PAPER NAME

Swoyam_Pokharel_2431342.docx

AUTHOR

-

WORD COUNT

4690 Words

CHARACTER COUNT

24909 Characters

PAGE COUNT

34 Pages

FILE SIZE

4.1MB

SUBMISSION DATE

Apr 20, 2025 9:00 PM GMT+5:45

REPORT DATE

Apr 20, 2025 9:01 PM GMT+5:45

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 3% Internet database
- 1% Publications database
- Crossref database
- Crossref Posted Content database
- 6% Submitted Works database

Academic Year	Module	Portfolio	Assessment Type
2025	5CS024/HJ1: Collaborative Development	1	Individual Report

Online Voting System - Developer

Name: Swoyam Pokharel

ID: 2431342

Group: 26

Tutor: Mr. Ankit Tamrakar

Acknowledgement

I'd like to acknowledge Herald College and University Of Wolverhampton for providing the opportunity to work on this project. It's genuinely been a valuable learning experience; getting hands-on with new tech stacks, experimenting freely and growing as a developer. Furthermore, working alongside a team has taught me the importance of communication , shared responsibility and iterative problem solving. This sprint was a great mix of personal growth and learning about methodologies of working in a team; it's been a rewarding experience.

Table Of Contents:

Table of Figures:

Self Appraisal Form

Student Number:	2431342	Name:	Swoyam Pokharel
Project:	Online Voting System	Date:	19th April 2025
Role:	Developer	Team:	L5CG26
Sprint (1 or 2)	1		

Personal Objectives - Performance Metric

These should be copied from your role description

Objectives	Evidence provided *(E.g. appendix A, file name etc.)*	Evaluation *Student / tutor*	
Picking the appropriate tech stack	For our voting system project, we selected a technology stack that balances performance, scalability, and developer efficiency, while aligning with our team's capabilities. Read More		
Tutor feedback:			
Developing core functionality	For this sprint, I completed all the core, foundational requirements for a system that allows users to vote. The user can login, cast a vote and get live, real-time vote updates; the admin can manage all major entities (users, citizens, elections , candidates). Read More		
Tutor feedback:			
		/20	/20

Collaboration Document:

Evidence of good collaboration

Good communication and file sharing

Most of the team's communication takes place on Discord and Google Chat. Discord serves as our primary platform for the project's discussions; we have a server where all the discussion happens and all members have full visibility over conversations. Regular meetings are also held on Discord, where the entire team gathers at a fixed time each day to review progress, align on tasks, and plan next steps. Project tracking happens on jira where each member marks their tasks done; along with the relevant proof. File sharing and version control are handled via GitHub, everyone has access to the repository where all the code is shared.
6
Appendix B

Continuing Personal Development (CPD)

To learn the tools I was working with, I went through official documentation for said tools. I used the provided documentation as key references and other websites such as reddit, stack overflow for very specific problems I was facing.

Appendix B

Issue Tracking

Although I wasn't assigned any issues directly by neither the PM or the BA, I still took proactive and reactive measures to tackle issues that either weren't documented or issues that were assigned to someone else, that I had to later pick up because of their unfamiliarity and the fact that the deadline was closing in. Apart from that, there were a bunch of other issues that I discovered and fixed on my own; issues that weren't officially documented such as fixing CORS problems, login bypass, and integrating the frontend and the backend.

Appendix B

Appendix A

Picking The Appropriate Tech Stack

For our voting system project, we selected a technology stack that balances performance, scalability, and developer efficiency, while aligning with our team's capabilities. Our stack includes Golang for the backend, Svelte/SvelteKit for the frontend, Turso and Redis for data management, websockets for real-time duplex communication and TailwindCSS with DaisyUI for UI styling.

The Backend:

As the sole backend developer in the team, I was a strong voice in the choice of the backend's language, and ultimately we settled with Golang.

Golang was chosen due to its high performance , efficient concurrency model and its minimal runtime. Furthemore, my familiarity with golang was a plus. Go's built-in support for concurrency through goroutines and channels make it very good for handling high volumes of concurrent requests, which is critical for a voting system expected to potentially handle an entire nation's election.

Other languages were considered but ultimately set aside for the following reasons:

- Node.js: Struggles with CPU-bound operations and lacks true multithreading support.
- Python: Interpreted and single-threaded, making it unsuitable for real-time, high-load systems.
- Rust: While highly performant, its complexity and steep learning curve would've slowed development.

Golang offered the best balance of performance, simplicity, scalability and familiarity for our use case.

The Frontend:

Svelte & Sveltekit

For the frontend, we chose Svelte along with SvelteKit to build a fast responsive and a lightweight user interface. The main reason Svelte was chosen was due to its syntax being very close to plain HTML, CSS and Javascript making it incredibly easy and intuitive for our team, especially since most members already had basic web development experience. Furthermore, its reactivity model removes the need for complex statement management libraries. On top of it all, svelte is more performant across the board with a lower memory footprint too, resulting in a cleaner and more performant codebase.

SvelteKit was chosen because it's the official meta framework for Svelte. It allows file based routing, server-side rendering (SSR) and data fetching paradigms, which significantly improved our development flow and performance. The data fetching model in SvelteKit is very intuitive and pairs very well with our Golang backend. SSR was a great touch on top, as it ensures initial faster load times and better SEO.

Other frameworks were considered but ultimately skipped due to:

- React: Introduced too much boilerplate and complexity, virtual dom added unnecessary performance overhead and it simply isn't as easy to pick up as Svelte.

- Vue: Easier than React but still suffers with the same trade offs ,

Websockets

Vanilla JS was not even a consideration because of the implicit need for our app to be highly reactive. To support real-time updates, we integrated websockets into the frontend. This allows the Golang backend to push live vote counts and election results to the users instantly, ensuring they get up to date information without needing to constantly refresh the page. Websockets provide a long living connection between the client and the server, enabling us to push live updates with low latency and allowing us to provide any user with a “per vote” update as soon as it happens.

Tailwindcss & Daisyui

For styling, we chose TailwindCSS paired with Daisyui. Tailwind’s utility first approach allowed us to style components without having to create a bunch of css files, keeping our codebase clean and maintainable. To accelerate development further, we integrated Daisyui, which is a component library built on top of tailwind, providing abstractions to use pre-designed, themeable UI components that helped us prototype quick. Furthermore, in the later sprint, we plan to settle for a theme, upon which all components styled will follow that same theme which will help us achieve cohesiveness and a modern look across the entire application.

The Databases – Turso & Redis:

For the database, we adopted a dual database setup with Turso and Redis; both were chosen for specific reasons tied to systems performance and scalability requirements.

Turso:

Turso serves as our primary database. It is a distributed database edge-hosted and built on top of libsql, which itself is a fork of the absurdly popular SQLite database improving on things traditional SQLite was missing such as embedded replicas and most importantly the ability to host in a server [1] . Using Turso, gave us the familiarity of SQLite combined with the modern capabilities such as replication, backups and global distribution. Turso allows us to have databases physically closer to the users geographically, which reduces latency and improves read performance which is especially important during high-traffic events like an election. Turso also allows multiple services to interact with the same database simultaneously, without the complexity of managing a centralized database [2]; which fits our use case perfectly as the aim of our Golang Backend is to be distributed across different servers. Turso handles storing the structured data such as user’s , citizen’s, candidate’s and election’s data.

Redis:

Redis is our secondary database, it was chosen because it’s uniquely suited for our use case of incrementing a counter. In hindsight, incrementing a counter seems no big deal, but when scaled to millions of users, traditional databases would simply be too slow or inefficient. Redis shines in this regard because it provides atomic operations to handle things such as incrementations making it very efficient and fast.[3]

Furthermore, its Pub/suB architecture enables an event-driven model for real time vote updates. A dedicated goroutine runs concurrently with the main thread as a Redis subscriber, and when a vote event is published, the go routine captures the event and instantly pushes the updated data to all connected clients via websockets [4]. This design ensures that users

receive live vote updates with minimal latency. Apart from this niche use case, we also plan to use redis as a cache to additionally improve performance

By combining Turso and Redis, we're able to achieve balance between consistency and reliability with real time responsiveness.

Developing Core Functionality

The core goal of our voting system was to let users sign up, view ongoing elections, and vote securely and efficiently. My role in this sprint was primarily backend, although I did contribute to the frontend as well. All the work is available in the github repository, below is a list of tasks I've handled along with the commits referencing them.

Week 1: Schema & Initial Setup

- Created and configured the Turso database
- Designed the database schema for:
  - Users
  - Citizens
  - Candidates
  - Elections
  - Admins
- Defined relations between tables for relational integrity, such as "each candidate must also be a citizen and have a valid election he/she is running on", "Each user must be a citizen" etc.

