**Team 8**
**Assignment 3: Architecture Document Part 2**
**SWE 6653**
**Fall 2023**

*Team Members and Contributions:*
**Ryan:** Editor, Data Section, UI Section, Design Rationale, Architecture Section
**Charles:** System Section
**Sai:** Architecture Section
**Krishna:** 3 Main Scenarios

**Table of Contents**

## 1. Introduction

### 1.1 Purpose:

This document describes the architectural design of our Online Voting System (OVS), aiming at non-governmental polls such as surveys, private organizational votes, and other unofficial elections. The OVS is designed with a focus on user-friendliness, security, and transparency.

### 1.2 Scope and System Overview:

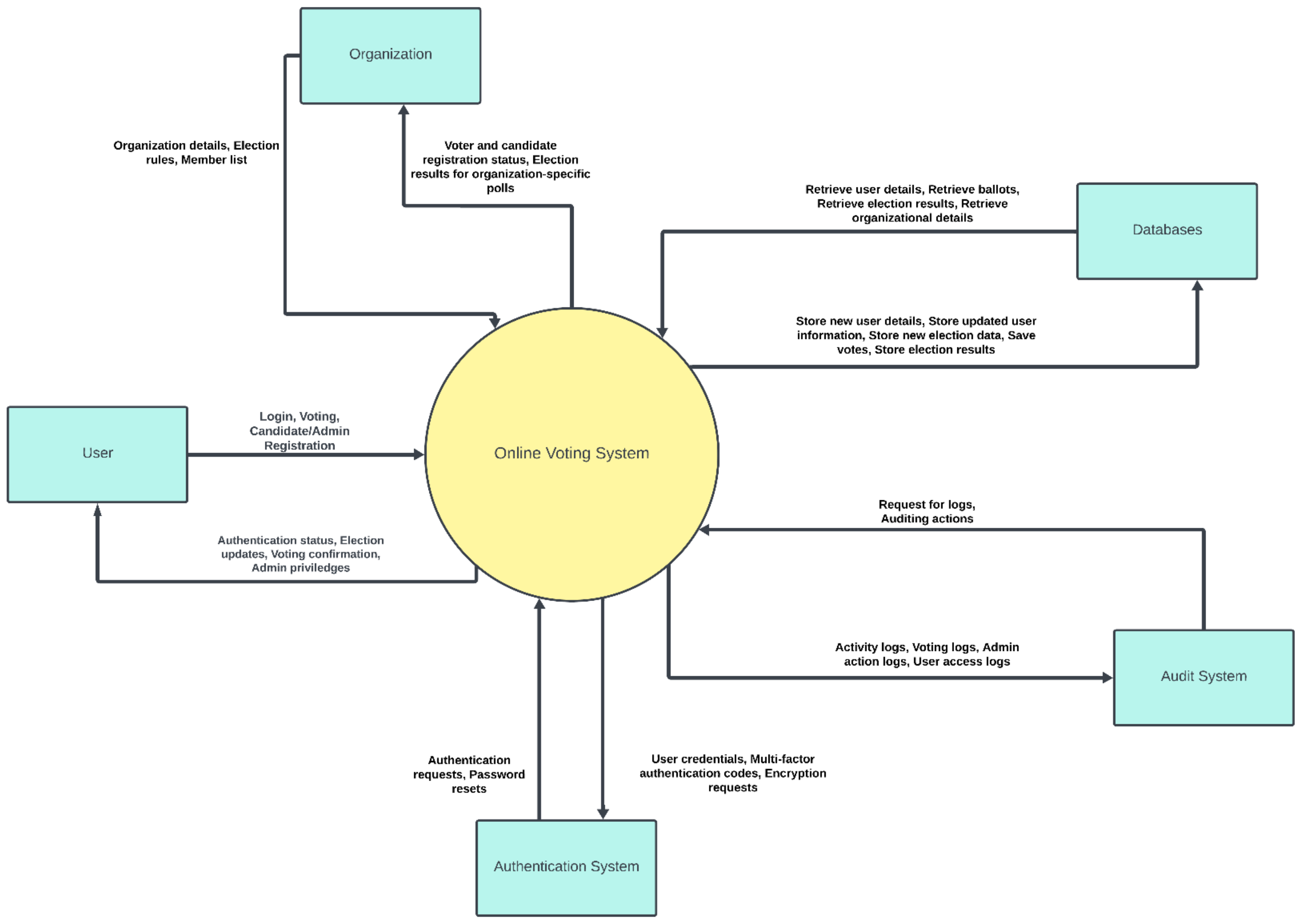
The platform is developed for non-official, remote voting. It offers key features like stringent user verification, accessibility, and a fortified interface armed with multi-factor authentication, encryption, and firewalls. For the sake of transparency, all user activities within the system are logged and sent to our auditing syst3em. Users initiate their journey by registering as a generic User. Once registered, they can either:

- Create an organization, granting them administrative privileges over it.
- Join an existing organization and register as a candidate for its elections.
- Simply join an organization as a regular user without pursuing an administrative or candidate role.

## 2. Domain Specification, E-Governance:

Previously, our system was aimed at public-sector e-governance, targeting national and local official elections. However, to navigate complex legal and logistical challenges, we've adjusted our aim to now focus on inter-organizational e-governance. Our system now focuses on unofficial voting processes like internal organizational polls, surveys, and other non-governmental elections.

## 3. Context Diagram


Organization
Organization details, Election rules, Member list
Voter and candidate registration status, Election results for organization-specific polls
Retrieve user details, Retrieve ballots, Retrieve election results, Retrieve organizational details
Databases
Store new user details, Store updated user information, Store new election data, Save votes, Store election results
User
Login, Voting, Candidate/Admin Registration
Online Voting System
Authentication status, Election updates, Voting confirmation, Admin priviledges
Request for logs, Auditing actions
Activity logs, Voting logs, Admin action logs, User access logs
Audit System
Authentication requests, Password resets
User credentials, Multi-factor authentication codes, Encryption requests
Authentication System

**4. Users and Actors:**

**User:**

This is a generic individual registered within the system.
Interactions include:

- Registering a new profile
- Editing their profile
- Verifying their identity
- Registering a new organization
- Joining an existing organization
- Receive results
- Voting in elections

**Admin:**

An admin has elevated privileges and can manage elections. Their tasks include:

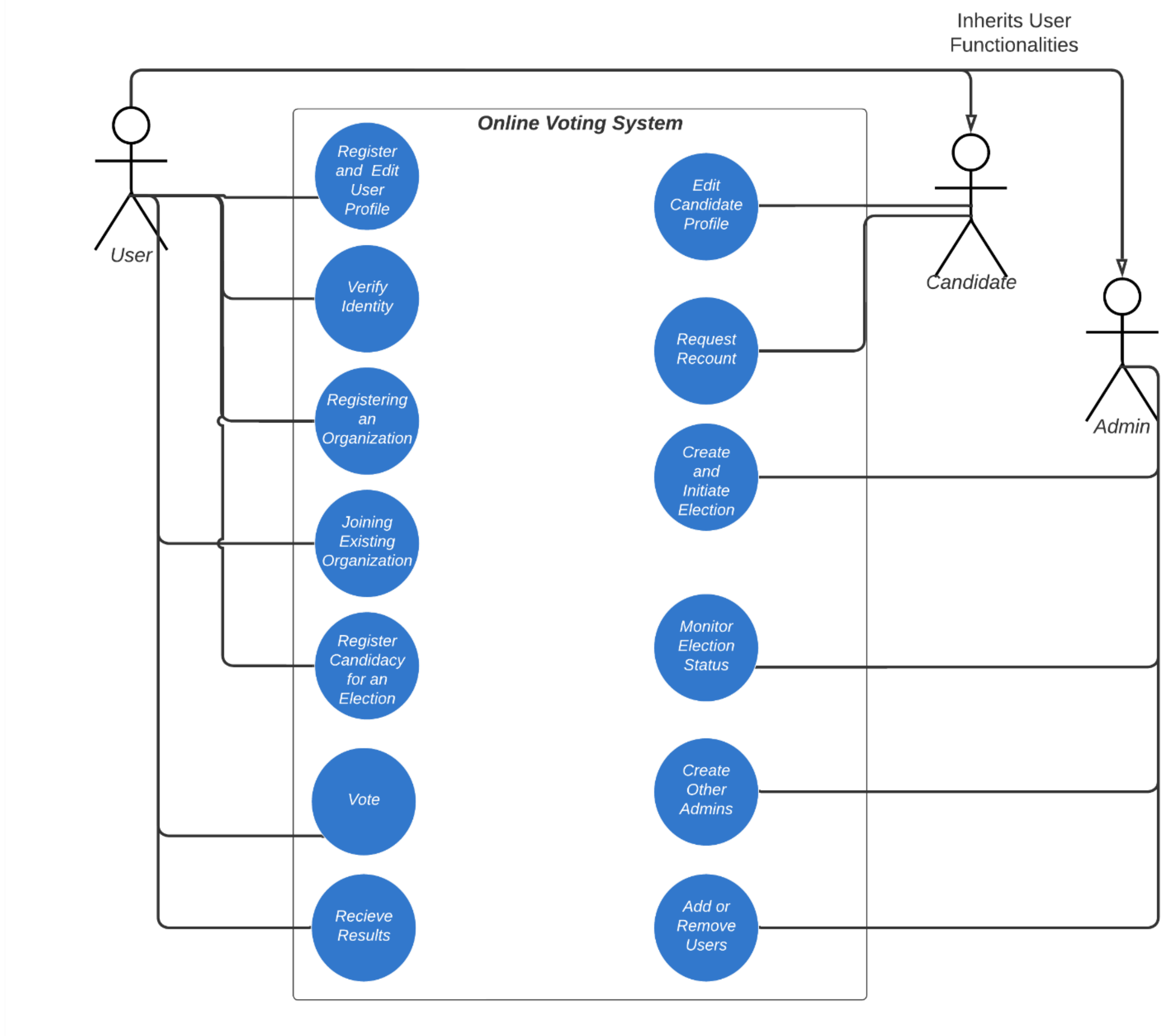
- Adding/removing users to an organization
- Elevating users to admin roles within an organization
- Selecting candidates for an election within an organization
- Creating/starting/stopping an election

