CSIT314 – Software Development Methodologies Group Project Final Report

2025 Q4 (Oct - Dec)

Tutorial Group: T07

Group Name: Crashout

Name	UOW ID	UOW Email	Signature
Foo Kok Tai Justin	8465800	Ktjfg709@uowmail.edu.au	
Eugene Lay Chai Chun	9070783	ecclc650@uowmail.edu.au	
Nur Syafiqah Binte Mustazam	8535759	nsbm983@uowmail.edu.au	
Jiyavudeen Mohamed Haneefa	8496407	jmh687@uowmail.edu.au	
Karissa Angeline Ramos Wong	8877889	karw858@uowmail.edu.au	
Tan Jun Rong Dillon	9093187	jrdt991@uowmail.edu.au	
Sim Zhan Qi	8930831	zqs501@uowmail.edu.au	

GitHub Project Link: https://github.com/Cakebut/CSIT314Crashout
Youtube Demo Link:https://www.youtube.com/watch?v=Y4NMnImlmLA
Taiga Link: https://tree.taiga.io/project/eugenelcc-314-name-project/backlog

Executive Summary

This report provides a comprehensive overview of the Software Development Methodologies group project. It is a documentation of the full lifecycle of the system's development and outcomes. It starts with fully compiled user stories defining the system's core functionalities in the point of view of each end user.

The project was based on four structured sprints. Each sprint was guided by the "4+1" architectural view model. For each sprint, a suite of design artefacts were produced. This includes use case diagrams, detailed use case descriptions, Boundary-Control-Entity (BCE) diagrams, sequence diagrams and wireframes. These diagrams collectively capture both functional and dynamic aspects of the system. Every user story comes with a corresponding test case. This ensures that every function has been validated from the user's perspective.

The program design segment additionally describes the system's architecture through Unified Modeling Language (UML) diagrams. This illustrates both the database schema and the program's class hierarchy. CI/CD pipelines were implemented using GitHub to uphold code quality, reliability and continuous integration. The details of which have been documented in this report.

The coordination and the progress of the project were managed systematically. Meeting minutes were done to record key decisions and milestones achieved. We used Gantt charts and Taiga to visually track our timeline, sprint goals and iterative progress.

This report concludes with an analytical reflection on the project. It addresses the ethical conditions which are related to both the new development process and final product. The report also discusses how data-driven software development principles allowed the decision making, strategy optimization and overall evolution of the system.

Member Contributions

Student Name	Contributions	Contribution Percentage
Foo Kok Tai Justin	Project Manager / Backend Developer / Database Architect - Project planning and Coordination - User Stories, Use case diagram & Description - Backend API development - Sprint Planning and Task allocation - Live Demo presentation lead - Documentation Review and Approval	100%
Eugene Lay Chai Chun	(DevOps/ CI-CD Engineer)/ Backend Developer/ Database Architect - Continuous Integration & deployment Setup (Github/Actions/Docker/PostgreSQL) - Backend API development & Integration - Backend debugging and Log monitoring - Use case, BCE & Sequence Diagram - CI Pipeline Maintenance - Data modeling & Validation - Backend API Integration with database - Performance optimization & Query testing	100%
Nur Syafiqah Binte Mustazam	Front-end / Documentation/ Backend Support - UI Implementation (React Native) - User flow design & Wireframes (FIGMA) - Frontend & Backend Integration testing - User acceptance testing (UAT) - Documentation (User Manual, Meeting Minutes)	100%
Jiyavudeen Mohamed Haneefa	Backend Developer / Documentation - Version control & Branch management - Backend development for CSR Rep - Use case, BCE & Sequence Diagram - Documentation (Sprints, Meeting Minutes)	100%
Karissa Angeline Ramos Wong	UIUX Designer /Documentation - Wireframe & Prototype design (FIGMA) - UX Flow & Accessibility review - Design style guide development - UAT Design feedback & Iterations - Documentation (Design Specification, WireFrame)	100%
Tan Jun Rong Dillon	Backend Developer / Documentation - Backend development for Platform Manager - Backend API development - Documentation (Use Case Diagrams, Meeting Mins)	100%
Sim Zhan Qi	Taiga Administrator/ Documentation - Project board Management (Taiga) - User acceptance testing support - Project Report layout & Proofreading - Documentation (Use Case Description & BCE & Sequence Diagram)	100%

Table of Contents

1. Introduction

This report presents the design, development, and evaluation of a Corporate Social Responsibility (CSR) Volunteering Matching System. The aim of the project is to address the need for an efficient digital platform connecting the CSR Representatives with Persons-In-Need (PIN) who seek assistance. While doing this, Platform Managers and User Administrators are also enabled to oversee operations, generate reports and manage user activities.

Using an agile scrum framework, in a team setting, this project follows a complete software development lifecycle. Beginning from requirements elicitation and analysis for design, implementation, testing and deployment. The system's entire design is based on the BCE framework and the "4+1" architectural model. This provides both structural and behavioural perspectives of the software.

During the development process, there was high emphasis on Test-Driven Development (TDD) and Continuous Integration/Continuous Delivery (CI/CD) practices. This was done to ensure quality and consistency. The team also used Github for automated testing and deployment. Taiga was used to facilitate sprint management and backlog tracking. Taiga also has evidence of the iterative team progress.

This report also consists of the team's collaborative workflow. This includes user stories, UML design diagrams, testing strategies and project management artefacts. Some of which are meeting minutes, sprint reviews and a Gantt chart. The team discussed ethical and professional considerations like data privacy, transparency and equitable service delivery. With this, there is a proposed data-driven enhancement to improve the matching efficiency and optimize the platform to its peak.

2. User Stories

2.1. User Admin

No.	USER STORY
1	As a User Admin, I want to log in so that I can access my admin features.
2	As a User Admin, I want to log out so that I can securely end my session.
3	As a User Admin, I want to create a user account so that new users can join the platform.
4	As a User Admin, I want to create new user roles (PIN, CSR Rep, User Admin, Platform Manager) so that users can be allocated to their respective roles
5	As a User Admin, I want to view user accounts so that I can see user information
6	As a User Admin, I want to update user accounts so that I can keep user information accurate.
7	As a User Admin, I want to suspend user accounts so that I can restrict access when needed.
8	As a User Admin, I want to disable user roles so that I can restrict user access if necessary.
9	As a User admin, I want to search for user accounts so that I can locate specific users.
10	As a User Admin, I want to view user roles so that I can access profile information.
11	As a User Admin, I want to export user activity log maintain audit records for compliance
12	As a User Admin, I want to search for user roles so that I can find specific user profiles.
13	As a User Admin, I want to approve user's new password changes so that user can re-access their account
14	As a User Admin, I want to delete roles so that I can remove unnecessary roles from the system.
15	As a User Admin, I want to generate a list of inactive users so that I can review unused accounts.

16	As a User Admin, I want to export user data to a CSV report so that I can keep a backup of the system's records.
17	As a User Admin, I want to receive notifications for failed login attempts so that I can detect potential security breaches.
18	As a User Admin, I want to view system activity logs so that I can audit user actions for accountability.

2.2. Person-in-Need (PIN)

No.	USER STORY
1	As a PIN, I want to log in so that I can manage my requests securely.
2	As a PIN, I want to log out so that I can safely end my sessions
3	As a PIN, I want to create help requests (e.g., transport, wheelchair assistance, daily needs) so that I can receive the necessary volunteer support.
4	As a PIN, I want to view all my requests so that I can monitor their progress and status.
5	As a PIN, I want to update my existing requests so that I can make changes when my needs change.
6	As a PIN, I want to delete a request so that I can remove requests that are no longer needed.
7	As a PIN, I want to search for a specific help request so that I can look up for specific request I need
8	As a PIN, I want to view the number of CSR Reps viewing my request so that I can understand the level of interest.
9	As a PIN, I want to view the number of times my request is shortlisted by CSR Reps so that I can gauge how likely I am to receive help.
10	As a PIN, I want to view through my past completed matches, filtered by service type or date range, so that I can find specific past help received.
11	As a PIN, I want to accept or cancel a CSR's offer to help my request so that I can control who provides the assistance I need.
12	As a PIN, I want to filter requests by status (e.g., available, pending, completed) so that I can track progress easily.

13	As a PIN, I want to mark a request as completed so that I can close cases that have been fulfilled.
14	As a PIN, I want to receive notifications when a CSR shortlists my request so that I know someone is interested in helping me.
15	As a PIN, I want to receive notifications when a CSR confirms to help so that I can prepare for the service.
16	As a PIN, I want to download my past service history so that I can keep a record for personal reference.
17	As a PIN, I want to edit my profile information (contact number, address, etc.) so that CSR volunteers can reach me easily.
18	As a PIN, I want to delete my account so that I can withdraw from the platform if I no longer need assistance.
19	As a PIN, I want to rate my experience with the CSR Rep so that I can give user feedback.

2.3. Corporate Social Responsibility (CSR) Representative

No.	**USER STORY**
1	As a CSR Rep, I want to log in so that I can access my volunteer activities securely.
2	As a CSR Rep, I want to log out so that I can securely end my session.
3	As a CSR Rep, I want to search for PIN requests so that I can find people in need of services that match my company's volunteering capacity.
4	As a CSR Rep, I want to view details of PIN requests so that I can understand the type of help required.
5	As a CSR Rep, I want to shortlist PIN requests so that I can review them later.
6	As a CSR Rep, I want to search through my shortlist so that I can easily locate a particular PIN request I saved earlier.
7	As a CSR Rep, I want to view my shortlist so that I can manage potential volunteer opportunities efficiently.
8	As a CSR Rep, I want to filter my completed volunteer services by service type or date, so that I can track past volunteer efforts.

9	As a CSR Rep, I want to view how many PINs I have successfully helped, so that I can measure my impact.
10	As a CSR Rep, I want to filter PIN requests by location so that I can find nearby volunteer opportunities.
11	As a CSR Rep, I want to filter PIN requests by urgency level so that I can prioritize critical cases.
12	As a CSR Rep, I want to receive notifications when a PIN accepts my volunteer request so that I can respond to it promptly.
13	As a CSR Rep, I want to assign a volunteer from my company to a PIN request so that the volunteer can contribute to the volunteer project.
14	As a CSR Rep, I want to update the status of a volunteer assignment so that PINs and platform Managers can track progress.
15	As a CSR Rep, I want to upload completion proof (e.g., photos) so that service delivery can be verified by the PIN.
16	As a CSR Rep, I want to view analytics of my volunteer impact (e.g., total hours or cases served) so that I can report CSR outcomes to my company.

2.4. Platform Manager

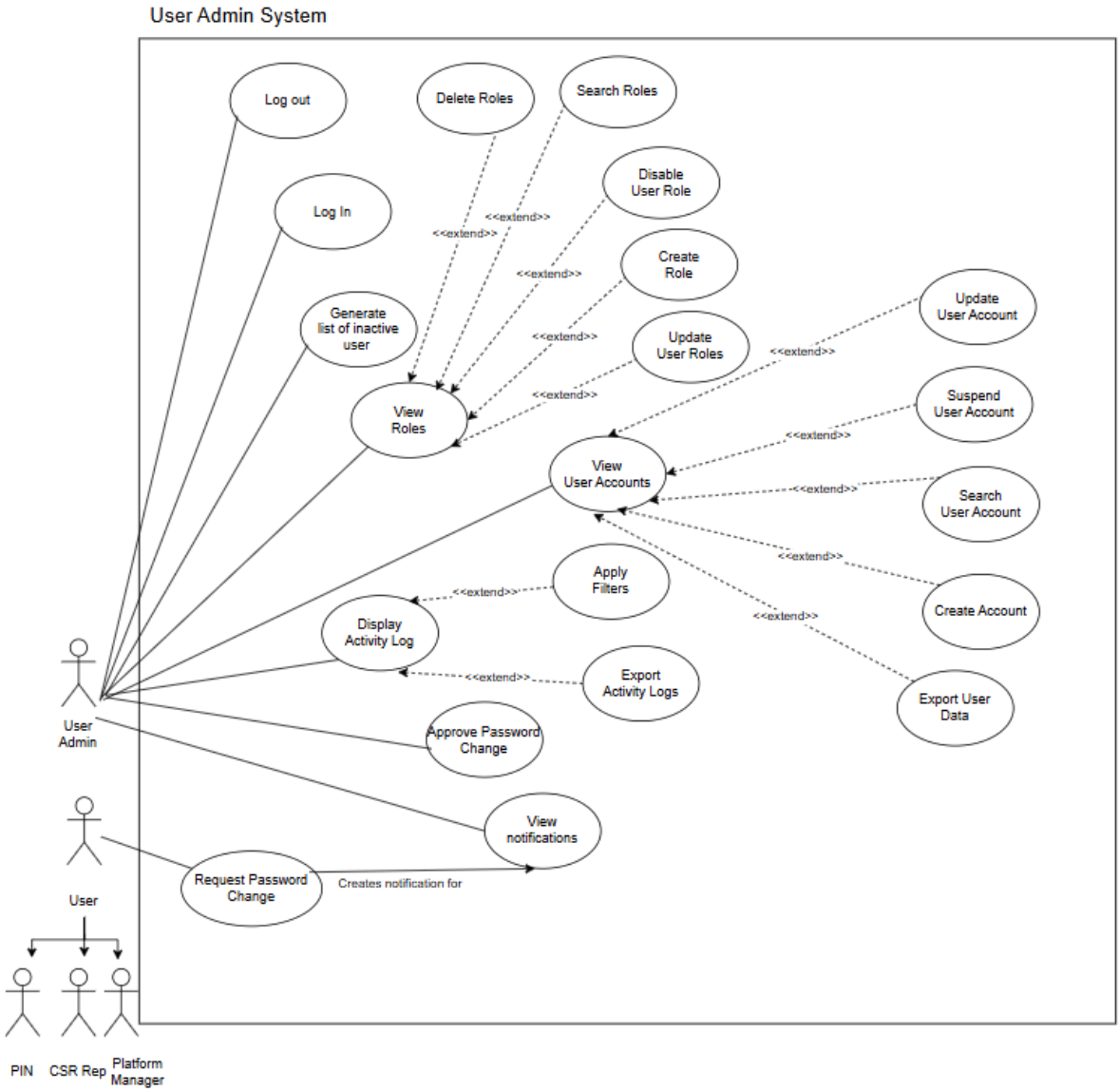
No.	**USER STORY**
1	As a Platform Manager, I want to create categories of volunteer services (e.g., medical, transport, household help) so that I can ensure the platform supports all relevant community needs.
2	As a Platform Manager, I want to view current volunteer service categories so that I can monitor the available services offered to Persons-In-Need.
3	As a Platform Manager, I want to update volunteer service categories so that I can keep the platform's offerings accurate and relevant.
4	As a Platform Manager, I want to delete outdated volunteer service categories so that I can remove services that are no longer offered or needed.
5	As a Platform Manager, I want to search service categories so that I can quickly locate and manage specific types of volunteer services.

6	As a Platform Manager, I want to generate daily reports so that I can view the daily statistics of the volunteer services requests.
7	As a Platform Manager, I want to generate weekly reports so that I can view the weekly statistics of the volunteer services requests.
8	As a Platform Manager, I want to generate monthly reports so that I can view the monthly statistics of the volunteer services requests.
9	As a Platform Manager, I want to log in to the system so that I can securely access and manage platform operations.
10	As a Platform Manager, I want to log out of the system so that I can safely end my administrative session and protect platform data.
11	As a Platform Manager, I want to generate custom reports based on date range or service types so that I can view custom statistics of the volunteer services requests.
12	As a Platform Manager, I want to view the number of active PINs and CSRs so that I can assess engagement levels.
13	As a Platform Manager, I want to view system-wide statistics (e.g., total matches, pending requests) so that I can monitor platform health.
14	As a Platform Manager, I want to view feedback from both PINs and CSRs so that I can identify areas for service improvement.
15	As a Platform Manager, I want to send announcements to all users so that I can communicate important updates (e.g., maintenance, events).
16	As a Platform Manager, I want to review system logs to ensure data integrity and detect unauthorized actions.
17	As a Platform Manager, I want to assign moderators to handle specific service categories so that management tasks can be distributed.
18	As a Platform Manager, I want the system to automatically remove help requests that have been open for more than a certain number of days so that outdated requests do not clutter the platform.

3. Diagrams and Use Case Description

3.1. User Admin

Use Case Diagram

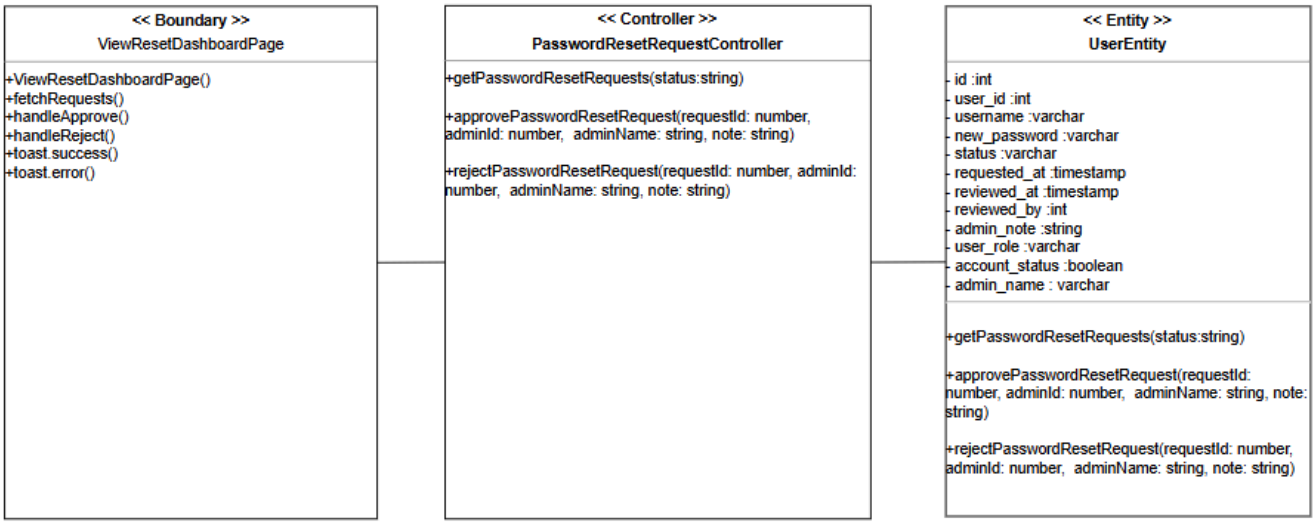


USER STORY (Taiga ID: #392)
As a User Admin, I want to approve user's new password changes so that user can re-access their account

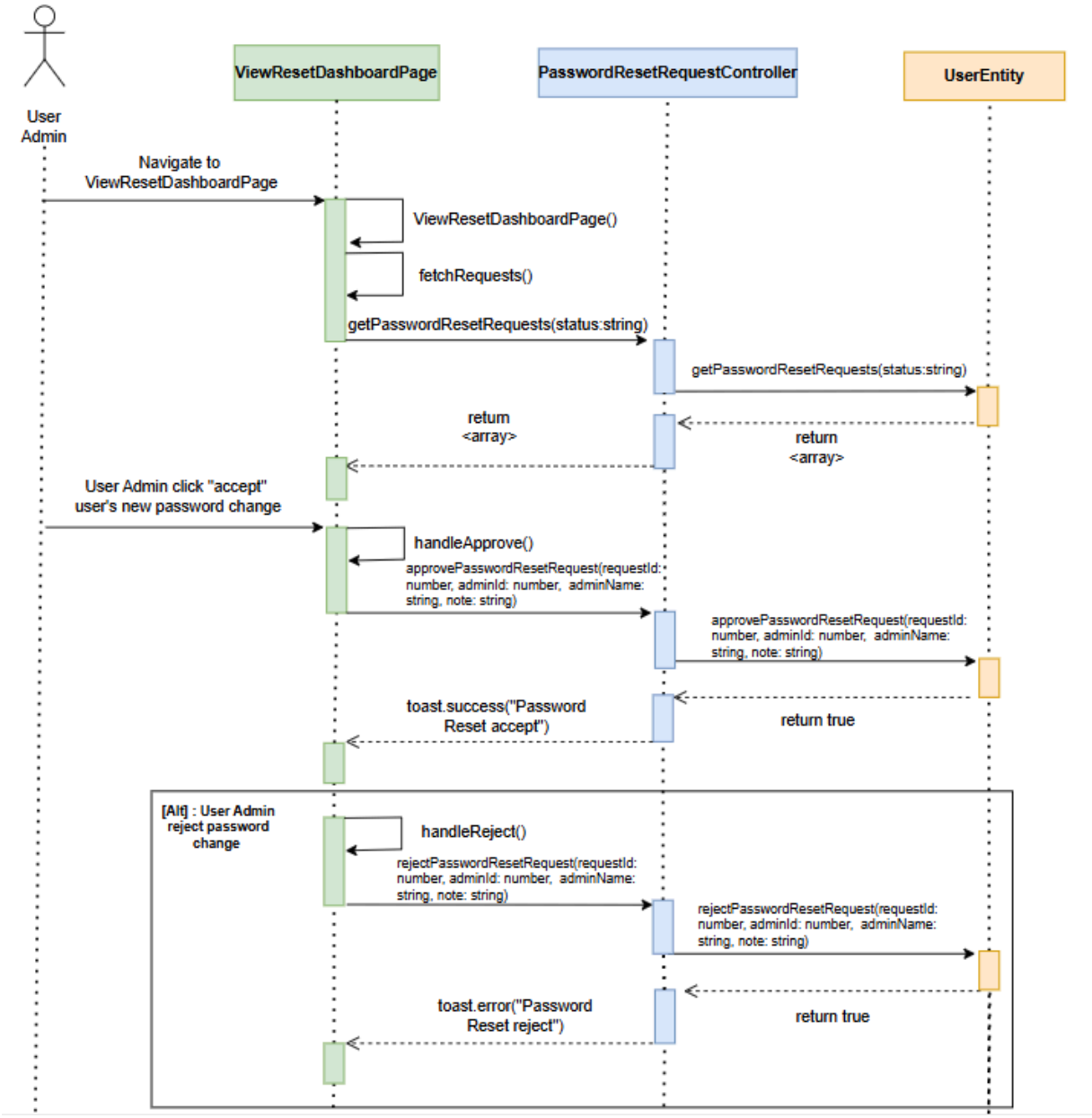
Name: User Admin - Approve changed password	**Taiga - ID: #503**
Stakeholders and goals: User Admin - Approve changed password	
Description: This use case enables User Admin to review and approve password change requests submitted by the user through the "Forgot Password" link on the login page. Users submit their username and desired new password. Which is then sent to the admin for verification and approval to ensure secure account recovery.	
Actors: User Admin	
Trigger: User submits a password change request through the "Forgot Password" link on the login page	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ User has request change of password ❖ User Admin is authenticated and logged into their admin portal ❖ User Admin has appropriate permissions to approve password changes	
Main flow: 1. User clicks on "Forgot Password" link 2. System displays reset password page 3. User enters their registered username and desired new password 4. User submits the password change request 5. System send the password change request to User Admin dashboard 6. The system displays a confirmation message to the user : "Your Password change request has been submitted to the administrator for approval." 7. The User Admin opens admin dashboard and view the password change request(username, new password) 8. System displays the user account details 9. User Admins reviews the password change request 10. User Admin can approve/ disapprove password change of the user from their dashboard 11. System updates the user 's password in the database 12. System displays success confirmation to the User Admin 13. User can now login with the new password	

14. Use case ends
Alternative/Exceptional flow: **A1: User Admin rejects the request** 11a. User Admin decides not approve the password change 11b .User Admin clicks on reject button to reject the request 11c. System marked password request as rejected

BCE (User Admin - Taiga ID: #507)



Sequence diagram (User Admin - Taiga ID: #506)



Code Snippet

Boundary

```
28  function ViewResetDashboardPageModal({ open, onClose, request, onUpdated }: ViewPasswordRequestProps) {
85    const handleApprove = async () => {
86      if (!adminNotes) {
87        alert("Please add admin notes before approving.");
88        return;
89      }
90      if (!adminId || !adminName) {
91        alert("Admin identity not found. Please log in again.");
92        return;
93      }
94      setBusy(true);
95      try {
96        const res = await fetch('/api/userAdmin/password-reset-approve', {
97          method: 'POST',
98          headers: { 'Content-Type': 'application/json' },
99          credentials: 'include',
00          body: JSON.stringify({ requestId: request.id, adminId, adminName, note: adminNotes }),
01        });
02        if (!res.ok) throw new Error(`Approve failed: ${res.status}`);
03      setShowConfirmDialog(false);
04      // update local copy so UI reflects reviewer and timestamp
05      const now = new Date().toISOString();
06      setLocalRequest({ ...localRequest, status: 'Approved', reviewed_by: adminId, admin_name: adminName, reviewed_at: now, admin_note: adminNotes });
07      setActionResult('approved');
08      setShowSuccessDialog(true);
09
10      if (onUpdated) onUpdated();
11        } catch (err) {
12          console.error(err);
13          alert('Failed to approve request');
14        } finally {
15          setBusy(false);
16        }
17         toast.success('Password reset request approved');
18      };
19
20    //HANDLE REJECT BUTTON
21    const handleReject = async () => {
22      if (!adminNotes) {
23        alert("Please add admin notes before rejecting.");
24        return;
25      }
26      if (!adminId || !adminName) {
27        alert("Admin identity not found. Please log in again.");
28        return;
29      }
30      setBusy(true);
31      try {
32        const res = await fetch('/api/userAdmin/password-reset-reject', {
33          method: 'POST',
34          headers: { 'Content-Type': 'application/json' },
35          credentials: 'include',
36          body: JSON.stringify({ requestId: request.id, adminId, adminName, note: adminNotes }),
37        });
38      if (!res.ok) throw new Error(`Reject failed: ${res.status}`);
39      // update local copy so UI reflects reviewer and timestamp
40      const now = new Date().toISOString();
41      setLocalRequest({ ...localRequest, status: 'Rejected', reviewed_by: adminId, admin_name: adminName, reviewed_at: now, admin_note: adminNotes });
42      if (onUpdated) onUpdated();
43      // show a small success/rejection dialog and lock the notes
```

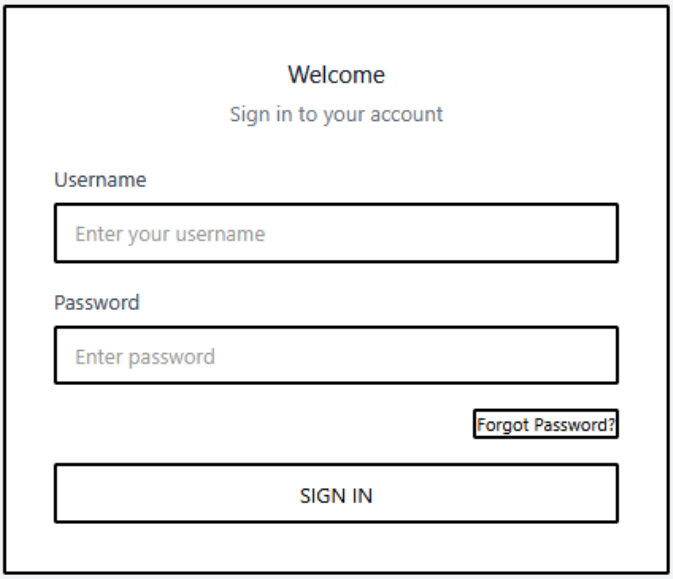
Controller

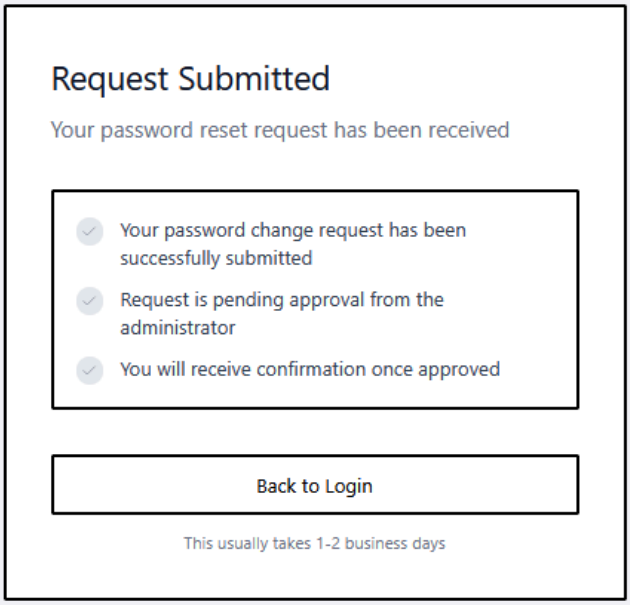
```
133  export class PasswordResetRequestController {
159    //ADMIN
160    // Admin fetches all password reset requests (optionally filter by status)
161    public async getPasswordResetRequests(status?: string) {
162      return await this.userEntity.getPasswordResetRequests(status);
163    }
164        You, yesterday • Added Reset password features
165    // Admin approves a password reset request
166    public async approvePasswordResetRequest(requestId: number, adminId: number,  adminName: str
167      return await this.userEntity.approvePasswordResetRequest(requestId, adminId, adminName, not
168    }
169
170    // Admin rejects a password reset request
171    public async rejectPasswordResetRequest(requestId: number, adminId: number,  adminName: strin
172      return await this.userEntity.rejectPasswordResetRequest(requestId, adminId, adminName, note
173    }
174  }
```

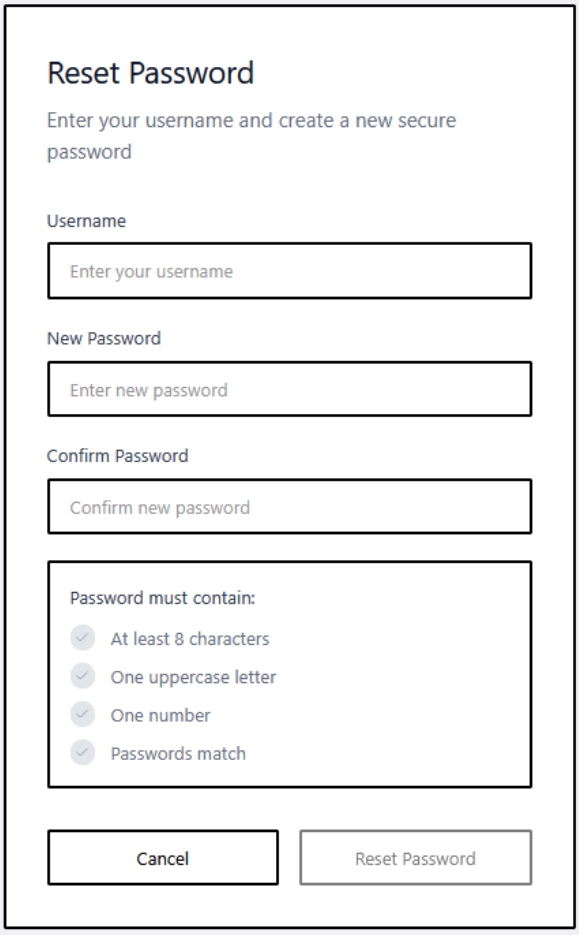
Entity

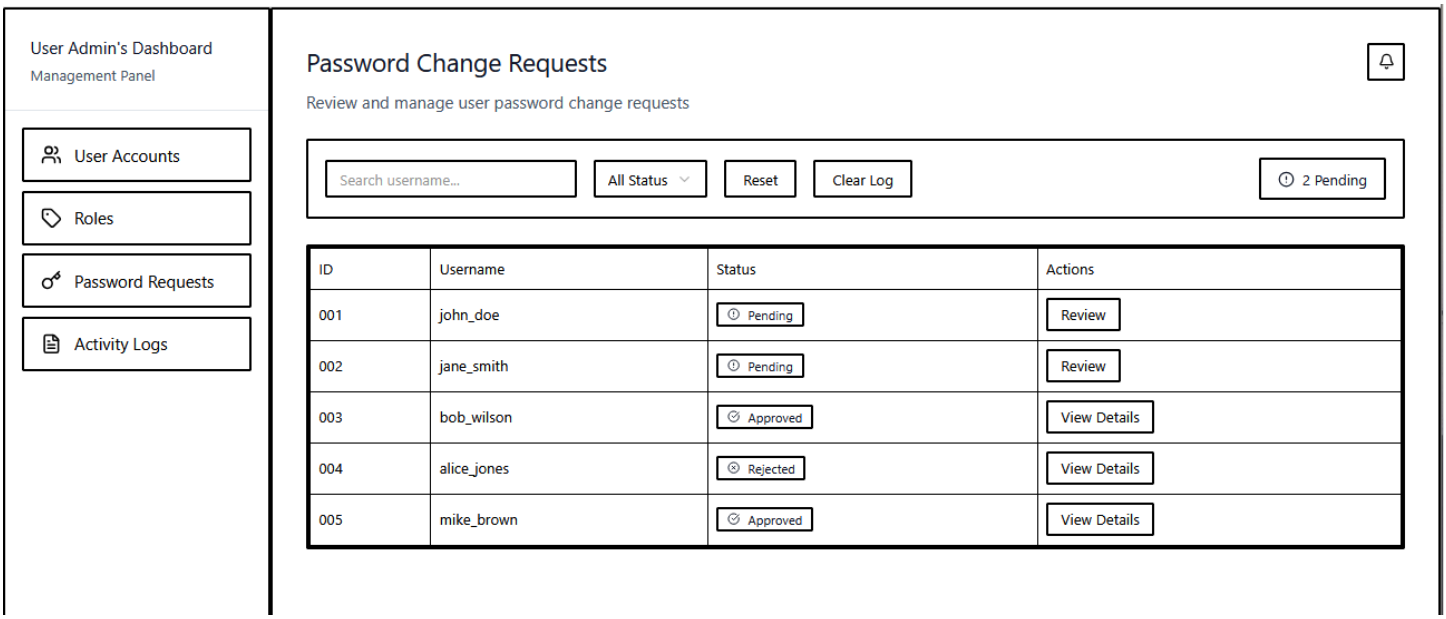
```
 9  export class UserEntity {
371    async approvePasswordResetRequest(requestId: number, adminId: number, adminName: string, note
372      try {
373        // 1. Get the request
374        const [request] = await db.select().from(passwordResetRequestsTable).where(eq(passwordRe
375        if (!request || request.status !== "Pending") return false;
376        // 2. Update user's password
377        await db.update(useraccountTable)
378          .set({ password: request.new_password })
379          .where(eq(useraccountTable.id, request.user_id));
380        // 3. Update request status
381        await db.update(passwordResetRequestsTable)
382          .set({ status: "Approved", reviewed_at: new Date(), reviewed_by: adminId, admin_name:
383          .where(eq(passwordResetRequestsTable.id, requestId));
384        // 4. Log to audit table
385        await db.insert(auditLogTable).values({
386          actor: adminName,
387          action: "Approve Password Reset",
388          target: request.username,
389          details: note, //`RequestId:${requestId}, Note:${note}`,
390          timestamp: new Date(),
391        });
392        return true;       You, yesterday • add stuff
393      } catch (err) {
394        console.error("Approve password reset request error:", err);
395        return false;
396      }
397    }
```

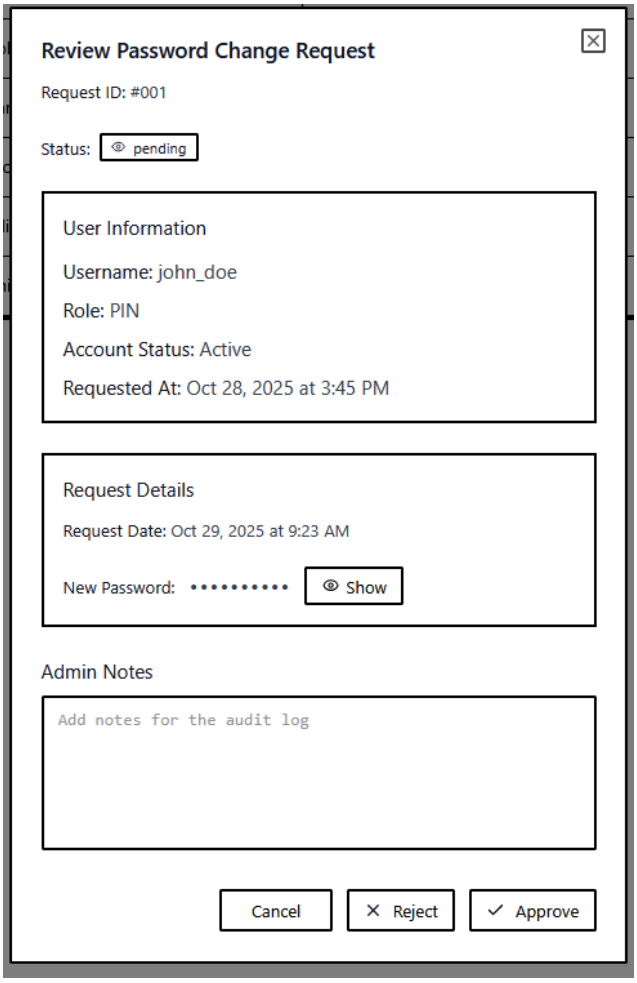
Wireframe (User Admin - Taiga ID #508):

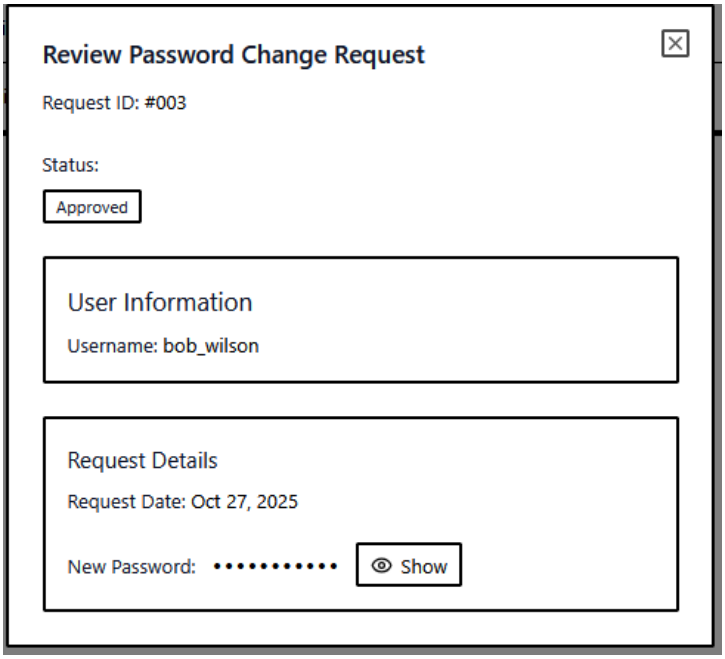







User Admin's Dashboard
Management Panel
User Accounts
Roles
Password Requests
Activity Logs
Password Change Requests
Review and manage user password change requests
Search username...
All Status
Reset
Clear Log
2 Pending
ID
Username
Status
Actions
001
john_doe
Pending
Review
002
jane_smith
Pending
Review
003
bob_wilson
Approved
View Details
004
alice_jones
Rejected
View Details
005
mike_brown
Approved
View Details

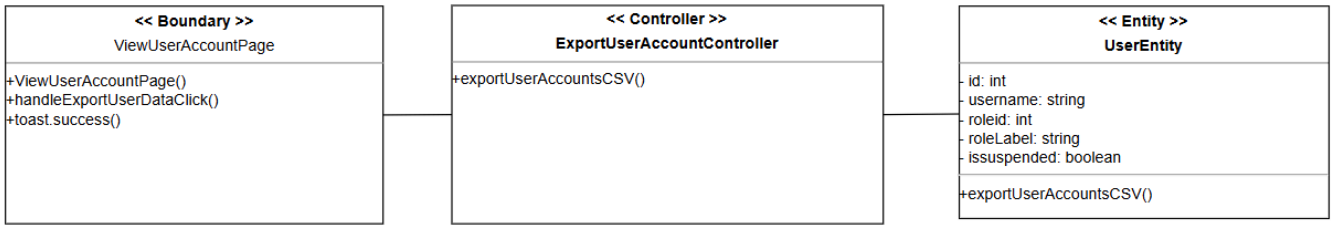

Review Password Change Request
Request ID: #001
Status: pending
User Information
Username: john_doe
Role: PIN
Account Status: Active
Requested At: Oct 28, 2025 at 3:45 PM
Request Details
Request Date: Oct 29, 2025 at 9:23 AM
New Password: ••••••••••
Show
Admin Notes
Add notes for the audit log
Cancel
Reject
Approve


Review Password Change Request
Request ID: #003
Status:
Approved
User Information
Username: bob_wilson
Request Details
Request Date: Oct 27, 2025
New Password: ••••••••••••
Show