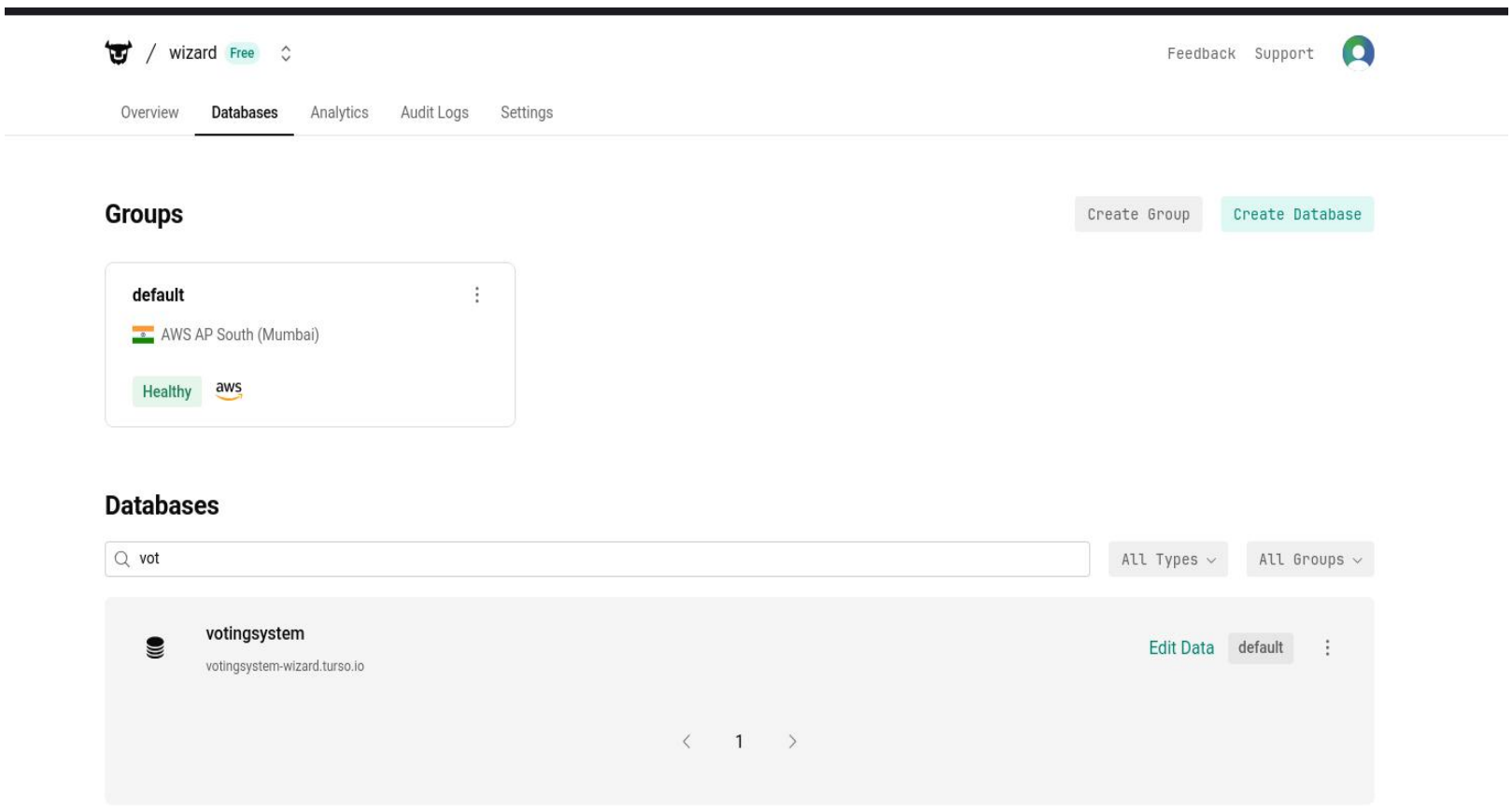


Fig 1: Screenshot of the Turso's Dashboard

The image above shows the creation of a database "votingsystem" in the region of mumbai.

```
→  PRAGMA table_info(users);
CID     NAME            TYPE            NOTNULL     DFLT VALUE      PK
0       userID          INTEGER         0           NULL            1
1       citizenID       VARCHAR(20)     1           NULL            0
2       password        VARCHAR(255)    1           NULL            0
3       phonenumber     VARCHAR(15)     1           '000000000'     0
4       tag             VARCHAR(50)     1           'untagged'      0
→  PRAGMA table_info(citizens);
CID     NAME                TYPE            NOTNULL     DFLT VALUE      PK
0       citizenID           VARCHAR(20)     0           NULL            1
1       fullName            VARCHAR(255)    1           NULL            0
2       dateOfBirth         DATE            1           NULL            0
3       placeOfResidence    TEXT            1           NULL            0
→  PRAGMA table_info(candidates);
CID     NAME            TYPE            NOTNULL     DFLT VALUE      PK
0       candidateID     INTEGER         0           NULL            1
1       citizenID       VARCHAR(20)     1           NULL            0
2       post            VARCHAR(255)    1           NULL            0
3       electionID      INTEGER         1           NULL            0
4       GroupName       TEXT            0           NULL            0
→  PRAGMA table_info(elections);
CID     NAME                TYPE            NOTNULL     DFLT VALUE      PK
0       electionID          INTEGER         0           NULL            1
1       title               VARCHAR(255)    1           NULL            0
2       startDate           DATE            1           NULL            0
3       endDate             DATE            1           NULL            0
4       votingRestrictions  VARCHAR(255)    1           NULL            0
→
[0] 0:turso*
```

Fig 2: Database Schema For The Entities

The above image shows the database schema for all the tables that exist on the database currently.

The database schema can be found here in a better, more comprehensive format.

Week 2: API Development

- Built API for Citizens:
    - Create New Citizen
    - Fetch A Citizen's Details
    - Get All Citizens Paginated
    - Delete A Citizen
    - Update A Citizen's Details
- Built API for Users
    - Create New User
    - Fetch A User's Details
    - Get All Users Paginated
    - Delete A User
    - Update User Details
- Built API for Candidates:
    - Create New Candidate
    - Fetch A Candidate's Details
    - Get All Candidates Paginated
    - Delete A Candidate
    - Update A Candidate's Details
- Built API for Elections:
    - Create A New Election
    - Fetch An Election's Details
    - Get All Running Elections
    - Update An Election's Details
    - Delete An Election

```
POST    | /api/secure/candidate                    |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleCreateCandidate
HEAD    | /api/secure/candidate/:id                |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleGetCandidate
PUT     | /api/secure/candidate/:id                |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleUpdateCandidate
GET     | /api/secure/candidate/:id                |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleGetCandidate
DELETE  | /api/secure/candidate/:id                |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleDeleteCandidate
HEAD    | /api/secure/candidatesPaginated/:offset  |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleGetCandidatesPaginated
GET     | /api/secure/candidatesPaginated/:offset  |      | github.com/PS-Wizard/ElectOneAPI/api/Candidates.HandleGetCandidatesPaginated
POST    | /api/secure/citizens                     |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleCreate
DELETE  | /api/secure/citizens/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleDelete
HEAD    | /api/secure/citizens/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleSearch
PUT     | /api/secure/citizens/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleUpdate
GET     | /api/secure/citizens/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleSearch
HEAD    | /api/secure/citizensPaginated/:offset    |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleGet
GET     | /api/secure/citizensPaginated/:offset    |      | github.com/PS-Wizard/ElectOneAPI/api/Citizens.HandleGet
POST    | /api/secure/election                     |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleCreateNewElection
HEAD    | /api/secure/election/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleGetElection
DELETE  | /api/secure/election/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleDeleteElection
GET     | /api/secure/election/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleGetElection
PUT     | /api/secure/election/:id                 |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleUpdateElectionDetails
HEAD    | /api/secure/electionsPaginated/:offset   |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleGetElectionsPaginated
GET     | /api/secure/electionsPaginated/:offset   |      | github.com/PS-Wizard/ElectOneAPI/api/Elections.HandleGetElectionsPaginated
POST    | /api/secure/user                         |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleCreateNewUser
DELETE  | /api/secure/user/:id                     |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleDeleteUser
GET     | /api/secure/user/:id                     |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleGetUser
PUT     | /api/secure/user/:id                     |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleUpdateUserDetails
HEAD    | /api/secure/user/:id                     |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleGetUser
GET     | /api/secure/usersPaginated/:offset       |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleGetUsersPaginated
HEAD    | /api/secure/usersPaginated/:offset       |      | github.com/PS-Wizard/ElectOneAPI/api/Users.HandleGetUsersPaginated
```

Fig 3: All the routes concerning all the major entities with their respective Verbs

The above image shows all the routes that exist for the API, with the verbs they expect and any route parameters.