**Candidate:**

Candidates are individuals running for various positions within an election. Their interactions with the system are:

- Registering their candidacy
- Monitoring election status

## 4.1 Use-Case Diagram


Inherits User Functionalities
Online Voting System
User
Candidate
Admin
Register and Edit User Profile
Verify Identity
Registering an Organization
Joining Existing Organization
Register Candidacy for an Election
Vote
Recieve Results
Edit Candidate Profile
Request Recount
Create and Initiate Election
Monitor Election Status
Create Other Admins
Add or Remove Users

## 5. Architecture and Components Description

### 5.1. Frontend Components:

Voter Interface: This interface enables users to:

- Log in.
- Access ballots.
- Submit votes.

Admin Interface: Through this interface, administrators can:

- Create polls.
- Manage ongoing polls.
- View poll results.

Candidate Interface: Designed for candidates, this interface allows them to:

- Register for electoral positions.
- Manage their profiles.
- Monitor the ongoing status and outcomes of the elections in which they're participating.

Identification Component: Responsible for:

- User registration.
- Login processes.

### 5.2. Backend Components:

Backend Server:

- API
- User logic
- Admin logic
- Ballot logic
- Routing logic
- Auditing logic
- Load balancing
- Voting logic
- Election logic

Authentication Server:

- Manages user authentication.
- Ensures only registered users access the platform and participate in voting.

Cloud Database: Responsible for storing:

- Voter data
- Ballot data
- Poll results

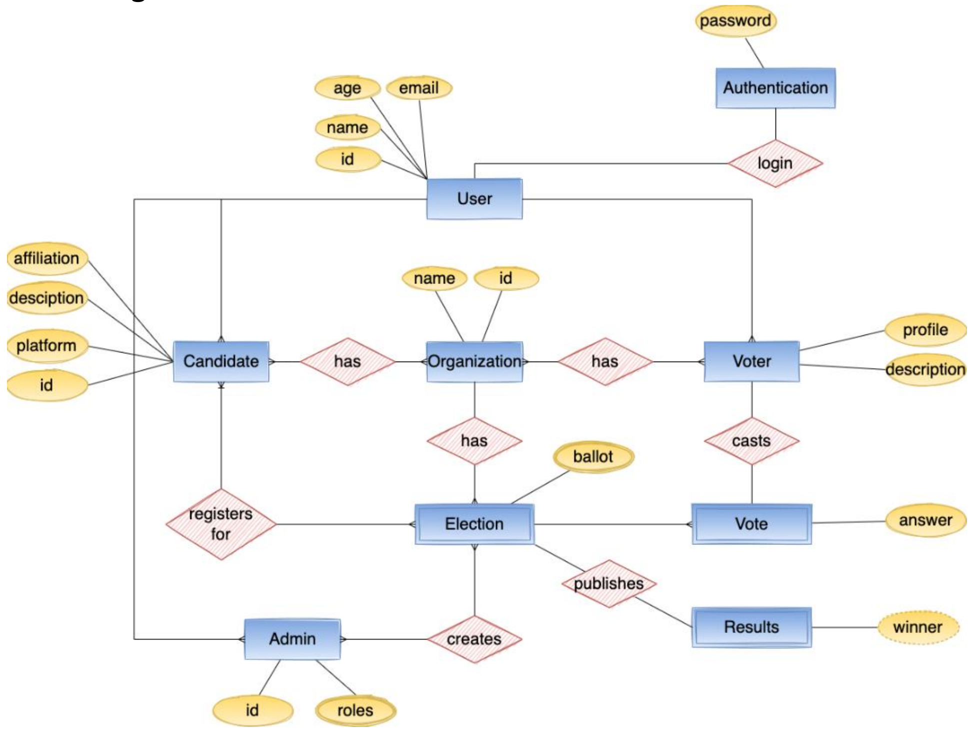
Audit Server: This server:

- Logs all system activities.
- Guarantees transparency and traceability.
- Includes components such as results logic and audit logic.

Audit Database: Specifically stores:

- System logs
- Activity data

## 6. ER Diagram



### 6.1 Main Entities Description

**Internal and External Entities**

<u>User:</u> Represents an individual registered with the system. A User interacts with the system by

- Registering a new profile
- Editing their profile
- Verifying their identity
- Registering a new Organization
- Joining an existing Organization

<u>Admin:</u> Represents the individuals that define the elections. Admins interact with the system by:

- Adding/Removing Users to an Organization
- Elevating Users to Admin roles within an Organization
- Selecting Candidates for an Election within an Organization
- Creating/Starting/Stopping an Election

<u>Voter:</u> Is a User within an Organization with the ability to vote in elections. They interact with the online voting system by:

- Casting votes

Candidate: Represents the individuals running for various positions in an election. They interact with the online voting system by:

- Registering their candidacy
- Monitoring election status

Election: Represents the electoral events organized by organization admins

Organization: Represents a group of users with a particular purpose. Voters and Candidates can belong to multiple organizations. Organizations can have multiple Admins.

Results: Represent the outcome of an election.

Authentication: Represents the information needed to successfully log in a user.

**6.2 Description of Relationships:**

Voter is a User

- Relationship Type: One-to-One
- Description: All Voters are Users

Candidate is a User

- Relationship Type: One-to-One
- Description: All Candidates are Users

Admin is a User

- Relationship Type: One-to-One
- Description: All Admins are Users

Voter casts a Vote

- Relationship Type: One-to-One
- Description: A Voter can cast a single vote for any given election

Organizations have Voters

- Relationship Type: Many-to-Many
- Description: Organizations can have many voters and a voter can belong to many organizations

Organizations have Candidates

- Relationship Type: Many-to-Many

- Description: Organizations can have many candidates and a candidate can belong to many organizations