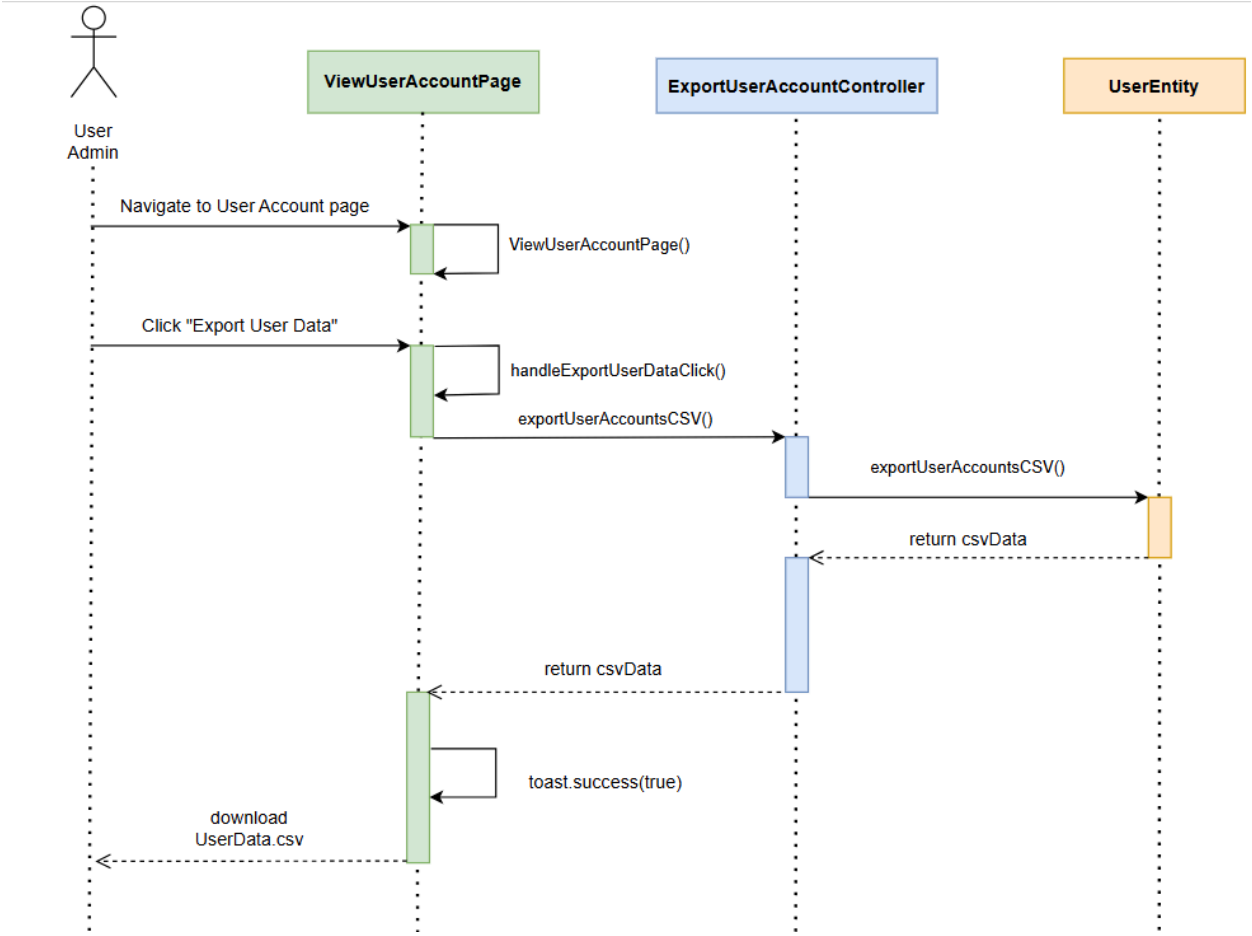
USER STORY (Taiga ID: #397)
As a user admin, I want to export user data to a CSV report so that i can keep a backup of the system's record

Name: User Admin - Export User Data to CSV Report	**Taiga ID: #587**
Stakeholders and goals: User Admin - Export User Data to CSV Report	
Description: This use case allows a User Admin to export user data from the system into a CSV (Comma - Separated Valued) file , enabling the system to create backups of system records or maintain historical records of the users information for compliance or audit purposes.	
Actors: User Admin	
Trigger: User Admin needs to create a backup of user records, generate a report for compliance/audit purposes, or export data for external analysis	
Pre-condition: ❖ User Admin is logged into the system ❖ User Admin has authorization to export user data ❖ At least one user record exists in the system	
Main flow: 1. User Admin navigates to the ViewUserPage 2. User Admin selects the "Export User Data" 3. System validates the request and begins generating CSV file 4. System processes user data and formats it into CSV structure 5. System completes the exports and presents download option 6. User Admin downloads the CSV file to their local system 7. System logs the export activity ({ID,Username,Role,Status}) 8. System displays success message "User data successfully exported. [X} records included." 9. Use case ends	
Alternative/Exceptional flow: NIL	

BCE (User Admin - Taiga ID: #590)



Sequence diagram (User Admin - Taiga ID: #591)



Code Snippet

Boundary

```
const ViewUserAccountPage: React.FC = () => {
  const handleExportUserDataClick = async () => {
    try {
      const res = await fetch(`/api/userAdmin/users/export`);
      if (!res.ok) throw new Error(`Export failed: ${res.status}`);
      const csv = await res.text();
      const lines = csv.split("\n");
      const recordcount = Math.max(0, lines.length - 1);
      const blob = new Blob([csv], { type: "text/csv;charset=utf-8;" });
      const url = window.URL.createObjectURL(blob);
      const a = document.createElement("a");
      a.href = url;
      a.download = `users_export_${new Date().toISOString()}.csv`;
      document.body.appendChild(a);
      a.click();
      a.remove();
      window.URL.revokeObjectURL(url);
      toast.success(`User data succesfully exported. [${recordcount}] records`);
    } catch (err) {
      console.error(err);
      toast.error("Failed to export users");
    }
  };
```

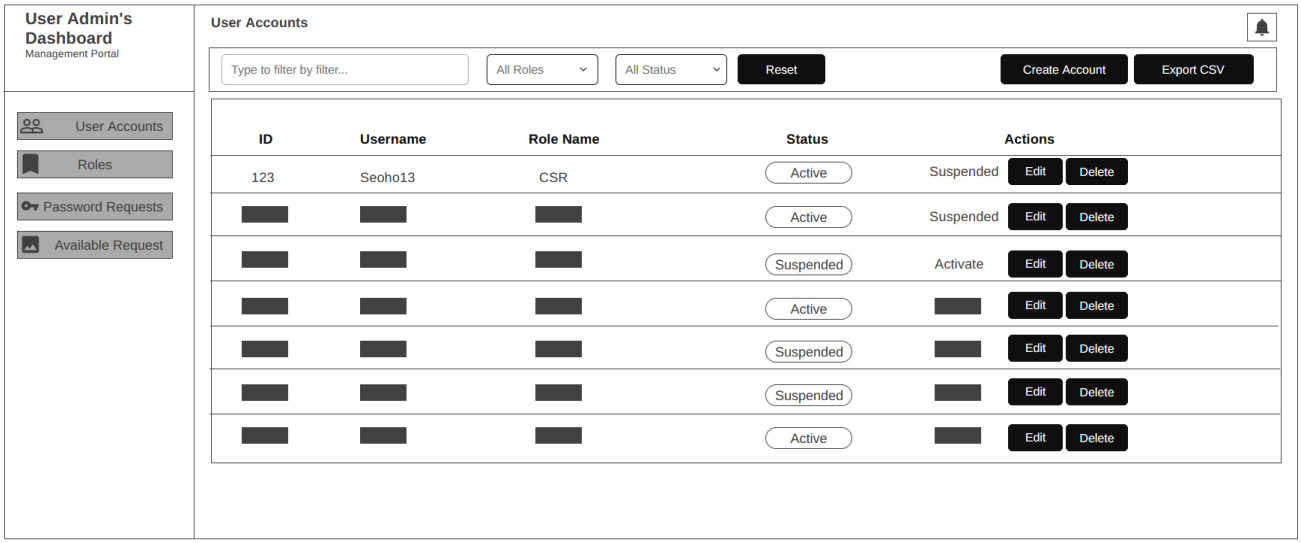
Controller

```
// Export user accounts as CSV
You, 2 hours ago | 1 author (You)
export class ExportUserAccountController {
  private userAccount = new UserEntity();
  public async exportUserAccountsCSV() {
    return await this.userAccount.exportUserAccountsCSV();
  }
}
```

Entity

```
export class UserEntity {

  // Export all user accounts as CSV
  async exportUserAccountsCSV(): Promise<string> {
    const users = await new UserEntity().getAllUserAccounts();
    const headers = ['ID', 'Username', 'Role', 'Status'];
    function escapeCsv(val: any) {
      if (val == null) return '';
      const str = String(val);
      const needsQuotes = /[",\n]/.test(str);
      const escaped = str.replace(/"/g, '""');
      return needsQuotes ? `"${escaped}"` : escaped;
    }
    const rows = users.map(u => [
      u.id,
      u.username,
      u.userProfile,
      u.isSuspended ? 'Suspended' : 'Active'
    ]);
    const csvData = [headers.join(','), ...rows.map(r => r.map(escapeCsv).join(','))].join('\n')
    return csvData;
  }
}       You, 2 weeks ago • Update Login BCE and entites
```

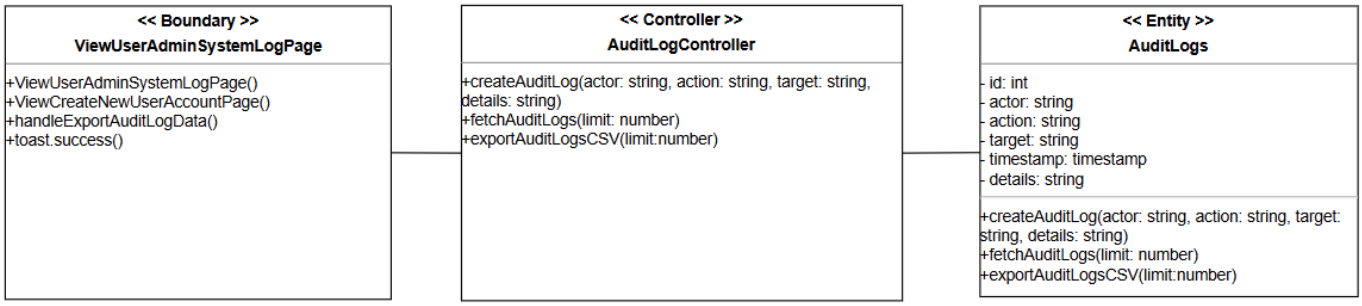
Wireframe: (User Admin - Taiga ID: #592)



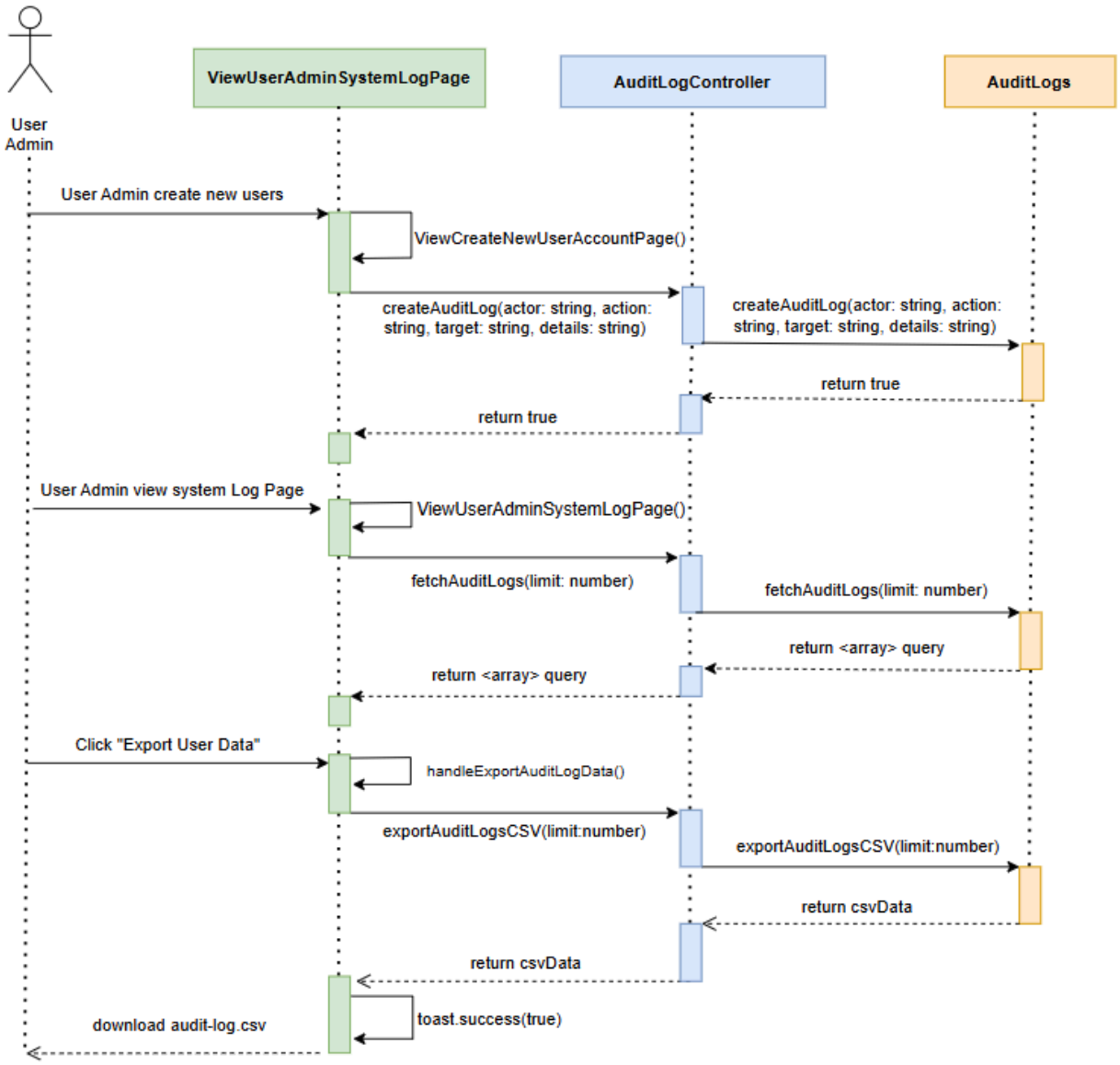
USER STORY (Taiga ID: #398)
As a user admin, I want to view system activity logs so that i can audit user actions for accountability

Name: User Admin - View System Activity Logs	**Taiga ID: #601**
Stakeholders and goals: User Admin	
Description: This use case allows the User Admin to view system activity logs that capture actions such as logins, logouts, account updates, role changes, and data exports. This helps the admin monitor user admins activities and ensure accountability within the system.	
Actors: User Admin	
Trigger: User Admin needs to review or audit recent user admins actions, investigate a security concern, or verify system usage patterns.	
Pre-condition: ❖ A User Admin account exists in the system. ❖ User Admin is authenticated and logged into the admin dashboard. ❖ User Admin has permission to view system activity logs. ❖ The system is actively recording user actions.	
Main flow: 1. User Admin navigates to the System Activity Logs section. 2. System displays available filter options (e.g., date range, username, action type, status). 3. The system displays a summary table showing key user activity, including username, number of sessions (login + logout), account suspension status, last suspension timestamp if applicable, and last password change timestamp. 4. User Admin may apply filters such as specific username, date range, or user role. 5. The system updates the table according to the selected filters. 6. User Admin reviews the activity summary for auditing purposes. 7. User Admin may export the summary table as a CSV file. 8. System logs the viewing or export action with timestamp and admin username for traceability. 9. Use case ends.	
Alternative/Exceptional flow: NIL	

BCE (User Admin - Taiga #604)



Sequence diagram (User Admin - Taiga #605)



Code snippet

Boundary

```
16   export default function ViewUserAdminSystemLogPage() {
79     // Export audit log data as CSV
80     const handleExportAuditLogData = async () => {
81       try {
82         const url = `/api/userAdmin/audit-log/export${
83           limit && limit > 0 ? `?limit=${limit}` : ""
84         }`;
85         const res = await fetch(url);
86         if (!res.ok) throw new Error("Failed to export");
87         const csv = await res.text();
88         const lines = csv.split("\n");
89         const recordCount = Math.max(0, lines.length - 1);
90         if (recordCount === 0) {
91           toast.error("No audit log data to export.");
92           return;
93         }
94         // Download CSV
95         const blob = new Blob([csv], { type: "text/csv" });
96         const downloadUrl = window.URL.createObjectURL(blob);
97         const link = document.createElement("a");
98         link.href = downloadUrl;
99         link.download = "audit-log.csv";
100        document.body.appendChild(link);
101        link.click();
102        document.body.removeChild(link);
103        window.URL.revokeObjectURL(downloadUrl);
104        toast.success(
105          `Audit log successfully exported. [${recordCount}] records included.`
106        );
107      } catch {
108        toast.error("Failed to export audit log data.");
109      }
110    };
```

```
//CREATE USER
router.post("/userAdmin/createUser", async(req: Request, res: Response) => {
  const { username, password, roleid } = req.body
  try {
    const actor = req.session?.username || "unknown";
    const obj = await createUserController.createUserFunc(username, password, roleid, actor);
    if (obj) {
      await auditLogController.createAuditLog(
        actor,
        "create user",
        username,
        `roleid: ${roleid}`
      );
      return res.status(201).json({success: obj})
    } else {
      return res.status(500).json({ success: false, error: "Account creation failed" });
    }
```

Controller

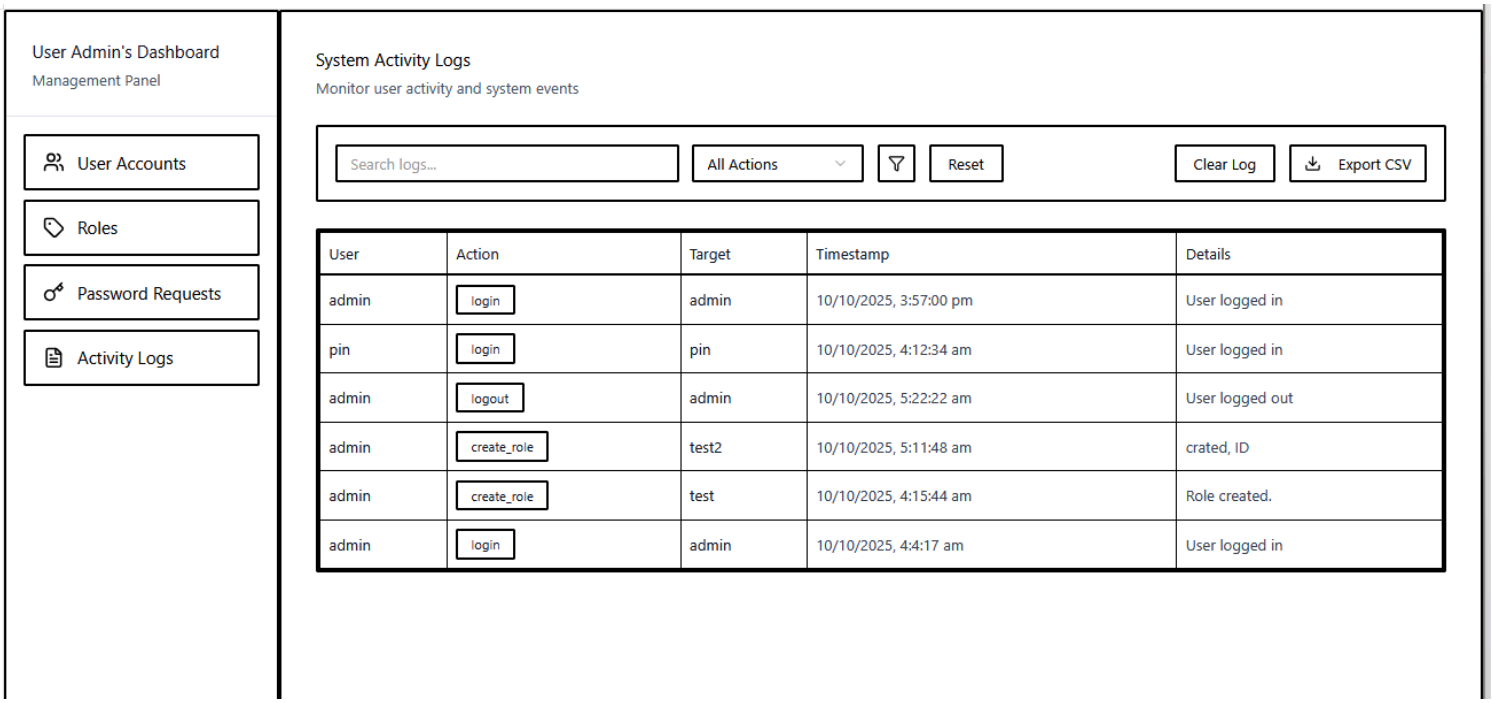
```
export class AuditLogController {
  private auditLogsEntity = new AuditLogs();

  async exportAuditLogsCSV(limit?: number) {
    // Call entity function to get CSV string
    return await this.auditLogsEntity.exportAuditLogsCSV(limit);      You, 31 minutes ago • Un
  }
  async createAuditLog(actor: string, action: string, target: string, details?: string) {
    try {
      if (details !== undefined) {
        return await this.auditLogsEntity.createAuditLog({ actor, action, target, details });
      } else {
        return await this.auditLogsEntity.createAuditLog({ actor, action, target });
      }
    } catch (error) {
      console.error("Error creating audit log:", error);
      throw new Error("Failed to create audit log");
    }
  }

  async fetchAuditLogs(limit?: number) {
    return await this.auditLogsEntity.fetchAuditLogs(limit);
  }
```

Entity

```
export class AuditLogs {
//Crate audit log entry
public async  createAuditLog({ actor, action, target, details }: {
  actor: string;
  action: string;
  target: string;
  details?: string;
}) {
  await db.insert(auditLogTable).values({
    actor,
    action,
    target,
    details,
    timestamp: new Date(),
  });

  return true;
}

//Fetch audit logs with optional limit
public async  fetchAuditLogs(limit?: number) {
  let query = db.select().from(auditLogTable).orderBy(desc(auditLogTable.timestamp));
  if (limit && limit > 0) {
    // drizzle-orm chaining: .orderBy(...).limit(...)
    return await db.select().from(auditLogTable).orderBy(desc(auditLogTable.timestamp)).limit(li
  }
  return await query;
}
```

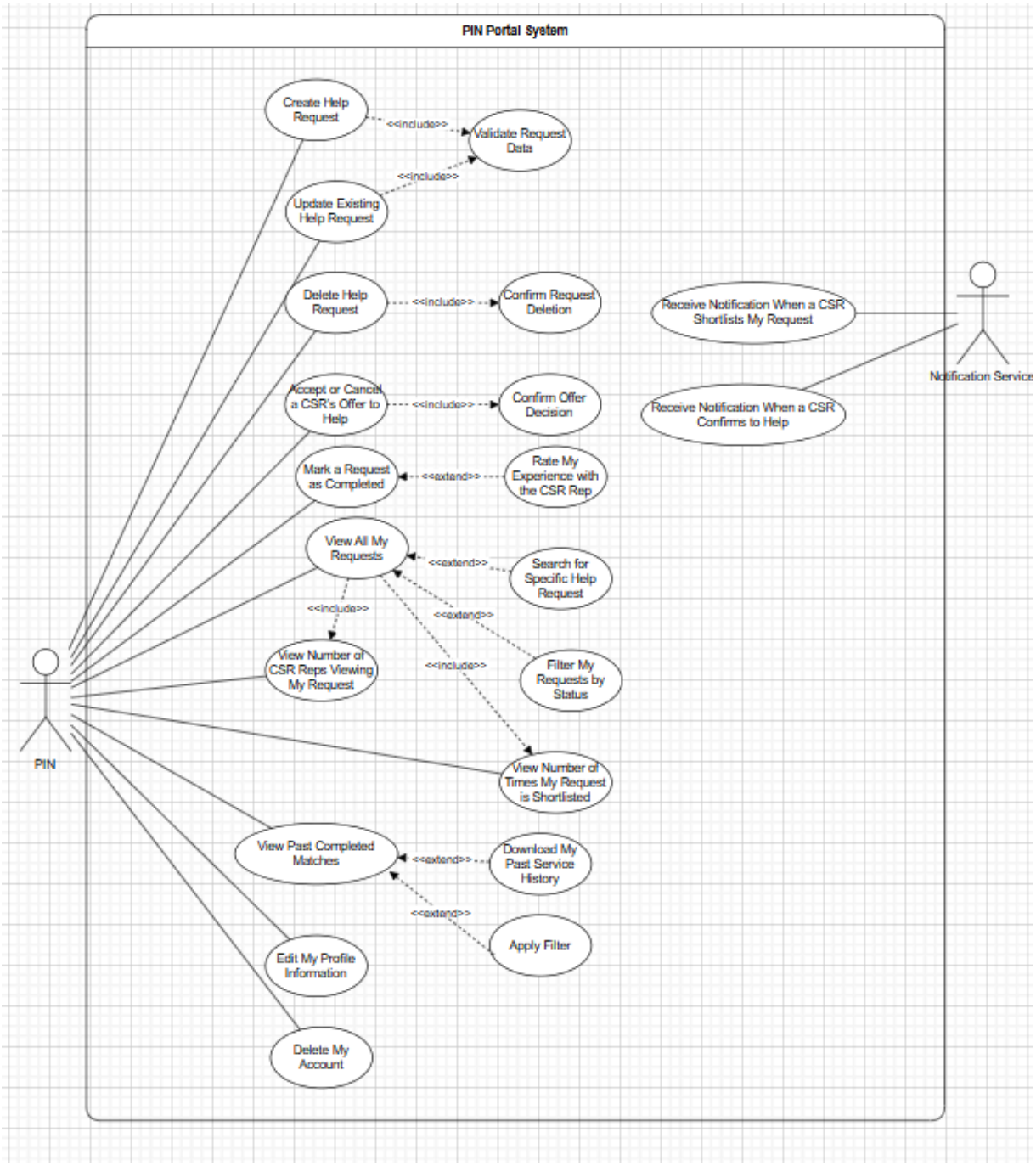
```
export class AuditLogs {

// public audit logs as CSV
public async exportAuditLogsCSV(limit?: number): Promise<string> {
  const logs = await this.fetchAuditLogs(limit);
  const headers = ['id', 'actor', 'action', 'target', 'timestamp', 'details'];
  function escapeCsv(val: any) {
    if (val == null) return '';
    const str = String(val);
    // Escape double quotes, wrap in quotes if contains comma, quote, or newline
    const needsQuotes = /[",\n]/.test(str);
    const escaped = str.replace(/"/g, '""');
    return needsQuotes ? `"${escaped}"` : escaped;
  }
  const rows = logs.map((log: any) => [
    log.id,
    log.actor,
    log.action,
    log.target,
    log.timestamp instanceof Date ? log.timestamp.toISOString() : log.timestamp,
    log.details || ''
  ]);
  const csvData = [headers.join(','), ...rows.map(r => r.map(escapeCsv).join(','))].join('\n');
return csvData;
}       You, 2 days ago • Added CSV export for both ViewUserList and Syst…
```

Wireframe: (User Admin - Taiga ID: #606)



3.2. Person in Need (PIN)

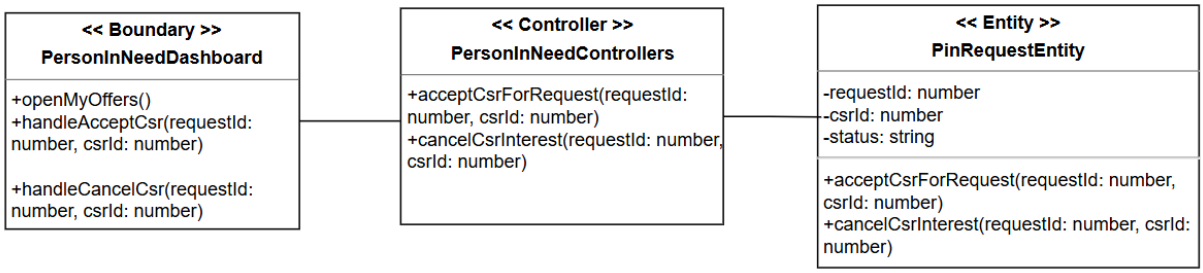
User Case Diagram



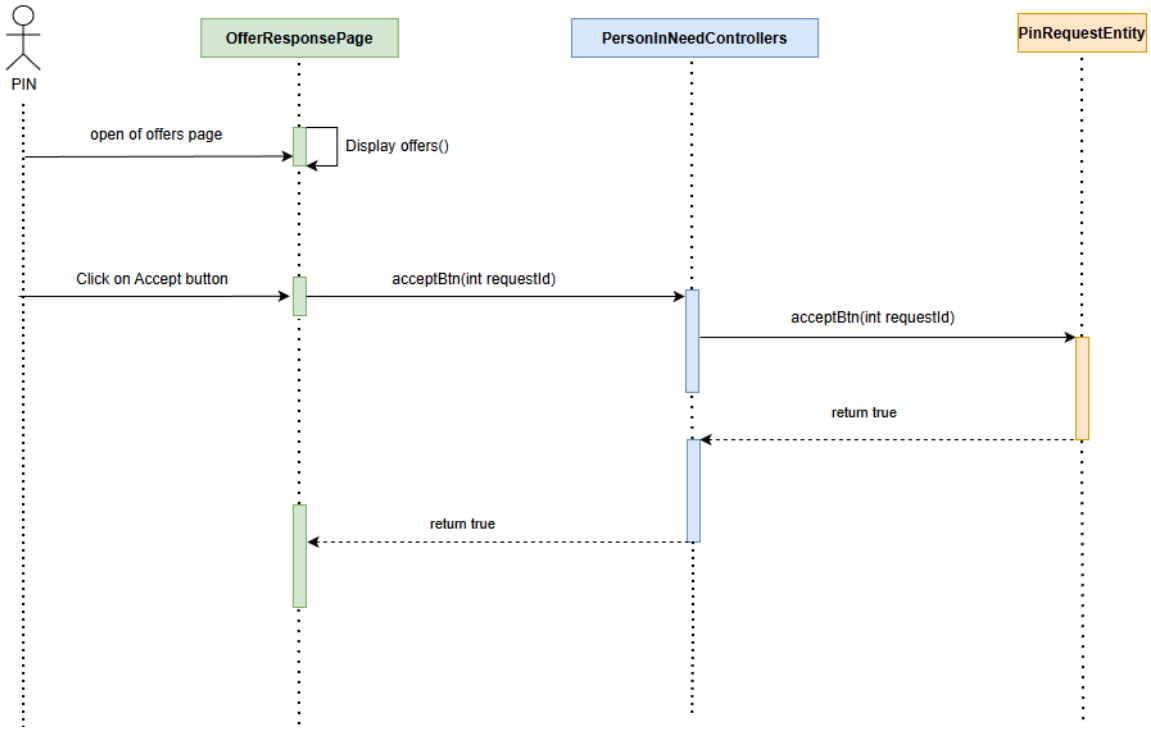
USER STORY (Taiga ID: #399)
As a PIN, I want to accept or cancel a CSR's offer to help my request so that I can control who provides the assistance I need.

Name: PIN - Respond to CSR Offer	**Taiga ID: #531**
Stakeholders and goals: PIN – To approve or cancel a CSR volunteer for their help request, ensuring that only suitable or desired volunteers provide assistance.	
Description: This use case allows a PIN to review offers from CSR reps who want to help with their submitted request. The PIN can either accept the CSR to proceed with the service or cancel the offer if they do not wish to proceed with that volunteer.	
Actors: PIN	
Trigger: A CSR expresses interest in assisting a PIN's submitted request.	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ PIN is authenticated and logged into their PIN portal ❖ The help request has at least one CSR offer pending. ❖ PIN has access to review and respond to CSR offers.	
Main flow: 1. PIN navigates to the My Requests section. 2. The system displays all requests with any CSR offers that are awaiting PIN approval. Requests. 3. PIN chooses to accept or reject the CSR offer. 4. If PIN accepts the CSR, the system changes the request status as Pending and assigns the CSR to the request. 5. If PIN rejects the CSR, the system keeps the request status as Available, allowing other CSR offers. 6. The system displays a confirmation message to the PIN indicating the action was successful. 7. Use case ends.	
Alternative/Exceptional flow: NIL	

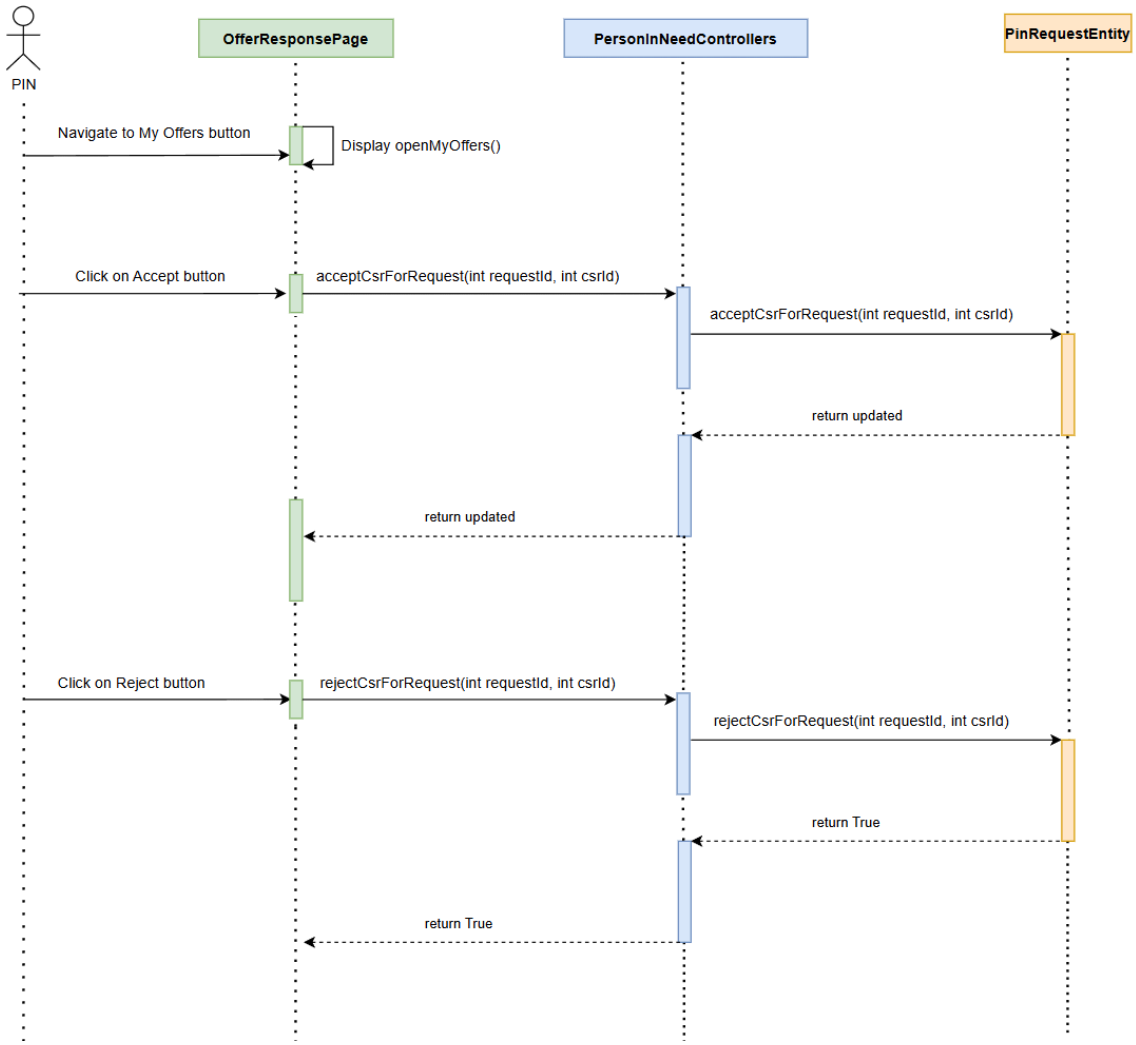
BCE (PIN - Taiga ID: #535)



Sequence Diagram (First Iteration - Sprint 1)



Sequence diagram (PIN - Taiga ID: #526 - Final Iteration)



Code snippet

Boundary

```
const PersonInNeedDashboard: React.FC = () => {
  const openMyOffers = () => {
    if (!userId) {
      setOffersError("No user is signed in.");
      setShowMyOffers(true);
      return;
    }
    setOffersLoading(true);
    setOffersError("");
    fetch(`${API_BASE}/api/pin/offers/${userId}`)
      .then(res => res.json())
      .then(data => {
        setOffers(data.data || []);
        setOffersLoading(false);
        setShowMyOffers(true);
      })
      .catch(() => {
        setOffersError("Could not load offers.");
        setOffersLoading(false);
        setShowMyOffers(true);
      });
  };
```

```
69  const PersonInNeedDashboard: React.FC = () => {
231   const handleAcceptCsr = async (requestId: number, csrId: number) => {
232     const res = await fetch(`${API_BASE}/api/pin/offers/${requestId}/accept`, {
233       method: "POST",
234       headers: { "Content-Type": "application/json" },
235       body: JSON.stringify({ csrId }),
236     });
237     if (res.ok) {
238       toast.success("CSR accepted for this request.");
239       // Refetch offers after a short delay to ensure backend updates are reflected
240       setTimeout(openMyOffers, 300);
241     } else {
242       toast.error("Failed to accept CSR.");
243     }
244   };
245
246   // Cancel a CSR's interest for a request
247   const handleCancelCsr = async (requestId: number, csrId: number) => {
248     const res = await fetch(`${API_BASE}/api/pin/offers/${requestId}/cancel`, {
249       method: "POST",
250       headers: { "Content-Type": "application/json" },
251       body: JSON.stringify({ csrId }),
252     });
253     if (res.ok) {
254       toast.success("CSR interest cancelled.");
255       setTimeout(openMyOffers, 300);
256     } else {
257       toast.error("Failed to cancel CSR interest.");
258     }
259   };
260
```

Controller

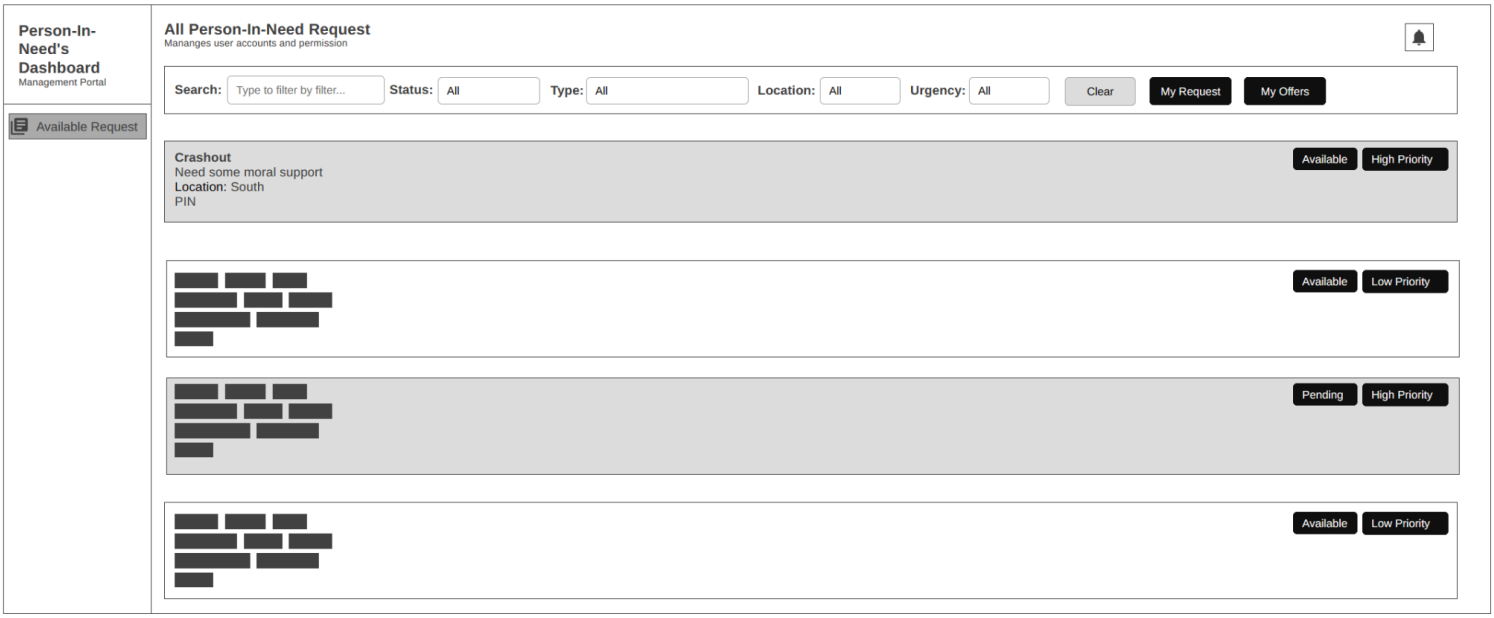
```
export class PersonInNeedControllers {
  // Accept a CSR for a request
  async acceptCsrForRequest(requestId: number, csrId: number) {
    return await PinRequestEntity.acceptCsrForRequest(requestId, csrId);
  }

  // Cancel a CSR's interest for a request
  async cancelCsrInterest(requestId: number, csrId: number) {
    return await PinRequestEntity.cancelCsrInterest(requestId, csrId);
  }
```

