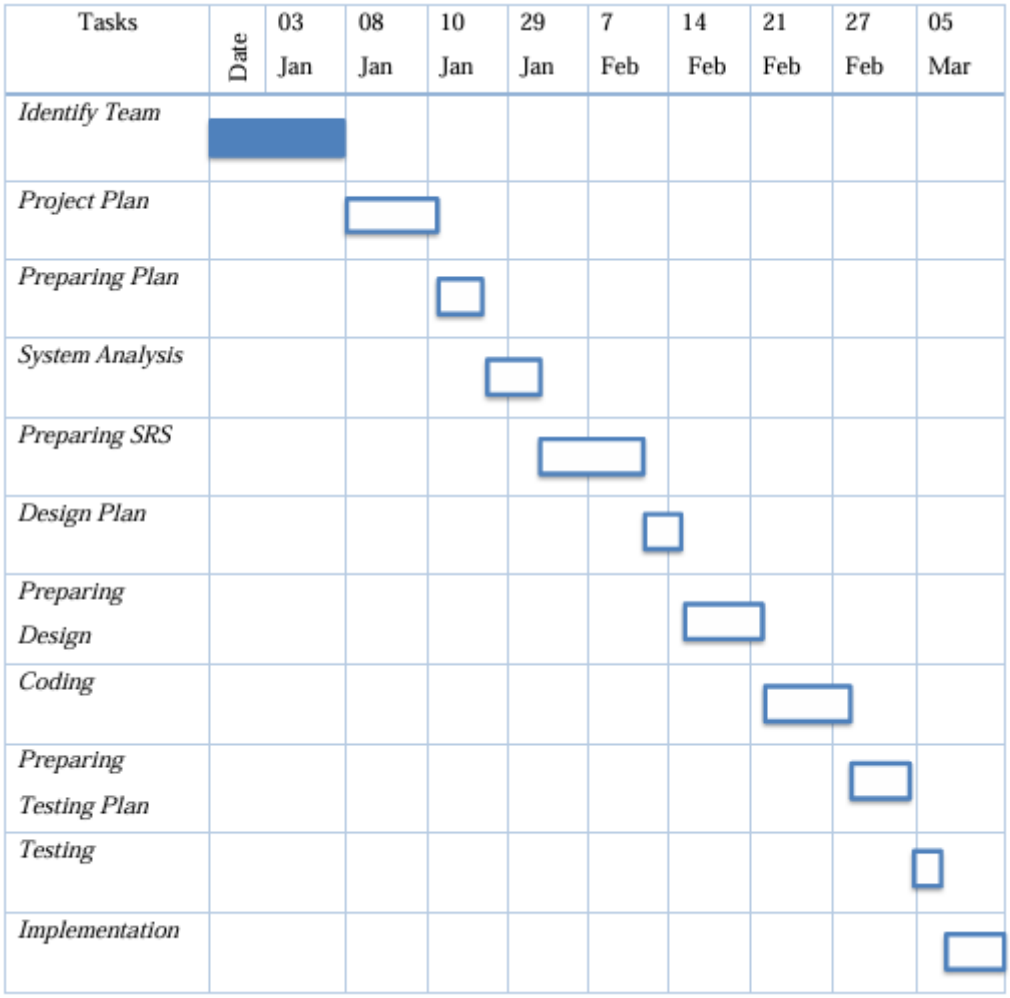


Figure : Jan 03, 2025

Meeting Minute

Date :	Jan 08,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

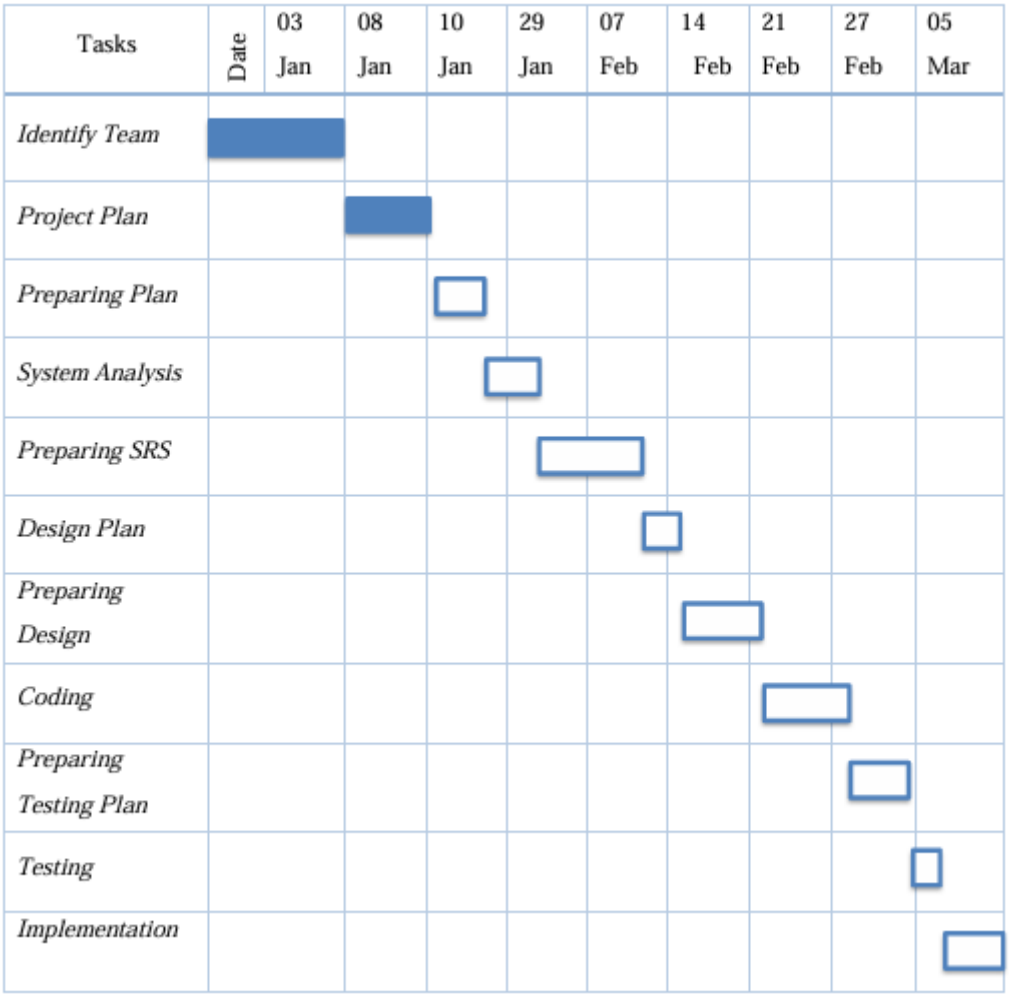