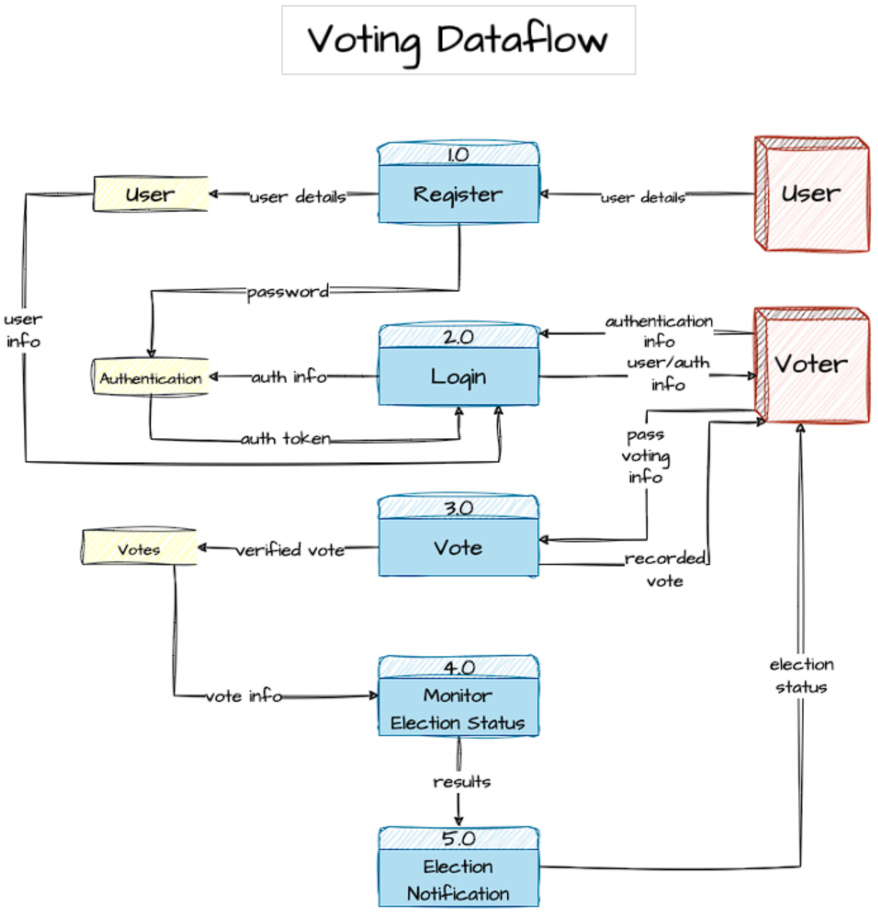
Admin creates Elections

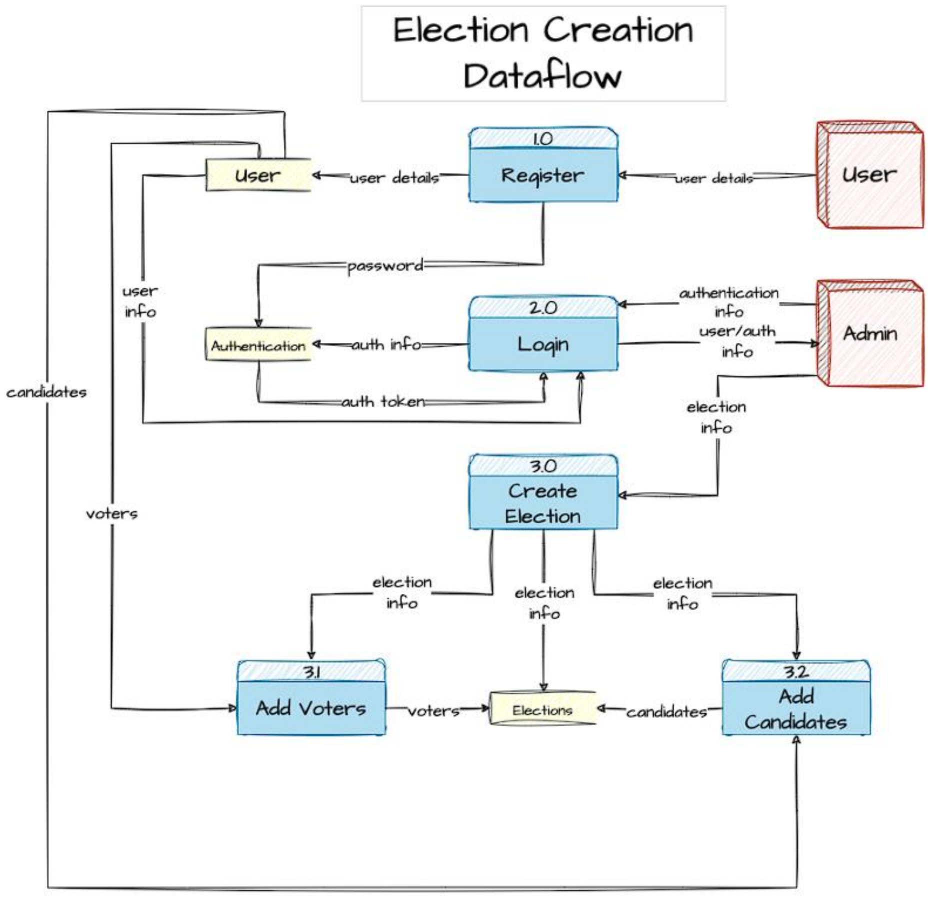
- Relationship Type: Many-to-Many
- Description: Admins can create multiple elections and elections can be created by multiple admins

Election publishes Results

- Relationship Type: One-to-One
- Description: Any Election can publish one result

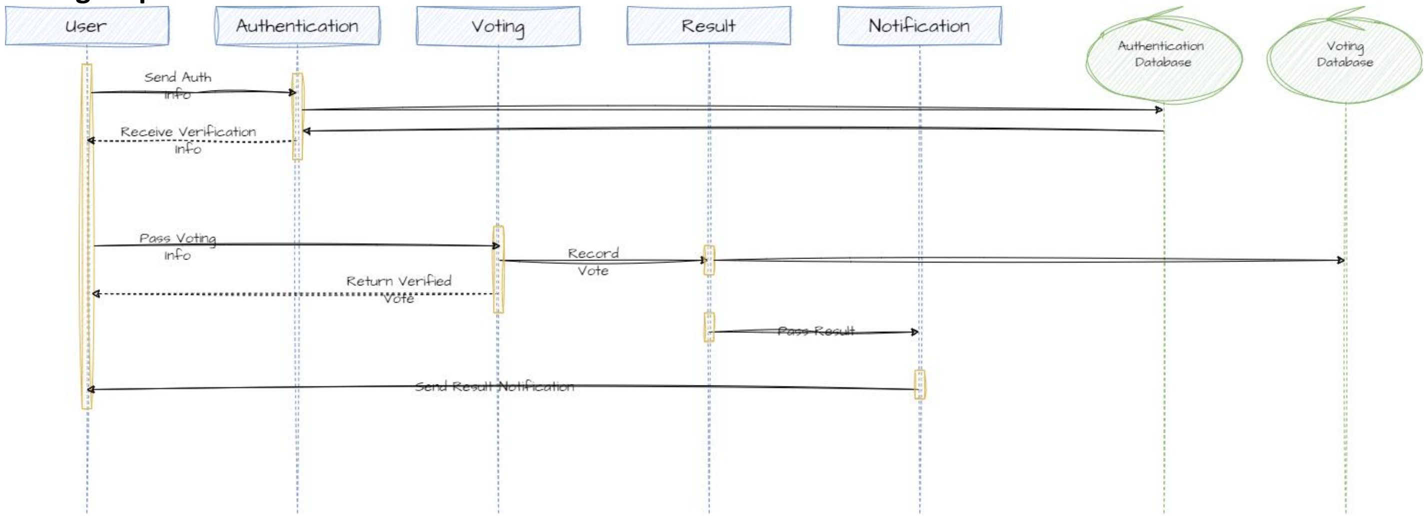
## 7. Data Flow Diagrams



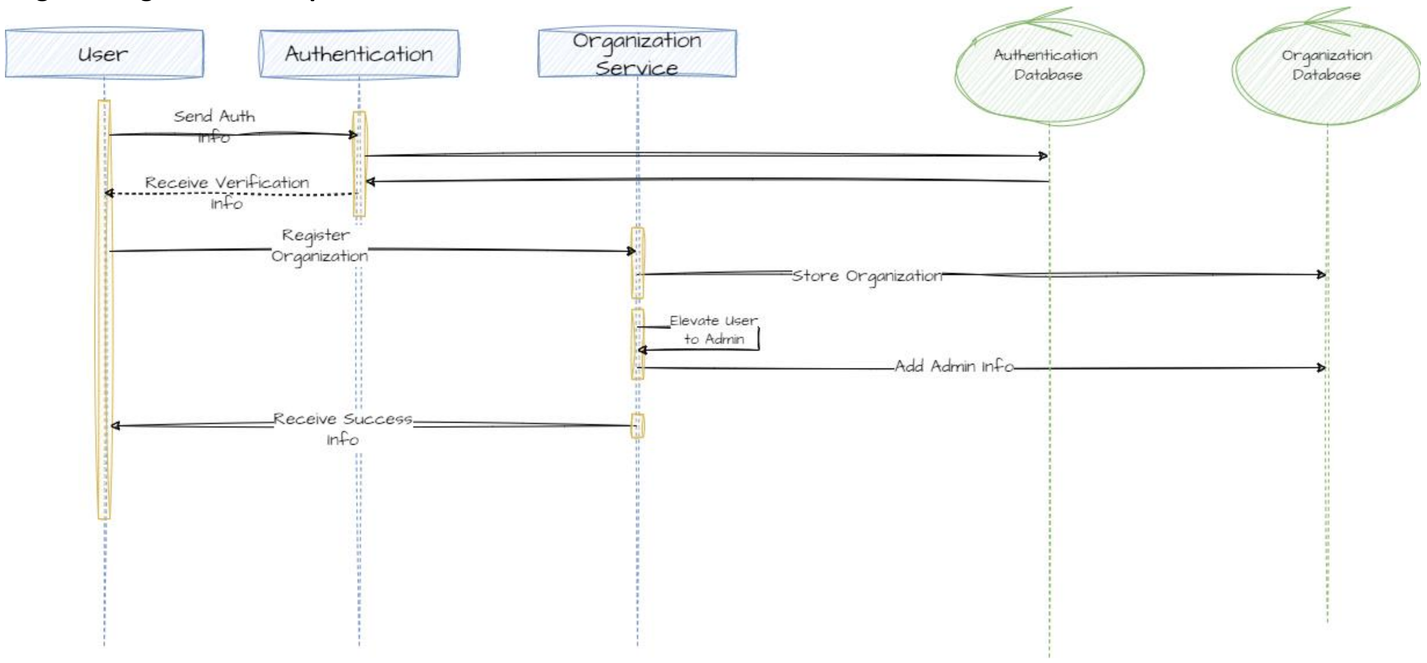

Election Creation Dataflow
User
1.0
Register
User
user details
user details
password
user info
2.0
Login
Authentication
auth info
auth token
authentication info
user/auth info
Admin
election info
candidates
voters
3.0
Create Election
election info
election info
election info
3.1
Add Voters
voters
Elections
candidates
3.2
Add Candidates