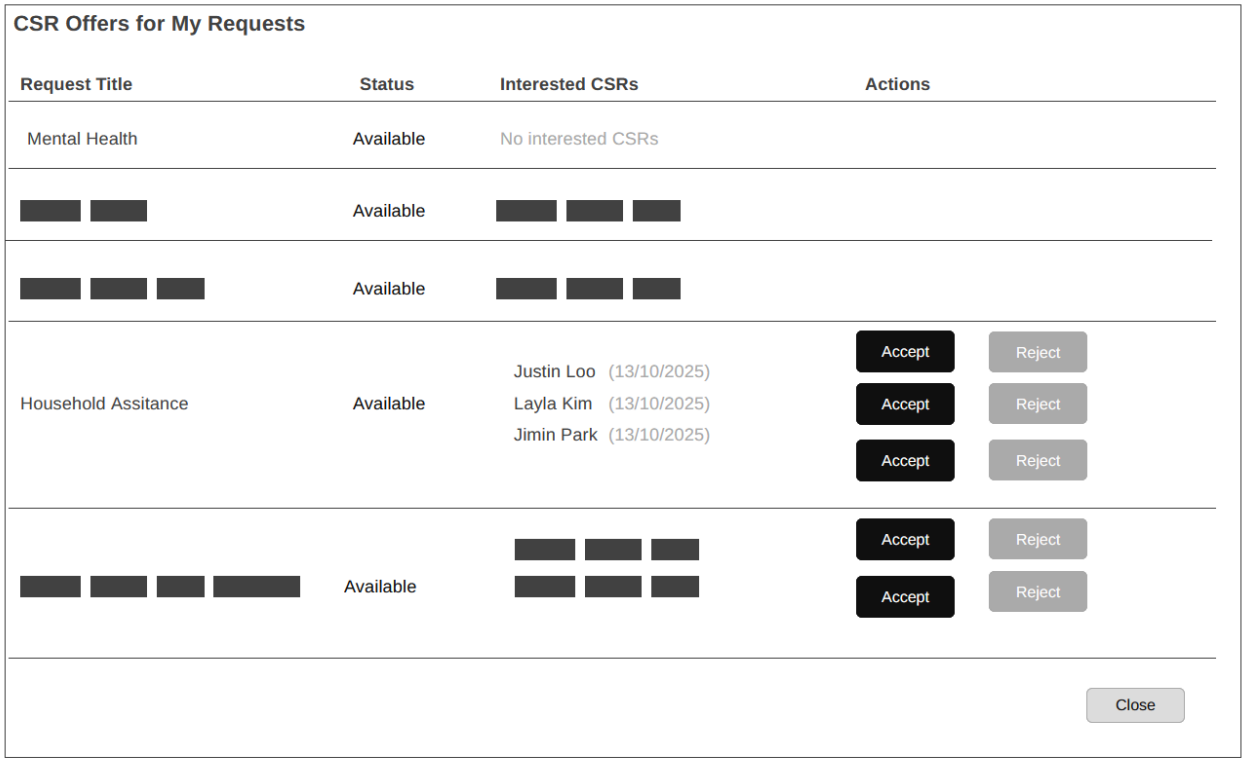
Entity

```
export class PinRequestEntity {
  static async acceptCsrForRequest(requestId: number, csrId: number) {
    // 1. Assign CSR and set status to Pending
    const [updated] = await db
      .update(pin_requestsTable)
      .set({ csr_id: csrId, status: 'Pending' })
      .where(eq(pin_requestsTable.id, requestId))
      .returning();

    // 2. Update csr_requestsTable: accepted CSR gets 'Accepted', all others get 'Rejected'
    const { and, not } = require('drizzle-orm');
    // Accept the chosen CSR
    await db.update(require('../db/schema/aiodb').csr_requestsTable)
      .set({ status: 'Accepted' })
      .where(and(
        eq(require('../db/schema/aiodb').csr_requestsTable.csr_id, csrId),
        eq(require('../db/schema/aiodb').csr_requestsTable.pin_request_id, requestId)
      ));

    // Send notification to accepted CSR
    await db.insert(require('../db/schema/aiodb').notificationTable).values({
      pin_id: updated.pin_id,
      csr_id: csrId,
      pin_request_id: requestId,
      type: 'accepted',
      createdAt: new Date(),
      read: 0,
    });

    // Find all other interested/rejected CSRs for this request
    const rejectedCsrs = await db
      .select({ csr_id: require('../db/schema/aiodb').csr_requestsTable.csr_id })
      .from(require('../db/schema/aiodb').csr_requestsTable)
      .where(and(
        not(eq(require('../db/schema/aiodb').csr_requestsTable.csr_id, csrId)),
        eq(require('../db/schema/aiodb').csr_requestsTable.pin_request_id, requestId)
      ));

    // Reject all other interested CSRs for this request
    await db.update(require('../db/schema/aiodb').csr_requestsTable)
      .set({ status: 'Rejected' })
      .where(and(
        not(eq(require('../db/schema/aiodb').csr_requestsTable.csr_id, csrId)),
        eq(require('../db/schema/aiodb').csr_requestsTable.pin_request_id, requestId)
      ));
```

```
export class PinRequestEntity {
  // Cancel a CSR's interest for a request
  static async cancelCsrInterest(requestId: number, csrId: number) {
    // Delete from csr_interestedTable
    await db.delete(require('../db/schema/aiodb').csr_interestedTable)
      .where(and(
        eq(require('../db/schema/aiodb').csr_interestedTable.pin_request_id, requestId),
        eq(require('../db/schema/aiodb').csr_interestedTable.csr_id, csrId)
      ));
    // Update status in csr_requestsTable
    await db.update(require('../db/schema/aiodb').csr_requestsTable)
      .set({ status: 'Rejected' })
      .where(and(
        eq(require('../db/schema/aiodb').csr_requestsTable.pin_request_id, requestId),
        eq(require('../db/schema/aiodb').csr_requestsTable.csr_id, csrId)
      ));

    // Get pin_id for notification
    const pinReq = await db.select({ pin_id: pin_requestsTable.pin_id })
      .from(pin_requestsTable)
      .where(eq(pin_requestsTable.id, requestId));
    const pin_id = pinReq[0]?.pin_id;
    if (pin_id) {
      await db.insert(require('../db/schema/aiodb').notificationTable).values({
        pin_id,
        csr_id: csrId,
        pin_request_id: requestId,
        type: 'rejected',
        createdAt: new Date(),
        read: 0,
      });
    }
    return true;
  }
```

Wireframe (PIN - Taiga ID: #536):

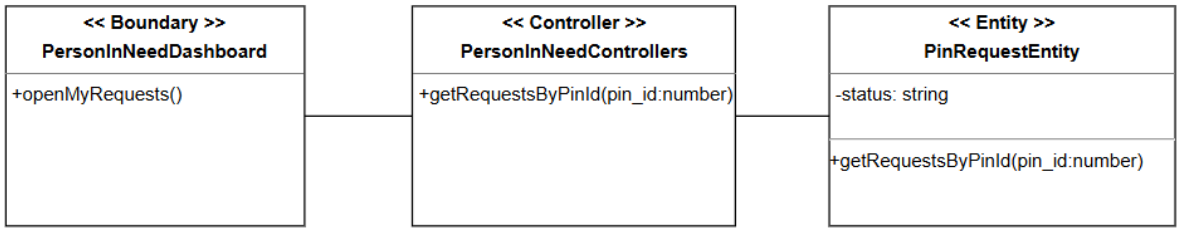




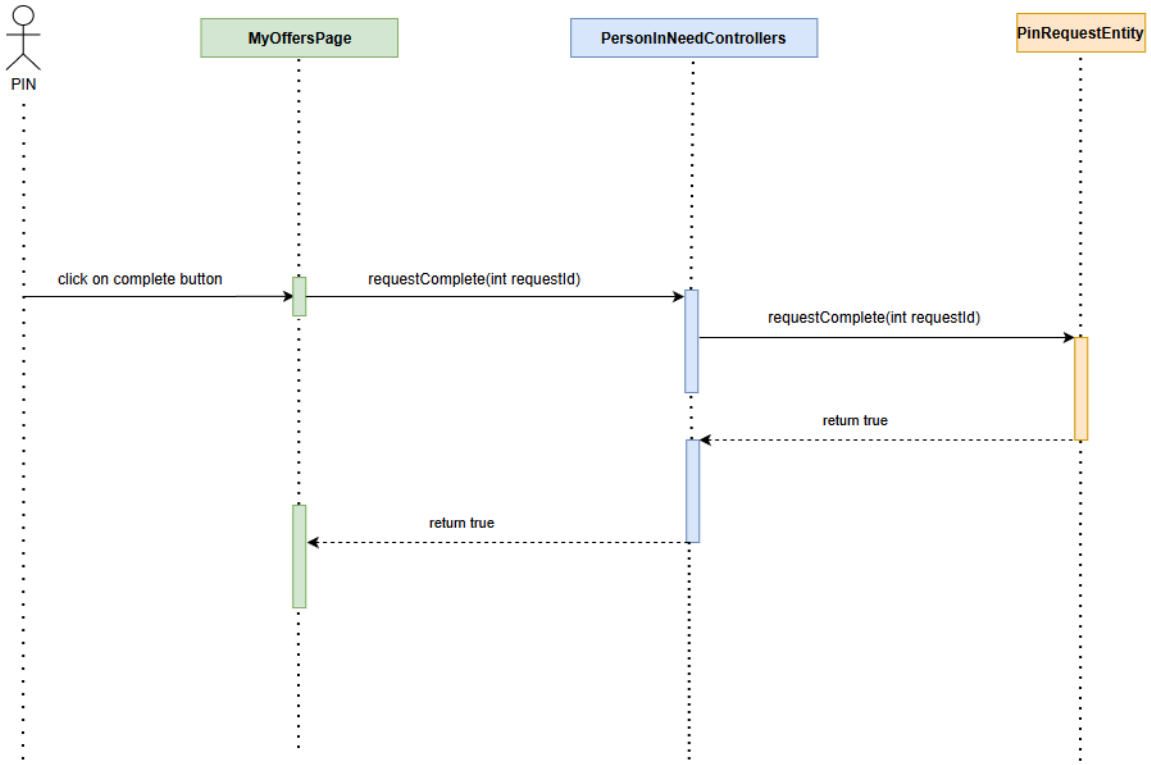
USER STORY (Taiga ID: #400)
As a PIN, I want to filter requests by status (e.g., available, pending, completed) so that I can track progress easily.

Name: PIN - View Requests by Status	**Taiga ID: #532**
Stakeholders and goals: PIN – To easily track the progress of their help requests by viewing them according to status: Available, Pending, or Completed.	
Description: This use case allows a PIN to view their submitted help requests filtered by status. It helps the PIN quickly see which requests are still open, which have pending CSR offers, and which are completed.	
Actors: PIN	
Trigger: PIN wants to monitor the progress of their help requests.	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ PIN is authenticated and logged into their PIN portal ❖ PIN account exists and is authenticated. ❖ Requests have valid status values (Available, Pending, Completed).	
Main flow: 1. PIN navigates to the My Requests section. 2. The system displays all of the PIN's requests with their current statuses. 3. PIN selects a status filter (Available, Pending, Completed). 4. System updates the list to display only requests matching the selected status. 5. PIN reviews the filtered requests to track progress. 6. Use case ends.	
Alternative/Exceptional flow: **A1. No requests for selected status:** 4a. If no requests match the selected status; display nothing 4b. Use case returns to Step 4 of the main flow.	

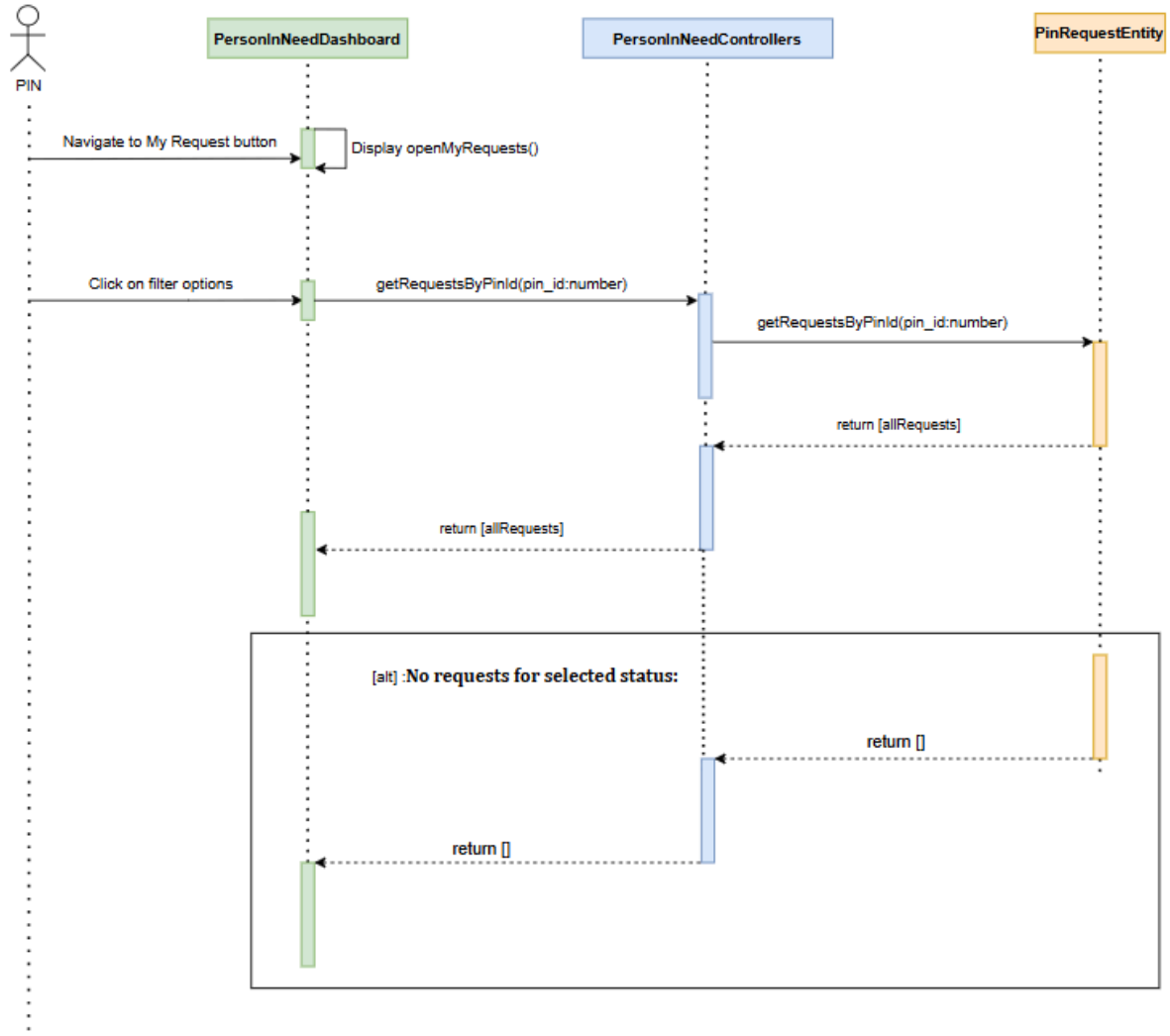
BCE (PIN - Taiga ID: #542)



Sequence Diagram (First Iteration - Sprint 1)



Sequence diagram (PIN - Taiga ID: #534 - <u>Final Iteration</u>)



Code snippet

Boundary

```
const PersonInNeedDashboard: React.FC = () => {
  const openMyRequests = () => {      Justin, 2 weeks ago • I DID THE PIN AND
    if (!userId) {
      setStatusMsg("No user is signed in.");
      return;
    }
    setStatusMsg("");
    fetch(`${API_BASE}/api/pin/requests/user/${userId}`)
      .then(res => res.json())
      .then(data => {
        setMyRequests(data.data || []);
        setShowMyRequests(true);
      })
      .catch(() => setStatusMsg("Network error while loading My Requests"));
  };
```

```
  </select>
  <select
    className="filter-control filter-select"
    value={myRequestsFilterStatus}
    onChange={e => setMyRequestsFilterStatus(e.target.value)}
  >
    <option value="">Status</option>
    <option value="Available">Available</option>
    <option value="Pending">Pending</option>
    <option value="Completed">Completed</option>
  </select>
  <select
```

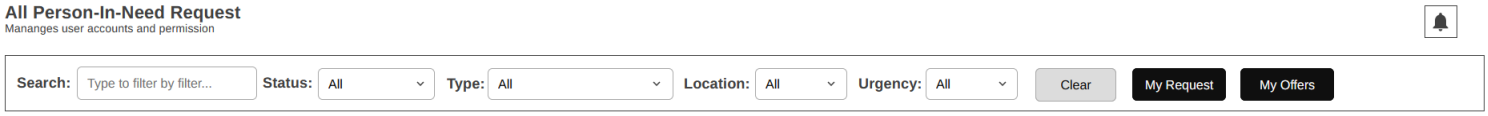
Controller

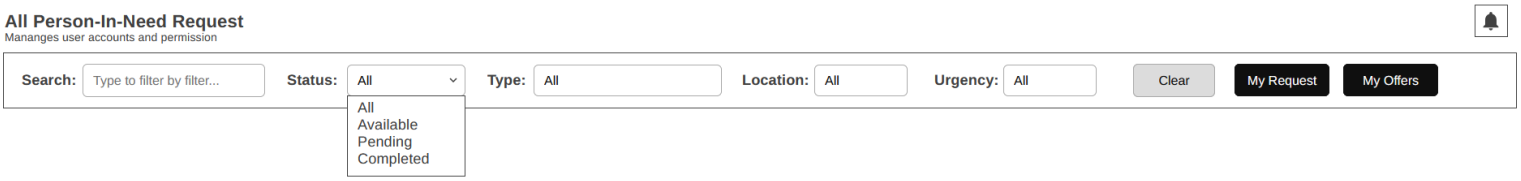
```
export class PersonInNeedControllers {

  async getRequestsByPinId(pin_id: number): Promise<PinRequest[]> {
    return await PinRequestEntity.getRequestsByPinId(pin_id);
  }
```

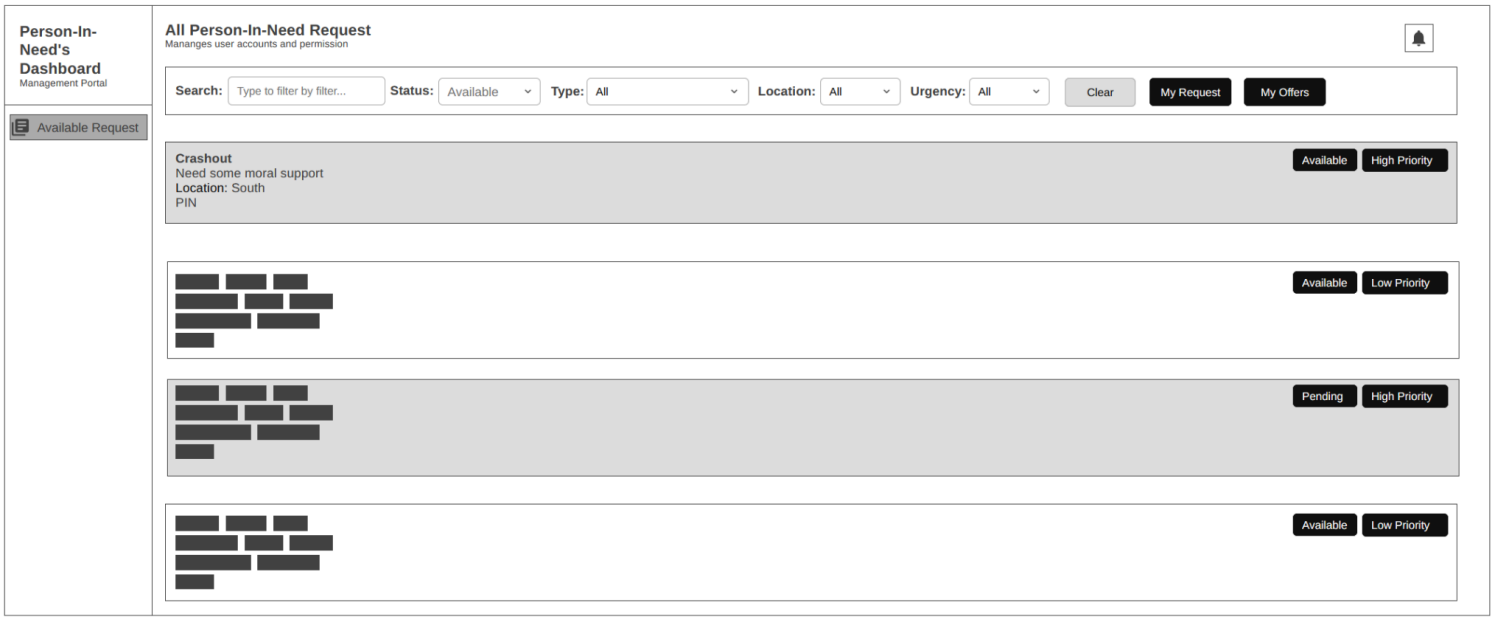
Entities

```
export class PinRequestEntity {
  static async getRequestsByPinId(pin_id: number): Promise<any[]> {
    // Join pin_requests with users to get username as pinName for a specific pin_id
    return await db
      .select({
        id: pin_requestsTable.id,
        pin_id: pin_requestsTable.pin_id,
        pinName: useraccountTable.username,
        title: pin_requestsTable.title,
        csr_id: pin_requestsTable.csr_id,
        categoryID: pin_requestsTable.categoryID,
        categoryName: service_typeTable.name,
        message: pin_requestsTable.message,
        locationID: pin_requestsTable.locationID,
        locationName: locationTable.name,
        urgencyLevelID: pin_requestsTable.urgencyLevelID,
        urgencyLabel: urgency_levelTable.label,
        view_count: pin_requestsTable.view_count,
        shortlist_count: pin_requestsTable.shortlist_count,
        createdAt: pin_requestsTable.createdAt,
        status: pin_requestsTable.status,
      })
      .from(pin_requestsTable)
      .leftJoin(useraccountTable, eq(pin_requestsTable.pin_id, useraccountTable.id))
      .leftJoin(service_typeTable, eq(pin_requestsTable.categoryID, service_typeTable.id))
      .leftJoin(locationTable, eq(pin_requestsTable.locationID, locationTable.id))
      .leftJoin(urgency_levelTable, eq(pin_requestsTable.urgencyLevelID, urgency_levelTable.id))
    .where(eq(pin_requestsTable.pin_id, pin_id));
  }
```

Wireframe (PIN - Taiga ID: #543):



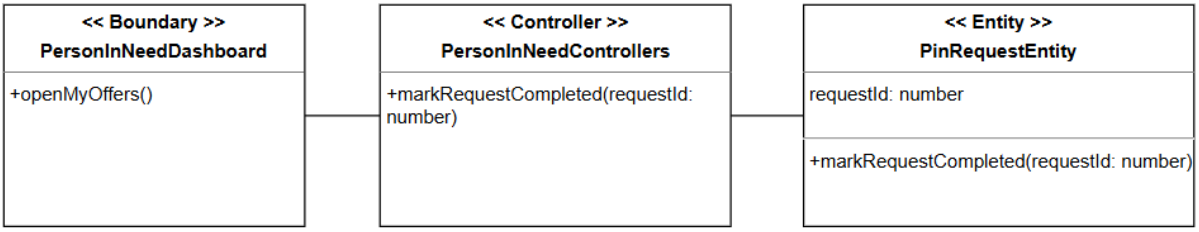




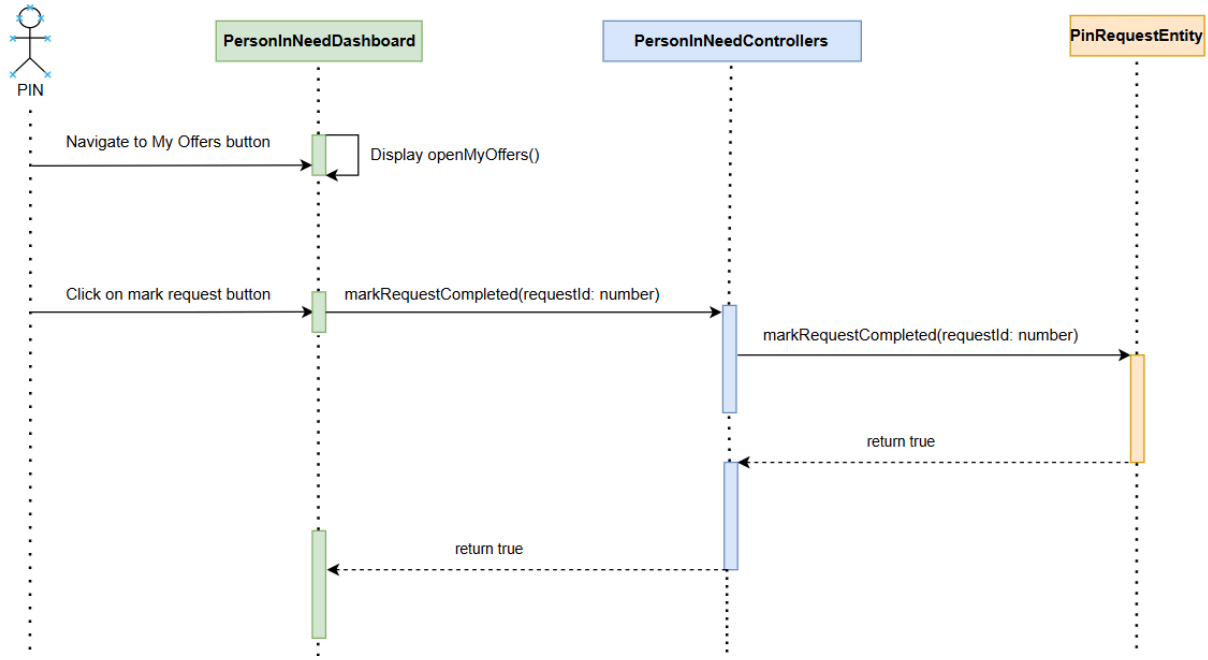
USER STORY (Taiga ID: #401)
As a PIN, I want to mark a request as completed so that I can close cases that have been fulfilled.

Name: PIN - Mark Request as Completed	**Taiga ID: #545**
Stakeholders and goals: PIN – To close a help request once the service has been fulfilled, ensuring the system accurately reflects completed requests and prevents further CSR assignments.	
Description: This use case allows a PIN to mark a help request as completed after the required assistance has been provided. This ensures that completed requests are properly recorded and no longer appear as available or pending in the system.	
Actors: PIN	
Trigger: PIN confirms that the requested help has been delivered and wants to close the request.	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ PIN is authenticated and logged into their PIN portal ❖ The request exists in the system and is assigned to a CSR (status: Pending).	
Main flow: 1. PIN navigates to the My Requests section. 2. The system displays all requests, highlighting those with Pending status. 3. PIN clicks the Mark as Completed button for a Pending request. 4. Use case ends.	
Alternative/Exceptional flow: NIL	

BCE (PIN - Taiga ID: #549)



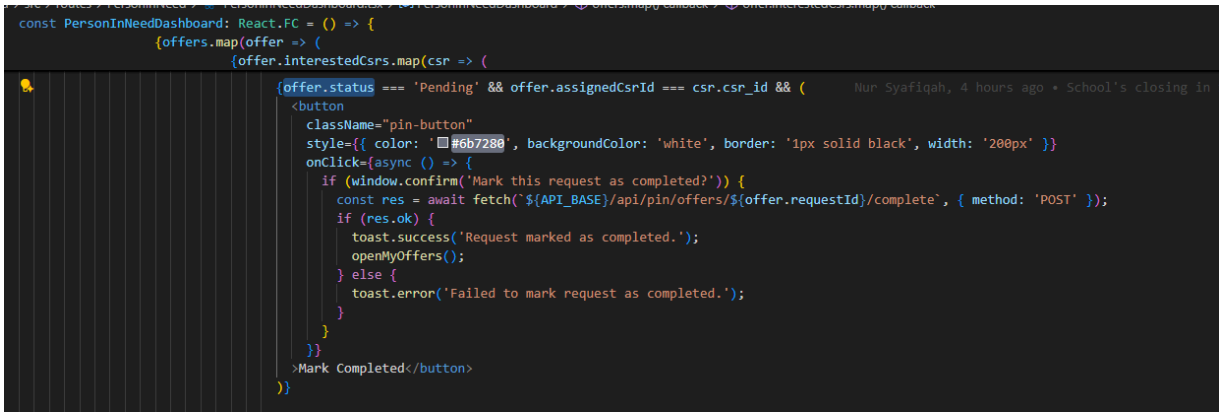
Sequence diagram (PIN - Taiga ID: #548)



Code Snippet

Boundary

```
69   const PersonInNeedDashboard: React.FC = () => {
307    const openMyOffers = () => {
308      if (!userId) {
309        setOffersError("No user is signed in.");
310        setShowMyOffers(true);
311        return;
312      }
313      setOffersLoading(true);
314      setOffersError("");
315      fetch(`${API_BASE}/api/pin/offers/${userId}`)
316        .then(res => res.json())
317        .then(data => {
318          setOffers(data.data || []);
319          setOffersLoading(false);
320          setShowMyOffers(true);
321        })
322        .catch(() => {
323          setOffersError("Could not load offers.");
324          setOffersLoading(false);
325          setShowMyOffers(true);
326        });
327    };
328
```



```
const PersonInNeedDashboard: React.FC = () => {
                {offers.map(offer => (
                    {offer.interestedCsrs.map(csr => (
                        {offer.status === 'Pending' && offer.assignedCsrId === csr.csr_id && (
                          <button
                            className="pin-button"
                            style={{ color: '#6b7280', backgroundColor: 'white', border: '1px solid black', width: '200px' }}
                            onClick={async () => {
                              if (window.confirm('Mark this request as completed?')) {
                                const res = await fetch(`${API_BASE}/api/pin/offers/${offer.requestId}/complete`, { method: 'POST' });
                                if (res.ok) {
                                  toast.success('Request marked as completed.');
                                  openMyOffers();
                                } else {
                                  toast.error('Failed to mark request as completed.');
                                }
                              }
                            }}
                          >Mark Completed</button>
                        )}
```

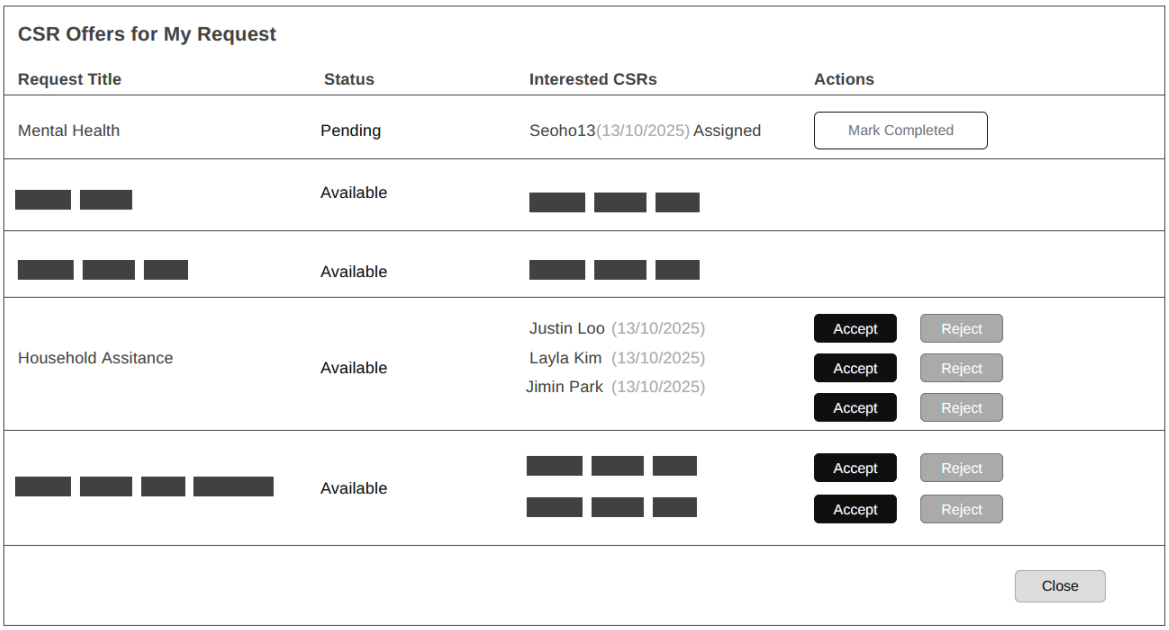
Controller

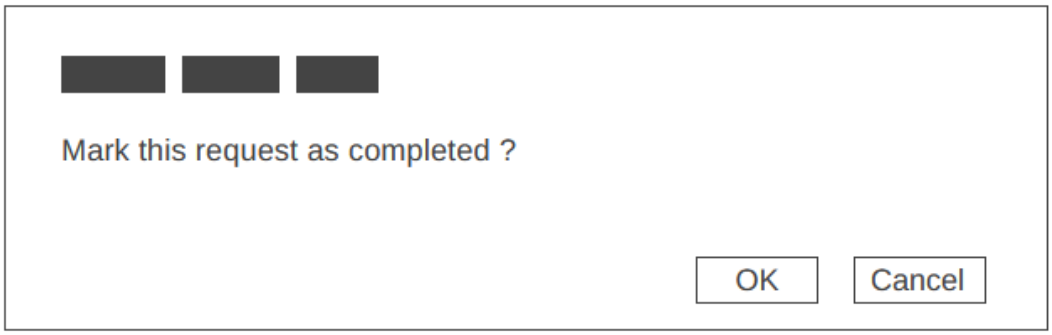
```
export class PersonInNeedControllers {
  // Mark a PIN request and its assigned CSR request as Completed
  async markRequestCompleted(requestId: number): Promise<boolean> {
    return await PinRequestEntity.markRequestCompleted(requestId);
  }
```

Entities

```
export class PinRequestEntity {
  // Mark a PIN request and its assigned CSR request as Completed      Justin, 5 days ago
  static async markRequestCompleted(requestId: number): Promise<boolean> {
    // 1. Update PIN request status to Completed
    const [updatedPin] = await db
      .update(pin_requestsTable)
      .set({ status: 'Completed' })
      .where(eq(pin_requestsTable.id, requestId))
      .returning();
    if (!updatedPin) return false;
    // 2. Find assigned CSR for this request
    const csrId = updatedPin.csr_id;
    if (csrId) {
      // 3. Update CSR request status to Completed
      await db.update(require('../db/schema/aiodb').csr_requestsTable)
        .set({ status: 'Completed' })
        .where(and(
          eq(require('../db/schema/aiodb').csr_requestsTable.csr_id, csrId),
          eq(require('../db/schema/aiodb').csr_requestsTable.pin_request_id, requestId)
        ));
    }
    return true;
  }
```

Wireframe (PIN - Taiga ID: #550):

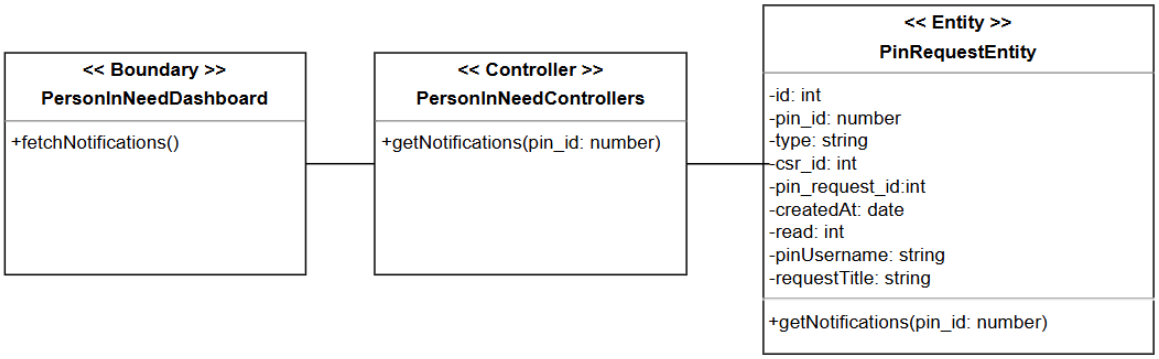




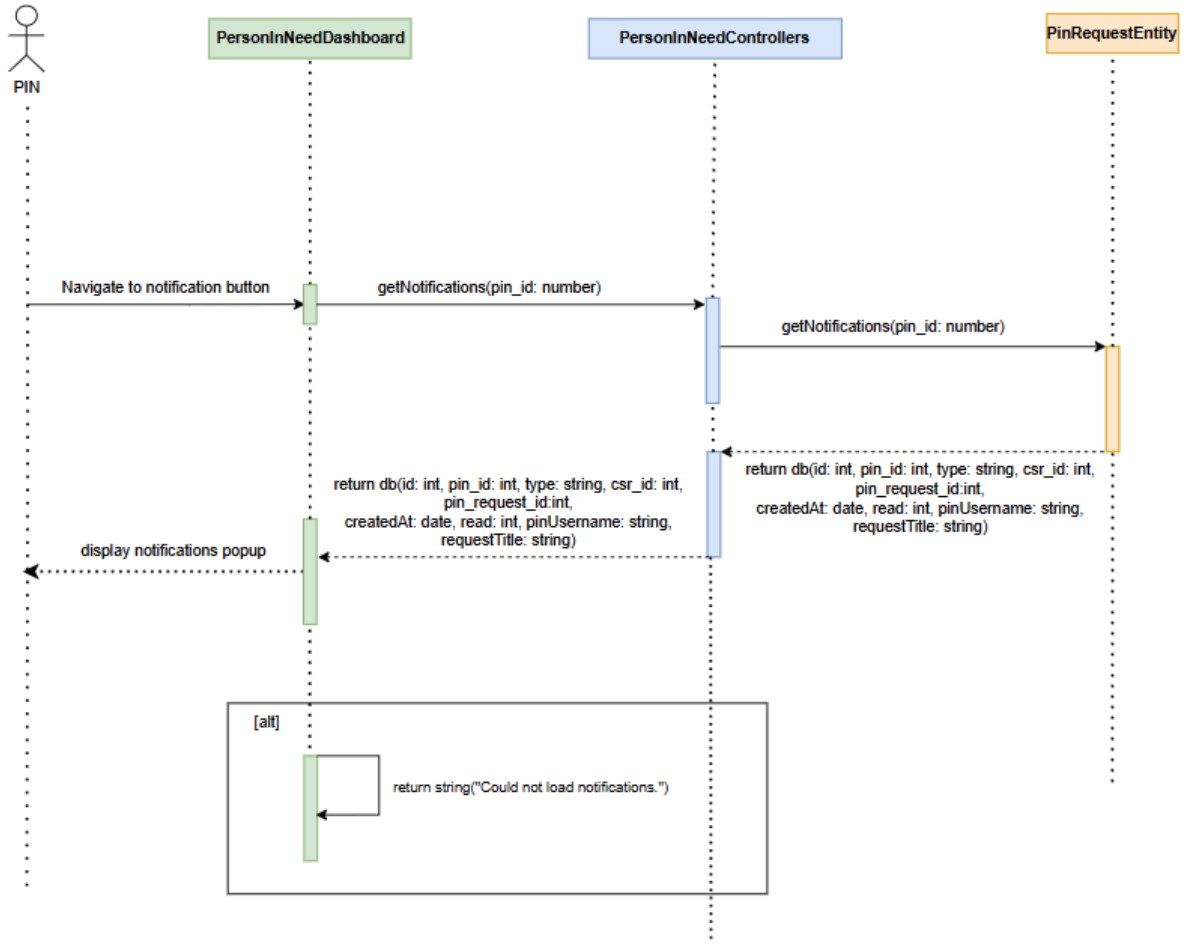
USER STORY (Taiga ID: #402)
As a PIN, I want to receive notifications when a CSR shortlists my request so that I know someone is interested in helping me.

Name: PIN - CSR Shortlist Notification	**Taiga ID: #608**
Stakeholders and goals: PIN – To be informed in their mail/notification system when a CSR shows interest in helping with a request, so they can track pending offers.	
Description: This use case allows a PIN to see notifications in their on-site mail system whenever a CSR shortlists one of their requests. This ensures the PIN knows which requests have interested volunteers and can take timely action.	
Actors: PIN	
Trigger: A CSR shortlists the PIN's help request for potential assistance.	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ PIN is authenticated and logged into their PIN portal ❖ PIN has access to the mail/notification system on the platform	
Main flow: 1. CSR shortlists a PIN's help request in the system. 2. System generates a notification in the PIN's mail/notification system 3. PIN clicks the notification pop up page to see the details 4. PIN clicks on that specific notification to remove it	
Alternative/Exceptional flow: NIL	

BCE (PIN - Taiga ID: #611)



Sequence Diagram (PIN - Taiga ID: #612)



Code snippet

Boundary

```
const PersonInNeedDashboard: React.FC = () => {
  const fetchNotifications = React.useCallback(() => {
    if (!userId) return;
    setNotiLoading(true);
    setNotiError("");
    fetch(`${API_BASE}/api/pin/notifications/${userId}`)
      .then(res => res.json())
      .then(data => {
        // Only keep 'interested' and 'shortlist' notification types for PIN dashboard
        const filtered = (data.data || []).filter((n: Notification) => n.type === 'interested' || n.type === 'shortlist');
        setNotifications(filtered);
        setNotiLoading(false);
        //setNotiHasUnread(filtered.some((n: Notification) => n.read === 0));
      })
      .catch(() => {
        setNotiError("Could not load notifications.");
        setNotiLoading(false);
      });
  }, [userId]);      You, last week • Update Pushh
```

Controller

```
export class PersonInNeedControllers {
  async getNotifications(pin_id: number) {
    return await PinRequestEntity.getNotifications(pin_id);
  }
```

Entities

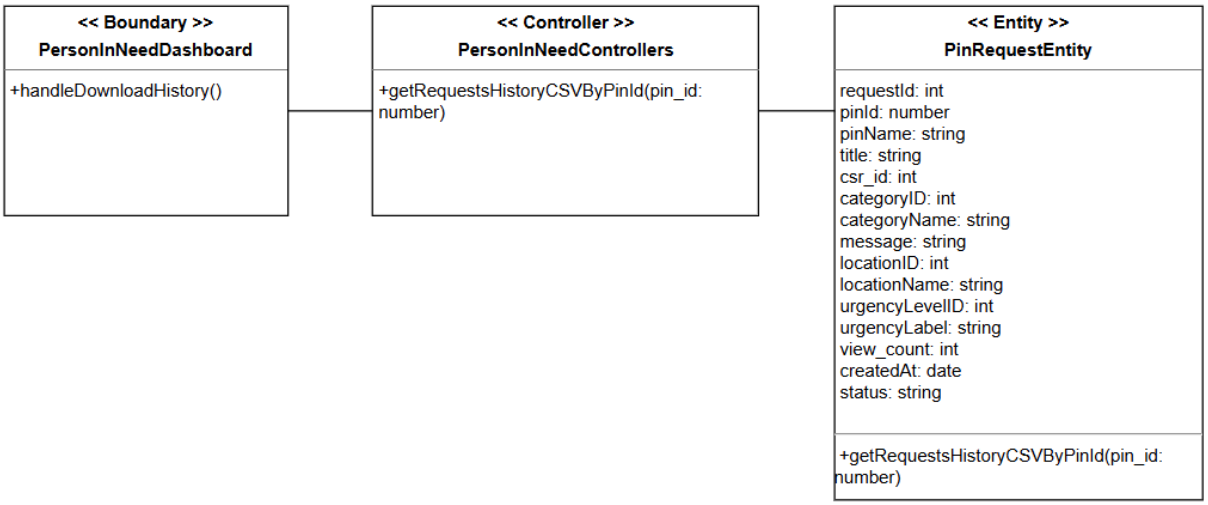
```
export class PinRequestEntity {
  static async getNotificationsForCsr(csr_id: number) {      Justin, 2 days ago • Move the stuff from controller
    // Join notificationTable with useraccountTable (PIN) and pin_requestsTable for username and title
    // Exclude notifications that are intended only for the PIN (e.g. 'interested_cancelled')
    // so CSR won't get a notification for their own cancel action.
    return await db
      .select({
        id: notificationTable.id,
        pin_id: notificationTable.pin_id,
        type: notificationTable.type,
        csr_id: notificationTable.csr_id,
        pin_request_id: notificationTable.pin_request_id,
        createdAt: notificationTable.createdAt,
        read: notificationTable.read,
        pinUsername: useraccountTable.username,
        requestTitle: pin_requestsTable.title,
      })
      .from(notificationTable)
      .leftJoin(useraccountTable, eq(notificationTable.pin_id, useraccountTable.id))
      .leftJoin(pin_requestsTable, eq(notificationTable.pin_request_id, pin_requestsTable.id))
      .where(and(not(eq(notificationTable.type, 'interested_cancelled')), eq(notificationTable.csr_id, csr_id)))
      .orderBy(desc(notificationTable.read), desc(notificationTable.createdAt));
  }
```

Wireframe (PIN - Taiga ID: #613):

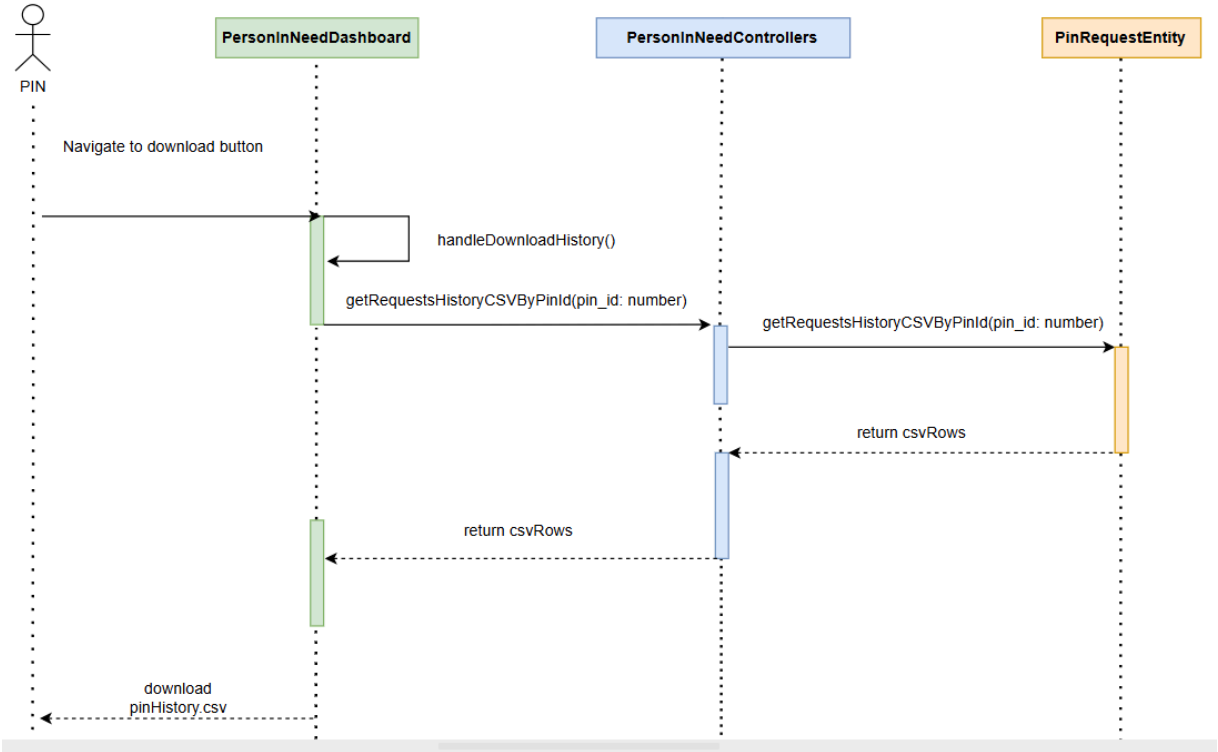
USER STORY (Taiga ID: #403)
As a PIN, I want to download my past service history so that I can keep a record for personal reference.