backend/routes/routes.go +9

```
1  1   package routes
2  2
3  3   import (
   4 +     candidates "github.com/PS-Wizard/ElectOneAPI/api/Candidates"
4  5     citizens "github.com/PS-Wizard/ElectOneAPI/api/Citizens"
5  6     elections "github.com/PS-Wizard/ElectOneAPI/api/Elections"
6  7     users "github.com/PS-Wizard/ElectOneAPI/api/Users"
7  8     "github.com/gofiber/fiber/v2"
8  9   )
9  10
10 11  func HandleRoutes(app *fiber.App) {
11 12    // Citizen Routes:
12 13    app.Get("/api/secure/citizens/:id", citizens.HandleSearch)
13 14    app.Get("/api/secure/citizensPaginated/:offset", citizens.HandleGet)
14 15    app.Post("/api/secure/citizens", citizens.HandleCreate)
15 16    app.Put("/api/secure/citizens/:id", citizens.HandleUpdate)
16 17    app.Delete("/api/secure/citizens/:id", citizens.HandleDelete)
17 18
18 19    // User Routes
19 20    app.Get("/api/secure/user/:id", users.HandleGetUser)
20 21    app.Get("/api/secure/usersPaginated/:offset", users.HandleGetUsersPaginated)
21 22    app.Post("/api/secure/user", users.HandleCreateNewUser)
22 23    app.Put("/api/secure/user/:id", users.HandleUpdateUserDetails)
23 24    app.Delete("/api/secure/user/:id", users.HandleDeleteUser)
24 25
   26 +    // Election Routes
25 27    app.Get("/api/secure/election/:id", elections.HandleGetElection)
26 28    app.Get("/api/secure/electionsPaginated/:offset", elections.HandleGetElectionsPaginated)
27 29    app.Post("/api/secure/election", elections.HandleCreateNewElection)
28 30    app.Put("/api/secure/election/:id", elections.HandleUpdateElectionDetails)
29 31    app.Delete("/api/secure/election/:id", elections.HandleDeleteElection)
   32 +
   33 +    //Candidate Routes
   34 +    app.Get("/api/secure/candidate/:id", candidates.HandleGetCandidate)
   35 +    app.Get("/api/secure/candidatesPaginated/:offset", candidates.HandleGetCandidatesPaginated)
   36 +    app.Post("/api/secure/candidate", candidates.HandleCreateCandidate)
   37 +    app.Put("/api/secure/candidate/:id", candidates.HandleUpdateCandidate)
   38 +    app.Delete("/api/secure/candidate/:id", candidates.HandleDeleteCandidate)
30 39  }
```

Fig 4: Screenshot of the updated routes after creation of said APIs.

The above image shows newly added additions in the routes.go file where all the routes are configured.

The proof of completion of said tasks can be found under commit 277425b. The relevant routes are also available in a postman collection here

Week 3: Integrating User & Admin Sign Up & Login , Creating Admin Dashboard, POC Authorization

- Minimal POC Client side route protection based on the existence of tokens.
- Created temporary header-token approach as a way to authorize certain admin-privilege actions for API requests

- Integrated Admin’s Login & Signup Page with Backend
- Integrated User’s Login & Signup Page with the Backend
- Created frontend admin views for:
    - Creating , Reading, Updating, Deleting, Searching for Citizen’s tables
    - Creating , Reading, Updating, Deleting, Searching for Election’s table
    - Creating , Reading, Updating, Deleting, Searching for Candidate’s tables
    - Creating , Reading, Updating, Deleting, Searching for User’s tables

backend/routes/routes.go +8

```
@@ -1,6 +1,7 @@
 package routes
+
 import (
+    auth "github.com/PS-Wizard/ElectOneAPI/api/Auth"
     candidates "github.com/PS-Wizard/ElectOneAPI/api/Candidates"
     citizens "github.com/PS-Wizard/ElectOneAPI/api/Citizens"
     elections "github.com/PS-Wizard/ElectOneAPI/api/Elections"
@@ -36,4 +37,11 @@ func HandleRoutes(app *fiber.App) {
     app.Post("/api/secure/candidate", candidates.HandleCreateCandidate)
     app.Put("/api/secure/candidate/:id", candidates.HandleUpdateCandidate)
     app.Delete("/api/secure/candidate/:id", candidates.HandleDeleteCandidate)
+
+    // Admin Login Routes:
+    app.Post("/api/admin/signup", auth.HandleCreateAdmin)
+    app.Post("/api/admin/login", auth.HandleAdminLogin)
+
+    app.Post("/api/userlogin", auth.HandleUserLogin)
+
 }
```

Fig 5: Updated routes after completing said tasks.

The above image shows additional routes added in the routes.go file concerning admin & user login / signup

backend/routes/middleware.go +34 -5

```
@@ -3,18 +3,47 @@ package routes
 import (
     "log"

+    auth "github.com/PS-Wizard/ElectOneAPI/api/Auth"
     "github.com/gofiber/fiber/v2"
+    "github.com/golang-jwt/jwt/v5"
 )

 func TokenValidationAdmin(ctx *fiber.Ctx) error {
-    token := ctx.Get("Authorization")
+    tokenString := ctx.Cookies("admin_token")
+    if tokenString == "" {
+        return ctx.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
+            "error": "Authorization cookie is missing",
+        })
+    }
+
+    token, err := jwt.Parse(tokenString, func(token *jwt.Token) (any, error) {
+        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
+            return nil, fiber.NewError(fiber.StatusUnauthorized, "Invalid token signing method")
+        }
+        return auth.JWT_SECRET, nil
+    })

-    if token != "Bearer adminsecrettoken" {
-        log.Println("Invalid Token")
+    if err != nil || !token.Valid {
+        log.Println("Invalid or expired token")
         return ctx.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
-            "error": "Unauthorized access, invalid token",
+            "error": "Invalid or expired token",
         })
     }
+
     return ctx.Next()
 }
```

Fig 6: POC middleware setup.

The above image shows the proof of concept authorization for using the API, it checks if there is a “Authorization” token with the value of “Bearer adminsecrettoken” as the means for authorization to use the API.

electoneui/src/routes/users/login/+page.svelte +51

```
@@ -0,0 +1,51 @@
+ <script>
+   let citizenID = "";
+   let password = "";
+   let error = "";
+
+   const login = async () => {
+     error = "";
+     if (!citizenID || !password) {
+       error = "Citizen ID and password are required";
+       return;
+     }
+
+     try {
+       const res = await fetch("https://localhost:3000/api/userlogin", {
+         method: "POST",
+         headers: { "Content-Type": "application/json" },
+         body: JSON.stringify({ citizenID, password }),
+       });
+
+       if (!res.ok) throw new Error("Invalid credentials");
+     } catch (err) {
+       error = err.message;
+     }
+   };
+ </script>
+
+ <div
+   class="flex flex-col items-center justify-center min-h-screen bg-gray-100 p-4"
+ >
+   <div class="card w-96 bg-white shadow-lg p-6 rounded-lg">
+     <h2 class="text-xl font-semibold text-center mb-4 text-black">Login</h2>
+     <input
+       type="text"
+       placeholder={error || "Citizen ID"}
+       bind:value={citizenID}
+       class="input bg-white text-black input-bordered w-full mb-2 placeholder-red-500"
+     />
+     <input
+       type="password"
+       placeholder={error && !citizenID ? "Password" : "Password"}
+       bind:value={password}
+       class="input bg-white text-black input-bordered w-full mb-4 placeholder-red-500"
+     />
+     <button on:click={login} class="btn btn-primary w-full">Login</button>
+     <a
+       href="/forgot-password"
```

Fig 7: Integrating Login for the user

The above image shows the login page for the user, integrated with the backend by an asynchronous function that calls the API.

24 files changed +663 -662 lines changed

```
2   -   export let data;
```

electoneui/src/routes/admin/citizens/+page.svelte

```
2   +   import { onMount } from "svelte";
3   +   import {
4   +     GetCitizens,
5   +     DeleteCitizen,
6   +     UpdateCitizen,
7   +     CreateCitizen,
8   +   } from "./api.js";
9   +
10  +   let citizens = [];
11  +   let newCitizen = {
12  +     citizenID: "",
13  +     fullName: "",
14  +     dateOfBirth: "",
15  +     placeOfResidence: "",
16  +   };
17  +   let editingCitizen = null;
18  +
19  +   async function loadCitizens() {
20  +     citizens = await GetCitizens();
21  +     console.log(citizens);
22  +   }
23  +
24  +   async function removeCitizen(id) {
25  +     await DeleteCitizen(id);
26  +     loadCitizens();
27  +   }
28  +
29  +   async function addCitizen() {
30  +     await CreateCitizen(newCitizen);
31  +     newCitizen = {
32  +       citizenID: "",
33  +       fullName: "",
34  +       dateOfBirth: "",
35  +       placeOfResidence: "",
36  +     };
37  +     loadCitizens();
38  +   }
39  +
40  +   async function saveCitizen() {
41  +     if (editingCitizen) {
42  +       await UpdateCitizen(editingCitizen.citizenID, editingCitizen);
43  +       editingCitizen = null;
44  +       loadCitizens();
45  +     }
46  +   }
47  +
48  +   onMount(loadCitizens);
```

Fig 8: Admin Dashboard For Managing Citizens.

The above image shows 1 out of 4 table views for the admin. This particular image shows the view for admin dashboard for managing the citizens table.