## 8. Sequence Diagrams

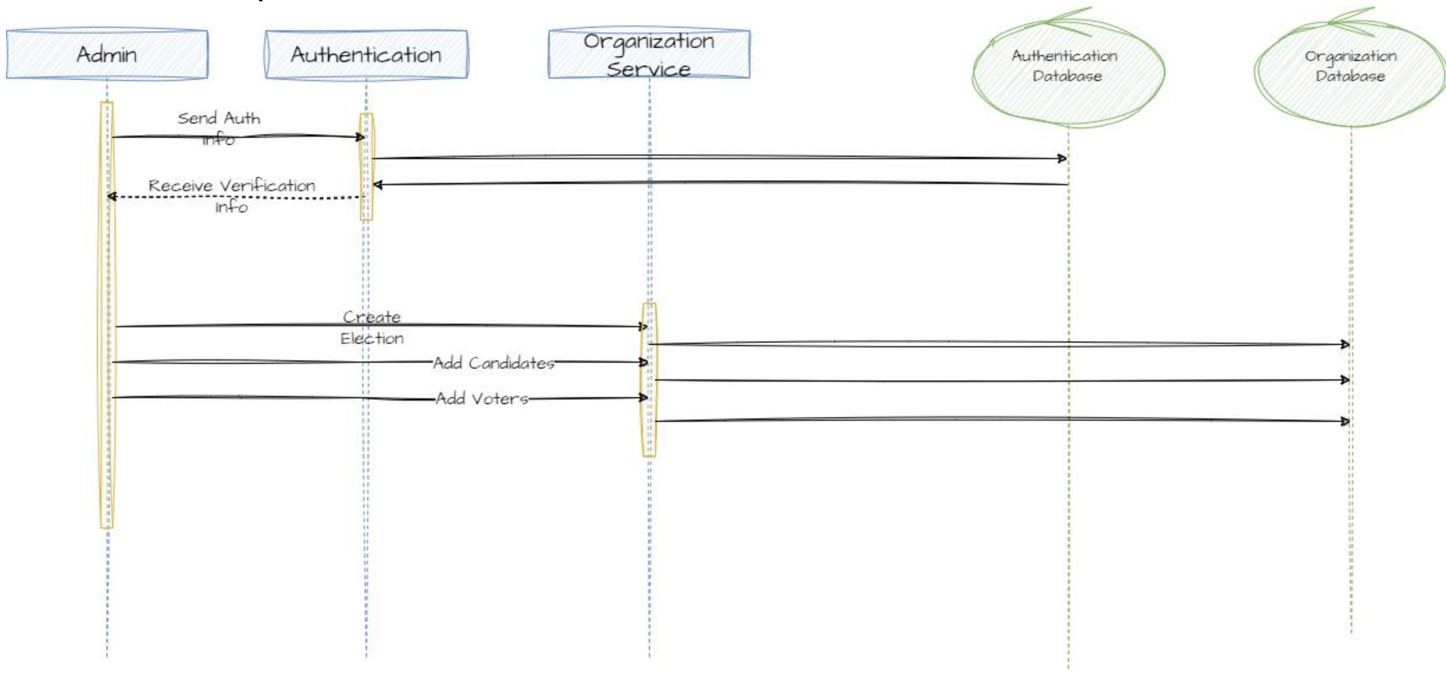
### Voting Sequence



### Register Organization Sequence



**Election Creation Sequence**

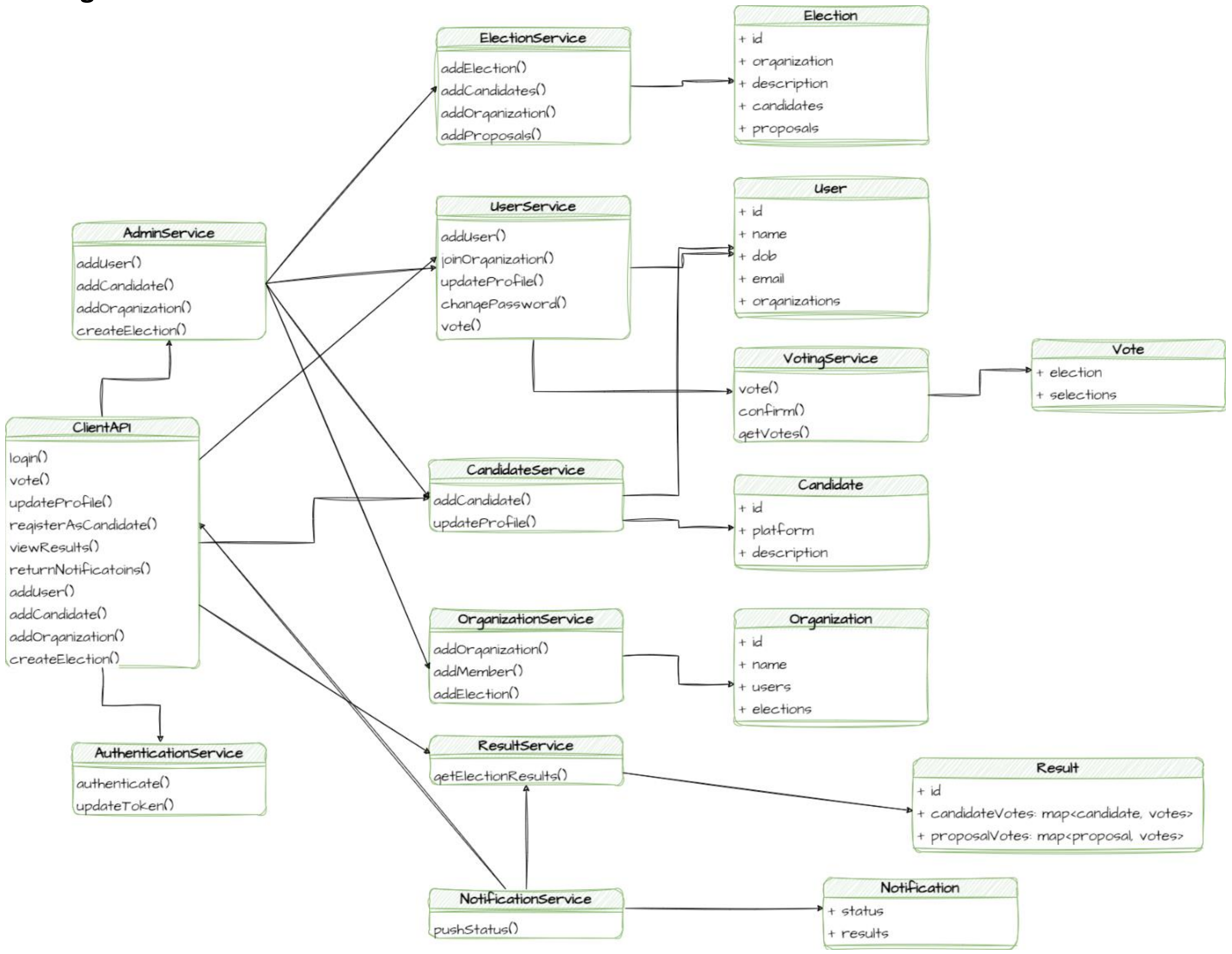


**PART 1 END**

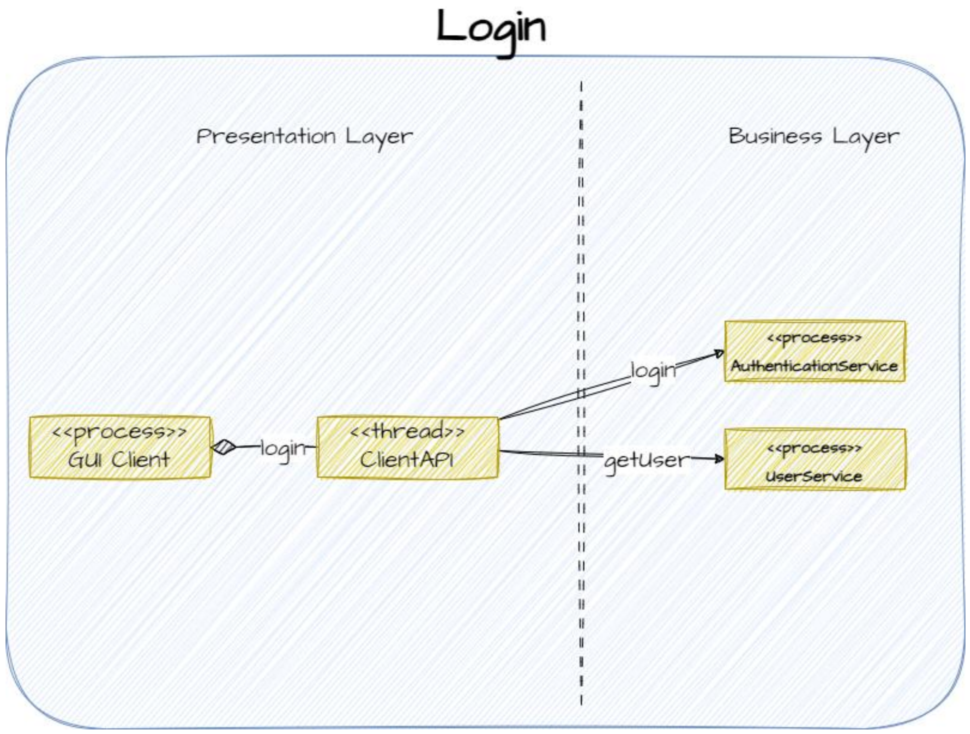
**PART 2 START**

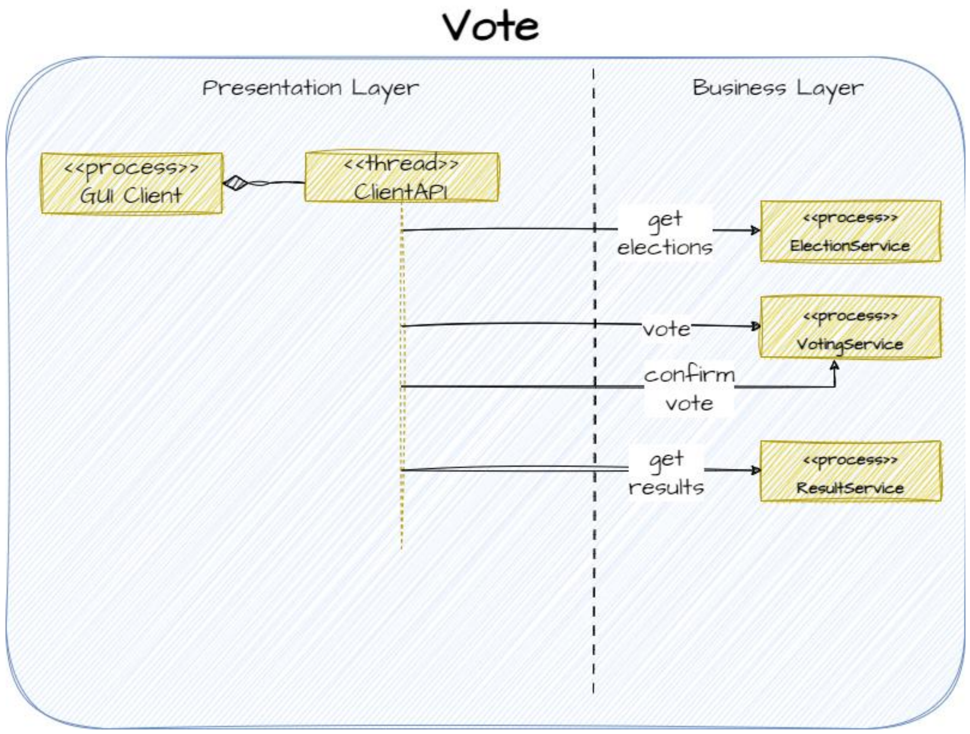
## 1. System Overview

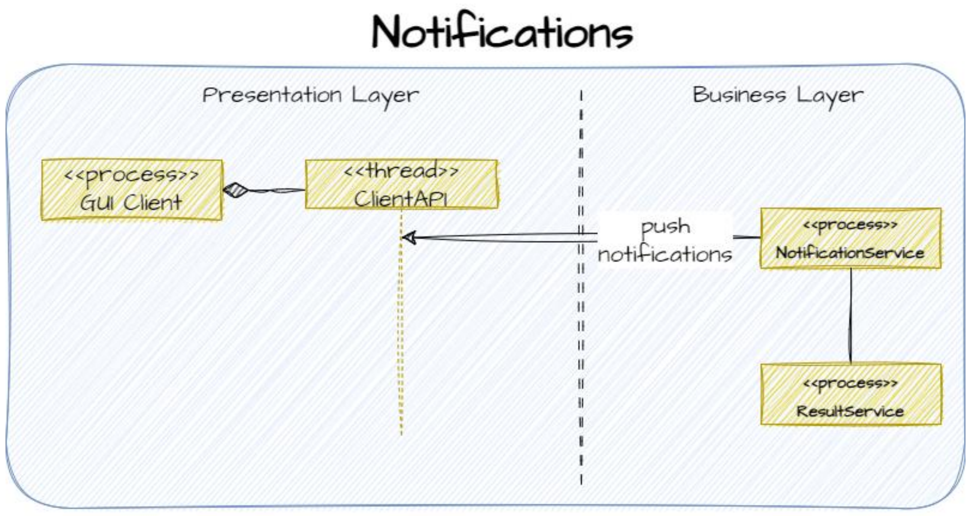
### 1.1 Logical view



## 1.2 Process Views



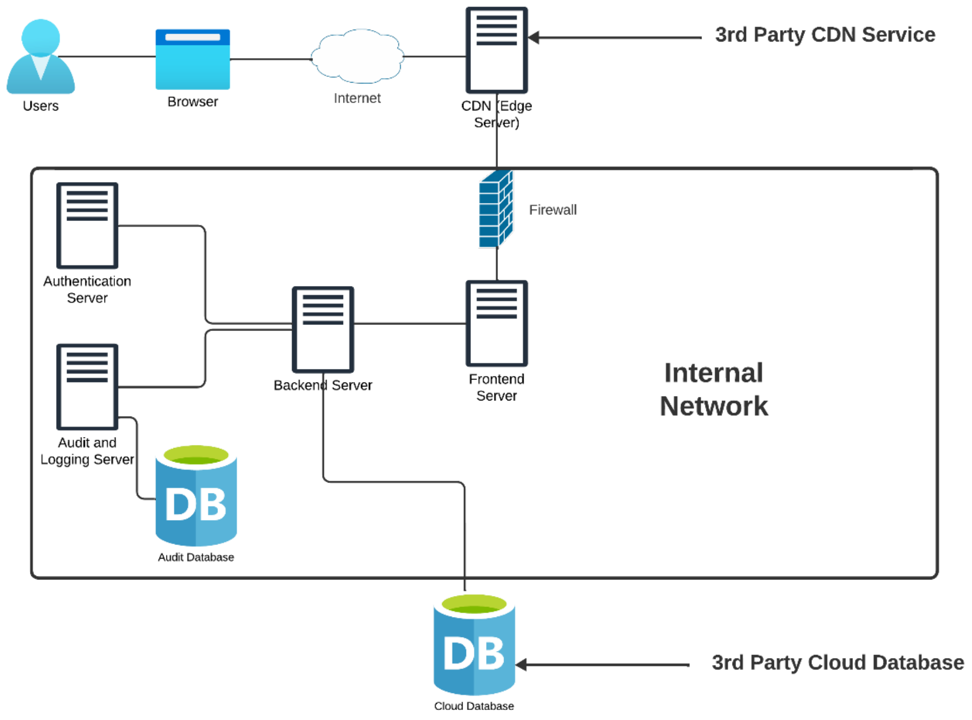




## 2. Architecture and Design

### 2.1 Component View

This is a high-level view of the major components; it's essentially a system diagram.



**2.1.1 Frontend Components:**

a. **Voter Interface:** For login, ballot access, and vote submission.
b. **Admin Interface**: Enables poll creation, management, and result viewing.
c. **Candidate Interface:** Allows electoral registration, profile management, and election monitoring.
d. **Identification Component:** Handles user registration and login processes.

**2.1.2 Backend Components:**

a. **Backend Server:** Manages APIs, user, admin, ballot, routing, auditing, load balancing, voting, and election logic.