Name: PIN - Download Past Service History	**Taiga ID: #580**
Stakeholders and goals: PIN – To keep a personal record of all completed help requests for reference.	
Description: This use case allows a PIN to download a file containing all of their completed requests. The file provides a record of services received and the CSR volunteers who assisted.	
Actors: PIN	
Trigger: PIN clicks the Download Past Service History button.	
Pre-condition: ❖ User account exist in the system ❖ User has access to the login page ❖ PIN is authenticated and logged into their PIN portal ❖ The system supports exporting completed requests to a downloadable file (e.g., CSV).	
Main flow: 1. PIN navigates to the My Requests section. 2. The system displays all requests, highlighting completed requests. 3. PIN clicks Download Past Service History. 4. The system generates a file containing all completed requests and immediately downloads it. 5. Use case ends.	
Alternative/Exceptional flow: NIL	

BCE (PIN - Taiga ID: #583)



Sequence diagram (PIN - Taiga ID: #584)



Code snippet

Boundary

```
const PersonInNeedDashboard: React.FC = () => {
  const handleDownloadHistory = async () => {
    try {
      const res = await fetch(`${API_BASE}/api/pin/requests/history?pin_id=${userId}`);
      if (!res.ok) throw new Error('Failed to download history');
      const blob = await res.blob();
      const url = window.URL.createObjectURL(blob);
      const a = document.createElement('a');
      a.href = url;
      a.download = 'service-history.csv';
      document.body.appendChild(a);
      a.click();
      a.remove();
      window.URL.revokeObjectURL(url);      Justin, last week • PIN download past service
    } catch {
      toast.error('Could not download service history.');
    }
  };
```

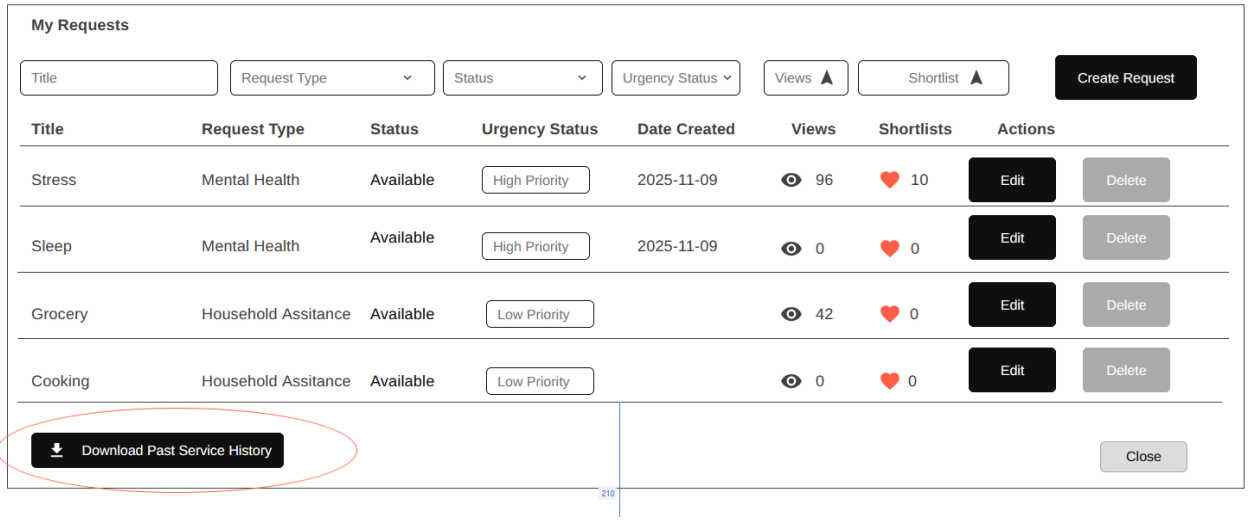
Controller

```
export class PersonInNeedControllers {
  async getRequestsHistoryCSVByPinId(pin_id: number): Promise<string> {
    return await PinRequestEntity.getRequestsHistoryCSVByPinId(pin_id);
  }
```

Entities

```
export class PinRequestEntity {
  static async getNotificationsForCsr(csr_id: number) {      Justin, 2 days ago • Move the stuff from controller to e
    // Join notificationTable with useraccountTable (PIN) and pin_requestsTable for username and title
    // Exclude notifications that are intended only for the PIN (e.g. 'interested_cancelled')
    // so CSR won't get a notification for their own cancel action.
    return await db
      .select({
        id: notificationTable.id,
        pin_id: notificationTable.pin_id,
        type: notificationTable.type,
        csr_id: notificationTable.csr_id,
        pin_request_id: notificationTable.pin_request_id,
        createdAt: notificationTable.createdAt,
        read: notificationTable.read,
        pinUsername: useraccountTable.username,
        requestTitle: pin_requestsTable.title,
      })
      .from(notificationTable)
      .leftJoin(useraccountTable, eq(notificationTable.pin_id, useraccountTable.id))
      .leftJoin(pin_requestsTable, eq(notificationTable.pin_request_id, pin_requestsTable.id))
      .where(and(not(eq(notificationTable.type, 'interested_cancelled')), eq(notificationTable.csr_id, csr_id)))
      .orderBy(desc(notificationTable.read), desc(notificationTable.createdAt));
  }
```

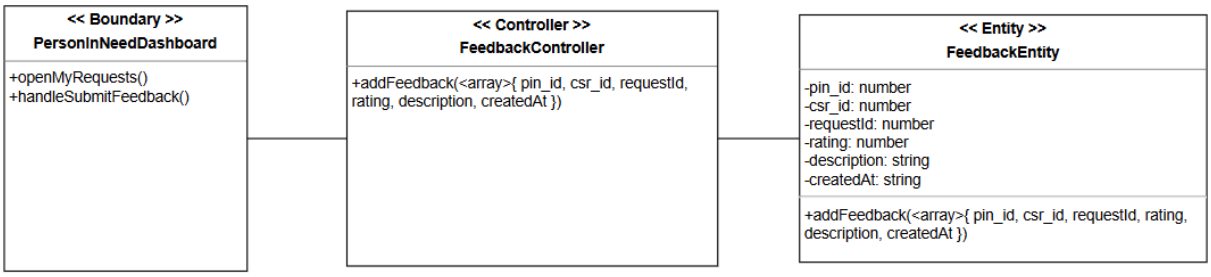
Wireframe (PIN - Taiga ID: #585):



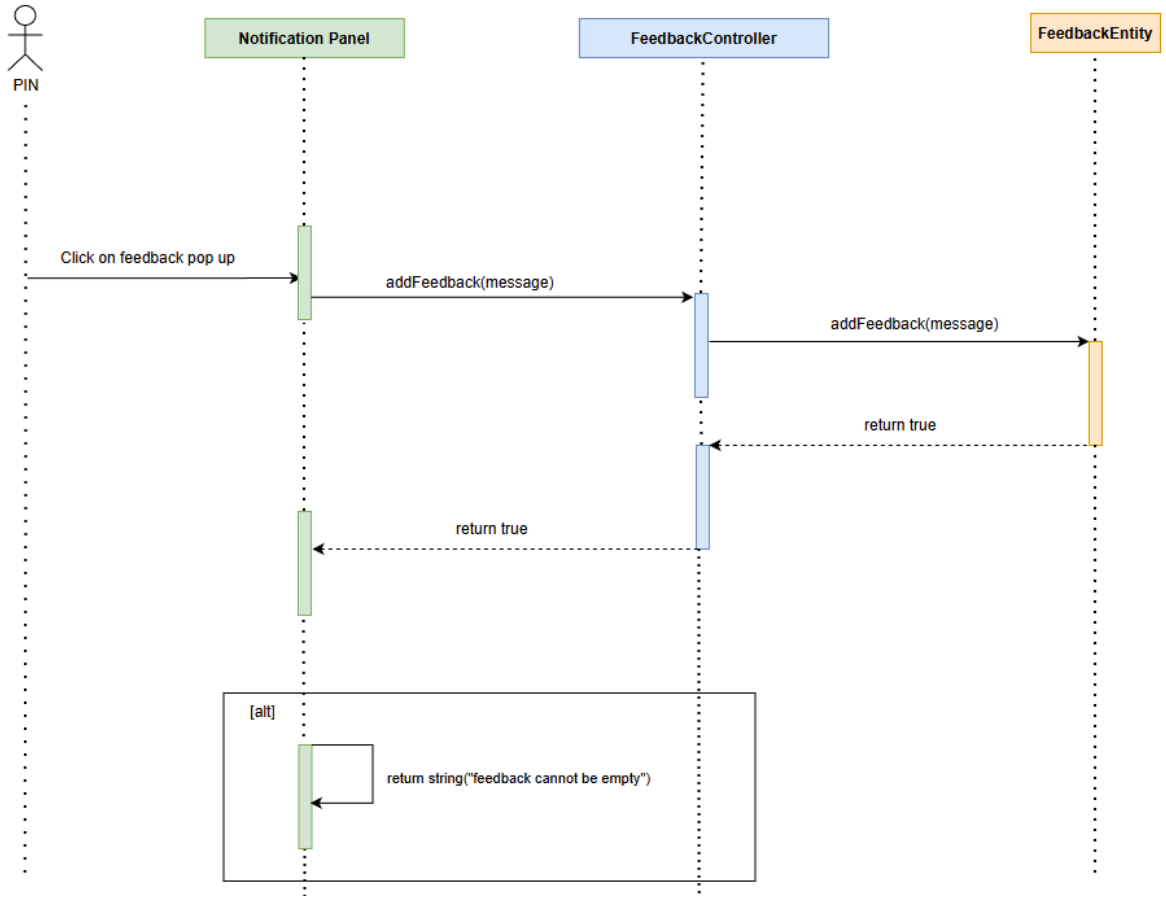
USER STORY (Taiga ID: #404)
As a PIN, I want to rate my experience with the CSR Rep so that I can give user feedback.

Name: PIN - Rate experience	**Taiga ID: #594**
Stakeholders and goals: PIN - Provide user feedback for improvement.	
Description: This use case allows the Person-in-Need to rate their experience and provide their own feedback for the CSR Representative, allowing the CSR Rep to understand what could be done better or improve on for the service they provide.	
Actors: Person-in-Need	
Trigger: The PIN clicks on "Rate the service" on the completed service dashboard.	
Pre-condition: ❖ The Person-in-Need is logged into the system. ❖ Have completed at least one request service completed by a CSR Representative. ❖ Requests have valid status values (Available, Pending, Completed).	
Main flow: 1. The Person-in-Need (PIN) navigates to their completed services tab. 2. The user clicks on the "Feedback" button next to the completed service. 3. A modal prompt comes up allowing the PIN to fill out the rating form. 4. The PIN submits the form. 5. A confirmation message appears on the screen: "Your rating has been submitted."	
Alternative/Exceptional flow: ❖ **A1. Failed to submit feedback** 4A. If user did not select the number of stars, it will not be able to send feedback	

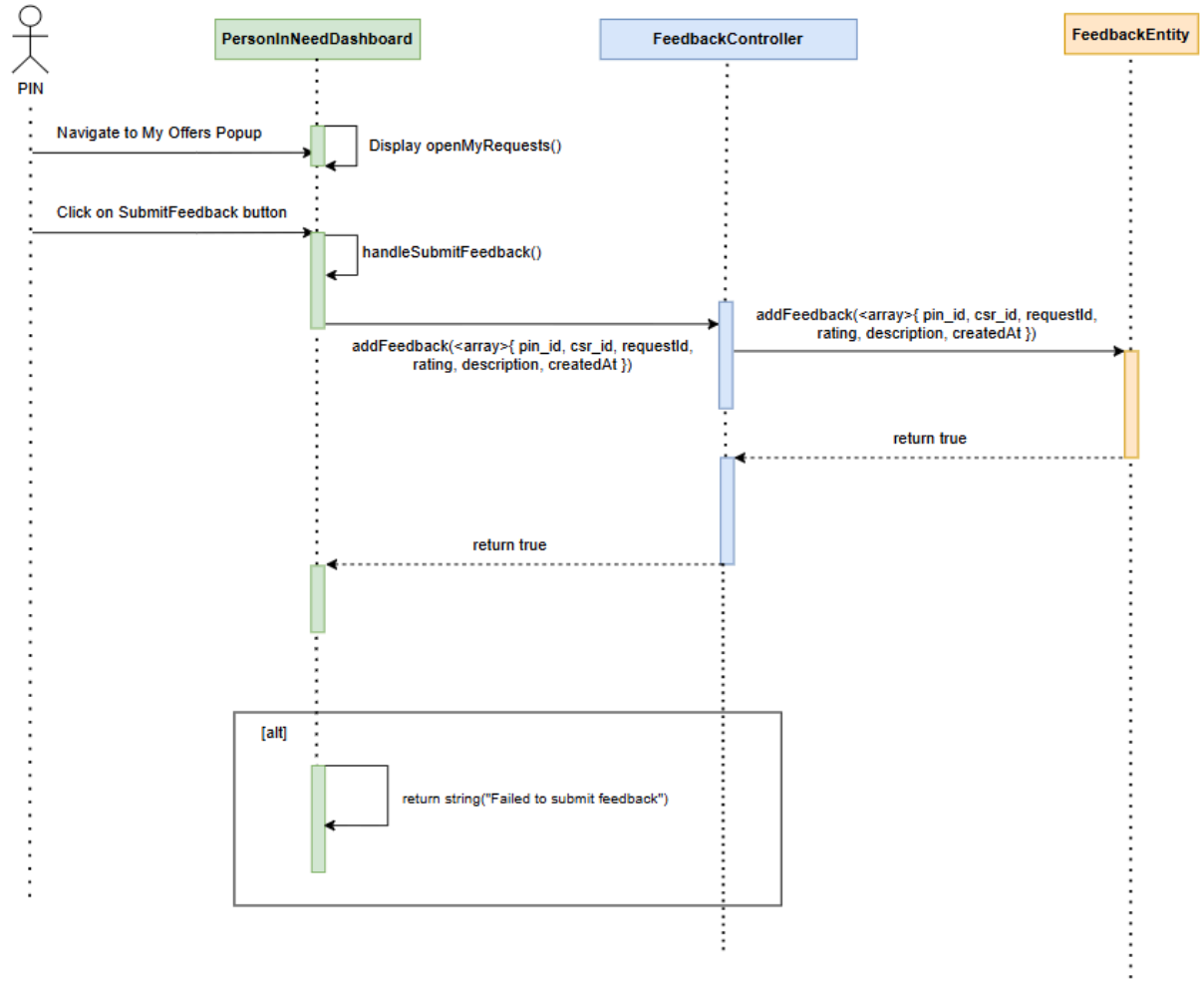
BCE Diagram (PIN - Taiga ID: #597)



Sequence Diagram (First Iteration - Sprint 1)



Sequence Diagram (PIN - Taiga ID: #598 - Final Iteration)



Code snippet
Boundary

```
const PersonInNeedDashboard: React.FC = () => {
  async function handleSubmitFeedback() {      Justin, 5 days ago • Feedback Featur
    if (!feedbackRequestId || !feedbackCsrId || feedbackRating === 0) return;
    setFeedbackLoading(true);
    const createdAt = new Date().toISOString();
    try {
      const res = await fetch(`${API_BASE}/api/pin/feedback`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          requestId: feedbackRequestId,
          csrId: feedbackCsrId,
          pinId: userId,
          rating: feedbackRating,
          description: feedbackText,
          createdAt,
        }),
      });
      if (res.ok) {
        toast.success('Feedback submitted!');
        // Refetch offers from backend to get updated feedback data
        openMyOffers();
        handleCancelFeedback();
      } else {
        toast.error('Failed to submit feedback.');
      }
    } catch {
      toast.error('Failed to submit feedback.');
    }
    setFeedbackLoading(false);
  }
```

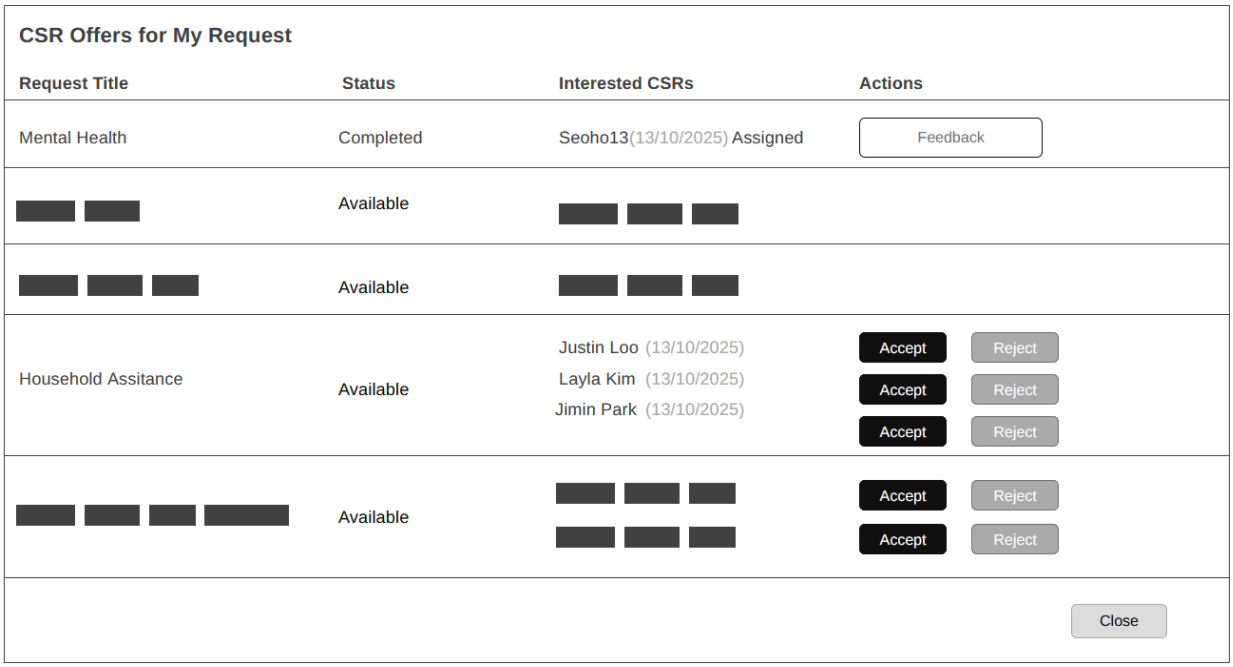
Controller

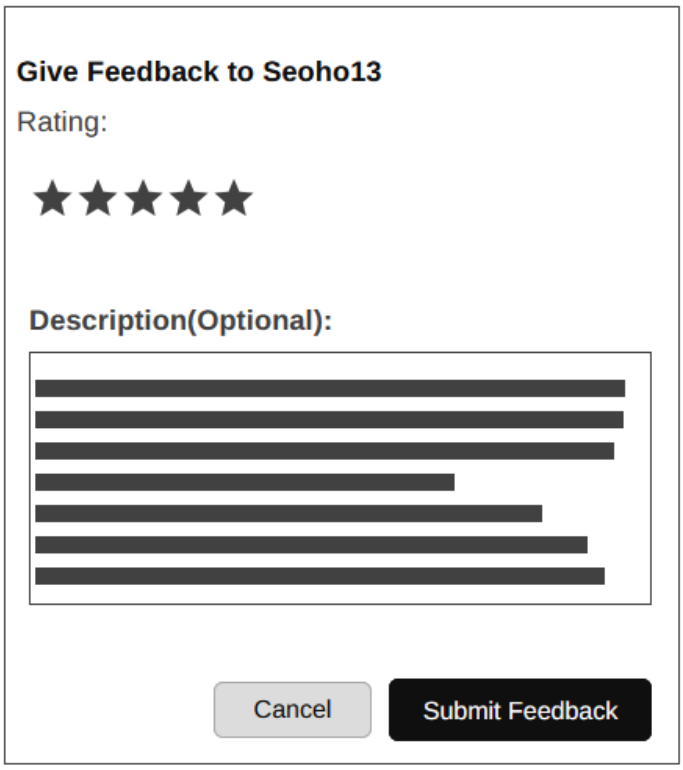
```
export class FeedbackController {
  // Service method: accepts a plain object and returns the entity result (throws on error)
  static async addFeedback({ pinId, csr_id, requestId, rating, description, createdAt }: {
    pinId: number;
    csr_id: number;
    requestId: number;
    rating: number;
    description?: string;
    createdAt: string;
  }) {
    // Basic validation
    if (!pinId || !csr_id || !requestId || !rating || !createdAt) {      You, 7 minutes ago • U
      throw new Error('Missing required fields');
    }
    // Delegate to entity (field names expected by entity)
    return await FeedbackEntity.addFeedback({
      pin_id: pinId,
      csr_id: csr_id,
      requestId: requestId,
      rating: rating,
      description: description ?? '',
      createdAt: createdAt,
    });
  }
}
```

Entities

```
export class FeedbackEntity {
  static async addFeedback({ pin_id, csr_id, requestId, rating, description, createdAt }: {
    pin_id: number;
    csr_id: number;
    requestId: number;
    rating: number;
    description?: string;
    createdAt: string;
  }) {
    await db.insert(feedbackTable).values({
      pin_id,
      csr_id,
      requestId,
      rating,
      description,
      createdAt: new Date(createdAt),
    });
    // Insert notification for CSR
    await db.insert(require('../db/schema/aiodb').notificationTable).values({
      pin_id,
      csr_id,
      pin_request_id: requestId,
      type: 'feedback',
      createdAt: new Date(createdAt),
      read: 0,
    });
    return { success: true };
  }
}
```

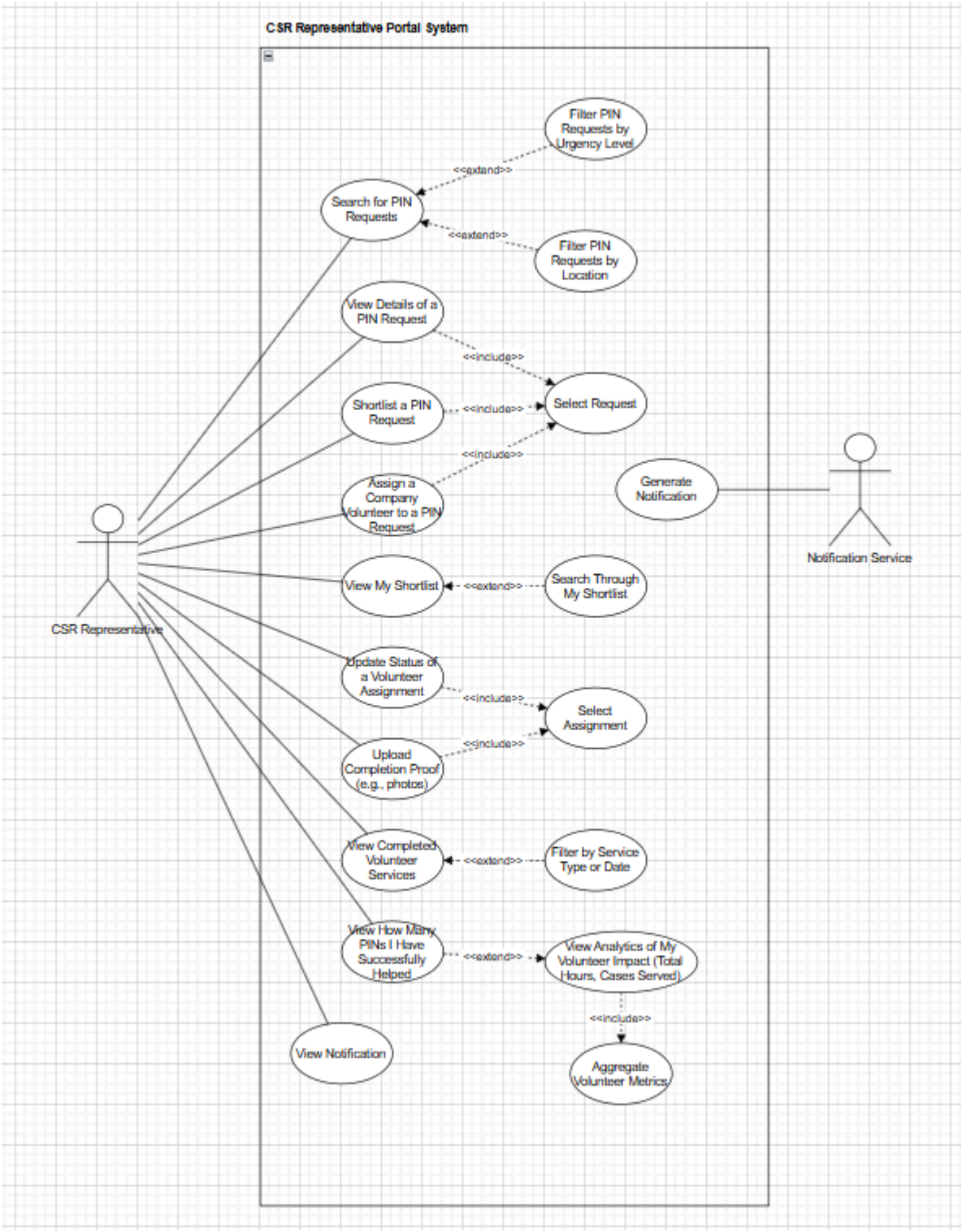
Wireframe (PIN - Taiga ID: #599)





3.3. Corporate Social Responsibility (CSR) Representative

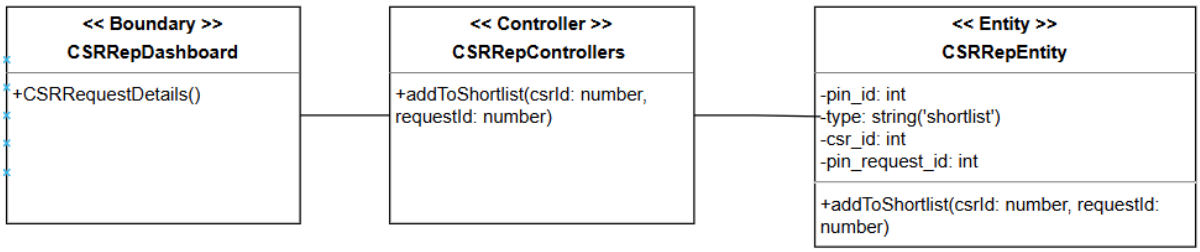
Use Case Diagram :



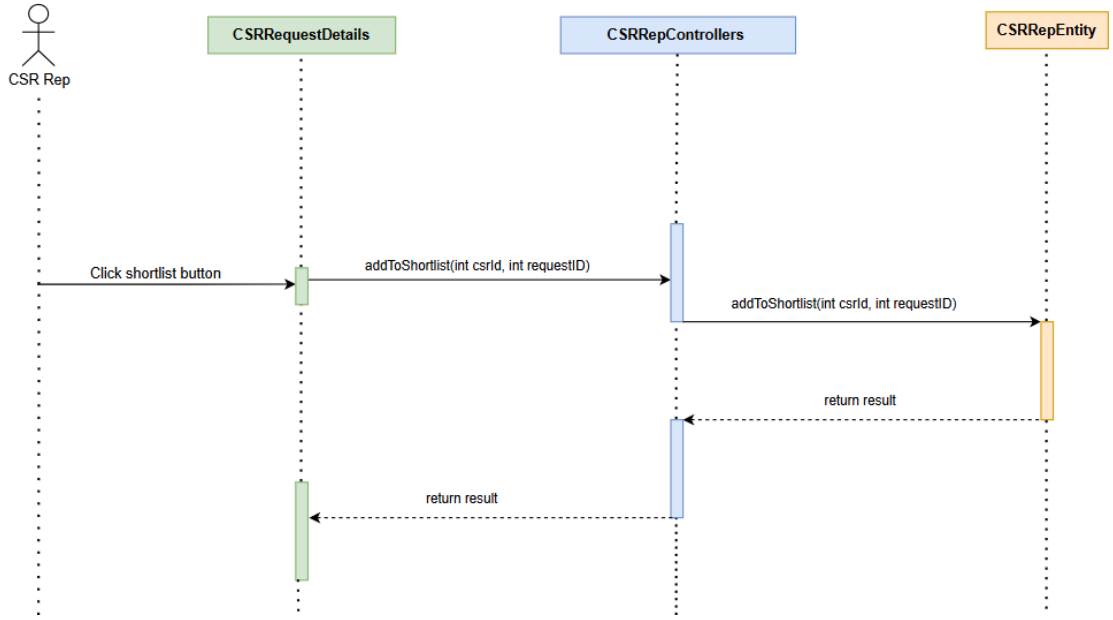
USER STORY (Taiga ID: #405)
As a CSR Rep, I want to shortlist PIN requests so that I can review them later.

Name: CSR Rep - Shortlist Pin Request	**Taiga ID: #615**
Stakeholders and goals: CSR Representative - Shortlist Pin Request	
Description: This use case allows a CSR Representative to save PIN requests to a personal shortlist for future reference, enabling them to efficiently track and review the request.	
Actors: Corporate Social Responsibility(CSR) Representative	
Trigger: CSR Rep identifies a PIN request they want to review later or consider for assignment planning	
Pre-condition: ❖ CSR Rep is logged into the system with appropriate permission ❖ At least one Person In Need Request exist in the system ❖ CSR Rep has access to view PIN Requests ❖ Shortlist functionality is enabled for the user's role	
Main flow: 1. CSR Rep navigates to the Available Request Page 2. System displays a list of available Person in Need requests 3. CSR Rep reviews PIN request details 4. CSR Rep selects the heart shape icon opinion for the chosen PIN request 5. System validates the request and adds the PIN request to the CSR Rep's My Shortlist page 6. CSR Rep can continue browsing or navigate to their shortlist to view saved items 7. Use case ends	
Alternative/Exceptional flow: NIL	

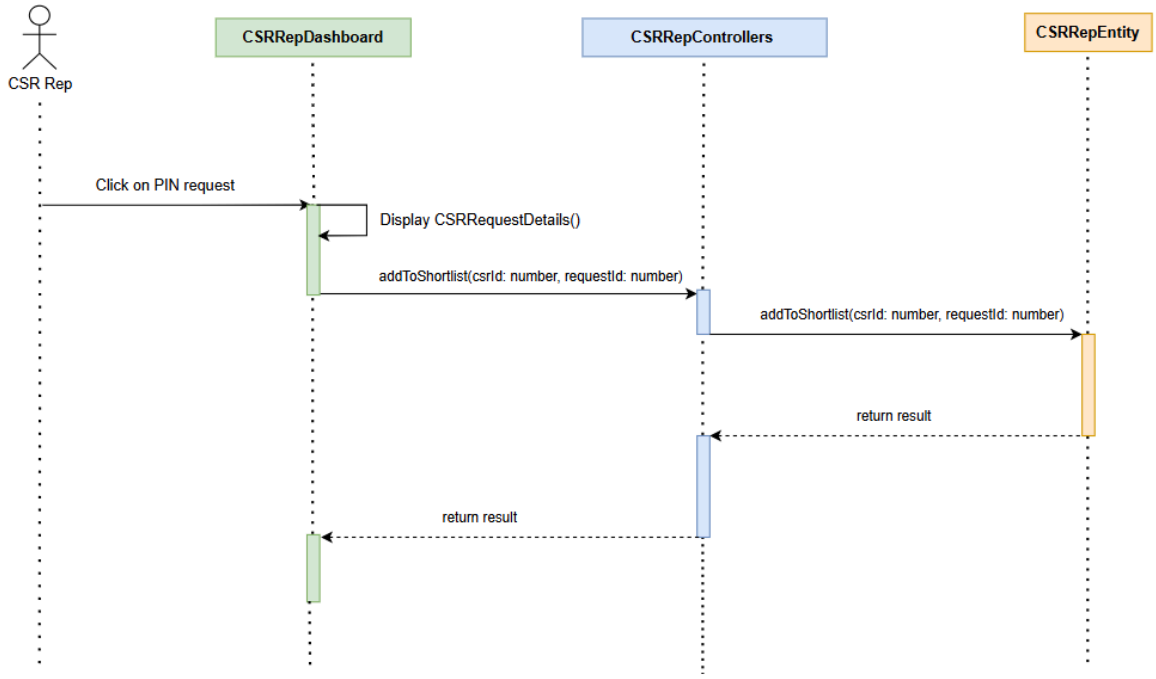
BCE Diagram: (CSR - Taiga ID: #618)



Sequence Diagram (First Iteration - Sprint 1)



Sequence Diagram: (CSR - Taiga ID: #619)



Code snippet

Boundary

```
on CSRRequestDetails({ request, onClose, csrId, shortlistedIds, interestedIds, reloadShortlist, reloadInterested }: CSRRequestDet
      onClick={async () => {
    }}
  >
    {interestedIds.includes(request.requestId) ? "Remove Interest" : "Interested"}
  </button>
  <button
    className="csr-heart-btn"
    style={{
      minWidth: 48,
      fontSize: 28,
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
      padding: '0 0.5em',
      background: 'none',
      border: 'none',
      cursor: csrId ? 'pointer' : 'not-allowed',
      opacity: csrId ? 1 : 0.5,
      transition: 'transform 0.1s',
    }}
    disabled={!csrId}
    onClick={async () => {
      if (!csrId) {
        alert("You must be logged in as a CSR rep to shortlist requests.");
        return;
      }
      let resp;
      try {
        if (shortlistedIds.includes(request.requestId)) {
          resp = await fetch(`http://localhost:3000/api/csr/${csrId}/shortlist/${request.requestId}`, { method: "DELETE" });
        } else {
          resp = await fetch(`http://localhost:3000/api/csr/${csrId}/shortlist/${request.requestId}`, { method: "POST" });
          if (resp.ok) {
            // Increment shortlist count in pin_requests
            await fetch(`http://localhost:3000/api/pin/requests/${request.requestId}/increment-shortlist`, { method: 'POST' });
          }
        }
        if (!resp.ok) {
          const err = await resp.text();
          alert(`Failed to update shortlist: ${err}`);
```

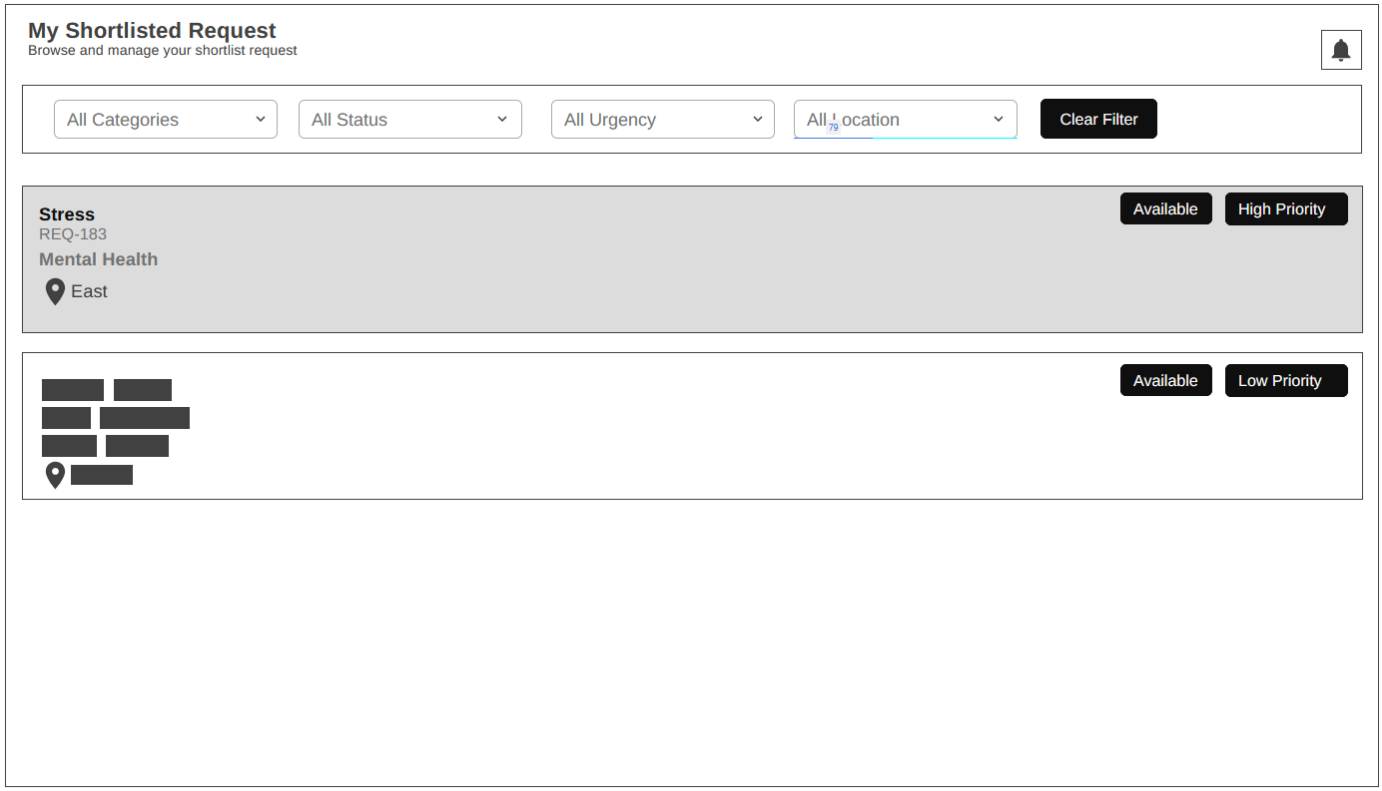
Controller

```
export class CSRRepControllers {

    async addToShortlist(csrId: number, requestId: number) {
        return await CSRRepEntity.addToShortlist(csrId, requestId);
    }
```

Entity

```
export class CSRRepEntity {

    static async addToShortlist(csrId: number, requestId: number) {
        // Insert ignore duplicate
        const result = await db.insert(csr_shortlistTable).values({ csr_id: csrId, pin_request_id: requestId }).execute();
        // Get pin_id for the request
        const req = await db.select({ pin_id: pin_requestsTable.pin_id }).from(pin_requestsTable).where(eq(pin_requestsTable.id,
        if (req && req[0] && req[0].pin_id) {
            await db.insert(require('../db/schema/aiodb').notificationTable).values({
                pin_id: req[0].pin_id,
                type: 'shortlist',
                csr_id: csrId,
                pin_request_id: requestId,
            }).execute();
        }
        return result;
    }
```

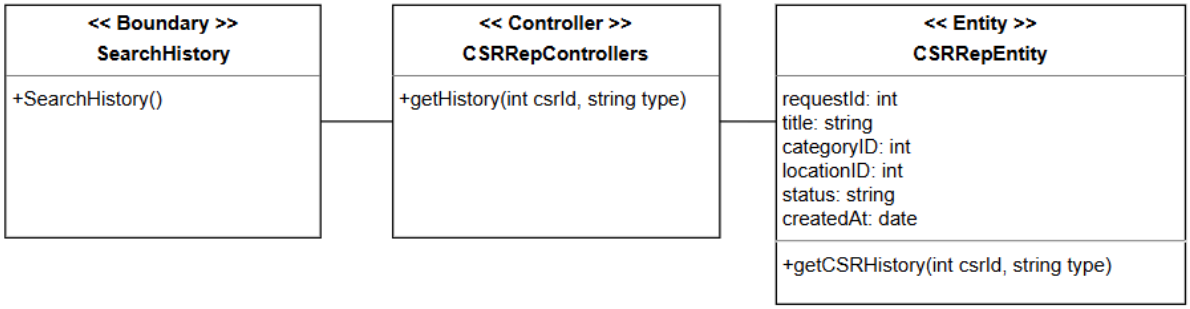
Wireframe (PIN - Taiga ID: #620):



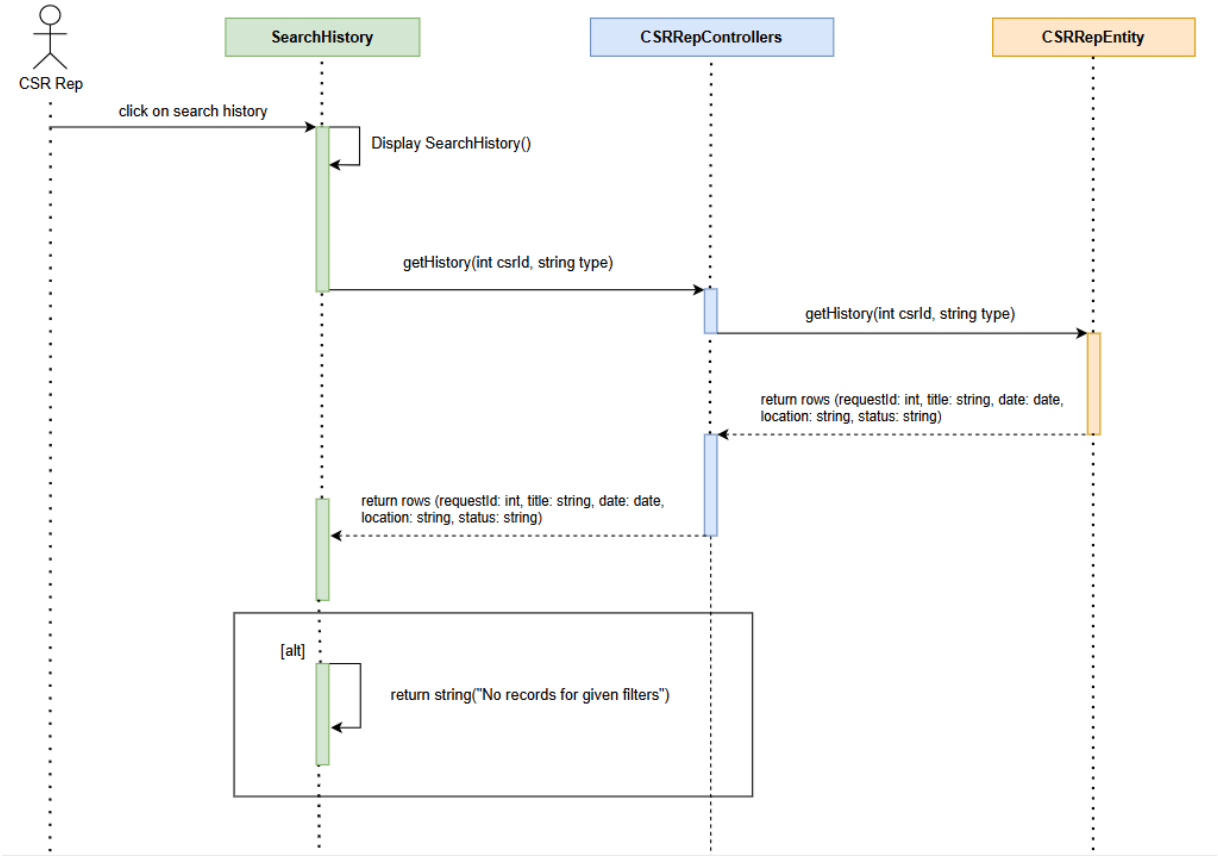
USER STORY (Taiga ID: #406)
As a CSR Rep, I want to filter my completed volunteer services by service type or date, so that i can track past volunteer efforts

Name: CSR Rep - Search History	**Taiga ID: #552**
Stakeholders and goals: CSR Representative - Search History	
Description: This use case enables a CSR Representative to search and view their personal history of completed volunteer service activities. The CSR Representative can filter results by service type or date range to review and track their past volunteering contributions.	
Actors: Corporate Social Responsibility(CSR) Representative	
Trigger: The CSR Representative selects the "Search History" or "View My History" option from their dashboard or navigation menu	
Pre-condition: ❖ The CSR Representative is logged into the system ❖ The CSR Representative has completed at least one volunteer service activity that has been recorded in the system ❖ The CSR Representative has appropriate permission to view their own service history	
Main flow: 1. The CSR Representative navigates to Search History page 2. System display the search interface with filter options (service type / date range) 3. The CSR Representative clicks on the "Search" or "Apply Filter" 4. System retrieves and displays matching volunteer service records 5. System shows results including: service name, date, location, service type, status 6. The CSR Representative reviews the search results	
Alternative/Exceptional flow: **A1. No Results found** 3a. If no records match the search criteria, systems display "No record for given filters".	