Relevant work can be found under commits 6b1fea5, eb614c1, for admin relevant tasks, 198e734 for API relevant tasks and under commit 8226d35, for client side relevant tasks.

Week 4: Protected Routes, Redis, HTTPS

- Replaced temporary header-token approach with proper JWT-based authentication

- For authorization, set up the JWT token to have different claims for user and admin roles, acting as our main way to distinguish between these 2 roles.

- Authorization is enforced via middleware that intercepts incoming requests, validates the provided token for authenticity & expiration, and checks if the token contains the necessary claims to perform the requested action. Only after passing these checks is the request allowed to proceed.

- Users can authenticate, cast votes, view their own profile, and browse available elections, candidates, and citizen records (read-only). They also receive real-time vote updates via WebSockets.

- Admins have full system access, an admin can create, read, update, and delete users, citizens, candidates, and elections. The only action they aren't authorized to do is to cast a vote.

- Integrated Redis for atomic vote count operations

- Created Vote Increment Endpoint to register votes

- Set Up a go-routine to act as the subscriber for redis pub/sub

- Added websocket support to publish live changes from the subscriber.

- Did Basic stress testing to validate Redis and WebSocket performance under load, ensuring their viability as a POC for real-time vote updates.

- Created A Self Signed Certificate from mkcert to port the entire application to HTTPS to bypass CORS issues.

```
13      func HandleRoutes(app *fiber.App) {
14          // Citizen Routes:
        -   app.Get("/api/secure/citizens/:id", citizens.HandleSearch)
        -   app.Get("/api/secure/citizensPaginated/:offset", citizens.HandleGet)
        -   app.Post("/api/secure/citizens", citizens.HandleCreate)
        -   app.Put("/api/secure/citizens/:id", citizens.HandleUpdate)
        -   app.Delete("/api/secure/citizens/:id", citizens.HandleDelete)
15      +   app.Get("/api/secure/citizens/:id", TokenValidationAdmin, citizens.HandleSearch)
16      +   app.Get("/api/secure/citizensPaginated/:offset", TokenValidationAdmin, citizens.HandleGet)
17      +   app.Post("/api/secure/citizens", TokenValidationAdmin, citizens.HandleCreate)
18      +   app.Put("/api/secure/citizens/:id", TokenValidationAdmin, citizens.HandleUpdate)
19      +   app.Delete("/api/secure/citizens/:id", TokenValidationAdmin, citizens.HandleDelete)
20
21          // User Routes
        -   app.Get("/api/secure/user/:id", users.HandleGetUser)
        -   app.Get("/api/secure/usersPaginated/:offset", users.HandleGetUsersPaginated)
        -   app.Post("/api/secure/user", users.HandleCreateNewUser)
        -   app.Put("/api/secure/user/:id", users.HandleUpdateUserDetails)
        -   app.Delete("/api/secure/user/:id", users.HandleDeleteUser)
22      +   app.Get("/api/secure/user/:id", TokenValidationBoth, users.HandleGetUser)
23      +   app.Get("/api/secure/usersPaginated/:offset", TokenValidationAdmin, users.HandleGetUsersPaginated)
24      +   app.Post("/api/secure/user", TokenValidationAdmin, users.HandleCreateNewUser)
25      +   app.Put("/api/secure/user/:id", TokenValidationAdmin, users.HandleUpdateUserDetails)
26      +   app.Delete("/api/secure/user/:id", TokenValidationAdmin, users.HandleDeleteUser)
27
28          // Election Routes
        -   app.Get("/api/secure/election/:id", elections.HandleGetElection)
        -   app.Get("/api/secure/electionsPaginated/:offset", elections.HandleGetElectionsPaginated)
        -   app.Post("/api/secure/election", elections.HandleCreateNewElection)
        -   app.Put("/api/secure/election/:id", elections.HandleUpdateElectionDetails)
        -   app.Delete("/api/secure/election/:id", elections.HandleDeleteElection)
29      +   app.Get("/api/secure/election/:id", TokenValidationBoth, elections.HandleGetElection)
30      +   app.Get("/api/secure/electionsPaginated/:offset", TokenValidationBoth, elections.HandleGetElectionsPaginated)
31      +   app.Post("/api/secure/election", TokenValidationAdmin, elections.HandleCreateNewElection)
32      +   app.Put("/api/secure/election/:id", TokenValidationAdmin, elections.HandleUpdateElectionDetails)
33      +   app.Delete("/api/secure/election/:id", TokenValidationAdmin, elections.HandleDeleteElection)
34
35          //Candidate Routes
        -   app.Get("/api/secure/candidate/:id", candidates.HandleGetCandidate)
        -   app.Get("/api/secure/candidatesPaginated/:offset", candidates.HandleGetCandidatesPaginated)
        -   app.Post("/api/secure/candidate", candidates.HandleCreateCandidate)
        -   app.Put("/api/secure/candidate/:id", candidates.HandleUpdateCandidate)
        -   app.Delete("/api/secure/candidate/:id", candidates.HandleDeleteCandidate)
36      +   app.Get("/api/secure/candidate/:id", TokenValidationBoth, candidates.HandleGetCandidate)
37      +   app.Get("/api/secure/candidatesPaginated/:offset", TokenValidationBoth, candidates.HandleGetCandidatesPaginated)
38      +   app.Post("/api/secure/candidate", TokenValidationAdmin, candidates.HandleCreateCandidate)
39      +   app.Put("/api/secure/candidate/:id", TokenValidationAdmin, candidates.HandleUpdateCandidate)
40      +   app.Delete("/api/secure/candidate/:id", TokenValidationAdmin, candidates.HandleDeleteCandidate)
41
42          // Admin Login Routes:
43          app.Post("/api/admin/signup", auth.HandleCreateAdmin)
44          app.Post("/api/admin/login", auth.HandleAdminLogin)
45
```

Fig 9: Updated routes after implementing finalized middleware.

The image shows changes in the routes.go file which now contains middlewares to build upon the API authorization logic from earlier. The routes can be seen requiring "TokenValidationAdmin" which makes those routes only accessible to the admin, "TokenValidationBoth" which makes those routes accessible to both the admin and the user.

```
+
+    "github.com/PS-Wizard/ElectOneAPI/api"
+ )
+
+ func incrementVote(candidateID, electionID string) error {
+    key := fmt.Sprintf("votes:%s:%s", candidateID, electionID)
+    log.Printf("Incrementing vote for key: %s", key) // Log the key being used
+    cmd := api.RDB.Incr(api.CTX, key)
+    num, err := cmd.Result()
+    log.Printf("Num: %d", num) // Log the key being used
+    if err != nil {
+       log.Printf("Failed To Increment Vote: %v", err)
+       return err
+    }
+    msg := fmt.Sprintf("Vote Count Updated For Election %s, Candidate %s: %d", electionID, candidateID, num)
+    err = api.RDB.Publish(api.CTX, "voteUpdates", msg).Err()
+    if err != nil {
+       log.Printf("Failed to publish vote count update: %v", err)
+       return err
+    }
+    fmt.Printf("Publisehd vote update:")
+    return nil
+ }
```

Fig 10: Function that handles vote increments utilizing atomic INCR from redis.

The above image shows the main logic that increments votes for a particular candidate using the atomic operation “INCR” provided by redis. It also shows the logic behind how after each vote, the event is published to all the subscribers.

```
+
+ func InitializeRedis(host, port, password string) {
+    addr := fmt.Sprintf("%s:%s", host, port)
+    RDB = redis.NewClient(&redis.Options{
+       Addr:     addr,
+       Password: password,
+       DB:       0,
+    })
+
+    _, err := RDB.Ping(CTX).Result()
+    if err != nil {
+       log.Fatalf("Failed To Connect To Redis: %v", err)
+    }
+    go startSubscriber()
+    log.Println("Redis Connection Established and publisher started")
+ }
+
+ func startSubscriber() {
+    pubsub := RDB.Subscribe(CTX, "voteUpdates")
+    defer pubsub.Close()
+    ch := pubsub.Channel()
+    for msg := range ch {
+       fmt.Printf("Received message: %s\n", msg.Payload)
+    }
+ }
+
+ func CloseRedis() {
+    if RDB != nil {
+       RDB.Close()
+       log.Println("Redis Connection Closed")
+    }
+ }
```