Figure : Jan 08, 2025

Meeting Minute

Date :	Jan 10,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

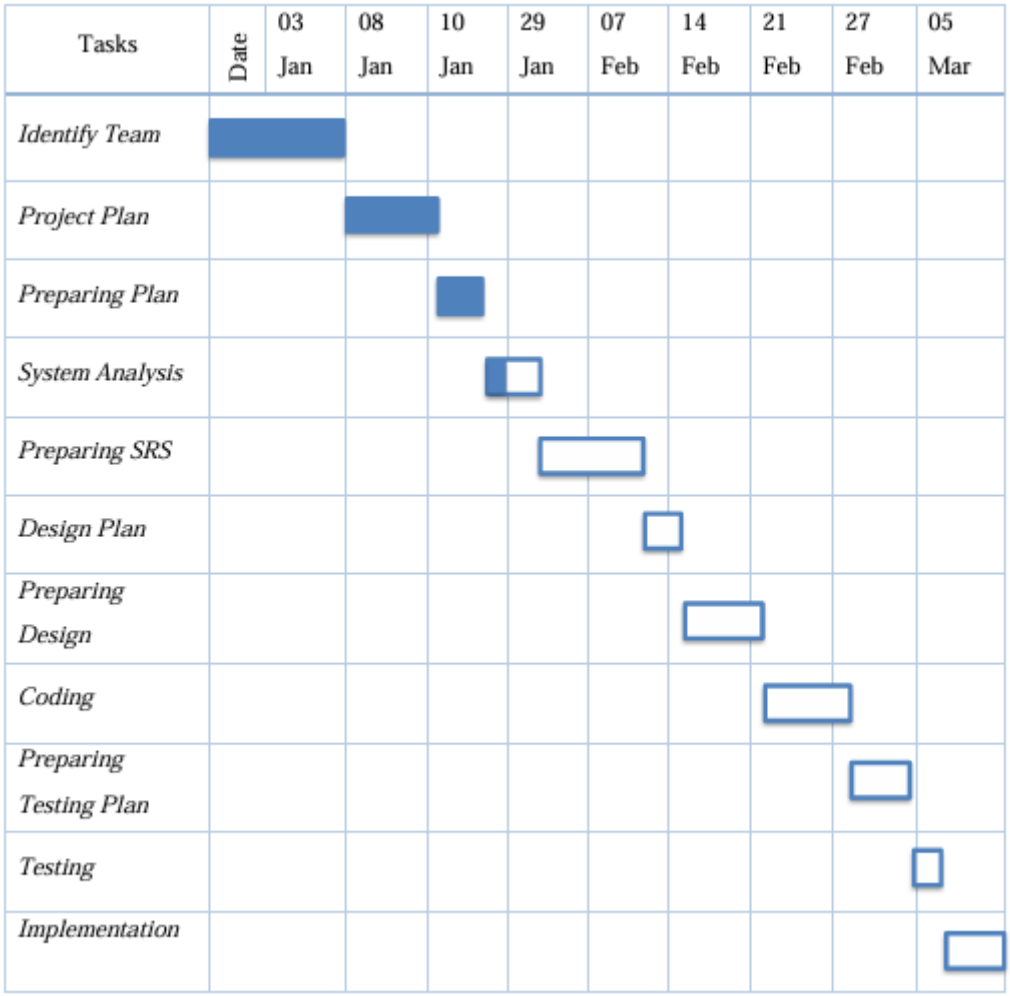