BCE Diagram (CSR - Taiga ID: #556):



Sequence Diagram (CSR - Taiga ID: #555):



Code snippet

Boundary

```
function SearchHistory() {
  const filtered = useMemo(() => {
    return DATA.filter(row => {
      const iend = !end || row.date <= end;
      return itype && istart && iend;
    });
  }, [DATA, svcType, start, end]);
  const totals = {
    total: filtered.length,
    pending: filtered.filter(r => r.status === "Pending").length,
    completed: filtered.filter(r => r.status === "Completed").length,
  };
  const clear = () => {
    setSvcType("All Types");
    setStart("");
    setEnd("");
  };
```

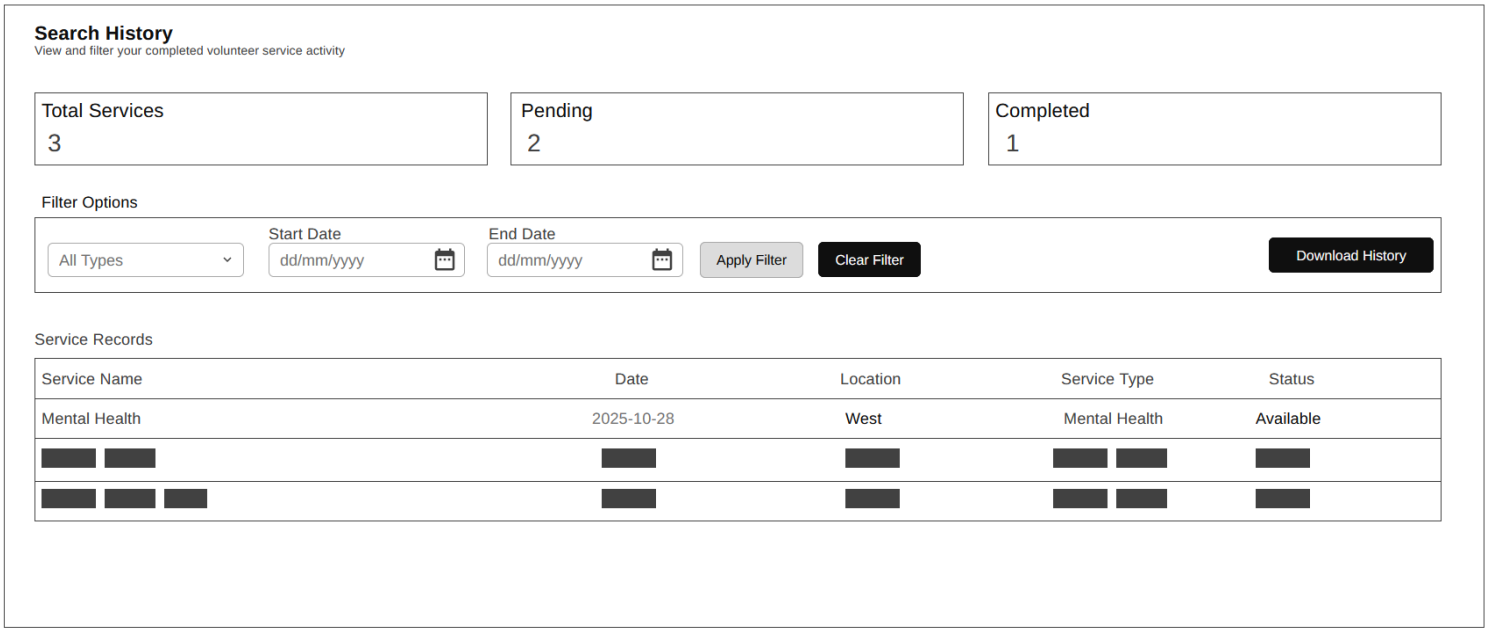
Controller

```
export class CSRRepControllers {
    // Get CSR history with filter
    async getHistory(csrId: number, type?: string) {
        return await CSRRepEntity.getCSRHistory(csrId, type);
    }
```

Entity

```
export class CSRRepEntity {
    // Get CSR history: all requests assigned to this CSR, with service type and location names, and filter by type
    static async getCSRHistory(csrId: number, type?: string) {
        // Get all requests assigned to this CSR
        let query = db
            .select({
                requestId: pin_requestsTable.id,
                title: pin_requestsTable.title,
                categoryID: pin_requestsTable.categoryID,
                locationID: pin_requestsTable.locationID,
                status: pin_requestsTable.status,
                createdAt: pin_requestsTable.createdAt,
            })
            .from(pin_requestsTable)
            .where(eq(pin_requestsTable.csr_id, csrId));

        // If type is specified and not "All Types", filter by service_type name
        if (type && type !== "All Types") {
            // Get service_type id for the given name
            const typeRow = await db.select().from(service_typeTable).where(eq(service_typeTable.name, type)).limit(1);
            if (typeRow && typeRow[0] && typeRow[0].id) {
                query = db
                    .select({
                        requestId: pin_requestsTable.id,
                        title: pin_requestsTable.title,
                        categoryID: pin_requestsTable.categoryID,
                        locationID: pin_requestsTable.locationID,
                        status: pin_requestsTable.status,
                        createdAt: pin_requestsTable.createdAt,
                    })
                    .from(pin_requestsTable)
                    .where(and(eq(pin_requestsTable.csr_id, csrId), eq(pin_requestsTable.categoryID, typeRow[0].id)));
            }
        }
```

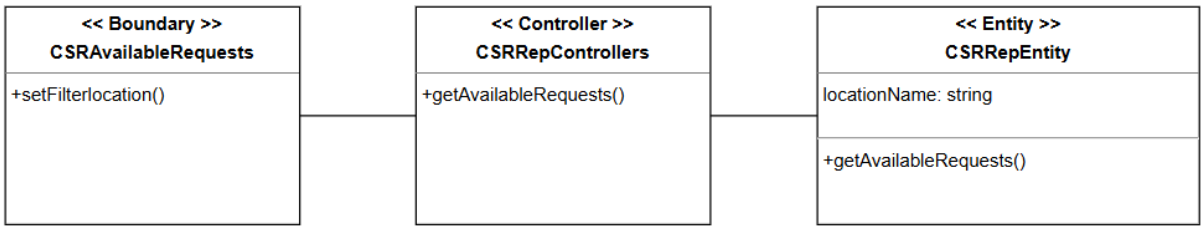
Wireframe (CSR - Taiga ID: #557):



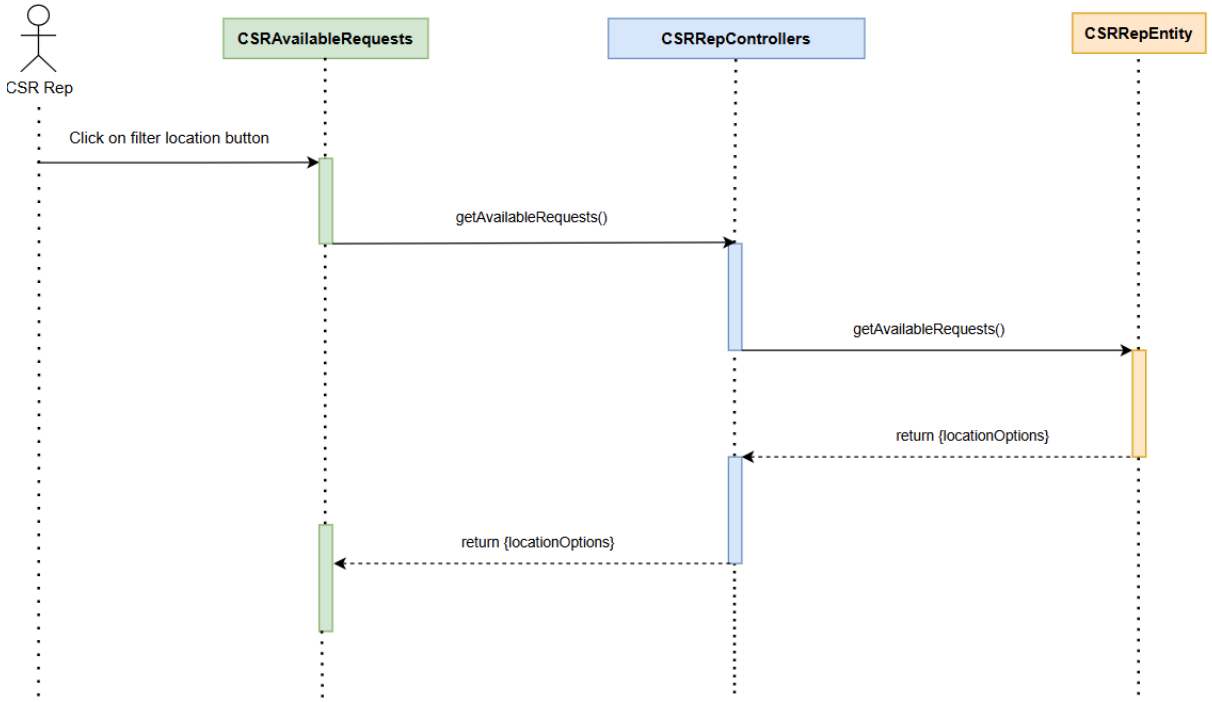
USER STORY (Taiga ID: #407)
As a CSR Rep, I want to filter PIN request by location so that i can find nearby volunteer opportunities

Name: Filter the pin request by Location	**Taiga ID: #559**
Stakeholders and goals: CSR Representative - Filter the pin request	
Description: This use case allows a CSR Representative to filter PIN requests based on location.	
Actors: Corporate Social Responsibility(CSR) Representative	
Trigger: CSR Representative needs to find volunteer opportunities near specific MRT Stations	
Pre-condition: ❖ CSR Representative navigate to the system with appropriate permissions ❖ CSR Representative has access to view PIN request ❖ At least one PIN Request exist in the system with Location information	
Main flow: 1. CSR Representative navigates to the PIN request section 2. System displays all available PIN requests with default view 3. CSR Representative selects the “Filter by location” 4. System presents Location (eg. North, South, East, West, Central) 5. System process the filter criteria and retrieve matching PIN requests’ 6. The system displays filtered PIN requests organized by district. 7. CSR Representative can select individual PIN requests to view full details 8. Use case ends	
Alternative/Exceptional flow:NIL	

BCE Diagram (CSR - Taiga ID: #562):



Sequence Diagram (CSR - Taiga ID: #563):



Code snippet

Boundary

```
const Available: React.FC = () => {
      <select value={filterLocation} onChange={e => setFilterLocation(e.target.value)} className="filter-available">
        <option value="">All Locations</option>
        {locationOptions.map(opt => <option key={opt} value={opt}>{opt}</option>)}
      </select>
      <select value={filterUrgency} onChange={e => setFilterUrgency(e.target.value)} className="filter-available">
        <option value="">All Urgency</option>
        {urgencyOptions.map(opt => <option key={opt} value={opt}>{opt}</option>)}
      </select>
      <select value={filterStatus} onChange={e => setFilterStatus(e.target.value)} className="filter-available">
        <option value="">All Status</option>
        <option value="Available">Available</option>
        <option value="Pending">Pending</option>
        <option value="Completed">Completed</option>
      </select>
      <button className="reset-available" onClick={() => { setFilterCategory(""); setFilterLocation(""); setFilterUrgency(""); 
    </div>
```

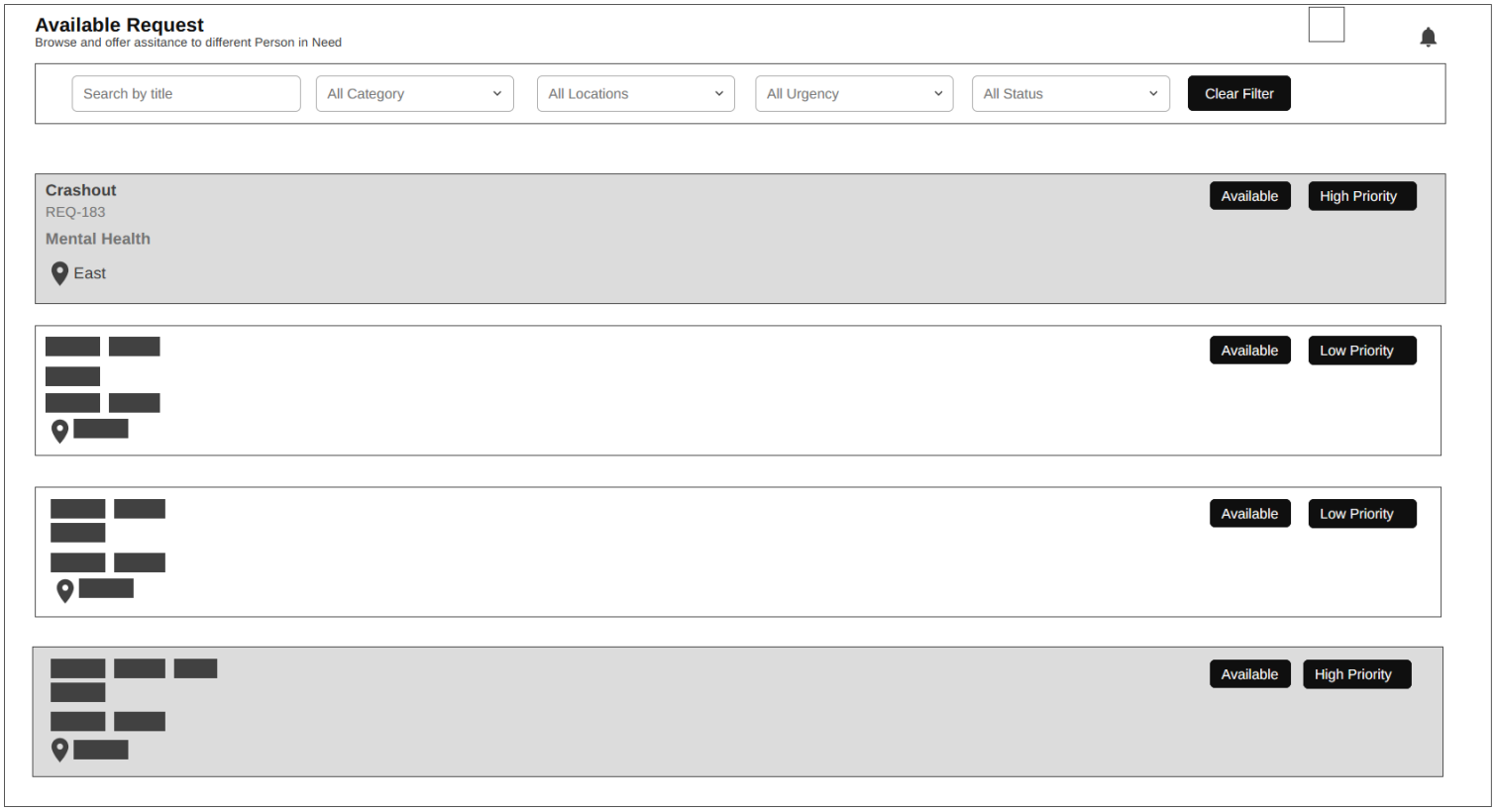
Controller

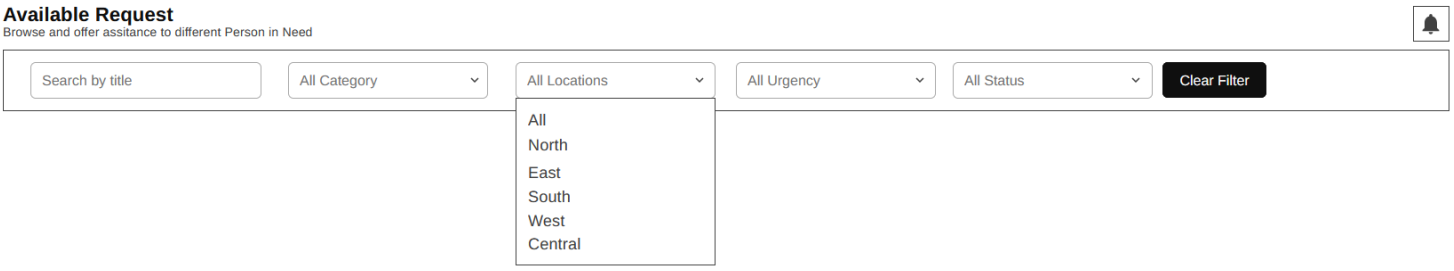
```
export class CSRRepControllers {
    }
    async getAvailableRequests() {
        return await CSRRepEntity.getAvailableRequests();
    }
```

Entity

```
export class CSRRepEntity {
        // Get all available requests (not assigned to a CSR, status Available), with joined labels
        static async getAvailableRequests() {
            const rows = await db
                    .select({
                        requestId: pin_requestsTable.id,
                        title: pin_requestsTable.title,
                        categoryID: pin_requestsTable.categoryID,
                        locationID: pin_requestsTable.locationID,
                        urgencyLevelID: pin_requestsTable.urgencyLevelID,
                        status: pin_requestsTable.status,
                        message: pin_requestsTable.message,
                        pinName: useraccountTable.username,
                    })
                .from(pin_requestsTable)
                .where(eq(pin_requestsTable.status, 'Available'))
                .leftJoin(useraccountTable, eq(pin_requestsTable.pin_id, useraccountTable.id));

            // Join to get category, location, urgency labels
            const serviceTypes = Object.fromEntries(
                (await db.select().from(service_typeTable)).map(st => [st.id, st.name])
            );
            const locations = Object.fromEntries(
                (await db.select().from(locationTable)).map(l => [l.id, l.name])
            );
            const urgencies = Object.fromEntries(
                (await db.select().from(urgency_levelTable)).map(u => [u.id, u.label])
            );

            return rows.map(r => ({
                requestId: r.requestId,
                title: r.title,
                categoryName: serviceTypes[r.categoryID] || '',
                location: r.locationID ? (locations[r.locationID] || '') : '',
                urgencyLevel: r.urgencyLevelID ? (urgencies[r.urgencyLevelID] || null) : null,
                status: r.status,
                message: r.message,
                pinName: r.pinName || null,
            }));
        }
```

Wireframe (CSR - Taiga ID: #564):

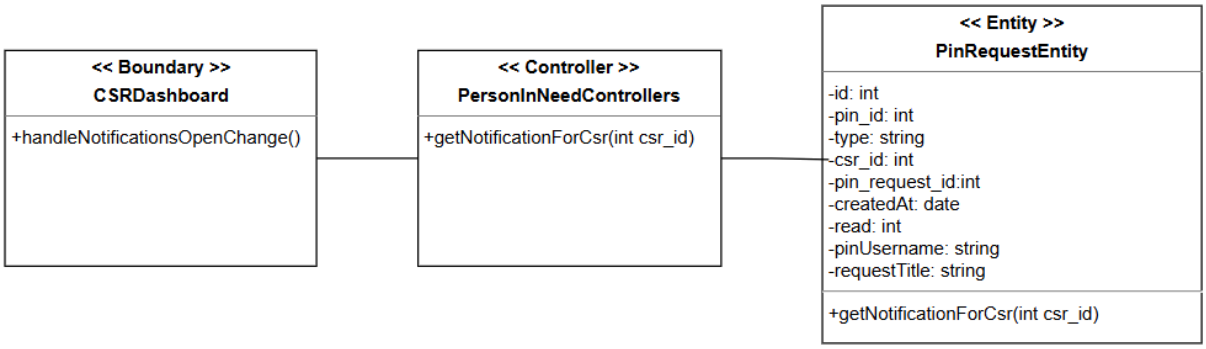




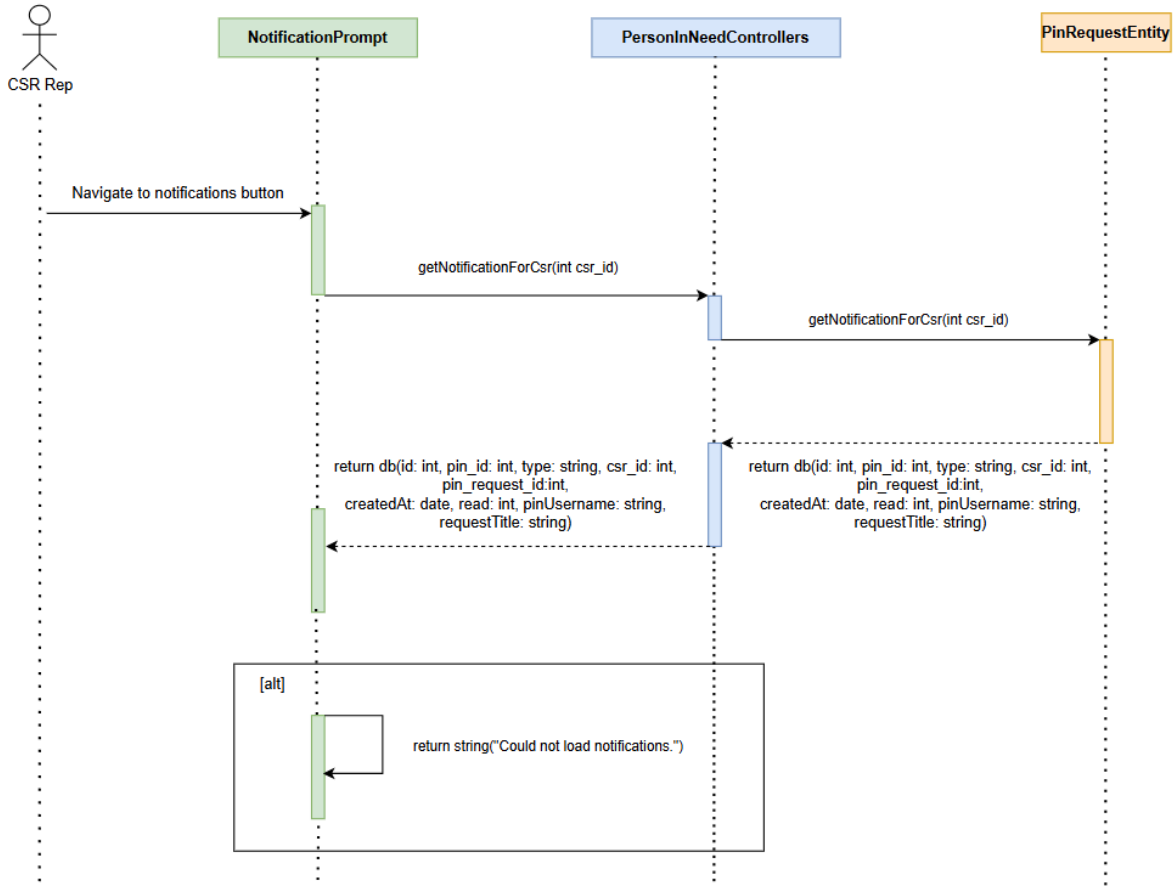
USER STORY (Taiga ID: #408)
As a CSR Rep, I want to receive notifications when a PIN accepts my volunteer request so that I can respond to it promptly.

Name: CSR Rep - Receive notifications upon being accepted.	**Taiga ID: #573**
Stakeholders and goals: CSR Representative - Receive notifications upon having their volunteer request accepted.	
Description: This use case allows the CSR Representative to receive a notification as a prompt whenever a Person-in-Need (PIN) accepts their volunteer request so that they can respond to it quickly and be able to provide support in a timely manner. This will provide effective engagement between the CSR Rep and the PIN and improve responsiveness.	
Actors: Corporate Social Responsibility (CSR) Representative	
Trigger: A Person-in-Need (PIN) accepts the CSR Representative's volunteer request.	
Pre-condition: ❖ The CSR Rep is logged into the system with the appropriate permission. ❖ The CSR Rep has requested to volunteer for one of the PIN's requests. ❖ The PIN has decided to accept the CSR Rep's volunteer request.	
Main flow: 1. A notification pop up appears on the screen: "Your volunteer request has been accepted." 2. CSR Rep can view the accepted request with the button: "View Request". 3. The system will display the request for further information and details of the accepted request with the Person-in-Need.	
Alternative/Exceptional flow: NIL	

BCE Diagram (CSR - Taiga ID: #576):



Sequence Diagram (CSR - Taiga ID: #577):



Code snippet

Boundary

```
function CSRDashboard({ onLogout }: CSRDashboardProps) {

    {/* Notification popover (restored Radix design) */}
    <div className="CSR-notification-popover-wrapper">
      <Popover.Root open={notificationsOpen} onOpenChange={handleNotificationsOpenChange}>
        <Popover.Trigger asChild>
          <button
            className="CSR-notification-button"
            aria-haspopup="true"
            aria-expanded={notificationsOpen}
            title="Notifications"
          >
            <Bell className="icon" />
            {unreadCount > 0 && (
              <span className="CSR-badge" aria-hidden>
                {unreadCount}
              </span>
            )}
          </button>
        </Popover.Trigger>
```

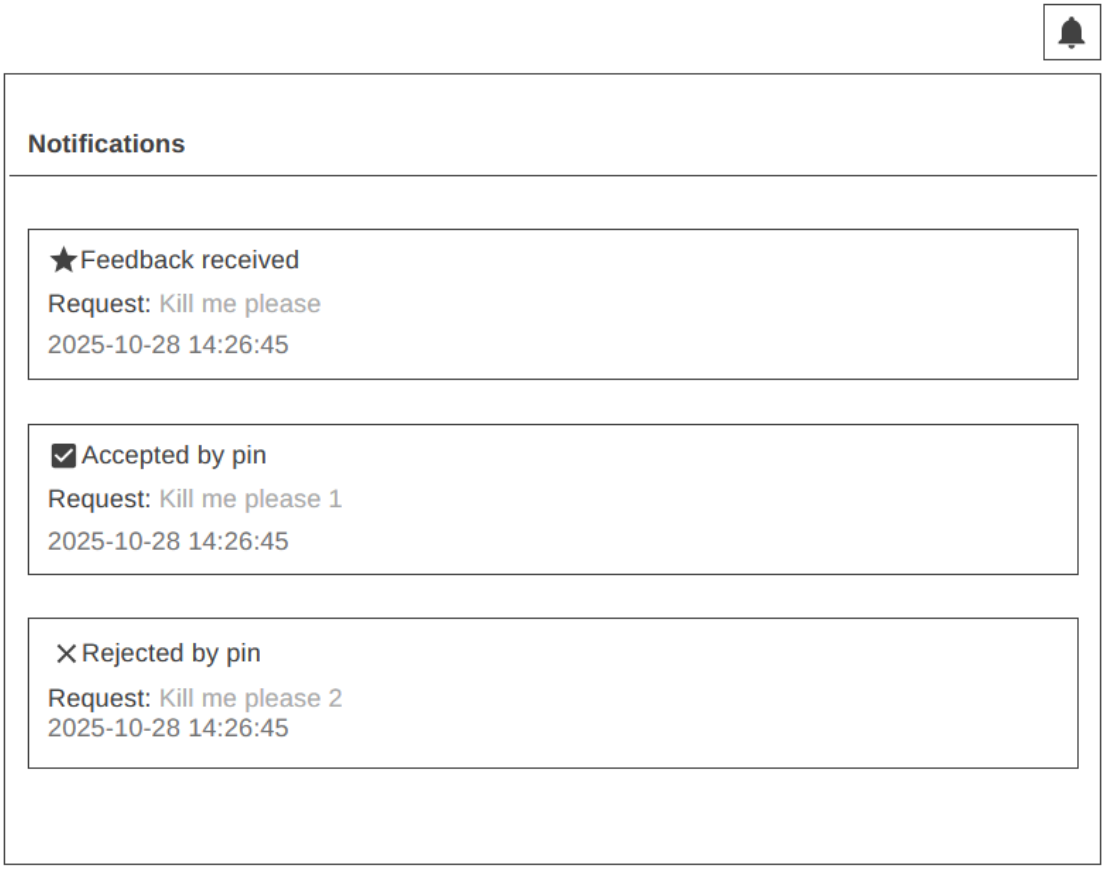
Controller

```
export class PersonInNeedControllers {
  // Fetch notifications for a CSR user
  async getNotificationsForCsr(csr_id: number) {
    return await PinRequestEntity.getNotificationsForCsr(csr_id);
  }
```

Entity

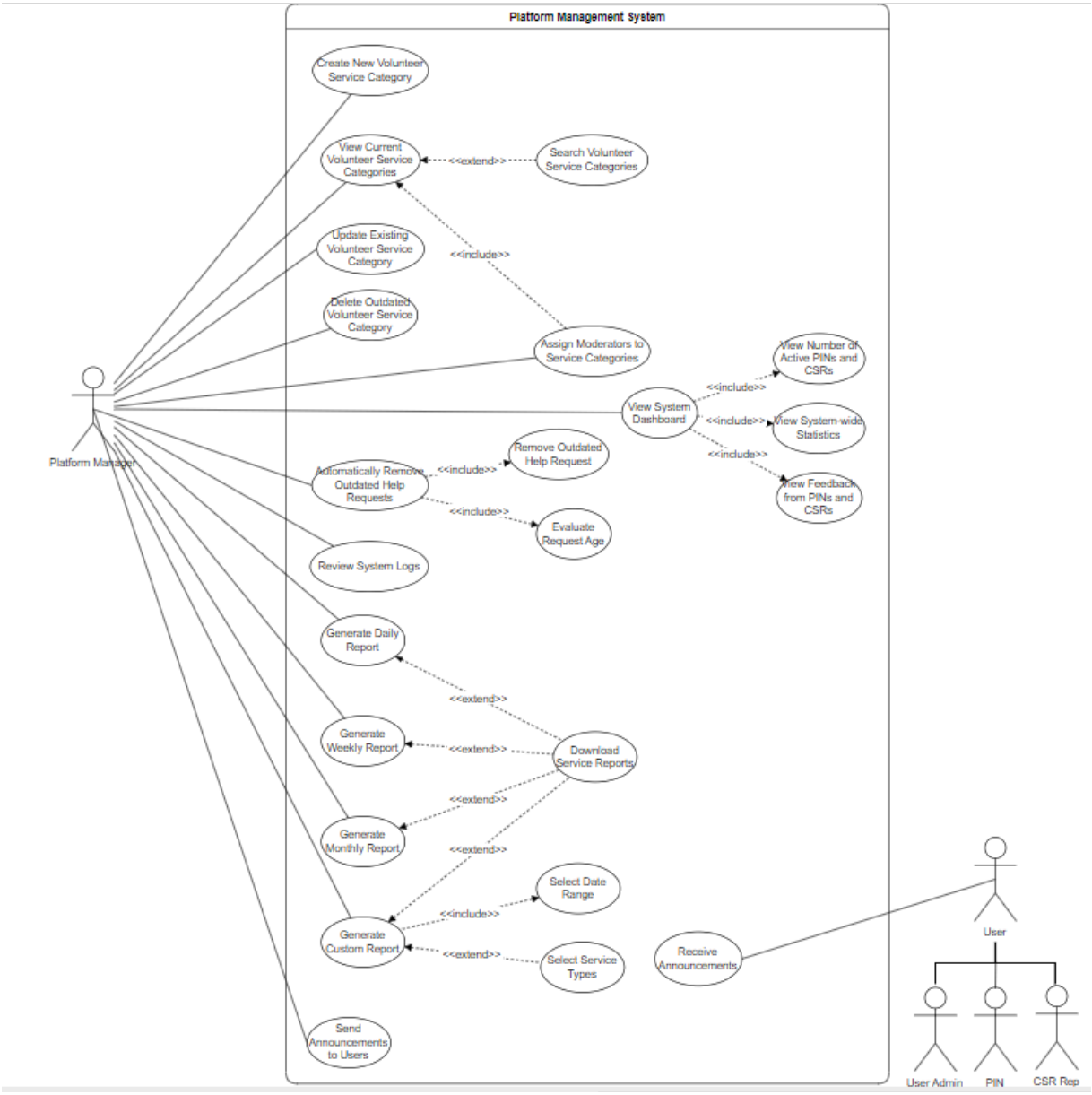
```
export class PinRequestEntity {
  // Fetch notifications for a CSR user
  static async getNotificationsForCsr(csr_id: number) {
    // Join notificationTable with useraccountTable (PIN) and pin_requestsTable for username and title
    // Exclude notifications that are intended only for the PIN (e.g. 'interested_cancelled')
    // so CSR won't get a notification for their own cancel action.
    return await db
      .select({
        id: notificationTable.id,
        pin_id: notificationTable.pin_id,
        type: notificationTable.type,
        csr_id: notificationTable.csr_id,
        pin_request_id: notificationTable.pin_request_id,
        createdAt: notificationTable.createdAt,
        read: notificationTable.read,
        pinUsername: useraccountTable.username,
        requestTitle: pin_requestsTable.title,
      })
      .from(notificationTable)
      .leftJoin(useraccountTable, eq(notificationTable.pin_id, useraccountTable.id))
      .leftJoin(pin_requestsTable, eq(notificationTable.pin_request_id, pin_requestsTable.id))
      .where(and(not(eq(notificationTable.type, 'interested_cancelled')), eq(notificationTable.csr_id, csr_id)))
      .orderBy(desc(notificationTable.read), desc(notificationTable.createdAt));
  }
```

Wireframe (CSR - Taiga ID: #578):



3.4. Platform Manager

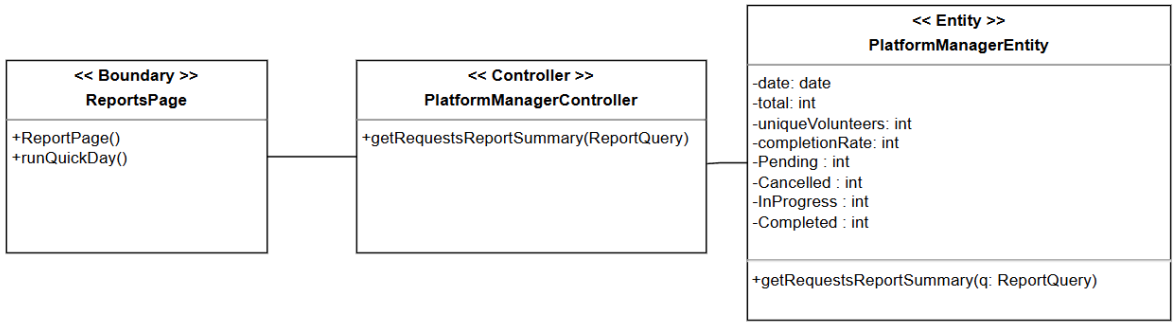
Use Case Diagram



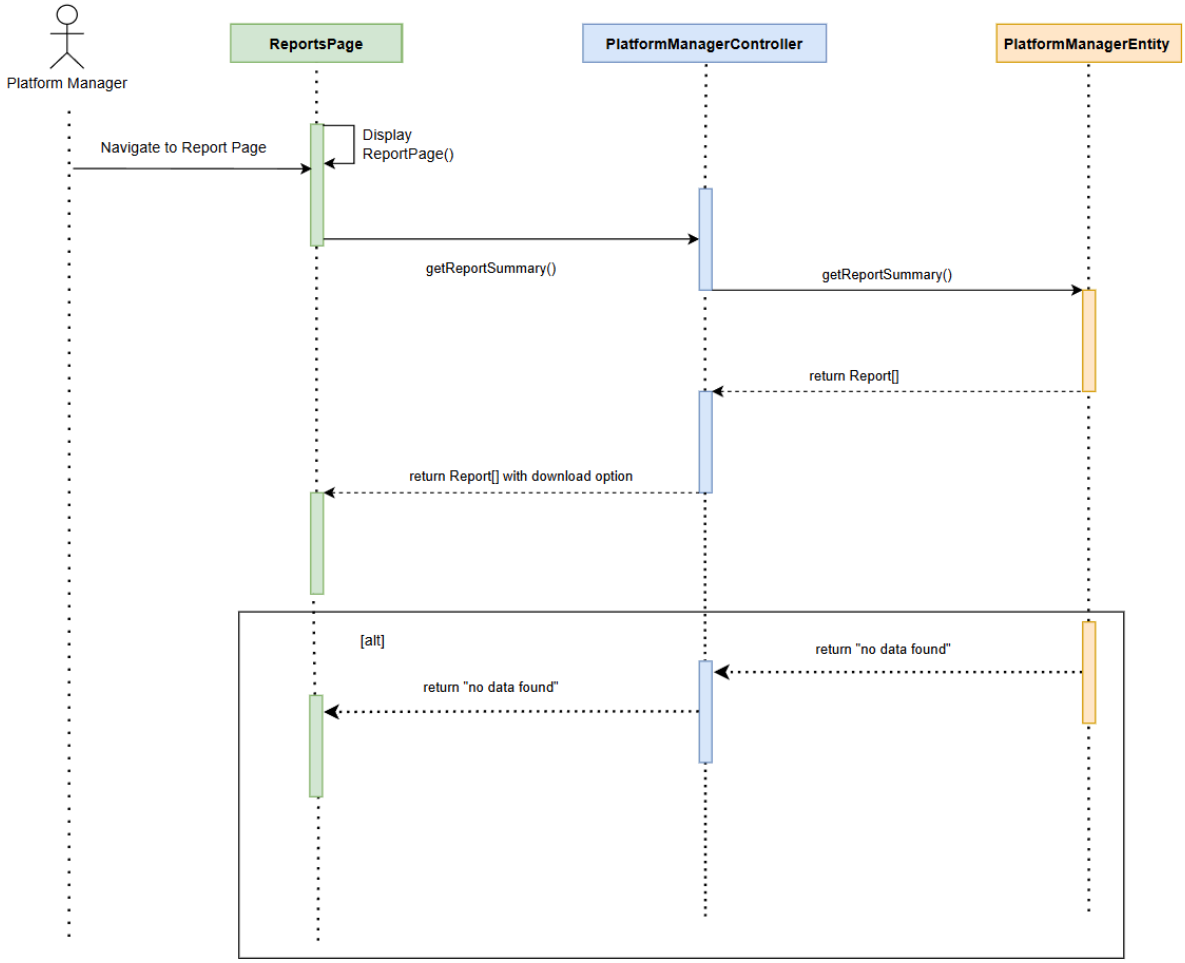
USER STORY (Taiga ID: #409)
As a Platform Manager, I want to generate daily reports so that I can view the daily statistics of the volunteer services requests.