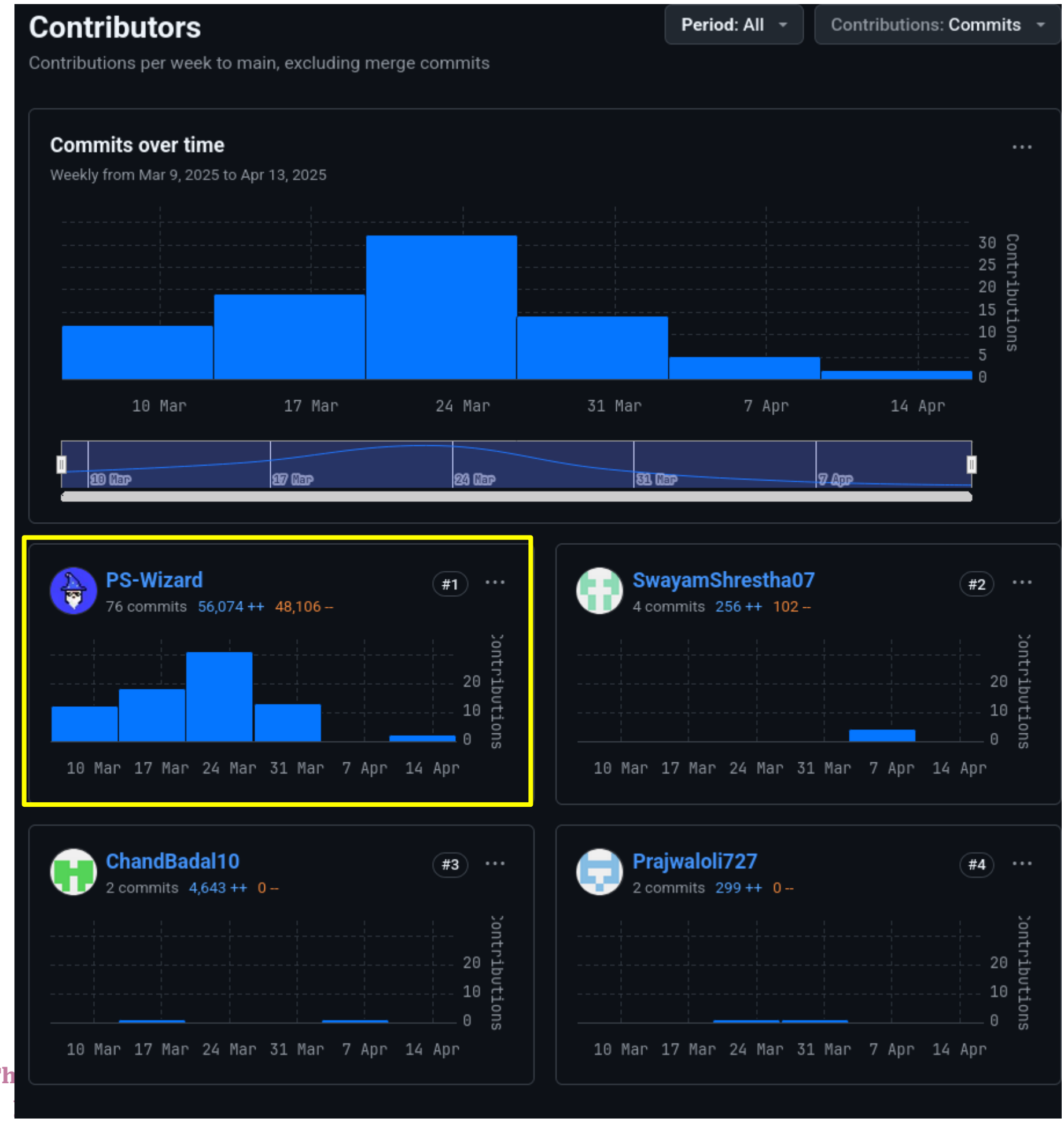
Fig 11: Function to initialize redis

The above image shows the logic of how a subscriber is initiated and how it listens to the events on the "voteUpdates" channel.

For a more in-depth look, commits 3d6fdb2 can be referred for redis relevant tasks , 198e734 for route relevant tasks and f774a13 for https relevant tasks.

Appendix B

Evidence of Use Of Version Control:



Th

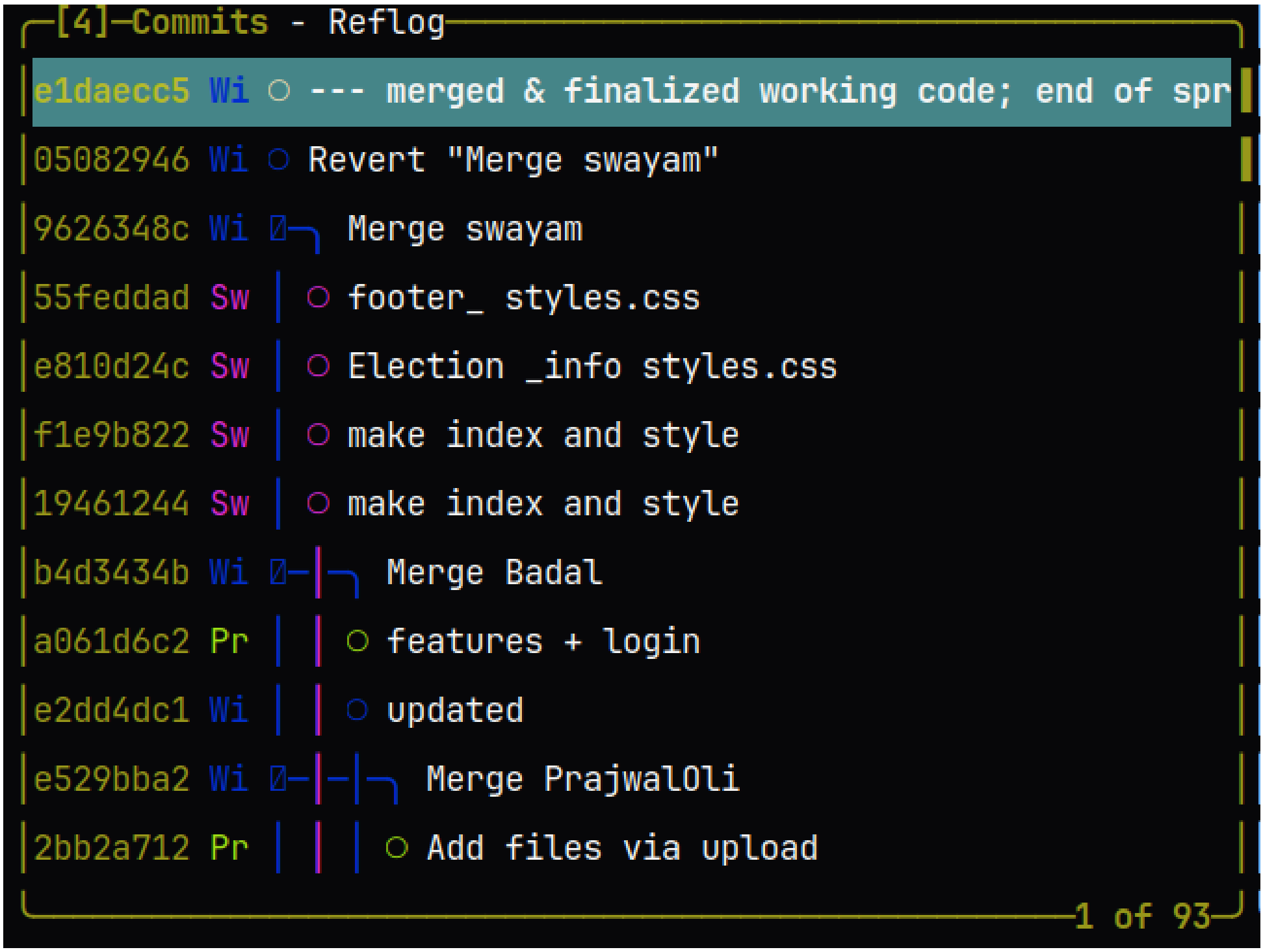


Fig 13: Screenshot of github reflog

The above image shows merging of different branches from other members handled by me.

```
commit 9626348c6fc3de841adcd9d840c84c8e3a64c18e
Merge: b4d3434 55fedda
Author: Wizard <p.swoyam.1@gmail.com>
Date:   Wed Apr 16 15:26:05 2025 +0545

    Merge swayam

commit b4d3434bf2b69f86453e5bf06c11a1c77ace7a1a
Merge: e529bba a061d6c
Author: Wizard <p.swoyam.1@gmail.com>
Date:   Wed Apr 16 15:26:05 2025 +0545

    Merge Badal

commit e529bba2e8c99bd76b0e72d7f48594adc1ce290f
Merge: f4e9d66 2bb2a71
Author: Wizard <p.swoyam.1@gmail.com>
Date:   Wed Apr 16 15:23:32 2025 +0545

    Merge PrajwalOli

commit f4e9d66c6d7ae686bf86607b4ed7886feac0aa73
Merge: 9f35a32 8476bac
Author: Wizard <p.swoyam.1@gmail.com>
Date:   Wed Apr 16 15:23:27 2025 +0545

    Merge Badal
:
```