Figure : Jan 10, 2025

Meeting Minute

Date :	Jan 29,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

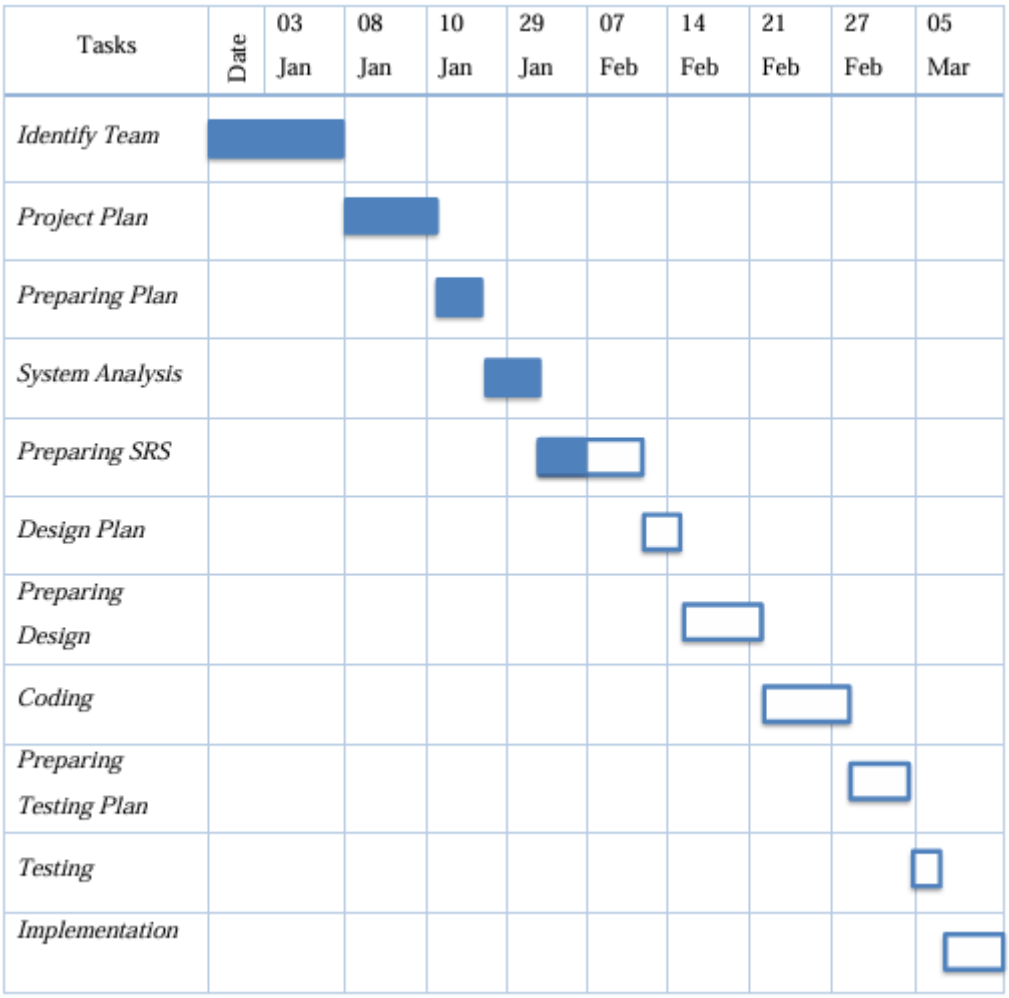


Figure : Jan 29, 2025

Meeting Minute