Name: Platform Manager - Generate Daily report	**Taiga ID: #510**
Stakeholders and goals: To enable the Platform Manager to generate and view summarized daily statistics of volunteer service requests.	
Description: This use case allows the Platform Manager to generate a daily volunteer services report. The system compiles and displays a summary of service requests for the day. After reviewing the report summary, the Platform Manager can download the report for record keeping.	
Actors: Platform Manager	
Trigger: Platform Manager selects "Generate Daily Report" option from the system dashboard.	
Pre-condition: ❖ Volunteer activity are recorded in the system. ❖ The Platform Manager is logged in. ❖ The system has access to the activity database for the day.	
Main flow: 1. The Platform Manager navigates to the Reports section. 2. The Platform Manager selects "Generate Daily Report". 3. The system retrieves statistics for that date. 4. The system displays the summary. 5. The Platform Manager can choose to download the report. 6. Use case ends.	
Alternative/Exceptional flow: ❖ **A1**: No volunteer data available for the selected day 4a. The system shows a "no data found" message. 4b. Use case ends.	

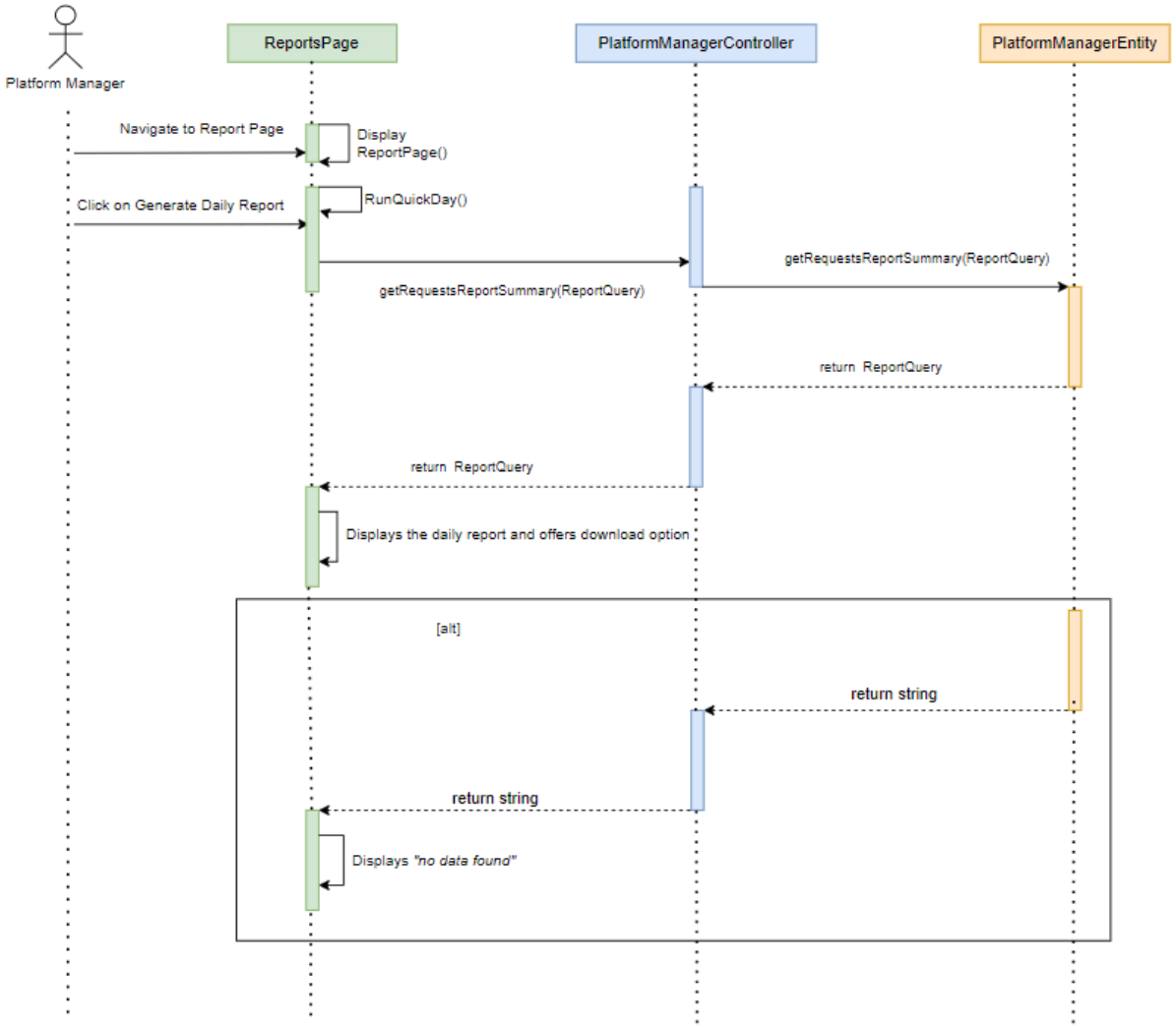
BCE (Platform Manager - Taiga ID: #514)



Sequence Diagram (First Iteration - Sprint 1)



Sequence Diagram (Platform Manager - Taiga ID: #513 - Final Iteration)



Code snippet

Boundary

```
export default function ReportsPage() {
  }
  function startOfToday() { const d = new Date(); d.setHours(0,0,0,0); return d; }
  function endOfToday() { const d = new Date(); d.setHours(23,59,59,999); return d; }
  function startOfWeek() {
    const d = new Date();
    const day = d.getDay(); // 0=Sun..6=Sat
    const diff = (day === 0 ? -6 : 1 - day); // Monday start
    d.setDate(d.getDate() + diff);
    d.setHours(0,0,0,0); return d;
  }
  function endOfWeek() { const d = startOfWeek(); d.setDate(d.getDate() + 6); d.setHours(23,59,59,999); return d; }
  function startOfMonth() { const d = new Date(); d.setDate(1); d.setHours(0,0,0,0); return d; }
  function endOfMonth() { const d = new Date(); d.setMonth(d.getMonth()+1, 0); d.setHours(23,59,59,999); return d; }
  // Removed unused startOfYear and endOfYear functions

  function runQuickRange(s: Date, e: Date) {
    const sStr = fmt(s);
    const eStr = fmt(e);
    syncDateRange(sStr, eStr);
    onGenerate(sStr, eStr, { ignoreType: true });
  }

  const runQuickDay = () => runQuickRange(startOfToday(), endOfToday());
  const runQuickWeek = () => runQuickRange(startOfWeek(), endOfWeek());
  const runQuickMonth = () => runQuickRange(startOfMonth(), endOfMonth());
```

Controller

```
export class PlatformManagerController {

  async getRequestsReportSummary(q: ReportQuery) {
    return this.entity.getRequestsReportSummary(q);
  }
```

Entity

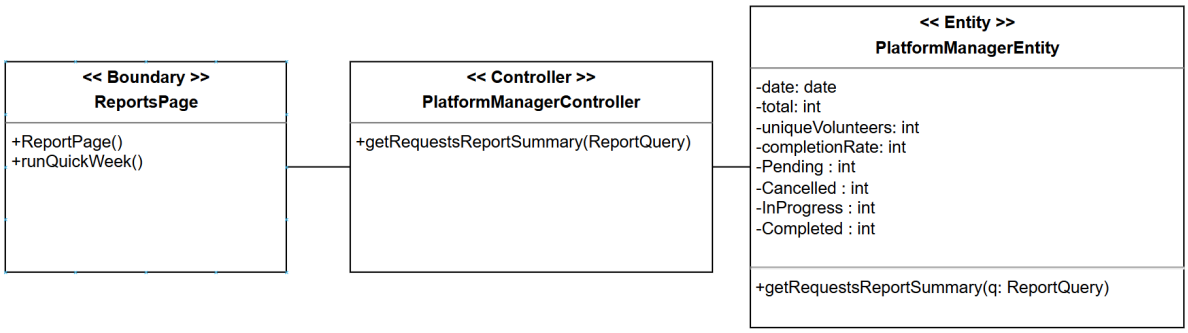
```
export class PlatformManagerEntity {
  async getRequestsReportSummary(q: ReportQuery) {
    const { start, end } = this.parseReportDateRange(q);
    const toTS = (d: Date) => {
      const y = d.getFullYear();
      const m = String(d.getMonth() + 1).padStart(2, '0');
      const day = String(d.getDate()).padStart(2, '0');
      const hh = String(d.getHours()).padStart(2, '0');
      const mm = String(d.getMinutes()).padStart(2, '0');
      const ss = String(d.getSeconds()).padStart(2, '0');
      const ms = String(d.getMilliseconds()).padStart(3, '0');
      return `${y}-${m}-${day} ${hh}:${mm}:${ss}.${ms}`;
    };
    const startStr = toTS(start);
    const endStr = toTS(end);

    let typeIds: number[] | null = null;
    if (q.typeNames && q.typeNames.length) {
      const rows = await this.db
        .select({ id: service_typeTable.id, name: service_typeTable.name })
        .from(service_typeTable)
        .where(inArray(service_typeTable.name, q.typeNames));
      typeIds = rows.map(r => r.id);
      if (!typeIds.length) return this.buildEmptyRequestsReportSummary();
    }
```

Wireframe (Platform Manager - Taiga ID: #515):

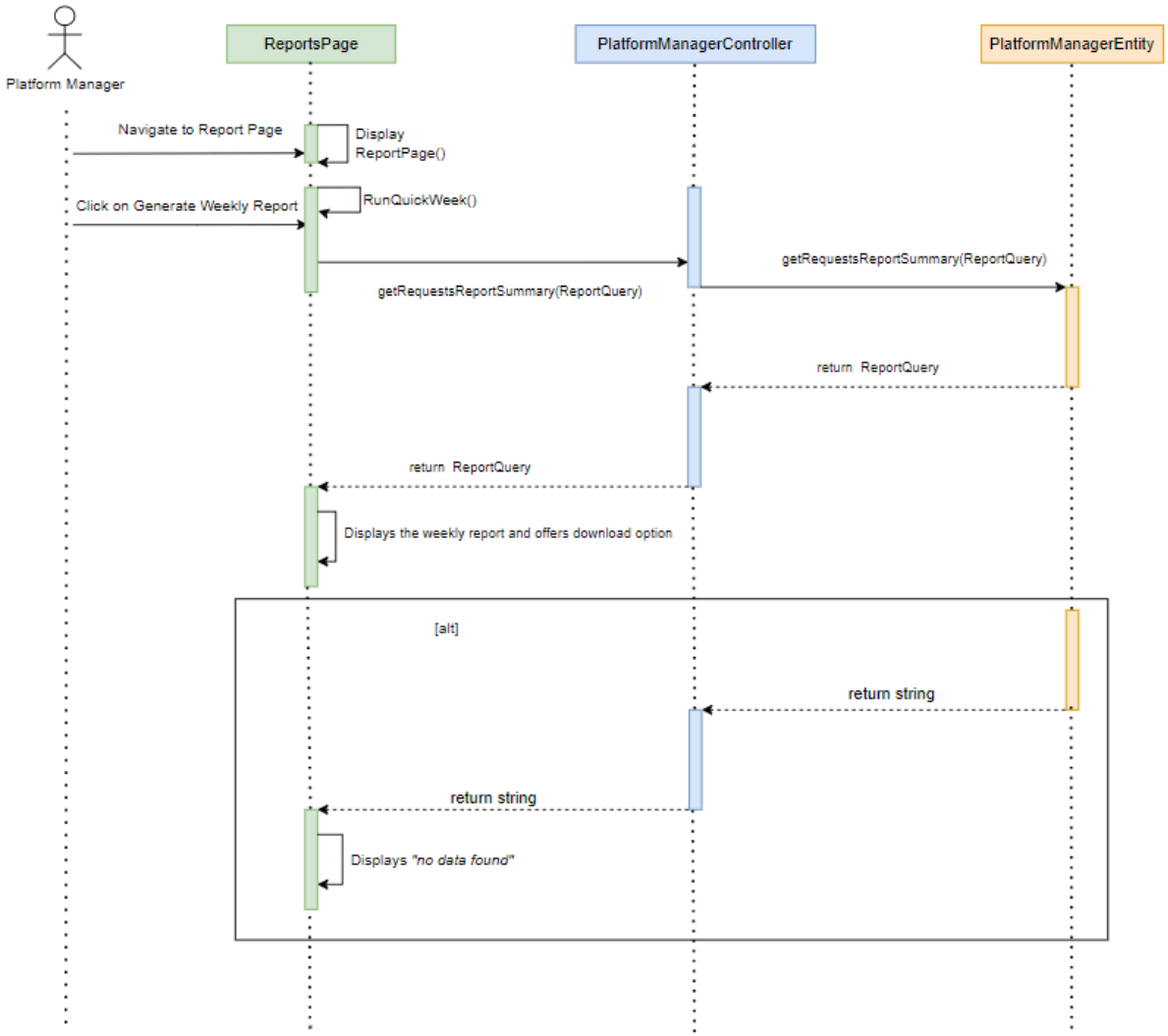
USER STORY (Taiga ID: #410)
As a Platform Manager, I want to generate weekly reports so that I can view the weekly statistics of the volunteer services requests.

Name: Platform Manager - Generate Weekly Report	**Taiga ID: #517**
Stakeholders and goals: To enable the Platform Manager to generate and view summarized current week's statistics of volunteer service requests.	
Description: This use case allows the Platform Manager to generate current week's volunteer services report. The system compiles and displays a summary of service requests for the week. After reviewing the report summary, the Platform Manager can download the report for record keeping.	
Actors: Platform Manager	
Trigger: Platform Manager selects "Generate Weekly Report" option from the system dashboard.	
Pre-condition: ❖ Volunteer activities are recorded in the system. ❖ The Platform Manager is logged in. ❖ The system has access to the activity database for the specified week.	
Main flow: 1. The Platform Manager navigates to the Reports section. 2. The Platform Manager selects "Generate Weekly Report". 3. The system retrieves volunteer activity for that week. 4. The system calculates aggregated statistics. 5. The system displays weekly summaries. 6. The Platform Manager can choose to download the report. 7. Use case ends.	
Alternative/Exceptional flow: **A1: No volunteer data available for the selected week** 4a. System shows a "no data found" message. 4b. Use case ends	

BCE (Platform Manager - Taiga ID: #521)



Sequence Diagram (Platform Manager - Taiga ID: #520)



Code snippet

Boundary

```
export default function ReportsPage() {
  function fmt(d: Date) {
    const day = String(d.getDate()).padStart(2, '0');
    return `${y}-${m}-${day}`;
  }
  function startOfToday() { const d = new Date(); d.setHours(0,0,0,0); return d; }
  function endOfToday() { const d = new Date(); d.setHours(23,59,59,999); return d; }
  function startOfWeek() {
    const d = new Date();
    const day = d.getDay(); // 0=Sun..6=Sat
    const diff = (day === 0 ? -6 : 1 - day); // Monday start
    d.setDate(d.getDate() + diff);
    d.setHours(0,0,0,0); return d;
  }
  function endOfWeek() { const d = startOfWeek(); d.setDate(d.getDate() + 6); d.setHours(23,59,59,999); return d; }
  function startOfMonth() { const d = new Date(); d.setDate(1); d.setHours(0,0,0,0); return d; }
  function endOfMonth() { const d = new Date(); d.setMonth(d.getMonth()+1, 0); d.setHours(23,59,59,999); return d; }
  // Removed unused startOfYear and endOfYear functions

  function runQuickRange(s: Date, e: Date) {
    const sStr = fmt(s);
    const eStr = fmt(e);
    syncDateRange(sStr, eStr);
    onGenerate(sStr, eStr, { ignoreType: true });
  }

  const runQuickDay = () => runQuickRange(startOfToday(), endOfToday());
  const runQuickWeek = () => runQuickRange(startOfWeek(), endOfWeek());
  const runQuickMonth = () => runQuickRange(startOfMonth(), endOfMonth());
```

Controller

```
export class PlatformManagerController {

  async getRequestsReportSummary(q: ReportQuery) {
    return this.entity.getRequestsReportSummary(q);
  }
```

Entity

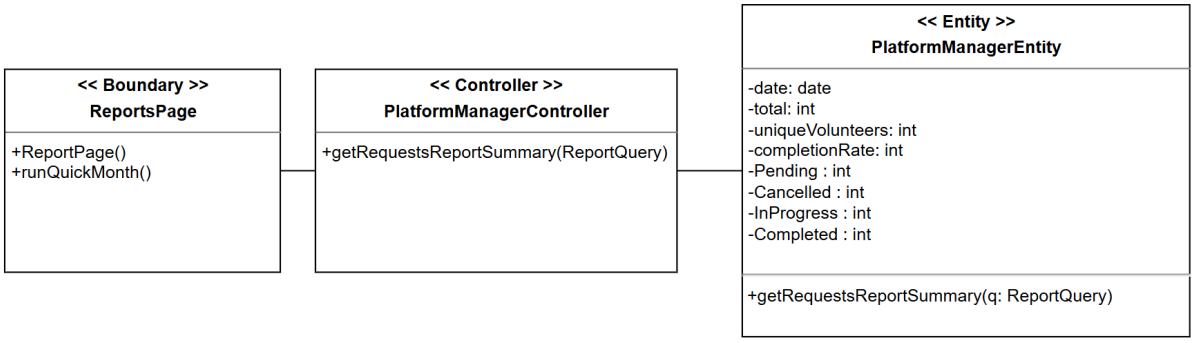
```
export class PlatformManagerEntity {
  async getRequestsReportSummary(q: ReportQuery) {
    const { start, end } = this.parseReportDateRange(q);
    const toTS = (d: Date) => {
      const y = d.getFullYear();
      const m = String(d.getMonth() + 1).padStart(2, '0');
      const day = String(d.getDate()).padStart(2, '0');
      const hh = String(d.getHours()).padStart(2, '0');
      const mm = String(d.getMinutes()).padStart(2, '0');
      const ss = String(d.getSeconds()).padStart(2, '0');
      const ms = String(d.getMilliseconds()).padStart(3, '0');
      return `${y}-${m}-${day} ${hh}:${mm}:${ss}.${ms}`;
    };
    const startStr = toTS(start);
    const endStr = toTS(end);

    let typeIds: number[] | null = null;
    if (q.typeNames && q.typeNames.length) {
      const rows = await this.db
        .select({ id: service_typeTable.id, name: service_typeTable.name })
        .from(service_typeTable)
        .where(inArray(service_typeTable.name, q.typeNames));
      typeIds = rows.map(r => r.id);
      if (!typeIds.length) return this.buildEmptyRequestsReportSummary();
    }
```

Wireframe (Platform Manager - Taiga ID: #522):

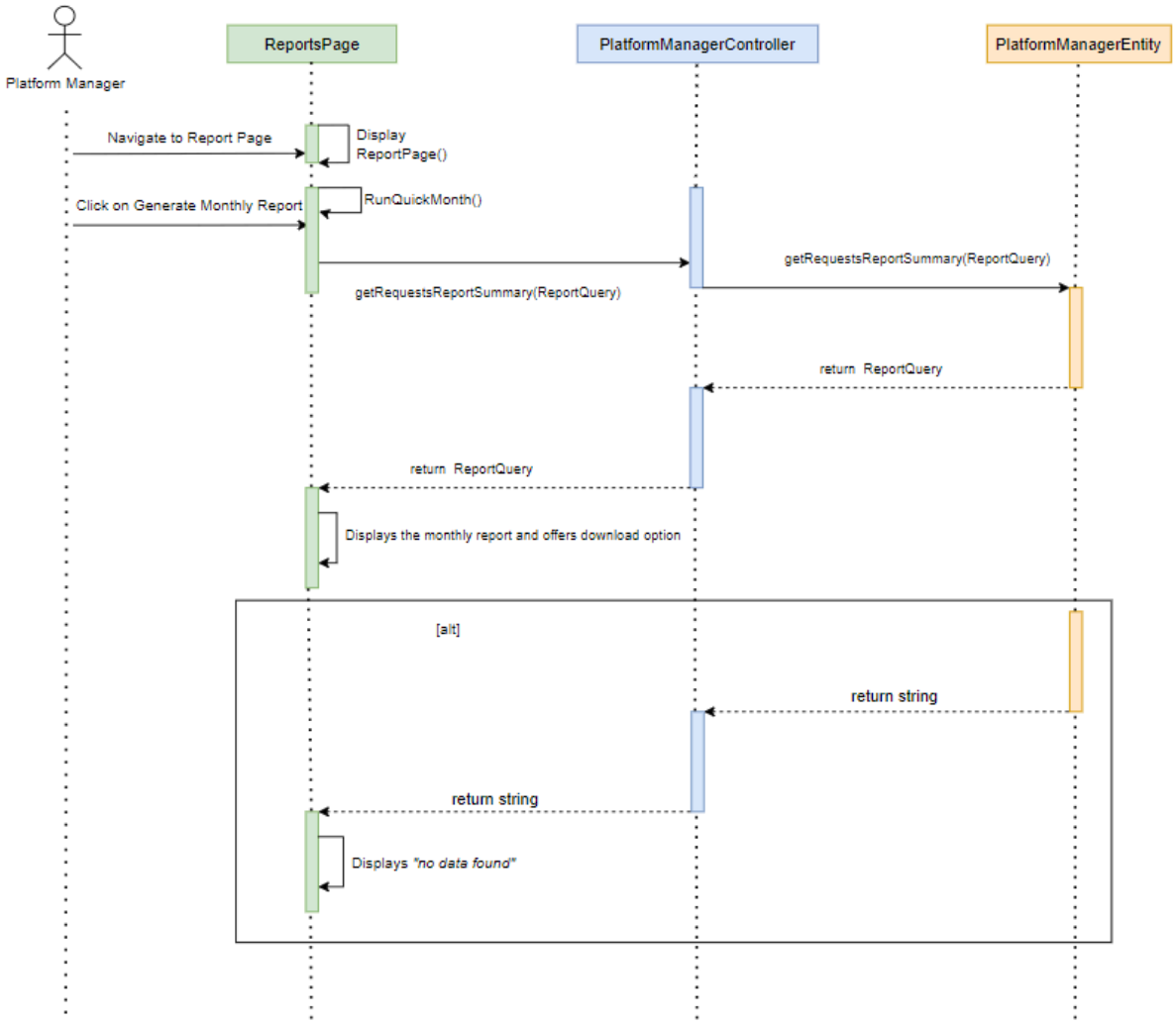
USER STORY (Taiga ID: #411)
As a Platform Manager, I want to generate monthly reports so that I can view the monthly statistics of the volunteer services requests.

Name: Platform Manager - Generate Monthly Report	**Taiga ID: #524**
Stakeholders and goals: To enable the Platform Manager to generate and view summarized current month's statistics of volunteer service requests.	
Description: This use case allows the Platform Manager to generate volunteer services report for the current month. The system compiles and displays a summary of service requests for the selected month. After reviewing the report summary, the Platform Manager can download the report for record keeping.	
Actors: Platform Manager	
Trigger: Platform Manager selects the "Generate Monthly Report" option from the system dashboard.	
Pre-condition: ❖ Volunteer activity are recorded in the system. ❖ The Platform Manager is logged in. ❖ The system has access to the activity database for the specified month.	
Main flow: 1. The Platform Manager navigates to the Reports section. 2. The Platform Manager selects "Generate Monthly Report." 3. The system retrieves volunteer activity for that month. 4. The system calculates aggregated statistics. 5. The system displays a monthly summary. 6. The Platform Manager can choose to download the report. 7. Use case ends.	
Alternative/Exceptional flow: **A1: No volunteer data available for the selected month.** 4a. System shows a "no data found" message. 4b. Use case ends.	

BCE (Platform Manager - Taiga ID: #528)



Sequence Diagram (Platform Manager - Taiga ID: #527)



Code snippet

Boundary

```
export default function ReportsPage() {
  function fmt(d: Date) {
  }
  function startOfToday() { const d = new Date(); d.setHours(0,0,0,0); return d; }
  function endOfToday() { const d = new Date(); d.setHours(23,59,59,999); return d; }
  function startOfWeek() {
    const d = new Date();
    const day = d.getDay(); // 0=Sun..6=Sat
    const diff = (day === 0 ? -6 : 1 - day); // Monday start
    d.setDate(d.getDate() + diff);
    d.setHours(0,0,0,0); return d;
  }
  function endOfWeek() { const d = startOfWeek(); d.setDate(d.getDate() + 6); d.setHours(23,59,59,999); return d; }
  function startOfMonth() { const d = new Date(); d.setDate(1); d.setHours(0,0,0,0); return d; }
  function endOfMonth() { const d = new Date(); d.setMonth(d.getMonth()+1, 0); d.setHours(23,59,59,999); return d; }
  // Removed unused startOfYear and endOfYear functions

  function runQuickRange(s: Date, e: Date) {
    const sStr = fmt(s);
    const eStr = fmt(e);
    syncDateRange(sStr, eStr);
    onGenerate(sStr, eStr, { ignoreType: true });
  }

  const runQuickDay = () => runQuickRange(startOfToday(), endOfToday());
  const runQuickWeek = () => runQuickRange(startOfWeek(), endOfWeek());
  const runQuickMonth = () => runQuickRange(startOfMonth(), endOfMonth());
```

Controller

```
export class PlatformManagerController {

  async getRequestsReportSummary(q: ReportQuery) {
    return this.entity.getRequestsReportSummary(q);
  }
```

Entity

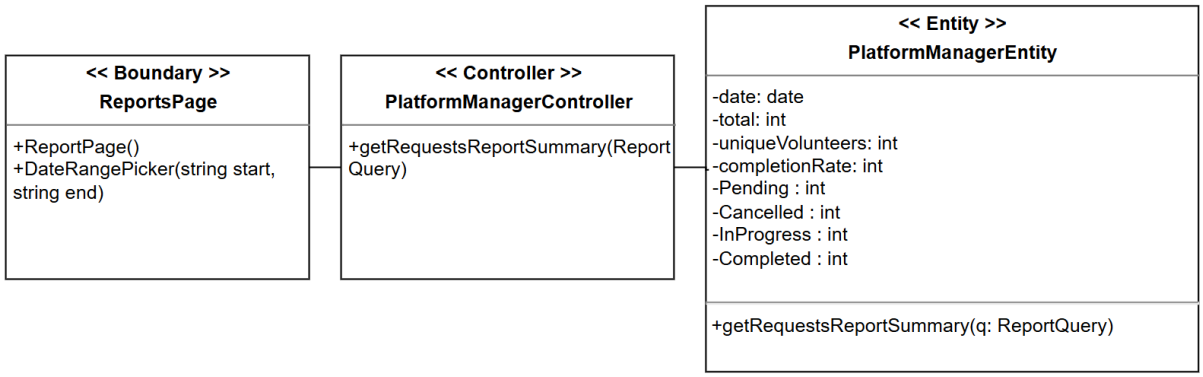
```
export class PlatformManagerEntity {
  async getRequestsReportSummary(q: ReportQuery) {
    const { start, end } = this.parseReportDateRange(q);
    const toTS = (d: Date) => {
      const y = d.getFullYear();
      const m = String(d.getMonth() + 1).padStart(2, '0');
      const day = String(d.getDate()).padStart(2, '0');
      const hh = String(d.getHours()).padStart(2, '0');
      const mm = String(d.getMinutes()).padStart(2, '0');
      const ss = String(d.getSeconds()).padStart(2, '0');
      const ms = String(d.getMilliseconds()).padStart(3, '0');
      return `${y}-${m}-${day} ${hh}:${mm}:${ss}.${ms}`;
    };
    const startStr = toTS(start);
    const endStr = toTS(end);

    let typeIds: number[] | null = null;
    if (q.typeNames && q.typeNames.length) {
      const rows = await this.db
        .select({ id: service_typeTable.id, name: service_typeTable.name })
        .from(service_typeTable)
        .where(inArray(service_typeTable.name, q.typeNames));
      typeIds = rows.map(r => r.id);
      if (!typeIds.length) return this.buildEmptyRequestsReportSummary();
    }
```

Wireframe (Platform Manager - Taiga ID: #529):

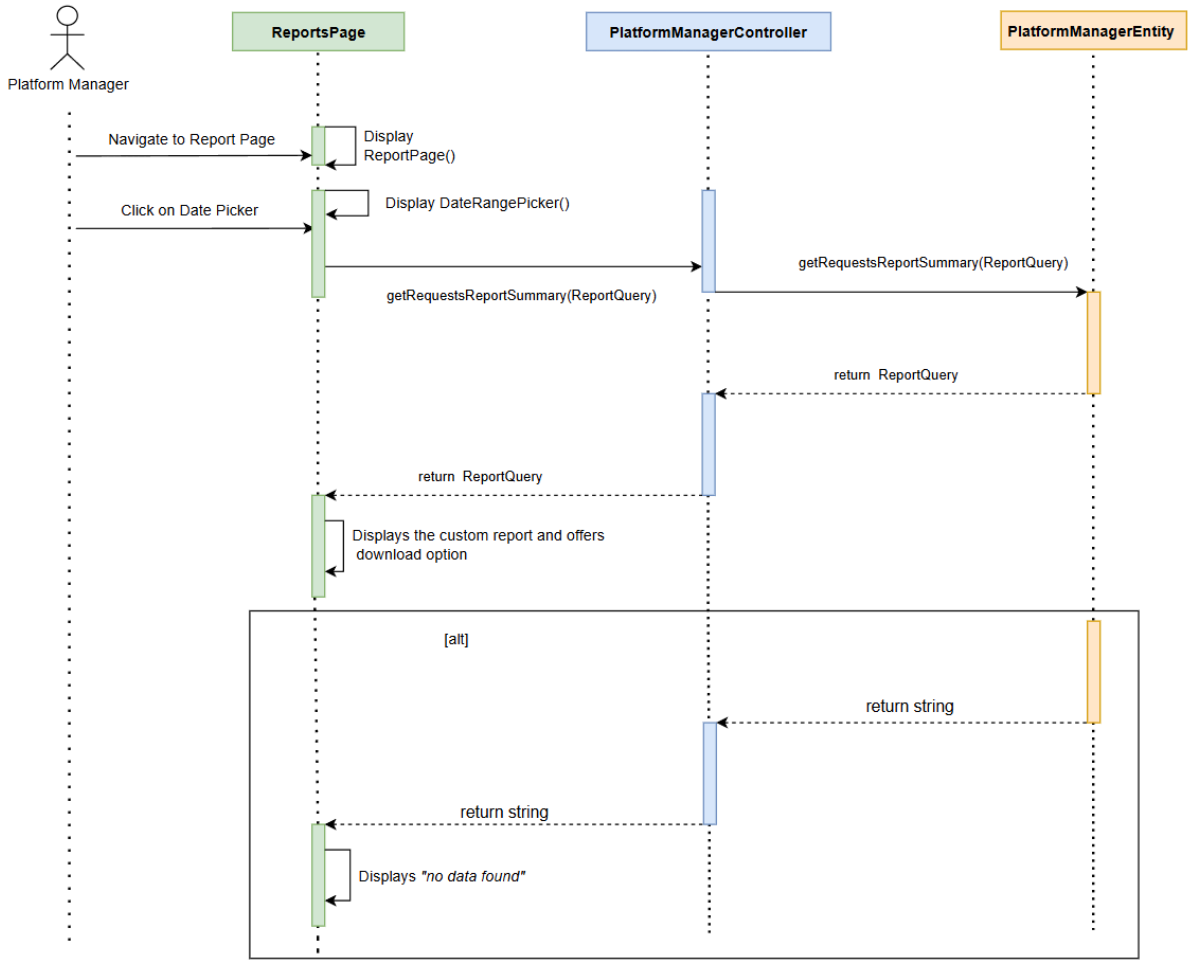
USER STORY (Taiga ID: #412)
As a Platform Manager, I want to generate custom reports based on date range or service types so that I can view custom statistics of the volunteer services requests.

Name: Platform Manager - Generate Custom Report	**Taiga ID: #566**
Stakeholders and goals: To enable the Platform Manager to generate and view summarized statistics of volunteer service requests filtered by specific date ranges and/or service types.	
Description: This use case allows the Platform Manager to generate a custom volunteer services report. The system compiles and displays a summary of service requests based on selected criteria such as custom date ranges and/or service types. After reviewing the report summary, the Platform Manager can download the report for record keeping and performance analysis.	
Actors: Platform Manager	
Trigger: Platform Manager selects "Generate Custom Report" option from the system dashboard.	
Pre-condition: ❖ Volunteer activity are recorded in the system. ❖ The Platform Manager is logged in. ❖ The system has access to the activity database for the specified criteria.	
Main flow: 1. The Platform Manager navigates to the Reports section. 2. The Platform Manager selects "Generate Custom Report." 3. The Platform Manager chooses a date range and/or specific service types. 4. The system retrieves volunteer activity based on the selected filters. 5. The system calculates aggregated statistics. 6. The system displays a custom summary report. 7. The Platform Manager can choose to download the report. 8. Use case ends.	
Alternative/Exceptional flow: **A1: No volunteer data matches the selected criteria.** 4a. The system shows a "No data found" message. 4b. Use case ends.	

BCE (Platform Manager - Taiga ID: #569)



Sequence Diagram (Platform Manager - Taiga ID: #570):



Code snippet

Boundary

```
function DateRangePicker({ start, end, onChange }: { start: string; end: string; onChange: (s: string, e: string) => void }) {
  const [open, setOpen] = useState(false);
  const [viewYear, setViewYear] = useState<number>(() => (start ? new Date(start + 'T00:00:00').getFullYear() : new Date().getFullYear()));
  const [viewMonth, setViewMonth] = useState<number>(() => (start ? new Date(start + 'T00:00:00').getMonth() : new Date().getMonth()));
  const [selStart, setSelStart] = useState<Date | null>(() => (start ? new Date(start + 'T00:00:00') : null));
  const [selEnd, setSelEnd] = useState<Date | null>(() => (end ? new Date(end + 'T00:00:00') : null));
  const [hover, setHover] = useState<Date | null>(null);

  function toISO(d: Date) {
    const y = d.getFullYear();
    const m = String(d.getMonth() + 1).padStart(2, '0');
    const da = String(d.getDate()).padStart(2, '0');
    return `${y}-${m}-${da}`;
  }

  function fmtLabel() {
    return start && end ? `${start} - ${end}` : 'Select range';
  }

  function daysInMonth(y: number, m: number) {
    return new Date(y, m + 1, 0).getDate();
  }

  function startOfMonth(y: number, m: number) {
    return new Date(y, m, 1);
  }

  function sameDay(a: Date, b: Date) {
    return a.getFullYear() === b.getFullYear() && a.getMonth() === b.getMonth() && a.getDate() === b.getDate();
  }

  function isBetween(d: Date, a: Date, b: Date) {
    const t = d.getTime();
    const t0 = a.getTime();
    const t1 = b.getTime();
    return t >= Math.min(t0, t1) && t <= Math.max(t0, t1);
  }

  function onDayClick(d: Date) {
    if (!selStart || (selStart && selEnd)) {
      setSelStart(d);
      setSelEnd(null);
    } else {
      if (d.getTime() < selStart.getTime()) {
        // reset start to earlier date
        setSelStart(d);
        setSelEnd(null);
      } else if (d.getTime() === selStart.getTime()) {
        // single day range
        setSelEnd(d);
      } else {
        setSelEnd(d);
      }
    }
  }
```

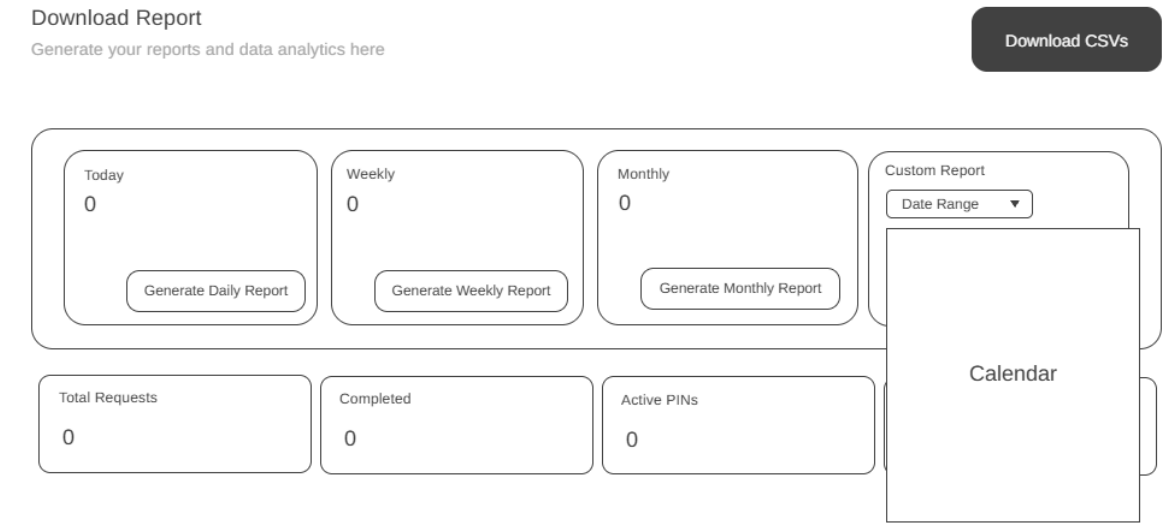
Controller

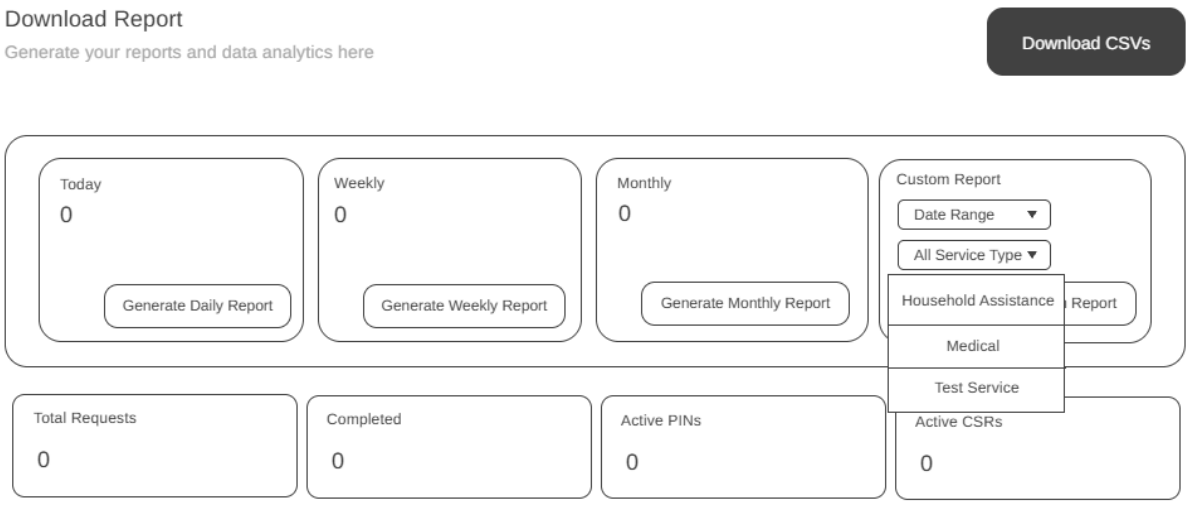
```
export class PlatformManagerController {

  async getRequestsReportSummary(q: ReportQuery) {
    return this.entity.getRequestsReportSummary(q);
  }
```

Entity

```
export class PlatformManagerEntity {
  async getRequestsReportSummary(q: ReportQuery) {
    const { start, end } = this.parseReportDateRange(q);
    const toTS = (d: Date) => {
      const y = d.getFullYear();
      const m = String(d.getMonth() + 1).padStart(2, '0');
      const day = String(d.getDate()).padStart(2, '0');
      const hh = String(d.getHours()).padStart(2, '0');
      const mm = String(d.getMinutes()).padStart(2, '0');
      const ss = String(d.getSeconds()).padStart(2, '0');
      const ms = String(d.getMilliseconds()).padStart(3, '0');
      return `${y}-${m}-${day} ${hh}:${mm}:${ss}.${ms}`;
    };
    const startStr = toTS(start);
    const endStr = toTS(end);

    let typeIds: number[] | null = null;
    if (q.typeNames && q.typeNames.length) {
      const rows = await this.db
        .select({ id: service_typeTable.id, name: service_typeTable.name })
        .from(service_typeTable)
        .where(inArray(service_typeTable.name, q.typeNames));
      typeIds = rows.map(r => r.id);
      if (!typeIds.length) return this.buildEmptyRequestsReportSummary();
    }
```

Wireframe (Platform Manager - Taiga ID: #571):

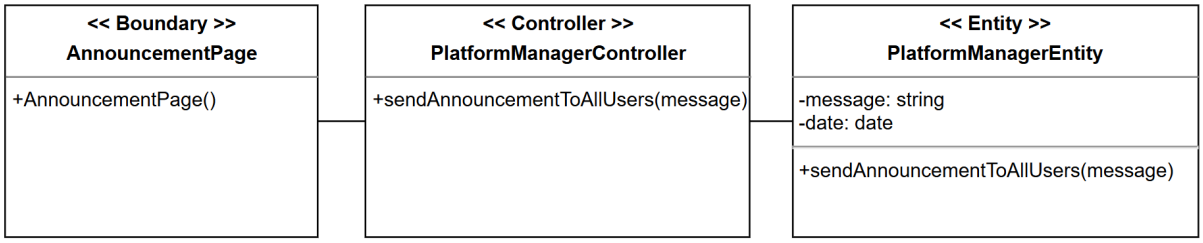




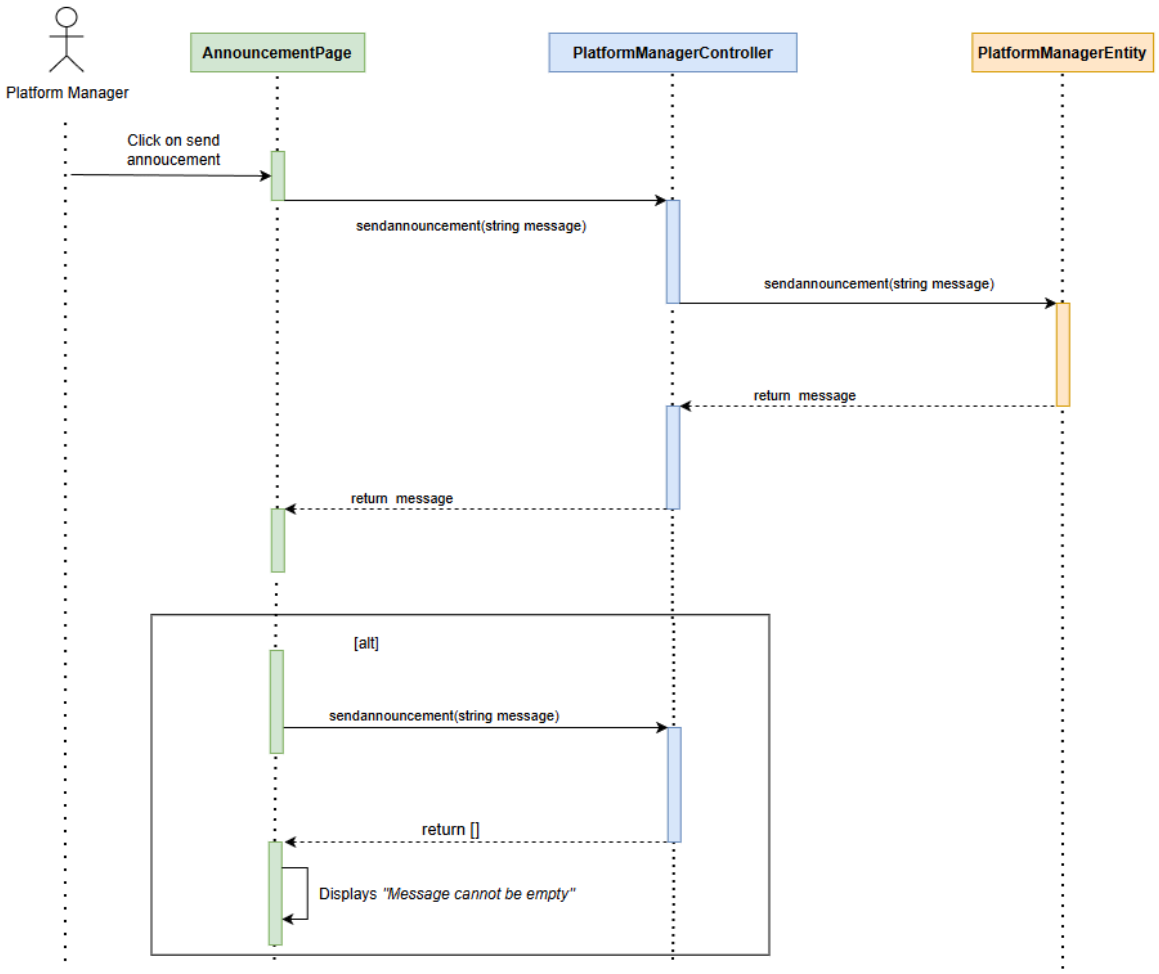
USER STORY (Taiga ID: #413)
As a Platform Manager, I want to send announcements to all users so that I can communicate important updates (e.g., maintenance, events).