Fig 14: Screenshot of github logs

The above image shows some of the github logs from our project.

```
| |
| |     Election _info styles.css
| |
| * commit f1e9b822671b6ab31f4600ed688934eed7a7dad1
| | Author: SwayamShrestha07 <np03cs4s240103@heraldcollege.edu.np>
| | Date:   Sun Apr 6 14:09:01 2025 +0545
| |
| |     make index and style
| |
| * commit 194612442d27b4744b7b2542765ae9b0bafac4d5
| | Author: SwayamShrestha07 <np03cs4s240103@heraldcollege.edu.np>
| | Date:   Sun Apr 6 14:08:57 2025 +0545
| |
| |     make index and style
| |
* |   commit b4d3434bf2b69f86453e5bf06c11a1c77ace7a1a
|\ \  Merge: e529bba a061d6c
| | | Author: Wizard <p.swoyam.1@gmail.com>
| | | Date:   Wed Apr 16 15:26:05 2025 +0545
| | |
| | |     Merge Badal
| | |
| * | commit a061d6c26b457d95728608e79becfff076d26190 (origin/features-page)
| | | Author: Prajwaloli727 <nepaligamer165@gmail.com>
| | | Date:   Fri Mar 28 11:57:14 2025 +0545
| | |
| | |     features + login
:
```

Fig 15: Screenshot of git log –graph

The above image shows the github branches from our project

Evidence Of Good Communication And File Sharing:

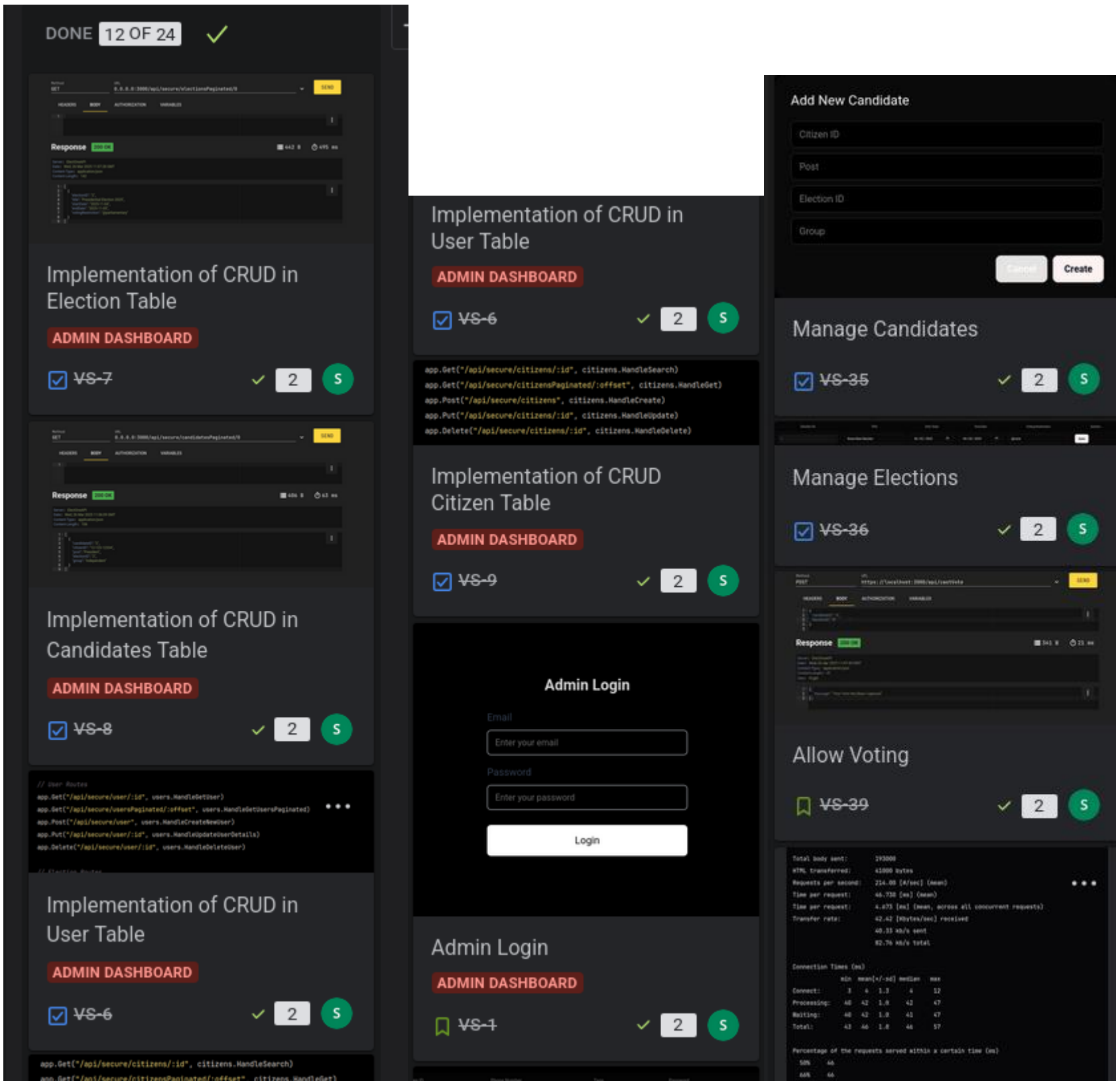


Fig 16: Screenshots of my jira tickets.

The above screenshots show my jira tickets that I've completed.

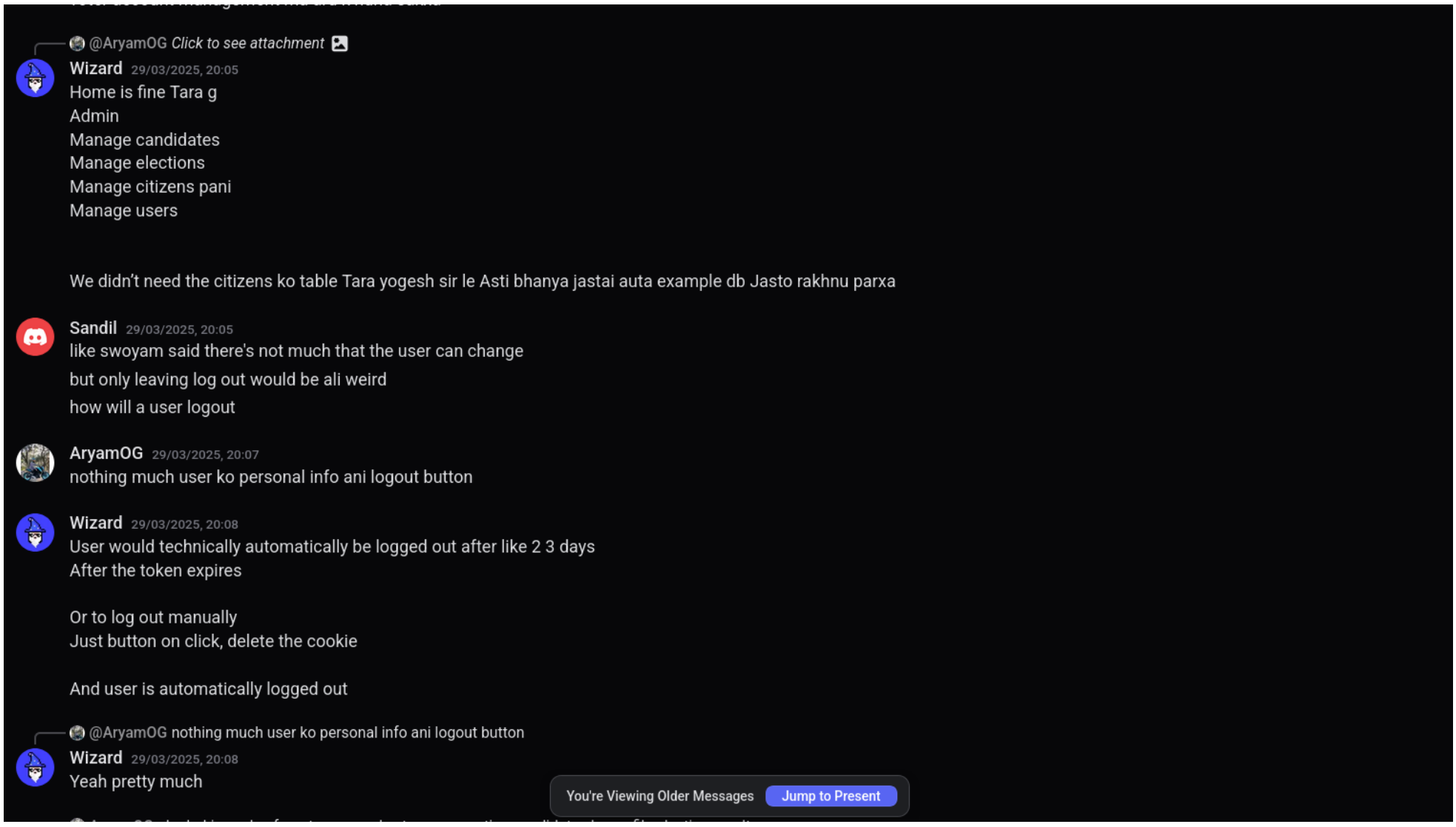


Fig 17: Discord Chat Discussion's Screenshot

The above image shows a discussion on what needs to be done and how user log out could be possibly handled.