b. **Authentication Server:** Ensures user authentication and authorized access.
c. **Cloud Database:** Stores voter, ballot, and poll result data.
d. **Audit Server:** Logs system activities and maintains transparency.
e. **Audit Database:** Contains system logs and activity data.

## 2.2 Connectors

**Event Connectors:**

When a user interacts with the frontend interfaces (Voter, Admin, Candidate), their actions (like submitting a vote or creating a poll) trigger events. For example, clicking the 'Submit Vote' button in the Voter Interface generates an event that is sent to the backend server.

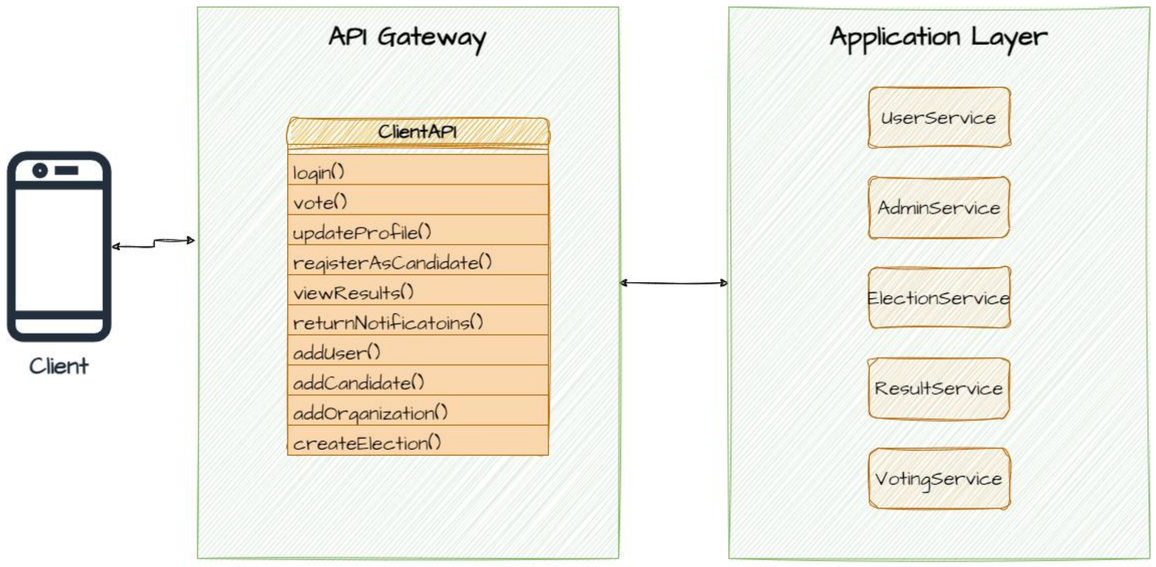
**Procedure Call Connectors:**

Procedure calls get triggered in the backend when a user either logs in, submits a vote, creates a poll, or creates an organization through the User Interface. For example, when a user clicks “submit vote” in the UI an event is triggered which initiates a procedure call to the backend, specifically to the ‘voting logic’ service. This procedure call facilitates the actions between the user’s submission and the service.