Name: Send announcement to users	**Taiga ID: #622**
Stakeholders and goals: - Platform Manager: communicate important information to all users quickly and reliably - All other users: Get notified of important information	
Description: Platform Manager composes an announcement and sends it to all users. The system would retrieve all recipients and deliver the announcement. The system will show a message to confirm successful delivery.	
Actors: Platform Manager	
Trigger: Platform Manager clicks on the “Send Announcement” button.	
Pre-condition: ❖ Platform Manager is logged in.	
Main flow: 1. The Platform Manager navigates to "Announcement" page. 2. The Platform Manager composes messages in the text box field. 3. The Platform Manager selects the “Send Announcement” button. 4. System retrieves the list of active users. 5. System will deliver the message to all users. 6. The system displays a message to confirm successful delivery.	
Alternative/Exceptional flow: **A1. Empty message** 4a. The system displays an error message “Message cannot be empty.”	

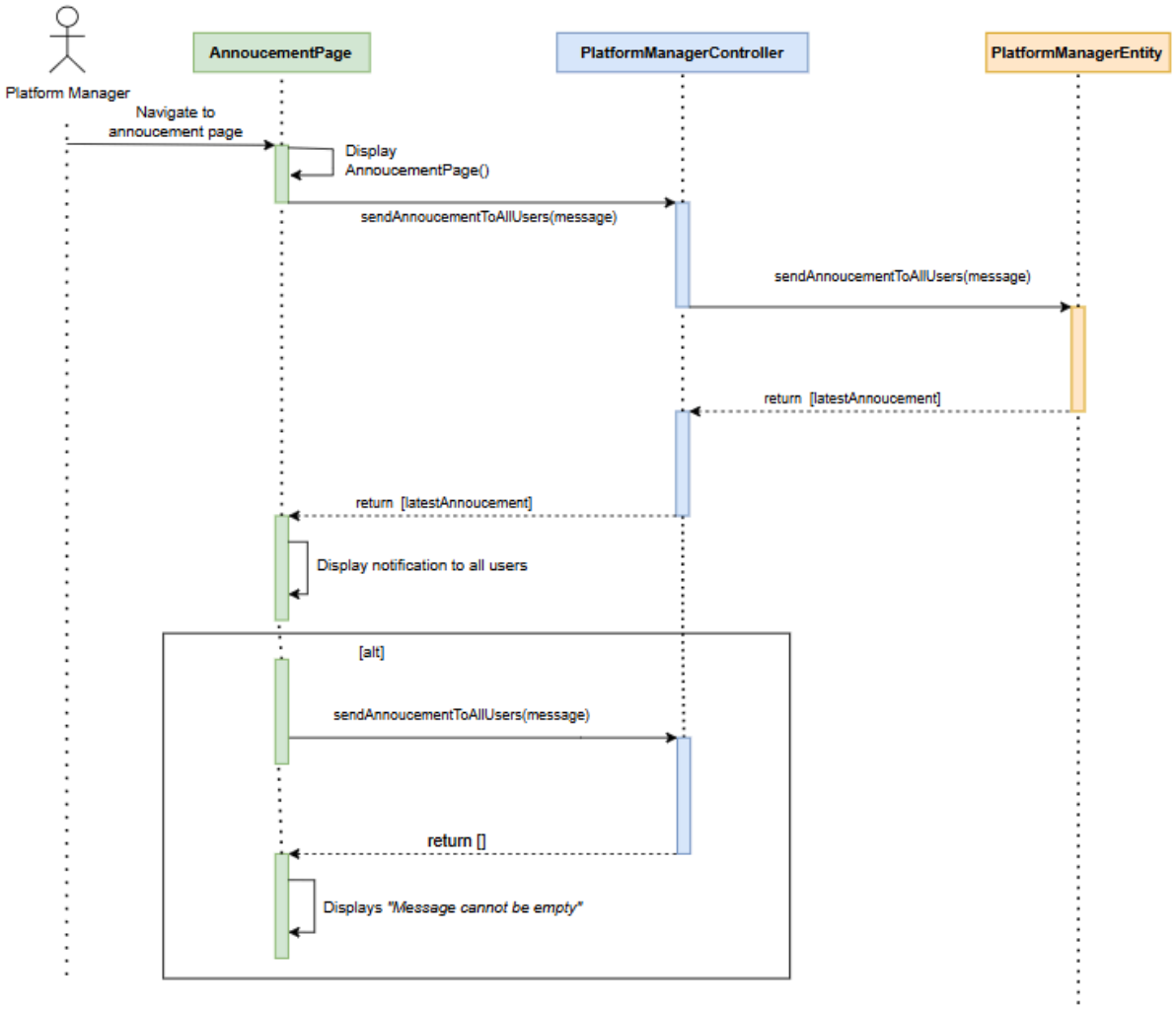
BCE (Platform Manager - Taiga ID: #625)



Sequence Diagram (First Iteration - Sprint 1)



Sequence Diagram (Platform Manager - Taiga ID: #626 - <u>Final Iteration</u>)



Code snippet

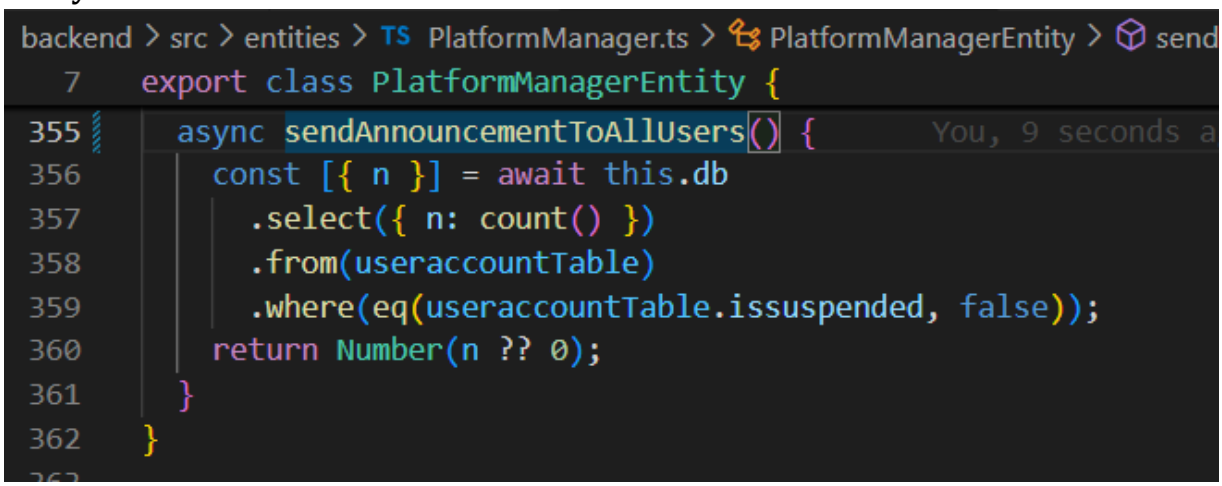
Boundary

```
export default function AnnouncementsPage() {
  const onSend = async () => {
    setInfo(null); setError(null);
    if (!message.trim()) { setError("Message cannot be empty"); return; }
    setBusy(true);
    try {
      const res = await fetch(`/api/pm/announcements/send`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ message })
      });
      const text = await res.text();
      let data: unknown = null;
      try { data = text ? JSON.parse(text) : null; } catch { /* non-JSON (e.g., HTML) */ }
      let errorMsg: string | undefined = undefined;
      let deliveredCount: number | undefined = undefined;
      if (isAnnouncementResponse(data)) {
        errorMsg = data.error;
        deliveredCount = data.deliveredCount;
      }
      if (!res.ok) throw new Error(errorMsg || `Failed to send (${res.status})`);
      const successMsg = `Sent to ${deliveredCount ?? '?'} users`;
      setInfo(successMsg);
      toast.success(successMsg);
      setMessage("");
      loadLatest();
    } catch (e) {
      const errMsg = e instanceof Error ? e.message : "Error occurred";
      setError(errMsg);
      toast.error(errMsg);
    } finally {
      setBusy(false);
```

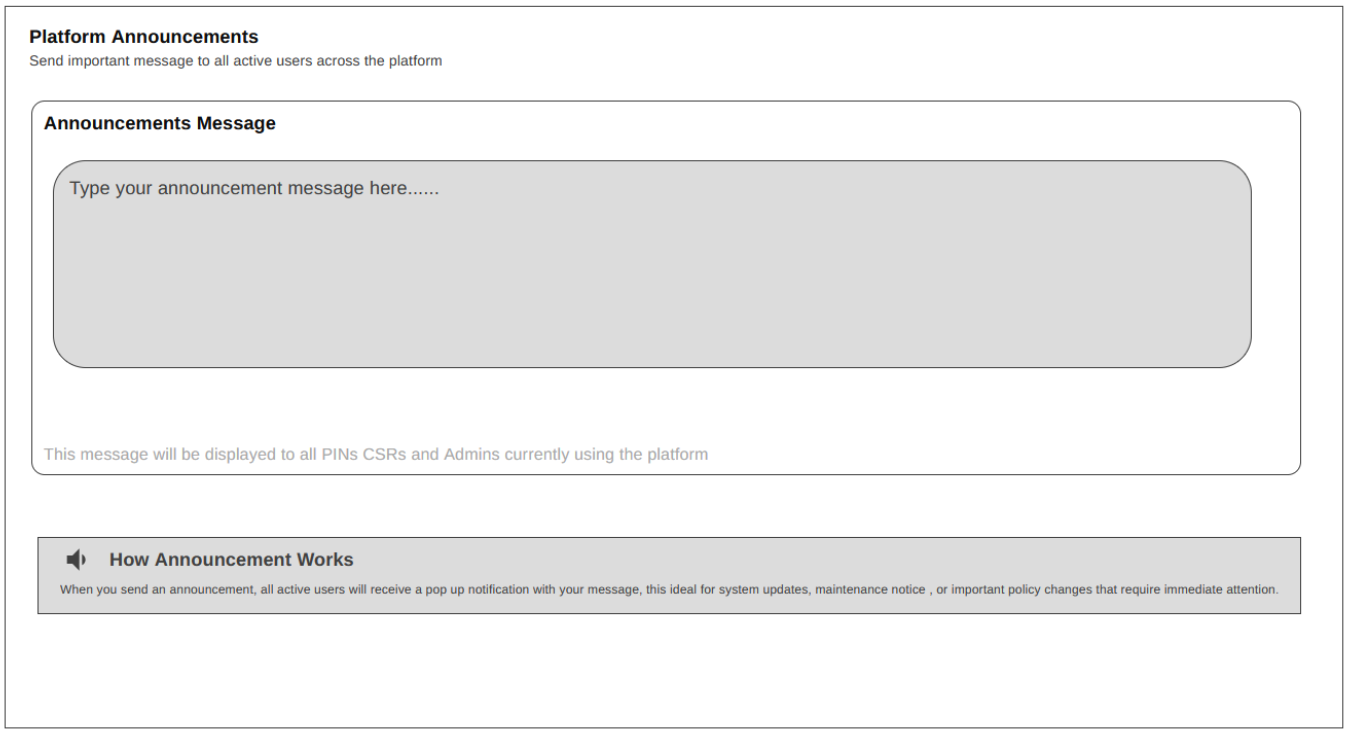
Controller

```
kend > src > controller > TS PlatformManagerControllers.ts > PlatformManagerController > getLatestAnnouncementSnap
6   export class PlatformManagerController {

1     async sendAnnouncementToAllUsers({ message }: { message?: string }) {
2       if (!message || !message.trim()) throw new Error("Message cannot be empty");
3       const deliveredCount = await this.entity.sendAnnouncementToAllUsers();
4       this.latestAnnouncement = { message: message.trim(), createdAt: new Date().toISOString()
5       return { deliveredCount };
6     }
```

Entity


backend > src > entities > TS PlatformManager.ts > PlatformManagerEntity > send
7 export class PlatformManagerEntity {
355 async sendAnnouncementToAllUsers() { You, 9 seconds a
356 const [{ n }] = await this.db
357 .select({ n: count() })
358 .from(useraccountTable)
359 .where(eq(useraccountTable.issuspended, false));
360 return Number(n ?? 0);
361 }
362 }

Wireframe (Platform Manager - Taiga ID: #627):



4. Sprint 1

4.1. Sprint 1 - 1st Meeting (06/10/2025)

Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

Discussion summaries	
Meeting Topics	**Details**
Role allocation	Project Manager/ Backend
Allocation of Tasks	- Justin : Project Manager/ Backend - Eugene : Backend / Database - Karissa : UI/UX Designer/ Documentation / Tester - Syafiqah : Frontend / UI/UX Designer - Zhan Qi : Documentation - Jiya : Backend / Documentation - Dillon : Backend/ Documentation
Initial Brainstorm of user stories	- Justin/Eugene : User Admin - Karissa/ Syafiqah : Person-In-Need (PIN) - Zhan Qi/ Jiya : CSR Representative - Dillon : Platform Management

Action Plans	
Tasks	**Delegation**
Each subgroup to refine user stories for their assigned category	As stated above
Prepare user story documentation for review	As stated above
Update Taiga with role allocation	Zhan Qi

Meeting Follow-ups	
Meeting Place	**Next Meeting Date**
Microsoft Teams	08/10/25

Meeting Follow-ups
Meeting Topics
1. Review of draft User Stories 2. Discussion methods of Backend development 3. Database creation 4. UI/UX Wireframe 5. Discussions on Frontend development

4.2. Sprint 1 - 2nd Meeting (08/10/2025)

Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

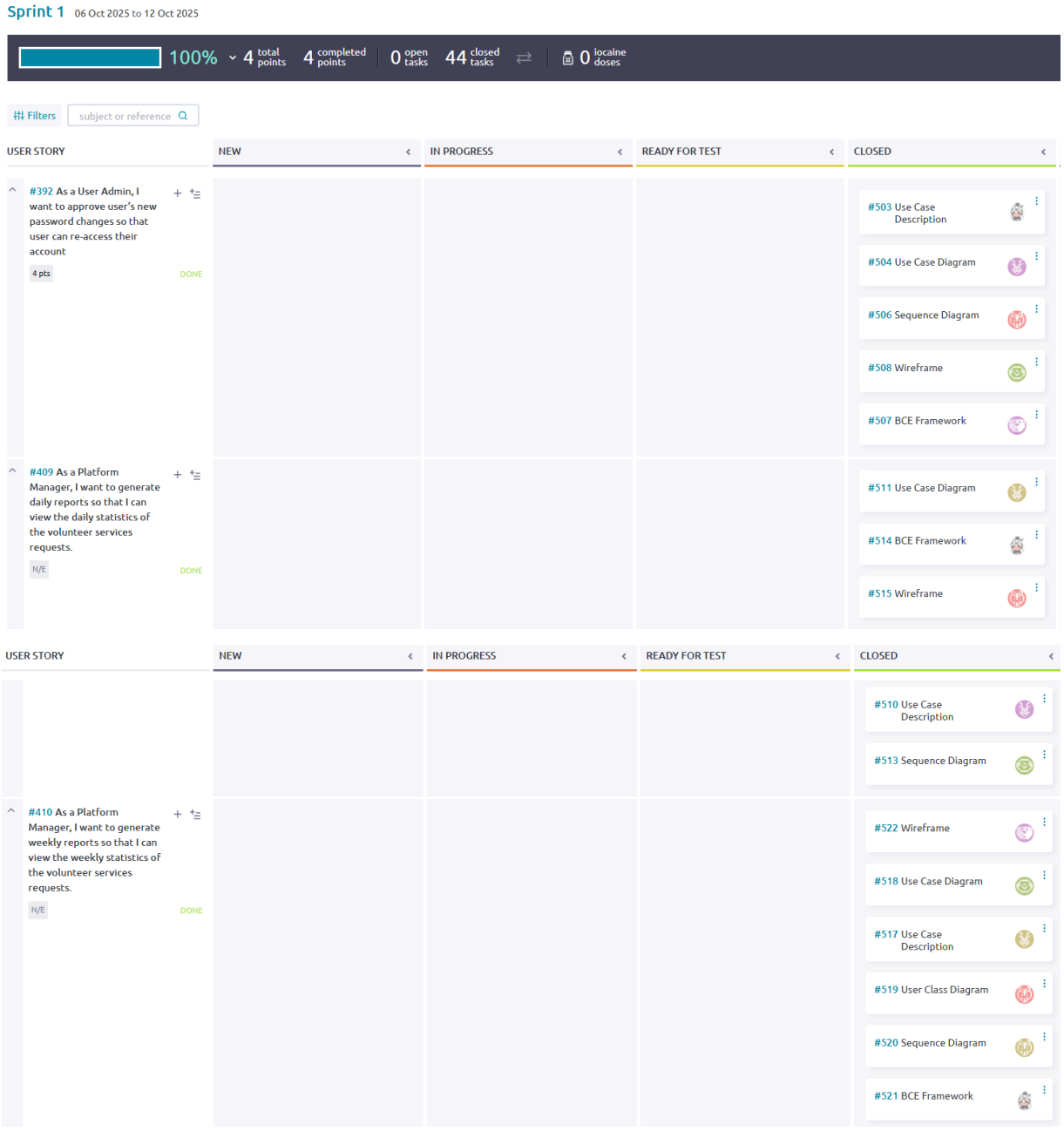
Discussion summaries	
Meeting Topics	**Details**
Review of draft User Stories	Reviewed all user stories to ensure each met acceptance criteria and followed standard format. Minor edits were made for clarity and scope.
Development of Use Case Description	Began writing detailed Use Case Descriptions for User Admin and PIN modules using Modelio. Ensured each use case linked to a specific story.
Development of Class Diagram	Outlined initial Entity classes based on user story data requirements.
Development of Sequence Diagram	Started Sequence Diagrams to illustrate actor–system interactions for login and data retrieval.
Discussion methods of Backend development	Agreed to use Node.js with Express and PostgreSQL via Drizzle ORM for maintainability and CI/CD compatibility.
Database creation	Draft of table schema for user accounts and CSR requests. Normalization review scheduled for next meeting.
UI/UX Wireframe	Karissa and Syafiqah commenced wireframe layout in Figma; defined color palette and component hierarchy.

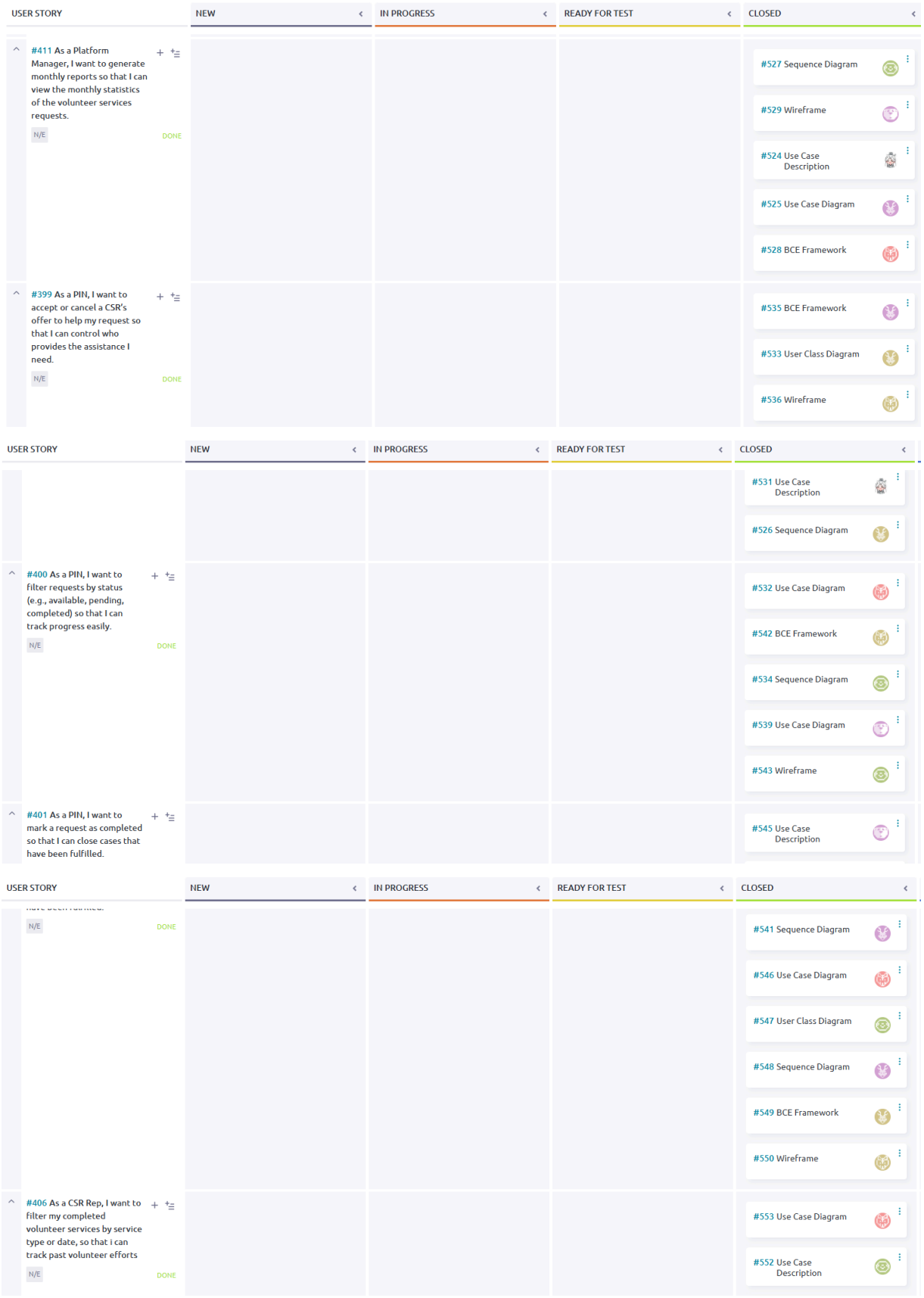
Discussion summaries	
Discussions on Frontend development	Confirmed React as framework with Bootstrap for rapid prototyping. Frontend to consume API routes once backend is stable.

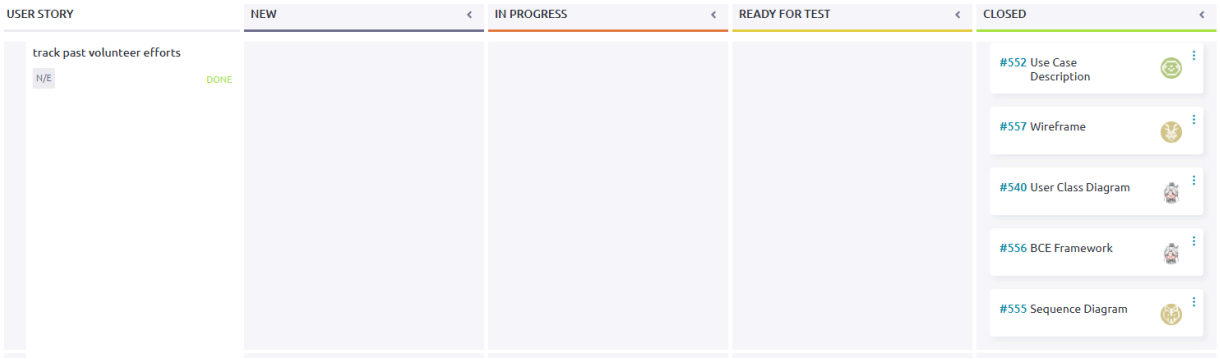
Action Plans	
Tasks	**Delegation**
Respective subgroup to complete UML Diagrams (Use Case, Class, Sequence)	As stated above
Finalise database schema	Eugene
Continue wireframe refinements	Karissa / Syafiqah
Consolidate documentation	All members

Meeting Follow-ups	
Meeting Place	**Next Meeting Date**
Block B Common Study Area	13/10/25
Meeting Topics	
1. Finalize system design artifacts 2. Begin backend API structure 3. Frontend layout linking 4. Prepare for Sprint 2 implementation	

4.3 Sprint 1 - Taiga




USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#411 As a Platform Manager, I want to generate monthly reports so that I can view the monthly statistics of the volunteer services requests.
N/E
DONE
#527 Sequence Diagram
#529 Wireframe
#524 Use Case Description
#525 Use Case Diagram
#528 BCE Framework
#399 As a PIN, I want to accept or cancel a CSR's offer to help my request so that I can control who provides the assistance I need.
N/E
DONE
#535 BCE Framework
#533 User Class Diagram
#536 Wireframe
USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#531 Use Case Description
#526 Sequence Diagram
#400 As a PIN, I want to filter requests by status (e.g., available, pending, completed) so that I can track progress easily.
N/E
DONE
#532 Use Case Diagram
#542 BCE Framework
#534 Sequence Diagram
#539 Use Case Diagram
#543 Wireframe
#401 As a PIN, I want to mark a request as completed so that I can close cases that have been fulfilled.
#545 Use Case Description
USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
N/E
DONE
#541 Sequence Diagram
#546 Use Case Diagram
#547 User Class Diagram
#548 Sequence Diagram
#549 BCE Framework
#550 Wireframe
#406 As a CSR Rep, I want to filter my completed volunteer services by service type or date, so that i can track past volunteer efforts
N/E
DONE
#553 Use Case Diagram
#552 Use Case Description


USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
track past volunteer efforts
N/E
DONE
#552 Use Case Description
#557 Wireframe
#540 User Class Diagram
#556 BCE Framework
#555 Sequence Diagram

5. Sprint 2

5.1. Sprint 2 - 1st Meeting (13/10/2025)

Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

Discussion summaries	
Meeting Topics	**Details**
Sprint Planning & Goals	Outlined development targets: database implementation, backend and frontend modules, and integration testing (Stage 1).
Database Implementation	Completed schema for User Admin, PIN, and CSR entities. Decided to use Drizzle ORM for consistent mapping.
Backend Development	Relevant developers to began coding API routes (login, CRUD operations) and route testing.
Frontend Development	Syafiqah and Karissa linked React components with backend endpoints; initial layout for login and user management screens completed.
Integration Plan	Defined schedule for API testing using Postman before Stage 1 integration.
Documentation	Sprint report section for UML consolidation and technical progress notes.

Action Plans	
Tasks	**Delegation**
Complete backend API functions for Login / CRUD operations	Respective backend developers
Implement database connection and test queries	Respective backend developers
Finish frontend forms and routing	Syafiqah / Karissa
Conduct initial integration test and log bugs	Respective backend developers

Meeting Follow-ups	
Meeting Place	**Next Meeting Date**

Meeting Follow-ups	
Block B Common Study Area	22/10/25
Meeting Topics	
1. Integration progress review 2. Bug tracking and refactoring 3. Documentation update 4. Sprint handover for testing	

5.2. Sprint 2 - 2nd Meeting (22/10/2025)

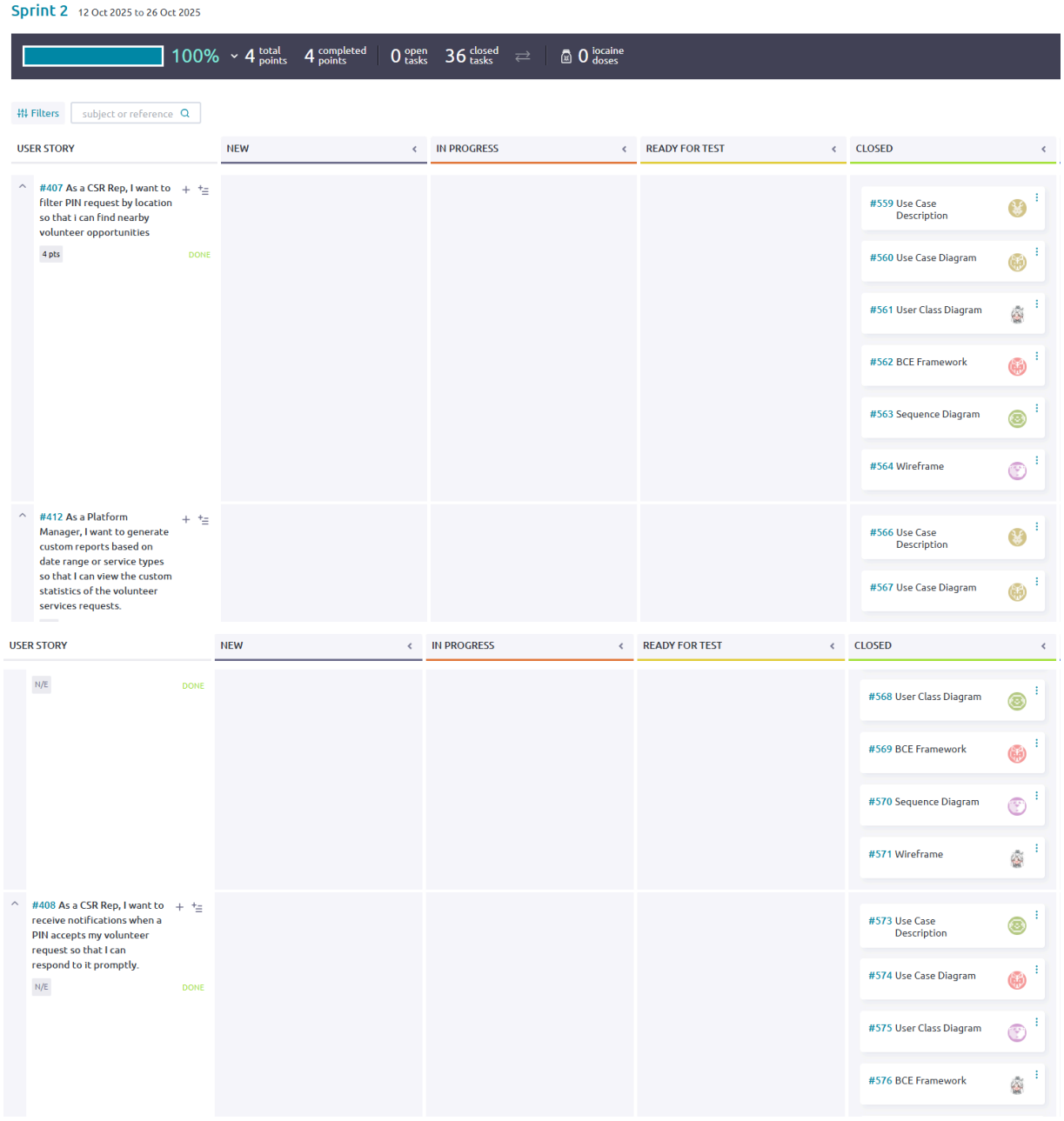
Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

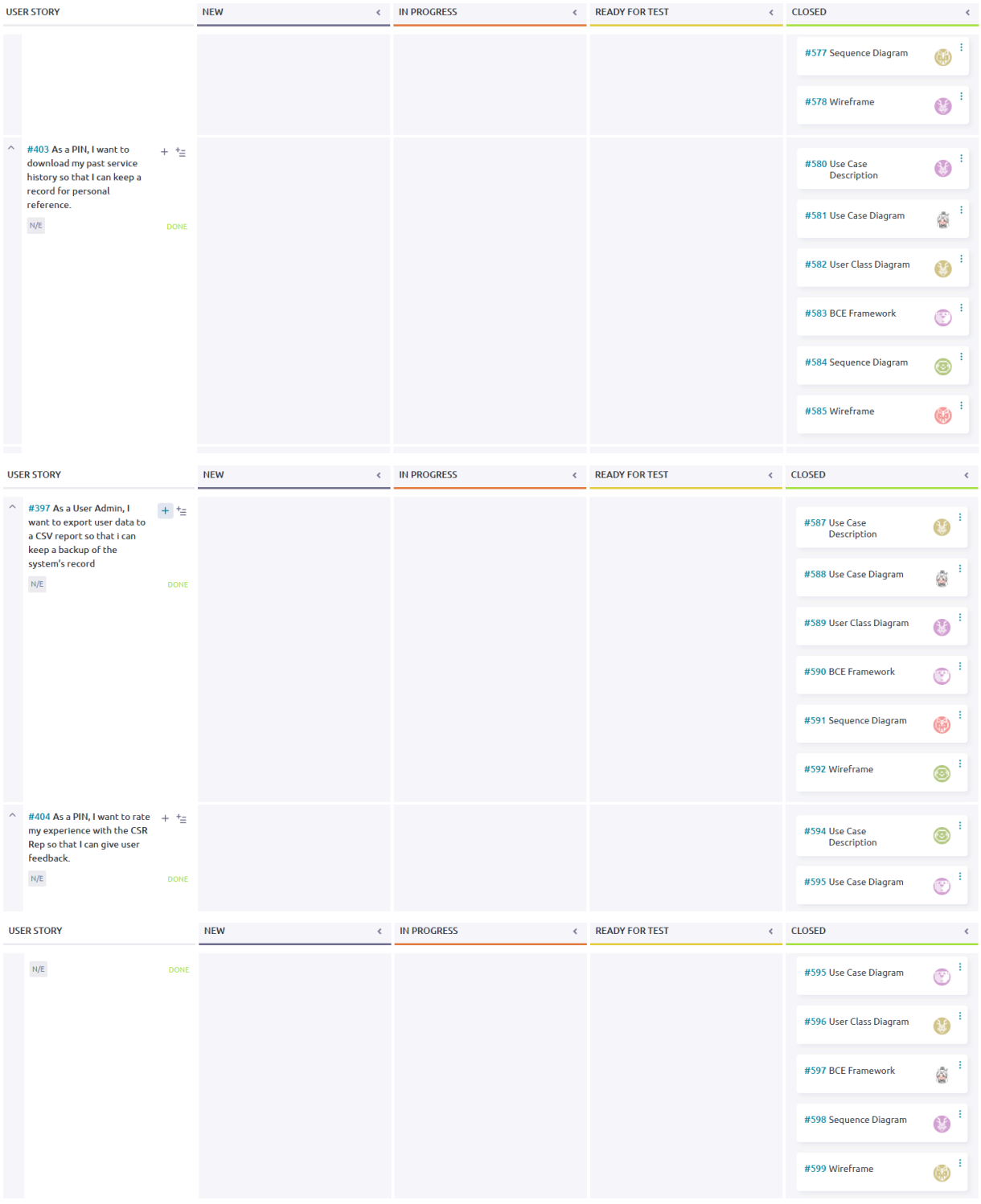
Discussion summaries	
Meeting Topics	**Details**
Integration Testing (Stage 1)	Backend and frontend modules successfully linked. CRUD operations verified.
Code Review	Developers perform code review to ensure naming and layer consistency.
Documentation	Respective members to update documentation to ensure consistency
Sprint Transition	Confirmed readiness for Sprint 3 integration and finalisation phase.

Action Plans	
Tasks	**Delegation**
Merge final branches into main repository	Respective developers
Prepare test cases for UAT	Respective developers
Update UML	Jiya
Plan for system deployment and demo	All members

Meeting Follow-ups	
Meeting Place	**Next Meeting Date**
Block B Common Study Area	27/10/25
Meeting Topics	
1. Full system integration 2. Stage 2 testing preparation 3. Presentation outline planning	

5.3. Sprint 2 - Taiga




USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#577 Sequence Diagram
#578 Wireframe
#403 As a PIN, I want to download my past service history so that I can keep a record for personal reference.
N/E
DONE
#580 Use Case Description
#581 Use Case Diagram
#582 User Class Diagram
#583 BCE Framework
#584 Sequence Diagram
#585 Wireframe
USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#397 As a User Admin, I want to export user data to a CSV report so that i can keep a backup of the system's record
N/E
DONE
#587 Use Case Description
#588 Use Case Diagram
#589 User Class Diagram
#590 BCE Framework
#591 Sequence Diagram
#592 Wireframe
#404 As a PIN, I want to rate my experience with the CSR Rep so that I can give user feedback.
N/E
DONE
#594 Use Case Description
#595 Use Case Diagram
USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
N/E
DONE
#595 Use Case Diagram
#596 User Class Diagram
#597 BCE Framework
#598 Sequence Diagram
#599 Wireframe

6. Sprint 3

6.1. Sprint 3 - 1st Meeting (27/10/2025)

Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

Discussion summaries	
Meeting Topics	**Details**
Integration Progress	Database connections and API flows validated for all user roles.
Frontend Enhancements	Improved dashboard views and form validation. Added report export feature.
Final Testing Plan	Scheduled Stage 2 integration and User Acceptance Testing.
Documentation	Update of Gantt chart and final diagrams for submission.
Video Demo Planning	Justin and Eugene assigned to record and edit demo walk-through.