Date :	Feb 07,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

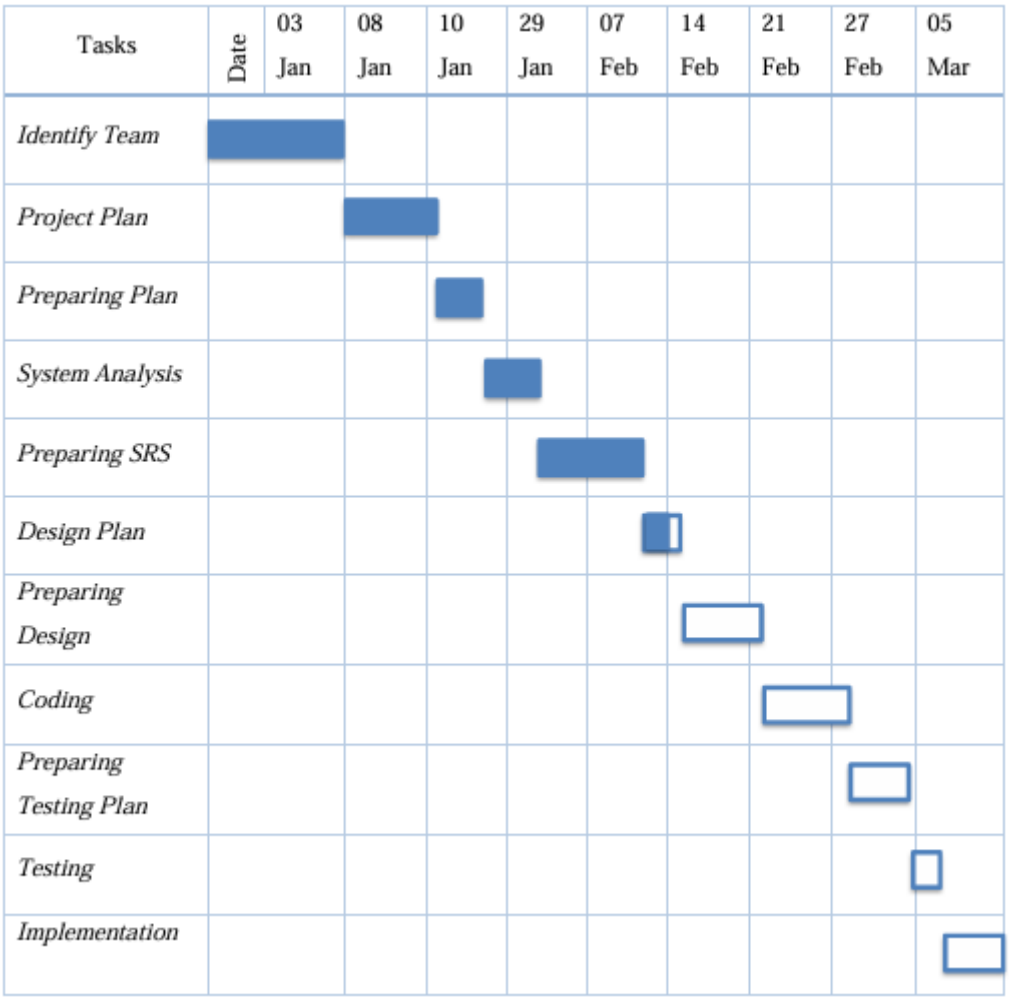


Figure : Feb 07, 2025

Meeting Minute