**Shared Data Access Connectors:**

These connectors help to facilitate the transfer of data between components as needed. So basically every interaction that requires data from the database will use these connectors. For example, when a user attempts to log in, his request needs access the database to check to see if that exists in the first place. Another example is when an election is completed; the shared data access connectors will facilitate the transfer of the election data to the database, the auditing server, and to the UI.

## 2.3 Interface



**2.4 Architectural Styles**

a. **Client-Server Architecture:** The frontend components interact with the backend services in a request-response pattern.

b. **Microservices Architecture:** Individual pieces of business logic can be seen as separate microservices that can be independently developed, deployed, and scaled.

c. **Service-Oriented Architecture (SOA)**: The backend components provide services that are reusable and can be integrated into multiple frontend applications.

**2.5 Design Patterns**

a) **MVC (Model-View-Controller)**: We planned to develop the front end using the MVC design pattern. This design pattern calls for the UI (View) to be separate from the business logic (Controller) and data (Model).

b) **Repository Pattern**: For data access logic, encapsulating the logic required to access the data source from the Cloud Database.

c) **Singleton Pattern**: The singleton pattern is good for classes that only have 1 instance in operation. For example, logging mechanisms within the audit and monitoring server will likely contain singleton-type classes.

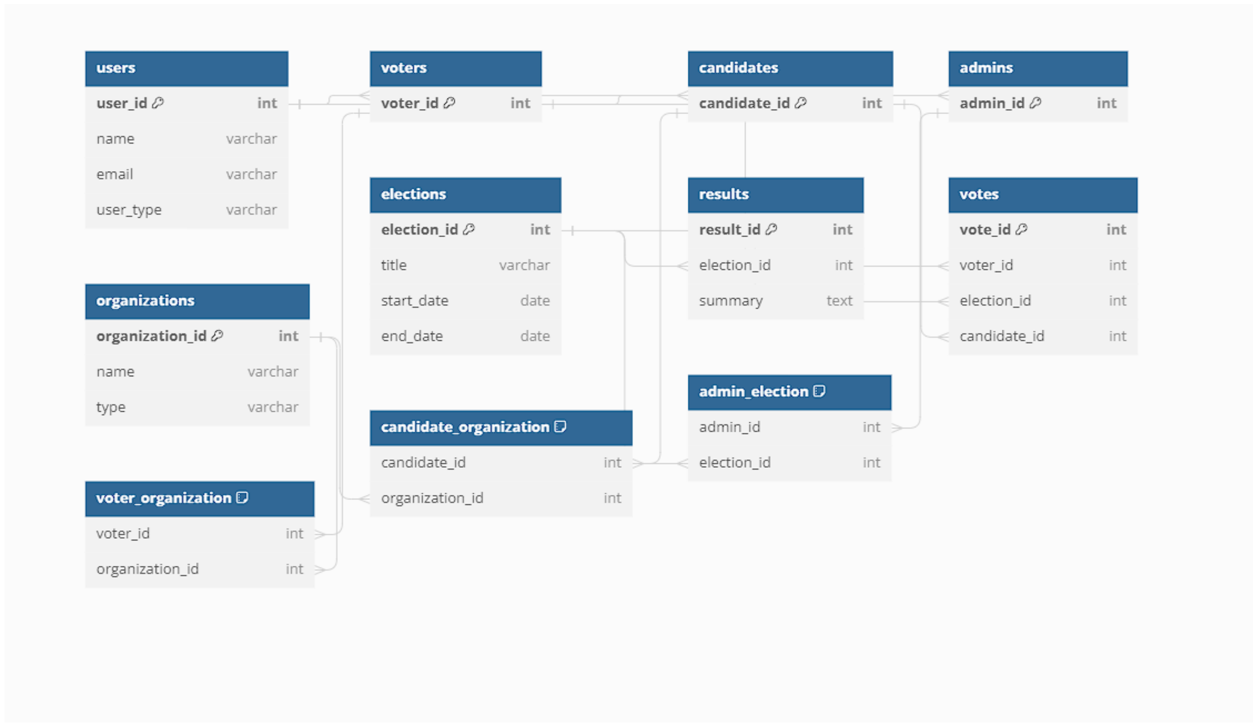
d) **Proxy Pattern**: This pattern can be used for auditing and authentication. A proxy can be used to check whether the client has the necessary access rights before forwarding the request to the target object. For example, if a user tries to login, his request will be sent to the proxy, which sends it to the auth server, which makes a decision on how to process the request, then the response is sent back to the user from the auth server through the proxy.

## 3. Data Components

### 3.1 Data Entities

1. *Users*
   Base entity representing all users, with specific roles identified.
2. *Voters*
   Extends from Users, represents individual voters.
3. *Candidates*
   Extends from Users, represents individual candidates.
4. *Admins*
   Extends from Users, represents administrators who can create elections.
5. *Organizations*
   Represents various organizations that can have associations with voters and candidates.
6. *Elections*
   Represents the elections created by admins.
7. *Results*
   Associated with each election, detailing the outcomes.
8. *Votes*
   Represents the votes cast by voters in elections.
9. *Voter_Organization*
   A junction table for managing the many-to-many relationship between Voters and Organizations.
10. *Candidate_Organization*
    A junction table for managing the many-to-many relationship between Candidates and Organizations.
11. *Admin_Election*
    A junction table for managing the many-to-many relationship between Admins and Elections.

### 3.2 Database Schema



## 4. User Interface

### 4.1 UI Interfaces

### A. Voter Interface:

Login Page <------ Major UI Component

*-Username/Email Field <------- UI Element*

*-Password Field*

*-Login Button*

*-Forgot Password Link*

Dashboard <------ Major UI Component

*-List of Available Ballots <------- UI Element*

*-Access to Past Votes*

Ballot Access

*-View Ballot Details*

*-Select Candidates/Options*

Vote Submission

*-Confirm Selection*

*-Submit Vote Button*

*-Vote Submission Confirmation*

**B. Admin Interface**

Login Page

*-Username/Email Field*

*-Password Field*

*-Login Button*

*-Forgot Password Link*

Dashboard

*- Create Poll Button*

*-Ongoing Poll List*

*-Access to Poll Results*

Poll Creation

*-Poll Details Form (Title, Description, Candidates, Duration)*

*-Create Poll Button*

Poll Management

*-View/Edit Poll Details*

*-Close Poll Early Option*

*-Delete Poll Option*

Poll Results Viewer

*-View Detailed Results*

*-Export Results Functionality*

**C. Candidate Interface**

Login Page

*-Username/Email Field*

*-Password Field*

*-Login Button*

*-Forgot Password Link*

Dashboard

*-Register for Election Button*

*-View Ongoing Elections*

*-Access to Election Results*

Election Registration

*-Election Selection Dropdown*

*-Registration Form (Biography, Promises, etc.)*
*-Submit Registration Button*

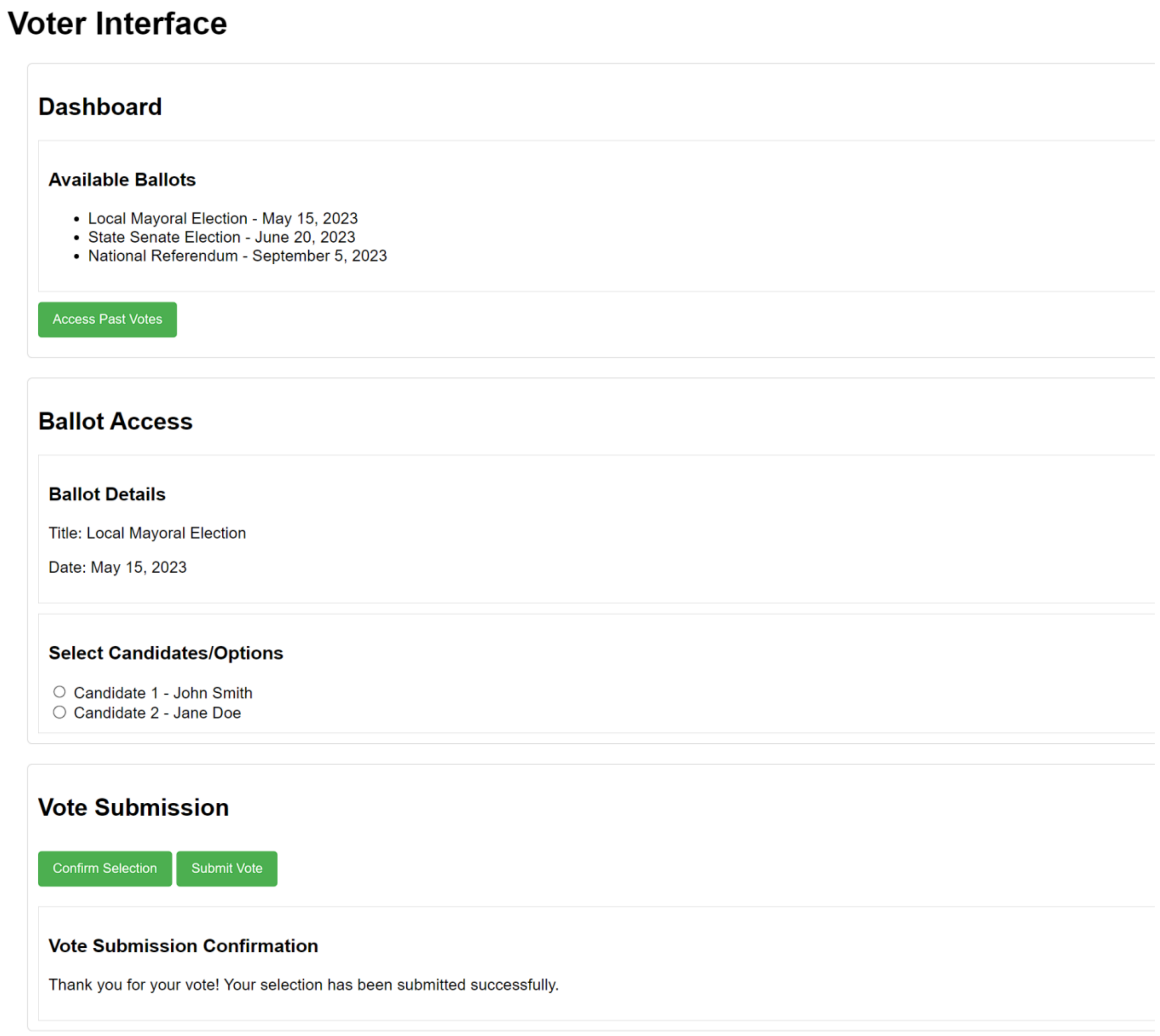
<u>Profile Management</u>
*-Edit Profile Information*
*-Upload Campaign Materials*