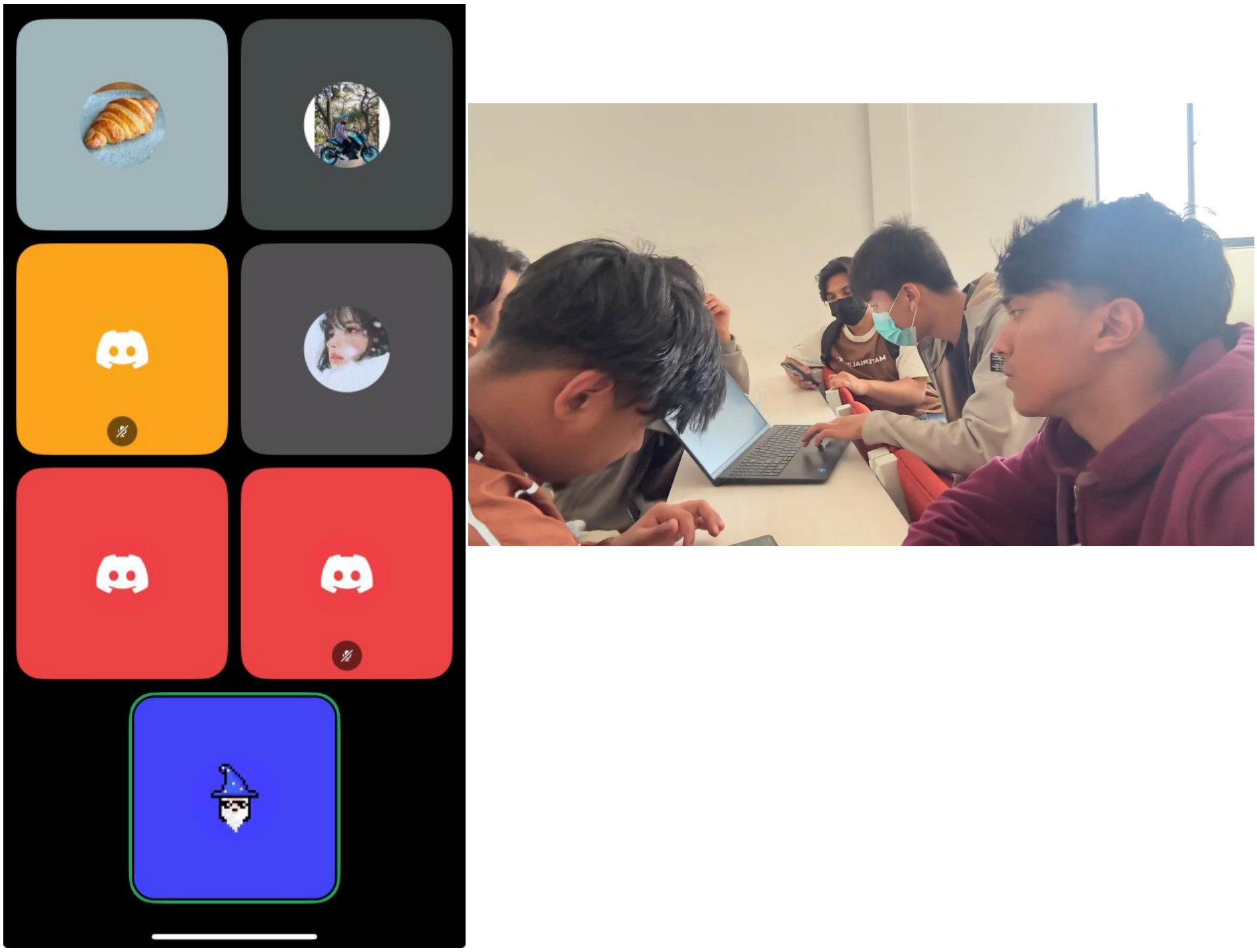
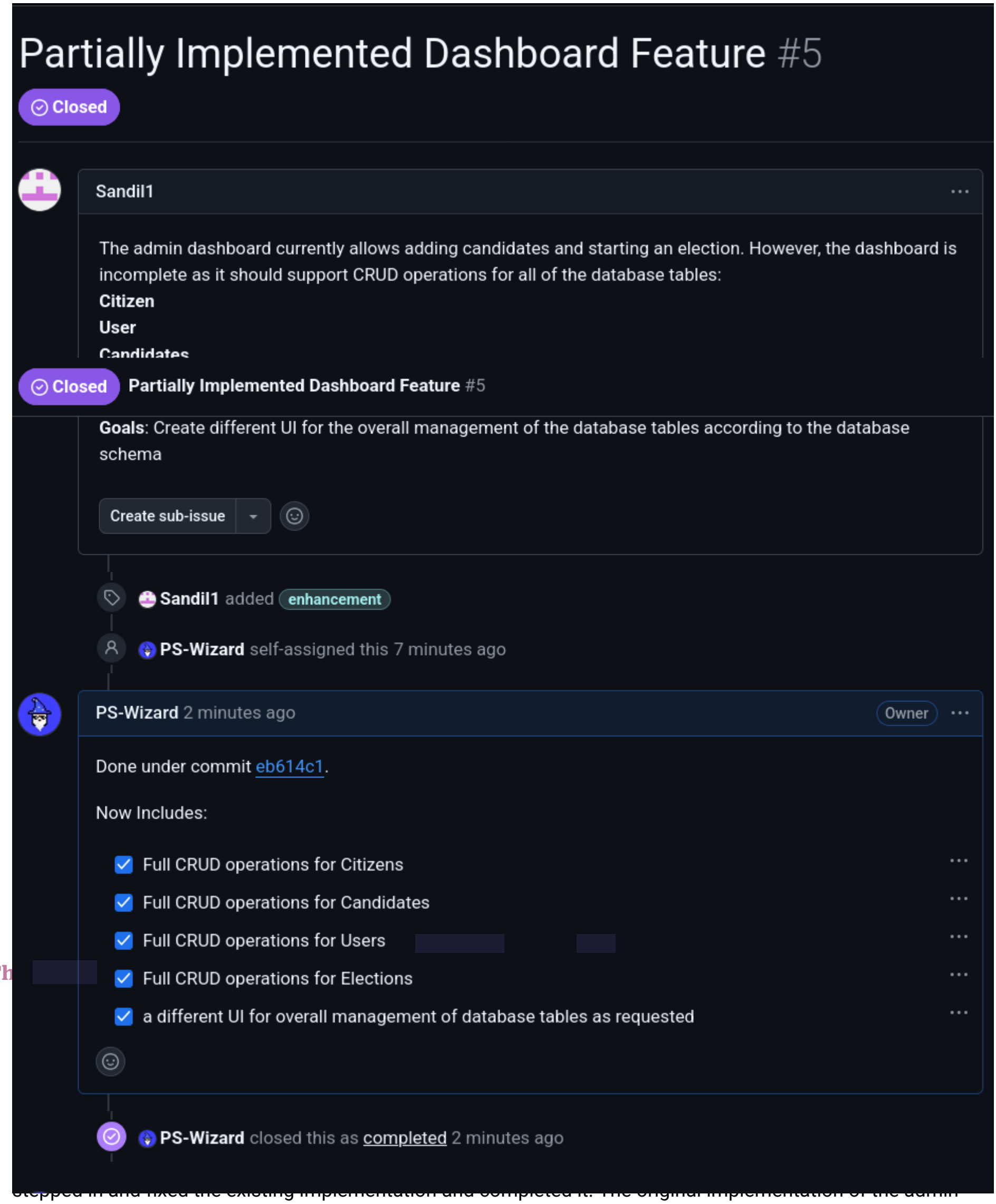


Fig 18: Team Meetings Both Virtual & Physical

The above images show meetings conducted both virtually on discord and physically.

Evidence Of Issue Tracking



Th

stepped in and fixed the existing implementation and completed it. The original implementation of the admin dashboard can be found here, with the updated version located here; these changes were introduced in commit eb614c1

In addition, I encountered and fixed several other issues that were not explicitly tracked or documented:

- CORS & Cookie Issue On Firefox: Discovered that Firefox was blocking cookies due to stricter SameSite policies, while in Chrome it was working fine. Had to switch to HTTPS to resolve it. This fix is introduced in commit f774a13

- Server-Sent Events Randomly Failing: Initially implemented SSE for live vote updates, but the connection was randomly dropping. Switched to websockets for better reliability, despite it adding complexity and possibly an overhead of a full duplex communication channel. This fix was introduced in 87a6e3f

- Unauthorized Dashboard Access: Users could access the dashboard without logging in. I implemented a client side token existence check that provides / limits access to the dashboard. This fix was introduced in 198e734

- Frontend Integration gaps: The frontend wasn't initially connected to the backend; login, signup, admin dashboard lacked API integration. I took the lead on integrating the backend and made several UI/UX refinements in the process. While there is no one commit that captures the entire scope, the differences are noticeable when comparing earlier versions of the frontend code from different branches to the final integrated main branch.