Date :	Feb 14,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

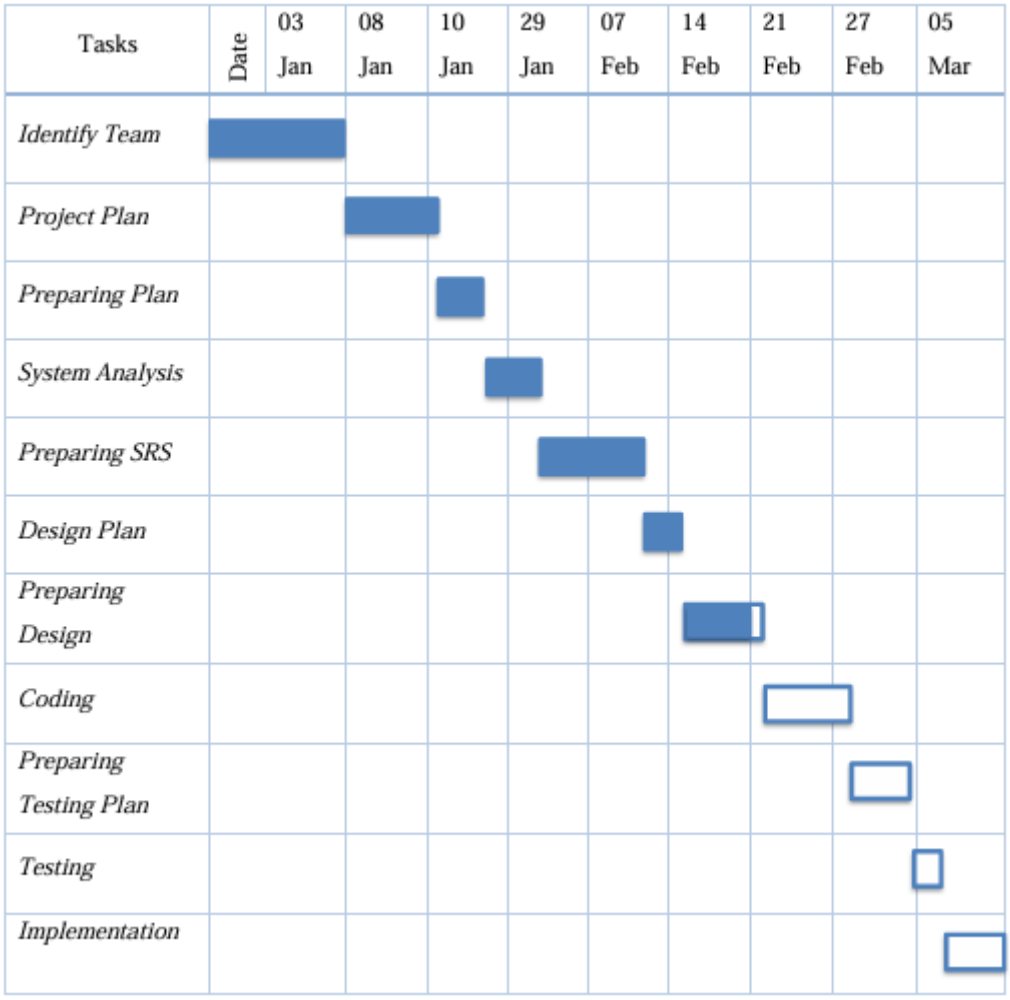


Figure : Feb 14, 2025

Meeting Minute

Date :	Feb 21,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