<u>Election Monitor</u>
*-Current Election Status*
*-View Vote Counts (if allowed)*

**D. Identification Interface (General)**
<u>User Registration</u>
*-Registration Form (Name, Email, Password, User Type)*
*-Submit Registration Button*

<u>Login Page</u>
*-Username/Email Field*
*-Password Field*
*-Login Button*
*-Forgot Password Link*

## 4.2 UI Graphics



### Voter Interface

#### Dashboard

**Available Ballots**

- Local Mayoral Election - May 15, 2023
- State Senate Election - June 20, 2023
- National Referendum - September 5, 2023

#### Ballot Access

**Ballot Details**

Title: Local Mayoral Election

Date: May 15, 2023

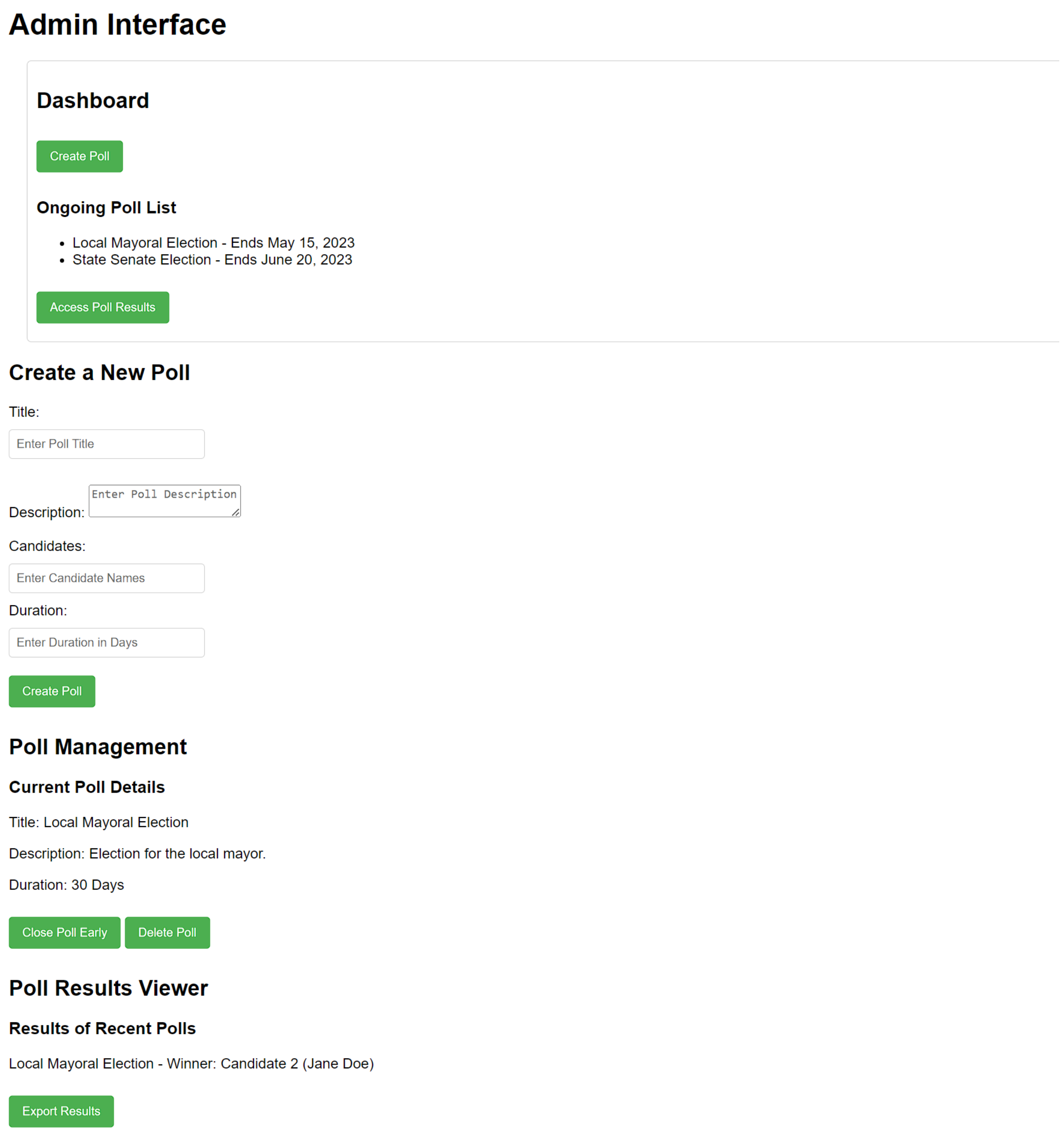
**Select Candidates/Options**

○ Candidate 1 - John Smith
○ Candidate 2 - Jane Doe

#### Vote Submission

**Vote Submission Confirmation**

Thank you for your vote! Your selection has been submitted successfully.



## Admin Interface

### Dashboard

Create Poll

#### Ongoing Poll List

- Local Mayoral Election - Ends May 15, 2023
- State Senate Election - Ends June 20, 2023

Access Poll Results

### Create a New Poll

Title:

Enter Poll Title

Description: Enter Poll Description

Candidates:

Enter Candidate Names

Duration:

Enter Duration in Days

Create Poll

### Poll Management

#### Current Poll Details

Title: Local Mayoral Election

Description: Election for the local mayor.

Duration: 30 Days

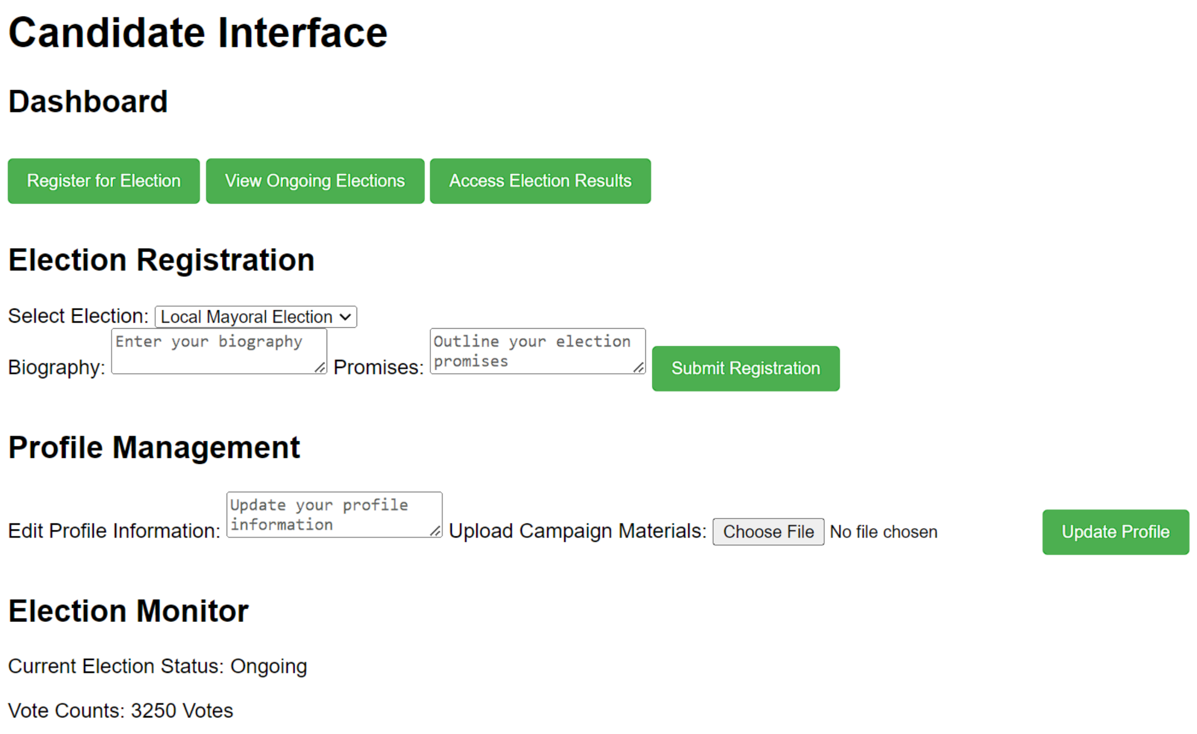
Close Poll Early Delete Poll

### Poll Results Viewer

#### Results of Recent Polls

Local Mayoral Election - Winner: Candidate 2 (Jane Doe)