These evidence shows both proactive and reactive issue solving from my part.

Moving on, technical issues aside, we really felt some non-technical issues arise during this sprint. While individual members contributed where possible, the team coordination was lacking. Initially, the project manager struggled with task tracking; multiple jira boards were created leading to confusion on where to log progress. One of the members ended up working off outdated boards. There was also a consistent lack of communication; when issues were found they weren't relayed properly between the PM and the BA, leading to incorrect ticket descriptions. This resulted in some of the members coding up their interpretation of the ticket, which inherently was wrong and caused their effort and time to be wasted. Overall, it was a learning experience working in a large team ; but this definitely highlighted the importance of strong leadership and better communication; something I hope to improve in the next sprint.

Evidence of Continuing Personal Development:

Over the course of this sprint, I had the chance to work with several technologies I had heard about before but never got around trying. One of the biggest learning experiences was working with Turso and Redis. I've always only worked with databases such as MySQL, PostgreSQL, Sqlite etc. In all of those cases, I've had to go through the hassle of installing them locally, which, being a nuisance in itself; sharing the database was even worse. Turso, being a distributed edge database, that is hosted remotely by default made it so that I could share the database to everyone and everyone will have the data I worked with and all can make any changes directly. Furthermore, Turso helped me understand how global replication and performance tuning can be approached in practice. Overall, Turso really changed how I think about modern databases, and how *"per user database"* is actually a viable strategy nowadays. It certainly has changed the way I will be approaching databases from now on.

Moving on to tackling the counter problem ; it seemed simple on paper, just incrementing the counter, but turns out even that simple task gets a bit complicated when millions of people are potentially going to be using the counter concurrently. I learnt that traditional databases

bottleneck quickly in use cases like this, that's where redis came in. I learnt that redis provides atomic operations for simple things such as incrementing a counter, which is great for our use case. Furthermore, it was a breath of fresh air not having any schema, any relations, just a simple “key:value” based database. Also, its pub/sub model ended up being key for publishing vote counts to all clients in real time. Also, I’ve later realized that, perhaps updating the client on “*each and every vote*” is not that good, instead I’m thinking of updating the client every X amount of votes; in batches. This reduces overhead for both the client and the server. This change will be made in the next sprint. Overall, Redis ended up being one of the most eye-opening tools I learnt this sprint, it taught me the importance of picking the right tool and how even simple things at scale become actual big challenges.

This was also my first time working with SvelteKit, and it was honestly a great experience. Its developer experience is straightforward, and it allowed us to build a reactive, responsive UI very quickly without the overhead that typically comes with other frameworks. I learned how to use its routing, SSR, and data-fetching paradigms effectively. With that being said, in the next sprint, we will probably switch to Svelte SPA for routing. Sveltekit is meant for a full stack svelte application; which in our case is kind of redundant as the backend is already handled in Go. I’ve realized that primarily we are using Sveltekit mainly for routing, which works but, SPA router would provide better experience for the client if routings is all we are doing with anyways; furthermore it helps clean up the codebase real quick, currently navigating the 27 directors is a pain. Overall, using Svelte and Sveltekit helped me understand where each one fits best, and working on it long enough has naturally built fluency in me, which has been a nice bonus.

On the backend side, I expanded my knowledge of JWT authentication and how to build scalable, secure middleware pipelines in Go. I had to handle HTTPS configuration and cookie management, particularly to make the app compatible across browsers with strict security policies like Firefox. This forced me to understand how to deal with CORS, SameSite policies, and the transition to HTTPS; all of which were new areas for me.

Another major area of growth was working with WebSockets. I had dabbled in them before, but this project required me to integrate them into an app that pushes live updates to possibly thousands of users. I explored how Redis can act as the bridge for pushing events across multiple clients via pub/sub, which gave me a deeper understanding of distributed systems and event driven architectures. That said, I now see how pushing an update per vote might be too much at scaling, and I’m considering batching updates (as in every X votes or seconds) next sprint.

Even though we didn’t use microservices yet, I’ve been studying them and plan to convert this MVP into a microservice architecture in the next sprint. This way, it increases fault tolerance; which I believe is crucial for an election system; even in cases where log-in fails, already logged in users should still be able to use the product, and this isolation can be achieved with microservices. Furthermore, I’ve been exploring Go contexts , which I’ve realized are essential for managing request timeouts and making systems responsive.

In retrospect I’ve also now realised things that I could’ve done better. For one, I probably should’ve gone with a microservice focused architecture from the start. Another thing is that, instead of working from inside out (starting from the API) I should’ve gone outside in. Had I started with the load balancer / gateway first I wouldn’t be in the current situation where the codebase is all aligned around CORS between localhost and the API without the gateway even in the picture. Changing that will now be a bit more complicated. I also wish I had given caching more attention, even though Turso handles our load fine, implementing something like a simple cache-aside/lazy-loading would’ve been a good proof of concept, just as a means to clear out where things would fall in place.

All in all, this sprint has allowed me to experiment with a bunch of technologies I've wanted to try but always put off on. This has been very beneficial for my personal development, and I'm

excited to tackle the next sprint with all the things I've learned. In the references section, I've mentioned the resources I've referred to and used as a guide throughout this sprint:

References

Adhikari, P., 2024. Implementing WebSockets in Golang: Real-Time Communication for Modern Applications. *WiseMonks*. [online] 14 Aug. Available at: <https://medium.com/wisemonks/implementing-websockets-in-golang-d3e8e219733b> [Accessed 20 March 2025].

Anon. 2025a. *ab - Apache HTTP server benchmarking tool*. [online] Apache HTTP Server Version 2.4. Available at: <https://httpd.apache.org/docs/2.4/programs/ab.html> [Accessed 2 May 2025].

Anon. 2025b. *Getting Started*. [online] Available at: <https://golang-jwt.github.io/jwt/> [Accessed 12 March 2025].

Anon. 2025c. *go-redis guide (Go)*. [online] Docs. Available at: <https://redis.io/docs/latest/develop/clients/go/> [Accessed 22 March 2025].

Anon. 2025d. *Welcome to Svelte • Svelte Tutorial*. [online] Available at: <https://svelte.dev/tutorial/svelte/welcome-to-svelte> [Accessed 21 March 2025].

Anon. 2025e. *What is SvelteKit? • Svelte Tutorial*. [online] Available at: <https://svelte.dev/tutorial/kit/introducing-sveltekit> [Accessed 25 March 2025].

FiloSottile, 2025. *GitHub - FiloSottile/mkcert: A simple zero-config tool to make locally trusted development certificates with any names you'd like*. [online] GitHub. Available at: <https://github.com/FiloSottile/mkcert> [Accessed 29 March 2025].

Maintainers, G.W.T., 2025. *Gorilla, the golang web toolkit*. [online] Available at: <https://gorilla.github.io/> [Accessed 13 March 2025].

Redis, 2023. *Redis - The Real-time Data Platform*. [online] Redis. Available at: <https://redis.io/> [Accessed 15 March 2025].

redis, 2025. *GitHub - redis/go-redis: Redis Go client*. [online] GitHub. Available at: <https://github.com/redis/go-redis> [Accessed 12 March 2025].

Turso, 2025. *Welcome to Turso Cloud*. [online] Turso. Available at: <https://docs.turso.tech/introduction> [Accessed 11 March 2025].

7% Overall Similarity

Top sources found in the following databases:

- 3% Internet database
- Crossref database
- 6% Submitted Works database
- 1% Publications database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	University of Wolverhampton on 2025-04-20 Submitted works	<1%
2	The University of Wolverhampton on 2024-04-22 Submitted works	<1%
3	University of Sydney on 2024-04-27 Submitted works	<1%
4	Aberystwyth University on 2018-04-29 Submitted works	<1%
5	Dundee and Angus College on 2025-03-28 Submitted works	<1%
6	University of Wolverhampton on 2025-04-19 Submitted works	<1%
7	University of Wolverhampton on 2021-04-16 Submitted works	<1%
8	University of Wolverhampton on 2021-04-16 Submitted works	<1%

9	Curtin University of Technology on 2023-10-22 Submitted works	<1%
10	The University of Wolverhampton on 2022-05-15 Submitted works	<1%
11	University of Greenwich on 2024-11-30 Submitted works	<1%
12	University of Wolverhampton on 2025-04-20 Submitted works	<1%
13	The University of Wolverhampton on 2022-05-14 Submitted works	<1%
14	The Open University on 2025-03-24 Submitted works	<1%
15	medium.com Internet	<1%
16	University of Wolverhampton on 2021-04-17 Submitted works	<1%
17	Victoria University on 2020-05-10 Submitted works	<1%
18	University of Wolverhampton on 2021-05-16 Submitted works	<1%
19	University of Wolverhampton on 2025-04-18 Submitted works	<1%
20	University of Wolverhampton on 2021-04-16 Submitted works	<1%