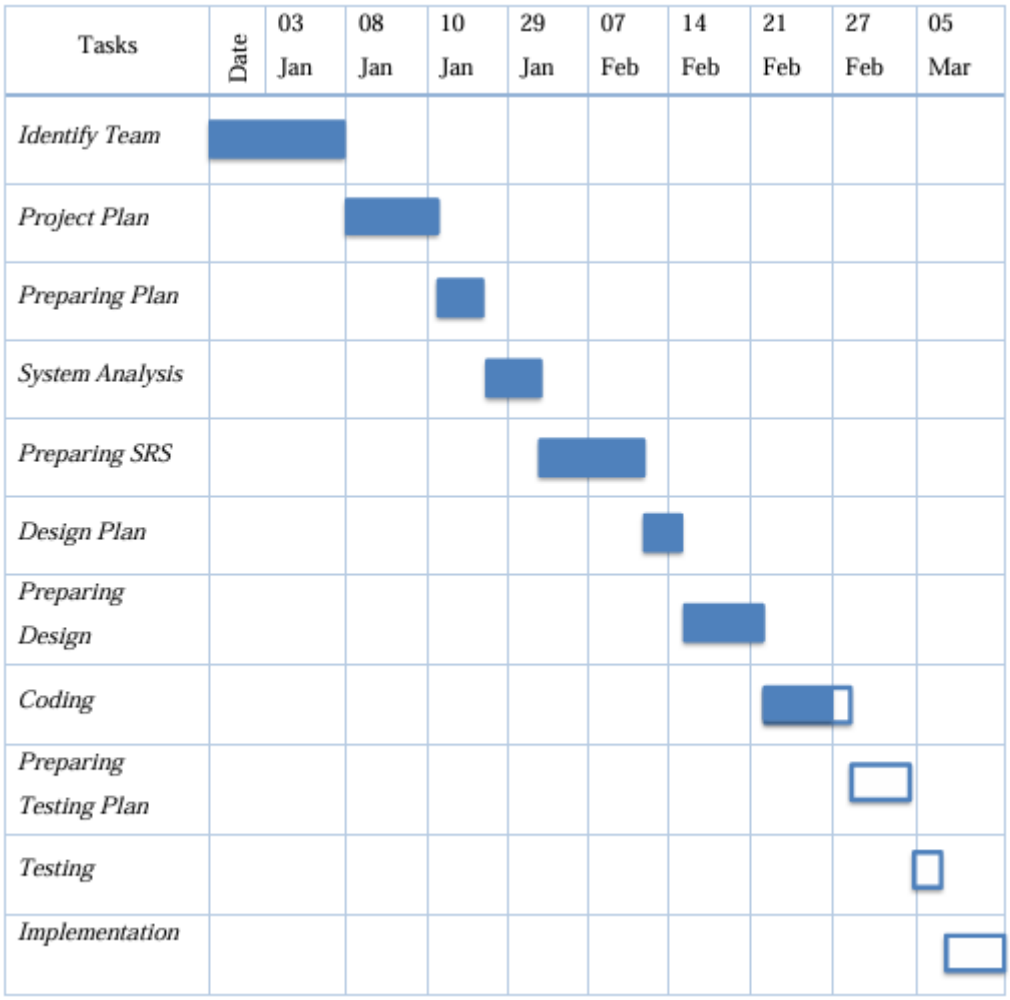


Figure : Feb 21, 2025

Meeting Minute

Date :	Feb 27,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