Export Results



Candidate Interface

Dashboard

Register for Election | View Ongoing Elections | Access Election Results

Election Registration

Select Election: Local Mayoral Election

Biography: Enter your biography Promises: Outline your election promises Submit Registration

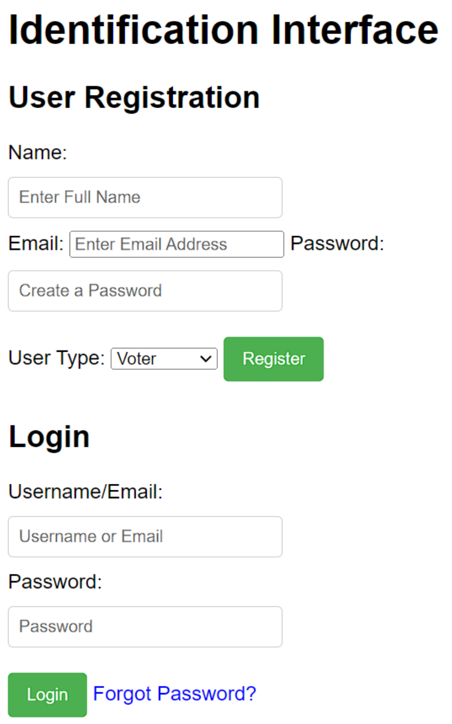
Profile Management

Edit Profile Information: Update your profile information Upload Campaign Materials: Choose File No file chosen Update Profile

Election Monitor

Current Election Status: Ongoing

Vote Counts: 3250 Votes



Identification Interface

User Registration

Name: Enter Full Name

Email: Enter Email Address Password: Create a Password

User Type: Voter Register

Login

Username/Email: Username or Email

Password: Password

Login Forgot Password?

## 5. Three Main Scenarios:

### 5.1 Normal

**Normal Scenario Description:**

In the normal scenario, users engage in routine operations within the Online Voting System (OVS) under ideal conditions. This involves routine tasks such as user registration, organization creation, election management, candidate registration, and the casting of votes. Users interact with the system as expected, following the established workflows and protocols.

a. **Routine Operations:**
   1. Users register new profiles.
   2. Admins manage elections, including adding/removing users, selecting candidates, and starting/stopping elections.
   3. Candidates register for electoral positions and monitor election status.
   4. Voters access the system, cast their votes, and receive election results.
   5. Organizations are created and users join existing organizations.

b. **Expected System Behavior:**
   1. Seamless user registration and profile management.
   2. Admins can efficiently manage elections and organizational structures.
   3. Candidates can easily register and monitor their campaign progress.
   4. Voters experience a straightforward process for accessing and participating in elections.
   5. Organizations are successfully created, and users join them without issues.

c. **Ideal Conditions:**
   1. Stable network connectivity.
   2. All system components (front-end, back-end, databases) are operational.
   3. Users provide accurate information during registration and election processes.
   4. Adequate server resources are available to handle user interactions.
   5. Security measures are effective in protecting user data and system integrity.

### 5.2 Error

a. **Error Detection:** The system employs robust error detection mechanisms, including input validation checks and continuous monitoring for system faults. Real-time error detection is integrated into critical processes such as user registration, voting, and election management.

b. **User Feedback:** Users receive clear and informative error messages when issues arise. These messages guide users on how to rectify errors or seek assistance. The feedback is designed to be user-friendly and easily understandable.

c. **Error Handling:**

1. Rollbacks: In case of transactional errors, the system initiates rollbacks to maintain data consistency.
2. Retries: Users are given opportunities to correct errors and retry actions.
3. Recoveries: The system has automated recovery processes to restore normal operations after encountering errors.

**d. Logging and Monitoring:** All error events are logged for analysis and auditing purposes. Monitoring tools track system performance and alert administrators to potential issues, facilitating proactive error resolution.

**5.3 Special Point of Variation**

a. **Unusual User Behavior:** The system is designed to handle unusual user behavior, such as multiple login attempts, in a way that ensures security without causing inconvenience to legitimate users. Anomalies are flagged for further investigation.

b. **High Network Stress/Load:** The system can dynamically scale resources to handle increased network stress or load during peak usage times, ensuring consistent performance and responsiveness.

c. **Security Breaches:** In the event of a security breach, the system implements predefined security protocols, including isolation of affected components, notifying users, and working towards a swift resolution. Intrusion detection systems and encryption measures are in place to mitigate security risks.

d. **Feature Variation:** The OVS system is adaptable to feature variations, accommodating client requests for new versions with additional features. The integration of new features follows a systematic development and testing process to ensure seamless incorporation without compromising system stability.

## 6. Design Rationale:

### 6.1 Security

As the designers of the software, we have considered several different measures that can be combined into a security plan when utilizing the OVS system. These are the general measures that we would employ if we were operating this OVS system for a large organization.

a. **Properly Configured Network Topology:** The OVS should utilize a segmented network architecture which separates the major components such as the: database, application servers, and authentication servers. This reduces the risk of lateral movement in case of a breach of one of the resources.

b. **Perimeter Defense:** Perimeter Defense typically includes security measures such as firewalls for monitoring and controlling incoming and outgoing network traffic, as well as intrusion detection systems (IDS) and intrusion prevention systems (IPS) to identify and block potential threats.

c. **Authentication Servers:** These are servers that implement measures such as single-factor and multi-factor authentication (MFA) to verify the identity of a user attempting to register or login to the system. Using auth servers helps to ensure the confidentiality of user accounts and other information within the system.

d. **Encryption:** All data, both in transit and at rest, should be encrypted using industry-standard protocols (such as TLS for data in transit and AES for data at rest). This also helps to maintain the confidentiality of information/data within the system.

e. **Regular Security Audits and Compliance:** Security audits should be performed regularly to identify and fix vulnerabilities within the system. Regular compliance checks should also be performed, this helps to ensure that the system abides by data protection and privacy laws (such as GDPR).

### 6.2 Performance

Performance is a critical non-functional component of the system. We need to make sure that that it is in an optimal state of performance while it's in operation. It's not enough to just be reliable, as in 99.99% up time, it also needs to perform well during that uptime.

a. **Content Delivery Network (CDN):** To increase performance, we have decided to include a 3rd party CDN service in our system's architecture. CDNs cache content closer to the user, reducing load times for static resources.

   For example, Amazon has data centers all around the world that can be used to deploy 'edge servers' which serve content from locations geographically closer to the end-user. These geographical regions are categorized into 'availability zones'. When a user requests data, such as an image or a video, instead of this request traveling to the main server, which could be thousands of miles away, it is routed to the nearest edge server. This significantly reduces latency, as the distance the data travels is much shorter. It also decreases the load on the origin server, because edge servers cache static content and offload traffic during peak times.

b. **Performance Testing:** If we were operating the OVS system, we would plan and perform regular performance testing. This includes stress testing different parts of the system to identify bottlenecks.

**6.3 Reliability**

a. **Redundancy:** Cloud databases and CDN's allow for high reliability through redundancy. Since most of the time we are dealing with virtual instances of these components, they are easy to replicate, rollback, and replace.
b. **Monitoring**: We can help to ensure that our system is functioning as expected through constant monitoring of the system. We may be able to utilize our Audit and Monitoring server for this.

**6.4 Availability**

a. **Web-Based Platform:** Anyone with a web browser and email can register and access the system from most places in the world. Users closer to availability zones will have better performance using the system.

**6.5 Scalability**

There are several aspects of our system that we believe will provide scalability in the future.

a. **Stateless Components:** Some low-level components such as the services within the backend server, are built with a 'Stateless' architecture. This allows each transaction by or with a service, to occur independently. This means the system can scale out horizontally by adding more servers without worrying about an individual server's state.

b. **CDN's:** Utilizing these types of networks to service end users is beneficial in several ways, one being that big data companies like Amazon and Microsoft (Azure) have data centers all over the world which offer scalable access to our system. For example, if our software system becomes popular in Canada, we can start hosting our content on CDN's within a Canadian availability zone.

c. **Cloud Database:** When using a cloud service like Amazon or Azure, you can adjust the amount of resources dedicated to running your system. The adjustments are made based on the demand that the system is experiencing; the adjustments can be performed manually or automatically.

d. **Microservices Architecture:** As mentioned in Part 2 Section 2, we are utilizing a microservice architecture. Here we break down Individual pieces of business logic into separate microservices that can be independently developed, deployed, and scaled.

**PART 2 END**