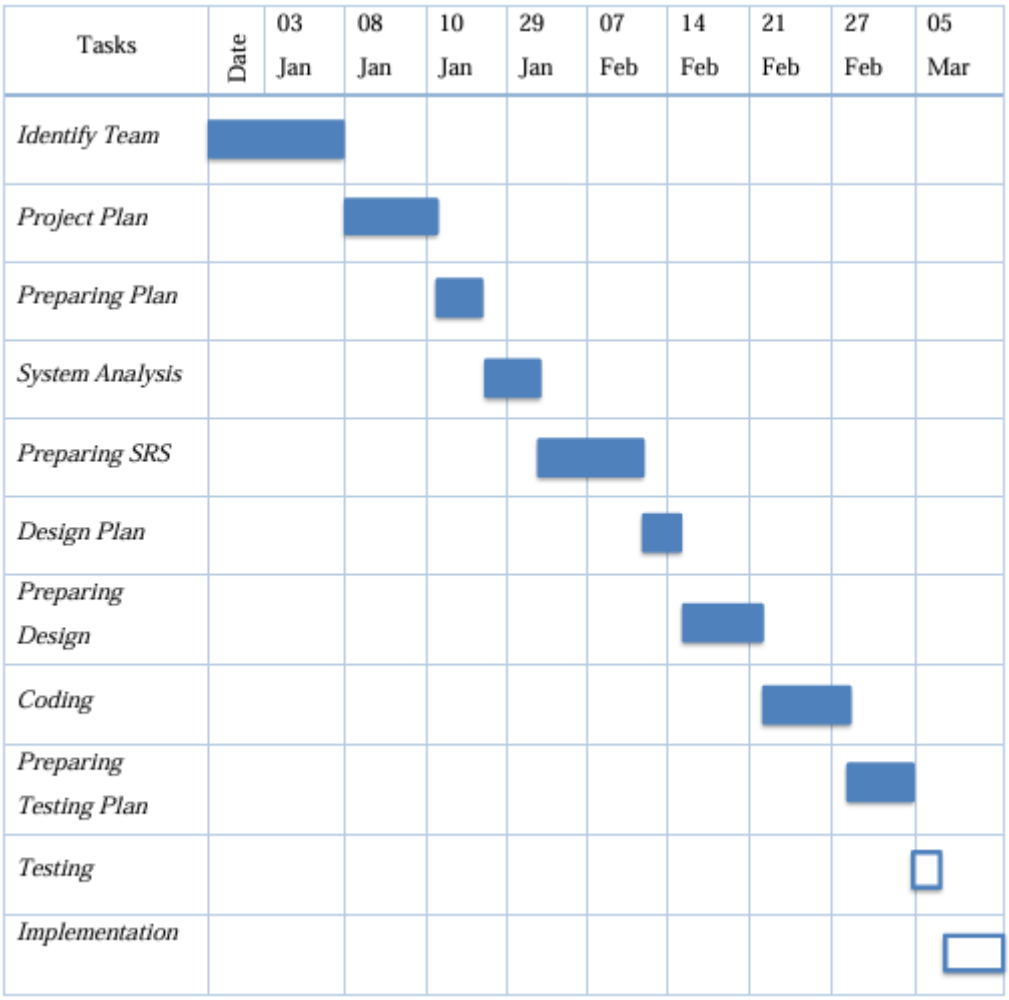


Figure : Feb 27,2025

Meeting Minute

Date :	Mar 05,2025
Time :	10:00 AM – 11.00 AM
Location :	Sree Narayana College, Cherthala
Topic :	TrueVoice

Participants

Guide Name :	Aswathy K
	Chinnu K Deepu
Students Name :	Adarsh B
	Akshay PT
	Goury Santhosh
	Muhammed Muhasin K

Completion of Assigned Task

Submitted to Department of Computer Science

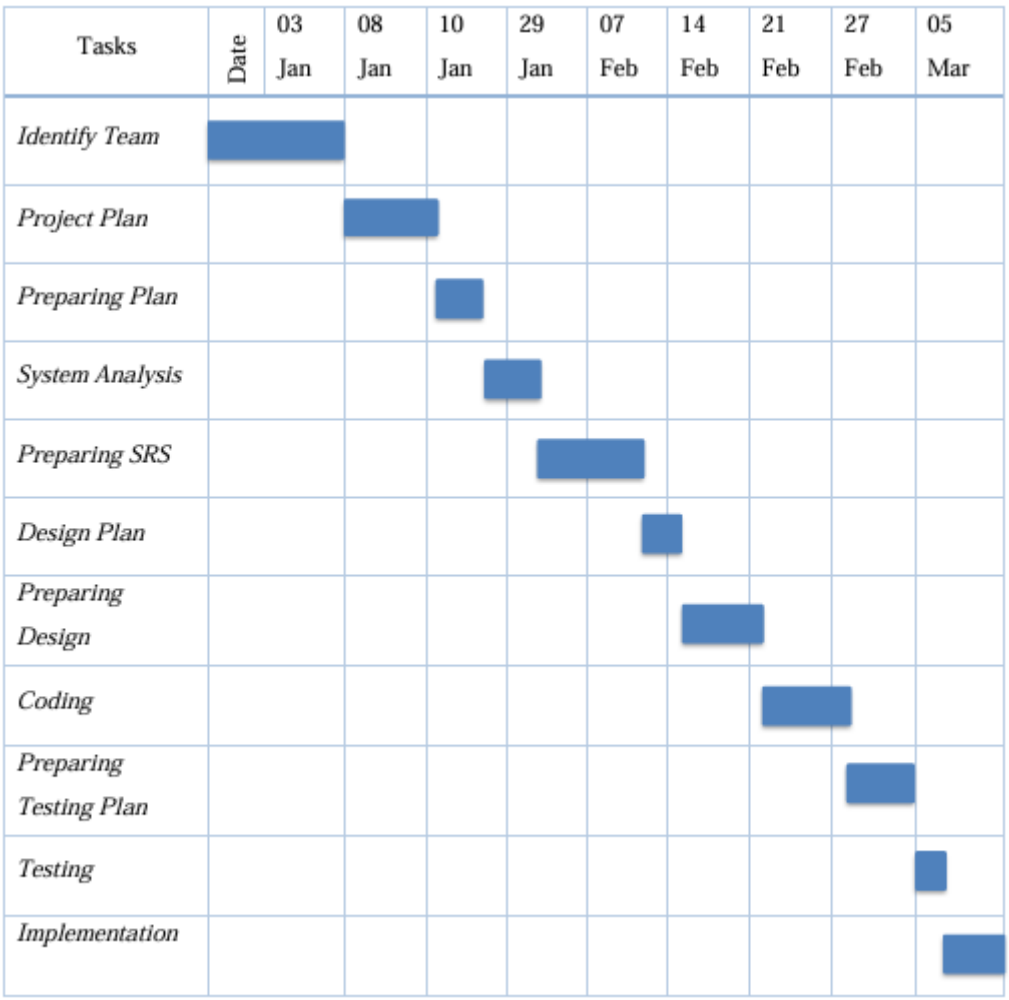


Figure : Mar 05, 2025