Action Plans	
Tasks	**Delegation**
Conduct Stage 2 integration tests	All Members
Record testing results and screenshots	All respective developers / Rissa
Complete video demo draft	Justin / Eugene
Prepare final documentation for mentor review	All members

Meeting Follow-ups	
Meeting Place	**Next Meeting Date**
Microsoft Teams	05/11/25

Meeting Follow-ups
Meeting Topics
1. UAT results discussion 2. Bug review and refactoring 3. Demo review and submission readiness

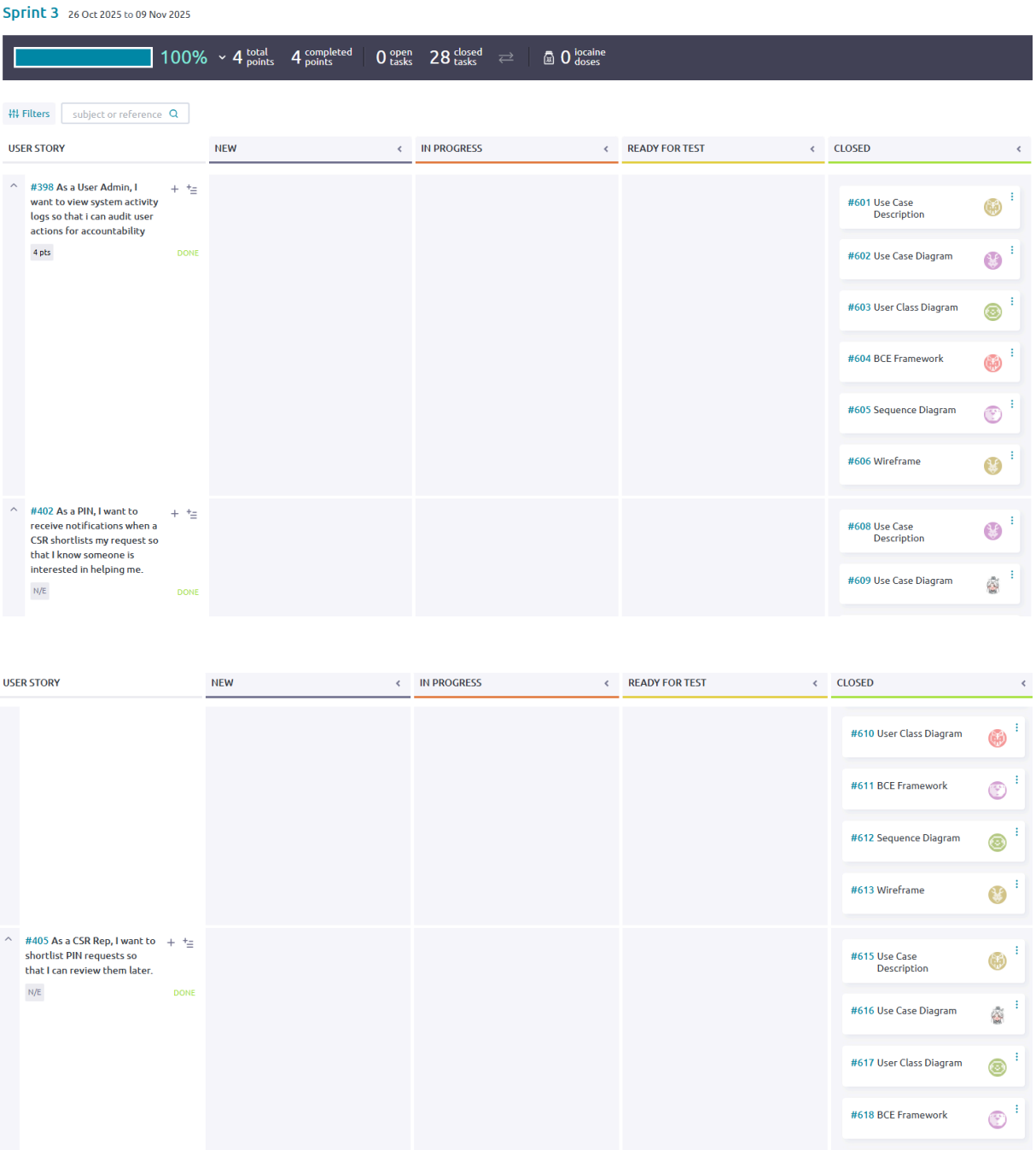
6.2. Sprint 3 - 2nd Meeting (05/11/2025)

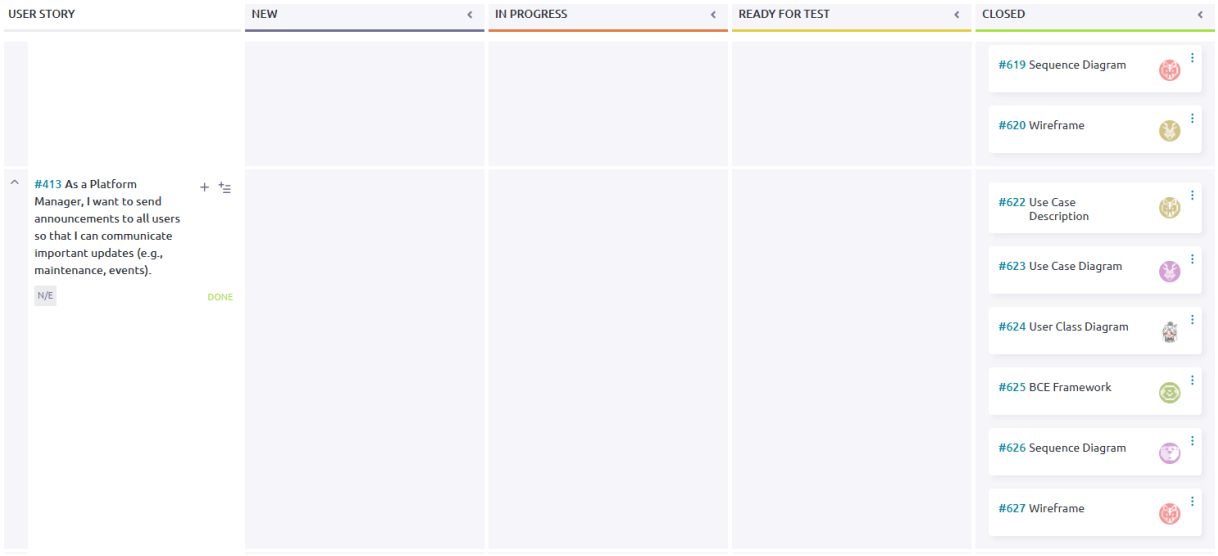
Meeting Attendance	
In Attendance	Foo Kok Tai Justin, Eugene Lay Chai Chun, Nur Syafiqah Binte Mustazam, Jiyavudeen Mohamed Haneefa, Karissa Angeline Ramos Wong, Tan Jun Rong Dillon, Sim Zhan Qi
Absent	None

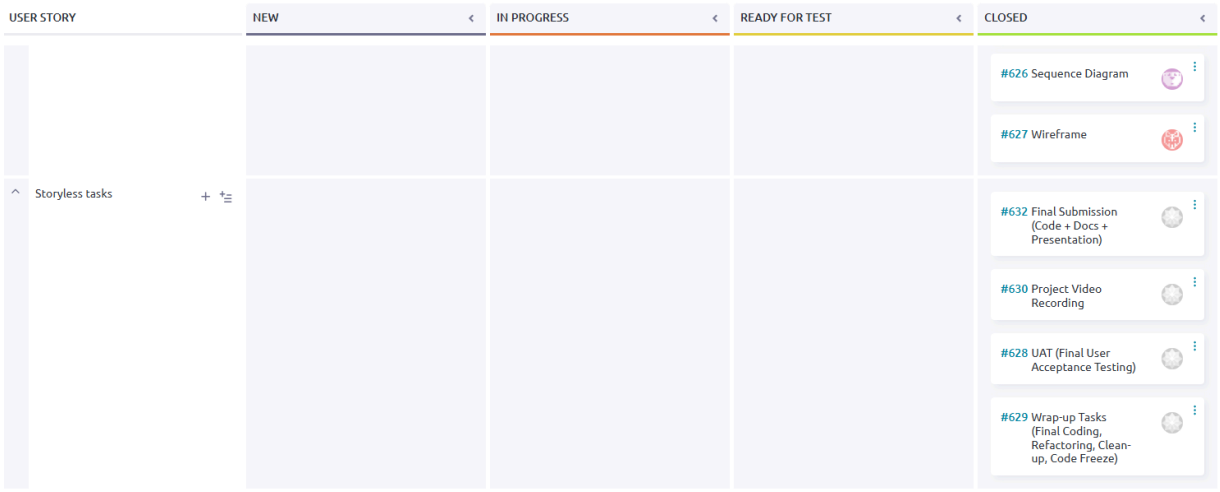
Discussion summaries	
Meeting Topics	**Details**
UAT Outcome	All test cases passed successfully; minor UI alignment adjusted.
Code Finalisation	Repository cleaned and tags applied for release version.
Presentation Preparation	Slides and demo video finalised; roles assigned for mentor review session.
Documentation Submission	Compiled final documents and uploaded to GitHub and Taiga.

Action Plans	
Tasks	**Delegation**
Deliver presentation and record session	All members
Verify final report sign-off	All Members
Submit all artifacts before deadline (13 Nov 2025)	Justin / Eugene

6.3. Sprint 3 - Taiga

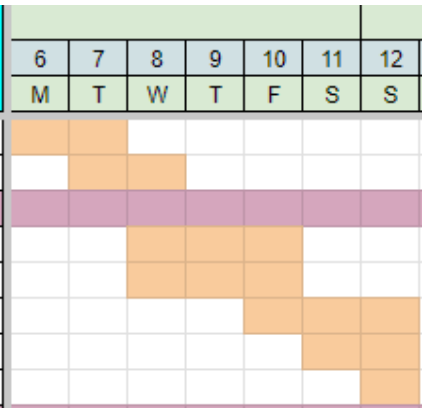



USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#619 Sequence Diagram
#620 Wireframe
#413 As a Platform Manager, I want to send announcements to all users so that I can communicate important updates (e.g., maintenance, events).
N/E
DONE
#622 Use Case Description
#623 Use Case Diagram
#624 User Class Diagram
#625 BCE Framework
#626 Sequence Diagram
#627 Wireframe

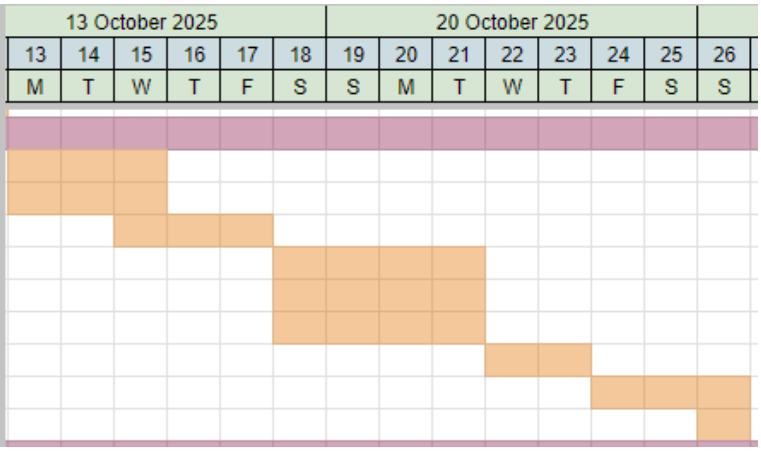

USER STORY
NEW
IN PROGRESS
READY FOR TEST
CLOSED
#626 Sequence Diagram
#627 Wireframe
Storyless tasks
#632 Final Submission (Code + Docs + Presentation)
#630 Project Video Recording
#628 UAT (Final User Acceptance Testing)
#629 Wrap-up Tasks (Final Coding, Refactoring, Clean-up, Code Freeze)

7. Gantt Chart

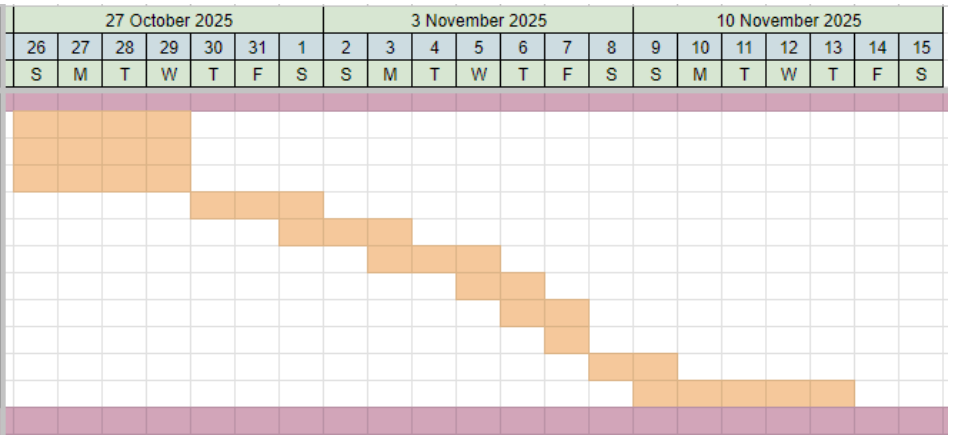
TASK TITLE	Duration (Days)	Start	End
User Stories	2	06/Oct/25	07/Oct/25
Setup Dev Env, Database and Codebase		07/Oct/25	08/Oct/25
Sprint 1			
Use Case Diagram	4	08/Oct/25	10/Oct/25
Use Case Description & BCE		08/Oct/25	10/Oct/25
Sequence Diagram		10/Oct/25	12/Oct/25
Wireframe (UI/UX)		11/Oct/25	12/Oct/25
Documentation Update		12/Oct/25	12/Oct/25



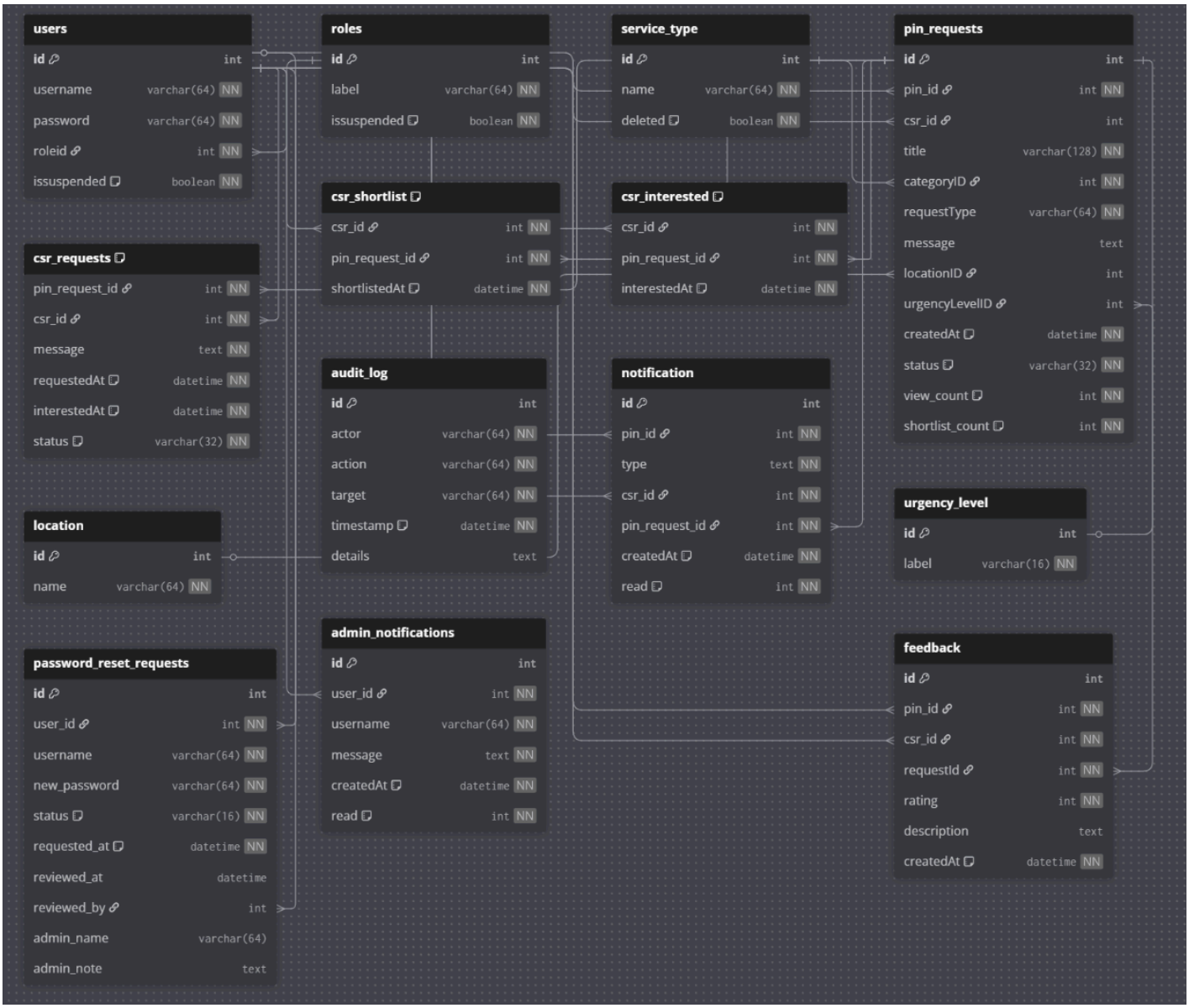
TASK TITLE	Duration (Days)	Start	End
Sprint 2			
Consolidate Use Case Diagram	13	13/Oct/25	15/Oct/25
Consolidate Use Case Description		13/Oct/25	15/Oct/25
Consolidate Sequence Diagram		15/Oct/25	17/Oct/25
Database Implementation		18/Oct/25	21/Oct/25
Backend Implementation		18/Oct/25	21/Oct/25
Frontend Implementation		18/Oct/25	21/Oct/25
Code Review & Refactor		22/Oct/25	23/Oct/25
Integration Testing - Stage 1		24/Oct/25	26/Oct/25
Documentation Update		26/Oct/25	26/Oct/25



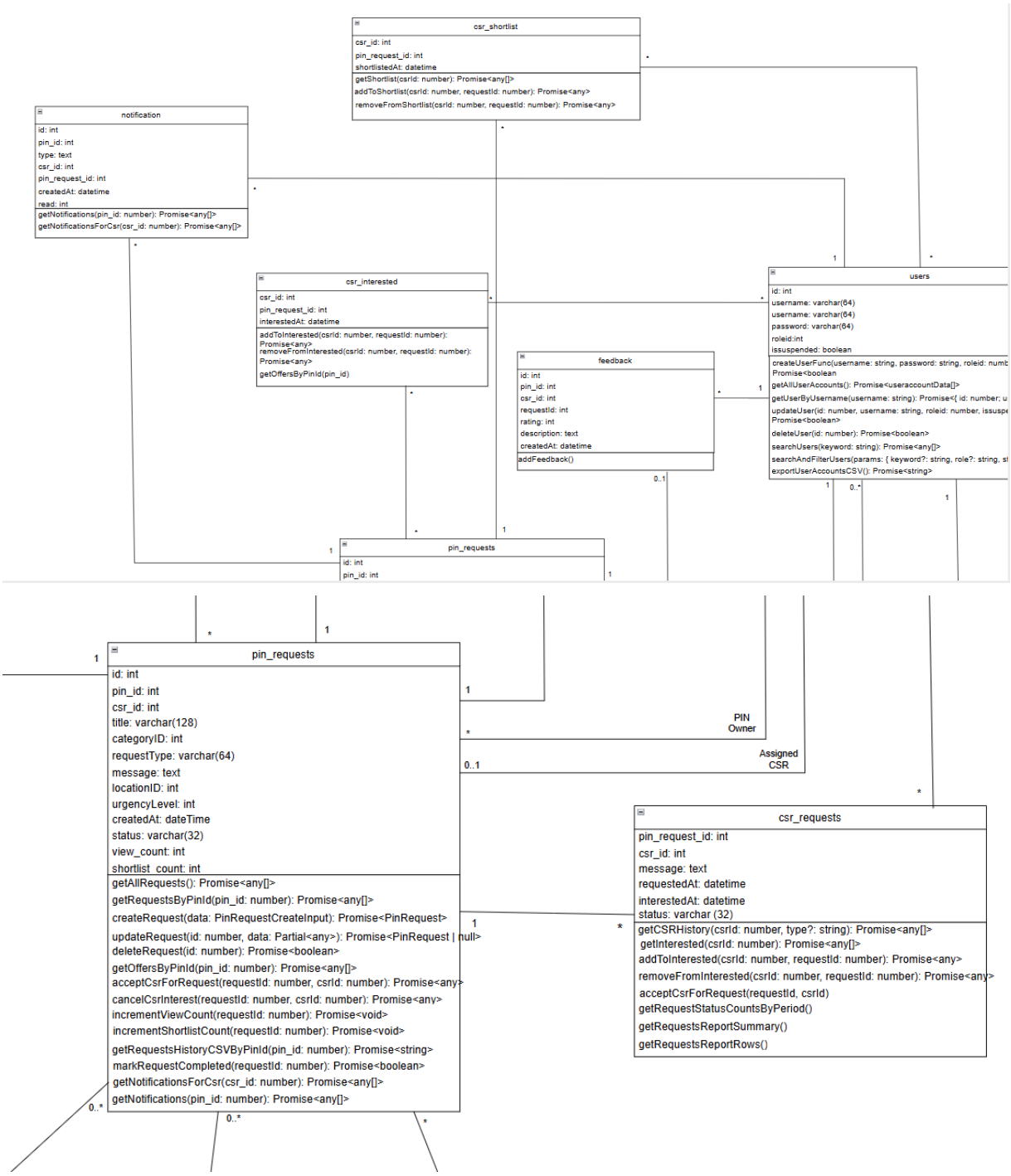
TASK TITLE	Duration (Days)	Start	End
Sprint 3			
Database Implementation	18	26/Oct/25	29/Oct/25
Backend Implementation		26/Oct/25	29/Oct/25
Frontend Implementation		26/Oct/25	29/Oct/25
Code Review & Refactor		30/Nov/25	01/Nov/25
Integration Testing - Stage 2		01/Nov/25	03/Nov/25
Final Code Review & Refactor (Clean-up)		03/Nov/25	05/Nov/25
Final Review & Bug Fixes		05/Nov/25	06/Nov/25
UAT (Final User Acceptance Testing)		06/Nov/25	07/Nov/25
Documentation Finalisation		07/Nov/25	07/Nov/25
Project Video Recording		08/Nov/25	09/Nov/25
Final Submission (Code + Docs + Presenta		09/Nov/25	13/Nov/25

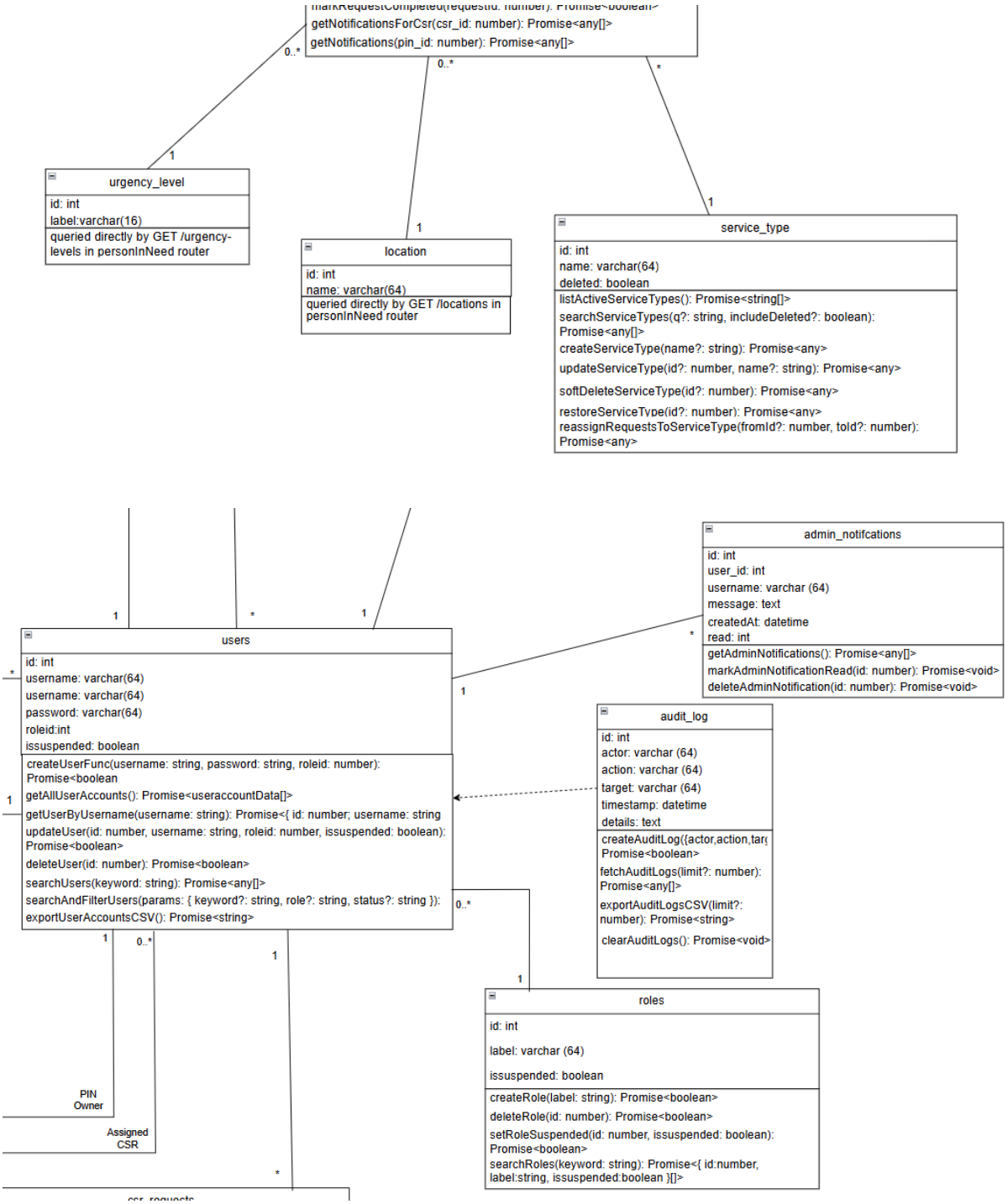


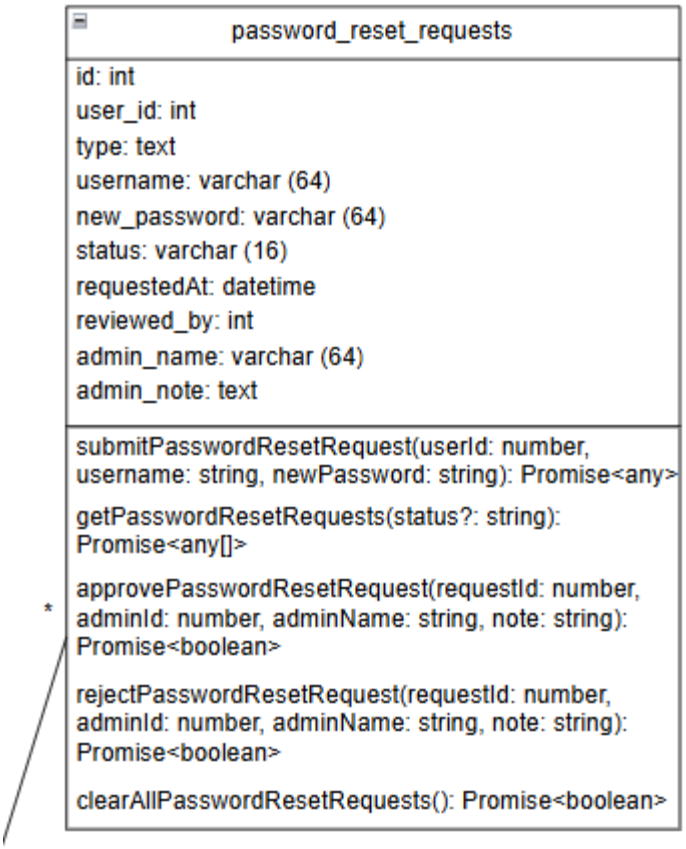
8. Entity Relation (ER)

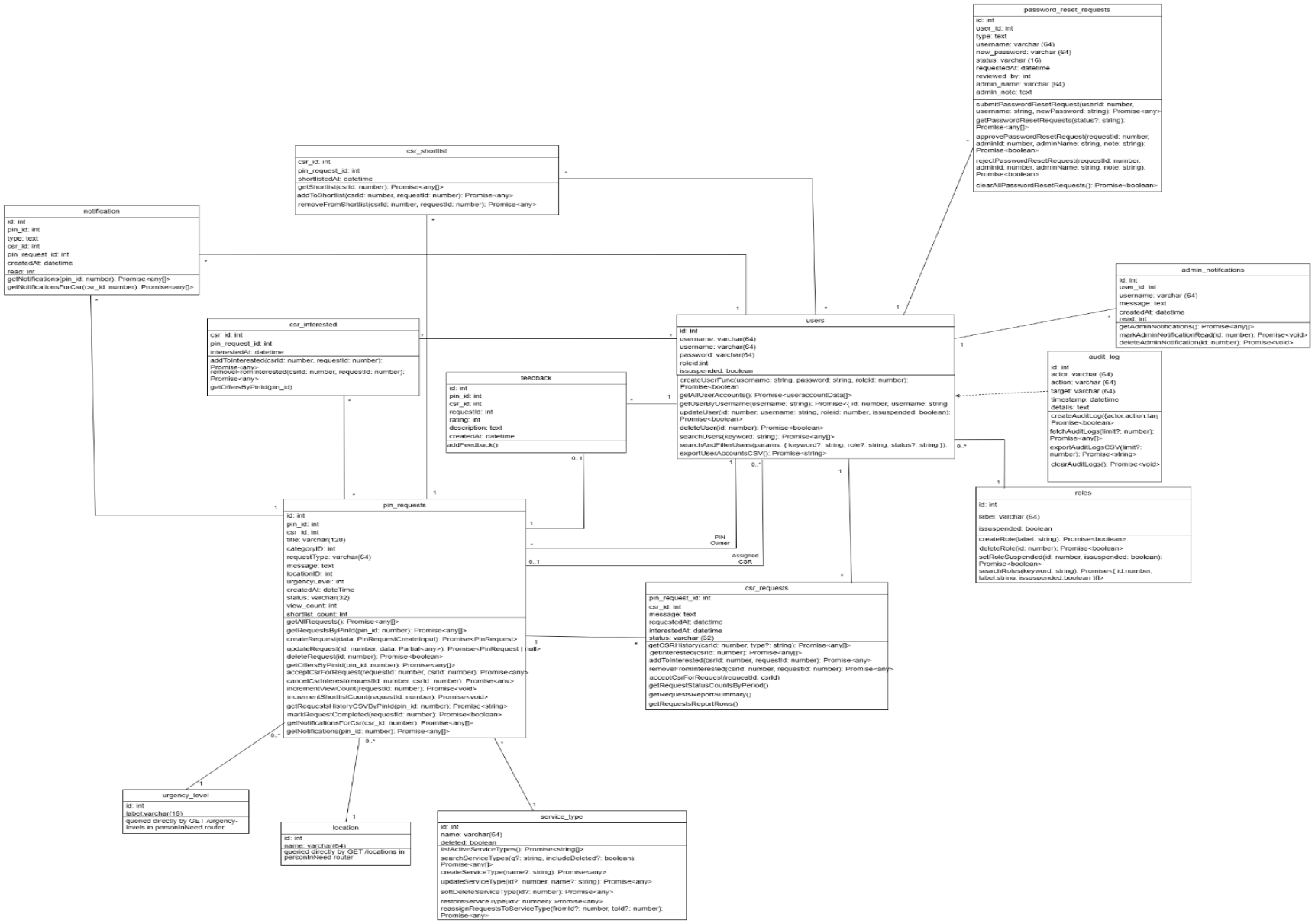


8.1. Class Diagram




getNotificationsForCsr(csr_id: number): Promise<any[]>
getNotifications(pin_id: number): Promise<any[]>
0..*
0..*
*
1
1
1
urgency_level
id: int
label:varchar(16)
queried directly by GET /urgency-levels in personInNeed router
location
id: int
name: varchar(64)
queried directly by GET /locations in personInNeed router
service_type
id: int
name: varchar(64)
deleted: boolean
listActiveServiceTypes(): Promise<string[]>
searchServiceTypes(q?: string, includeDeleted?: boolean): Promise<any[]>
createServiceType(name?: string): Promise<any>
updateServiceType(id?: number, name?: string): Promise<any>
softDeleteServiceType(id?: number): Promise<any>
restoreServiceType(id?: number): Promise<any>
reassignRequestsToServiceType(fromId?: number, toId?: number): Promise<any>
admin_notifcations
id: int
user_id: int
username: varchar (64)
message: text
createdAt: datetime
read: int
getAdminNotifications(): Promise<any[]>
markAdminNotificationRead(id: number): Promise<void>
deleteAdminNotification(id: number): Promise<void>
users
id: int
username: varchar(64)
username: varchar(64)
password: varchar(64)
roleid:int
issuspended: boolean
createUserFunc(username: string, password: string, roleid: number): Promise<boolean
getAllUserAccounts(): Promise<useraccountData[]>
getUserByUsername(username: string): Promise<{ id: number; username: string
updateUser(id: number, username: string, roleid: number, issuspended: boolean): Promise<boolean>
deleteUser(id: number): Promise<boolean>
searchUsers(keyword: string): Promise<any[]>
searchAndFilterUsers(params: { keyword?: string, role?: string, status?: string }):
exportUserAccountsCSV(): Promise<string>
audit_log
id: int
actor: varchar (64)
action: varchar (64)
target: varchar (64)
timestamp: datetime
details: text
createAuditLog({actor,action,tar
Promise<boolean>
fetchAuditLogs(limit?: number): Promise<any[]>
exportAuditLogsCSV(limit?: number): Promise<string>
clearAuditLogs(): Promise<void>
roles
id: int
label: varchar (64)
issuspended: boolean
createRole(label: string): Promise<boolean>
deleteRole(id: number): Promise<boolean>
setRoleSuspended(id: number, issuspended: boolean): Promise<boolean>
searchRoles(keyword: string): Promise<{ id:number, label:string, issuspended:boolean }[]>
PIN Owner
Assigned CSR


password_reset_requests
id: int
user_id: int
type: text
username: varchar (64)
new_password: varchar (64)
status: varchar (16)
requestedAt: datetime
reviewed_by: int
admin_name: varchar (64)
admin_note: text
submitPasswordResetRequest(userId: number, username: string, newPassword: string): Promise<any>
getPasswordResetRequests(status?: string): Promise<any[]>
approvePasswordResetRequest(requestId: number, adminId: number, adminName: string, note: string): Promise<boolean>
rejectPasswordResetRequest(requestId: number, adminId: number, adminName: string, note: string): Promise<boolean>
clearAllPasswordResetRequests(): Promise<boolean>
*


password_reset_requests
id: int
user_id: int
type: text
username: varchar (64)
new_password: varchar (64)
status: varchar (16)
requestedAt: datetime
reviewed_by: int
admin_name: varchar (64)
admin_note: text
submitPasswordResetRequest(userId: number, username: string, newPassword: string): Promise<any>
getPasswordResetRequests(status?: string): Promise<any[]>
approvePasswordResetRequest(requestId: number, adminId: number, adminName: string, note: string): Promise<boolean>
rejectPasswordResetRequest(requestId: number, adminId: number, adminName: string, note: string): Promise<boolean>
clearAllPasswordResetRequests(): Promise<boolean>
csr_shortlist
csr_id: int
pin_request_id: int
shortlistedAt: datetime
getShortlist(csrId: number): Promise<any[]>
addToShortlist(csrId: number, requestId: number): Promise<any>
removeFromShortlist(csrId: number, requestId: number): Promise<any>
notification
id: int
pin_id: int
type: text
csr_id: int
pin_request_id: int
createdAt: datetime
read: int
getNotifications(pin_id: number): Promise<any[]>
getNotificationsForCsr(csr_id: number): Promise<any[]>
admin_notifcations
id: int
user_id: int
username: varchar (64)
message: text
createdAt: datetime
read: int
getAdminNotifications(): Promise<any[]>
markAdminNotificationRead(id: number): Promise<void>
deleteAdminNotification(id: number): Promise<void>
csr_interested
csr_id: int
pin_request_id: int
interestedAt: datetime
addToInterested(csrId: number, requestId: number): Promise<any>
removeFromInterested(csrId: number, requestId: number): Promise<any>
getOffersByPinId(pin_id)
feedback
id: int
pin_id: int
csr_id: int
requestId: int
rating: int
description: text
createdAt: datetime
addFeedback()
users
id: int
username: varchar(64)
username: varchar(64)
password: varchar(64)
roleid:int
issuspended: boolean
createUserFunc(username: string, password: string, roleId: number): Promise<boolean
getAllUserAccounts(): Promise<useraccountData[]>
getUserByUsername(username: string): Promise<{ id: number; username: string
updateUser(id: number, username: string, roleId: number, issuspended: boolean): Promise<boolean>
deleteUser(id: number): Promise<boolean>
searchUsers(keyword: string): Promise<any[]>
searchAndFilterUsers(params: { keyword?: string, role?: string, status?: string }):
exportUserAccountsCSV(): Promise<string>
audit_log
id: int
actor: varchar (64)
action: varchar (64)
target: varchar (64)
timestamp: datetime
details: text
createAuditLog({actor,action,tar Promise<boolean>
fetchAuditLogs(limit?: number): Promise<any[]>
exportAuditLogsCSV(limit?: number): Promise<string>
clearAuditLogs(): Promise<void>
roles
id: int
label: varchar (64)
issuspended: boolean
createRole(label: string): Promise<boolean>
deleteRole(id: number): Promise<boolean>
setRoleSuspended(id: number, issuspended: boolean): Promise<boolean>
searchRoles(keyword: string): Promise<{ id:number, label:string, issuspended:boolean }[]>
pin_requests
id: int
pin_id: int
csr_id: int
title: varchar(128)
categoryID: int
requestType: varchar(64)
message: text
locationID: int
urgency_evel: int
createdAt: dateTime
status: varchar(32)
view_count: int
shortlist_count: int
getAllRequests(): Promise<any[]>
getRequestsByPinId(pin_id: number): Promise<any[]>
createRequest(data: PinRequestCreateInput): Promise<PinRequest>
updateRequest(id: number, data: Partial<any>): Promise<PinRequest | null>
deleteRequest(id: number): Promise<boolean>
getOffersByPinId(pin_id: number): Promise<any[]>
acceptCsrForRequest(requestId: number, csrId: number): Promise<any>
cancelCsrInterest(requestId: number, csrId: number): Promise<any>
incrementViewCount(requestId: number): Promise<void>
incrementShortlistCount(requestId: number): Promise<void>
getRequestsHistoryCSVByPinId(pin_id: number): Promise<string>
markRequestCompleted(requestId: number): Promise<boolean>
getNotificationsForCsr(csr_id: number): Promise<any[]>
getNotifications(pin_id: number): Promise<any[]>
PIN Owner
Assigned CSR
csr_requests
pin_request_id: int
csr_id: int
message: text
requestedAt: datetime
interestedAt: datetime
status: varchar (32)
getCSRHistory(csrId: number, type?: string): Promise<any[]>
getInterested(csrId: number): Promise<any[]>
addToInterested(csrId: number, requestId: number): Promise<any>
removeFromInterested(csrId: number, requestId: number): Promise<any>
acceptCsrForRequest(requestId, csrId)
getRequestStatusCountsByPeriod()
getRequestsReportSummary()
getRequestsReportRows()
urgency_level
id: int
label:varchar(16)
queried directly by GET /urgency-levels in personInNeed router
location
id: int
name: varchar(64)
queried directly by GET /locations in personInNeed router
service_type
id: int
name: varchar(64)
deleted: boolean
listActiveServiceTypes(): Promise<string[]>
searchServiceTypes(q?: string, includeDeleted?: boolean): Promise<any[]>
createServiceType(name?: string): Promise<any>
updateServiceType(id?: number, name?: string): Promise<any>
softDeleteServiceType(id?: number): Promise<any>
restoreServiceType(id?: number): Promise<any>
reassignRequestsToServiceType(fromId?: number, toId?: number): Promise<any>

9. Test Cases

9.1. User Admin

Test Date: 07/11/2025				Written By: Eugene			
Actor: User Admin				Tested By: Rissa			
USER STORY (#392): As a User Admin, I want to approve user's new password changes so that user can re-access their account							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-01	Approve Password Change	username: Bob new_password:Password123	1.Navigate to UA's Password page 2.View User's password request 3. Click "Accept" User's new password	User password is updated	User password is updated	PASS	
TC-01 {ALT1}	Reject Password Change	username: Bob new_password:Password123	1.Navigate to UA's Password page 2.View User's password request 3. Click "Accept" User's new password	User password remain the same	User password remain the same	PASS	

Test Date: 07/11/2025			Written By: Eugene				
Actor: User Admin			Tested By: Rissa				
USER STORY (#397): As a user admin, I want to export user data to a CSV report so that i can keep a backup of the system's record							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-02	Downloading CSV	Id:5 username:Bob Roleid:2 issuspended:false	1. UA authenticate themself 2. View list of users 3. Click on "Export CSV" btn to download CSV	CSV file is download with user data	CSV file is download with user data	PASS	
TC-02 {Alt1}	No Data to download CSV	Nil	1.UA authenticate themself 2.View list of users 3.Click on "Export CSV" btn to download CSV	Toast notification says, no data found	Toast notification says, no data found	PASS	

9.2. Person in Need (PIN)

Test Date: 07/11/2025				Written By: Justin			
Actor: Person in Need(PIN)				Tested By: Syafiqah			
USER STORY (#399) As a PIN, I want to accept or cancel a CSR's offer to help my request so that I can control who provides the assistance I need.							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-03	Accept a CSR's offer for a specific request.	pin_request_id=13 csr_id = 28 status = Pending	1. Navigate to the PIN "My Offers" page 2. Clicks on 'Accept" on that CSR user	It will assign and change the status of the CSR user to "Accepted"	toast.success ("CSR accepted for this request.");	PASS	
TC-03 {Alt1}	Cancel a CSR's offer for a specific request.	pin_request_id=18 csr_id = 41 status = Pending	1. Navigate to the PIN "My Offers" page 2. Clicks on 'Cancel" on that CSR user	It will not assign the CSR user and change the status of the CSR user to "Rejected"	toast.success ("CSR interest cancelled.");	PASS	

Test Date: 07/11/2025				Written By: Justin			
Actor: Person in Need (PIN)				Tested By: Syafiqah			
USER STORY (#403) As a PIN, I want to download my past service history so that I can keep a record for personal reference.							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-04	Download all past and present request records of the PIN that is currently logged in.	title : hello requestType: Transport status : Available UrgencyLevelID:3 createdAt: 2025-10-15 13:42:46.977 view_count : 14 shortlist_count : 5	1. Navigate to the PIN “My Requests” page 2. Clicks on “Download Past Service History”	CSV file is download with user data	CSV file is download with user data	PASS	
TC-04 {Alt1}	No past and present request records of the PIN that is currently logged in.	NIL	1. Navigate to the PIN “My Requests” page 2. Clicks on “Download Past Service History”	CSV file is download with user data	CSV file is download with user data	PASS	

9.3. Corporate Social Responsibility (CSR) Representative

Test Date: 07/11/2025				Written By: Jiya			
Actor: Corporate Social Responsibility(CSR) Representative				Tested By: Zhan Qi			
USER STORY (#405): As a CSR Rep, I want to shortlist PIN requests so that I can review them later.							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-05	Shortlisting a specific PIN request	pin_request_id = 14 csr_id = 28	1. Navigate to the CSR "Requests" section page 2. Click on that request number 3. Click on the heart shaped icon in the "Request Details" Page	The heart shaped icon will turn red and the request will be shown in "Shortlist" page.	The heart shaped icon turned red and the request will be shown in "Shortlist" page.	PASS	

Test Date: 07/11/2025				Written By: Jiya			
Actor: Corporate Social Responsibility(CSR) Representative				Tested By: Zhan Qi			
USER STORY (#408): As a CSR Rep, I want to receive notifications when a PIN accepts my volunteer request so that I can respond to it promptly.							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-06	Click accept for a request in the My Offers page for a specific CSR user	pin_id = 38 csr_id = 28 pin_request_id = 13	1. Navigate to the PIN "My Offers" page 2. Clicks on 'Accept" on that CSR user 3. Navigate to the CSR main dashboard page 4. Click on notification bell icon	The notification will show accepted by (pin_id) name with the request header and date time of the acceptance	It shows ✅ **Accepted by pin** Request: defendo debilito earum 2025-11-08 18:24:25	PASS	

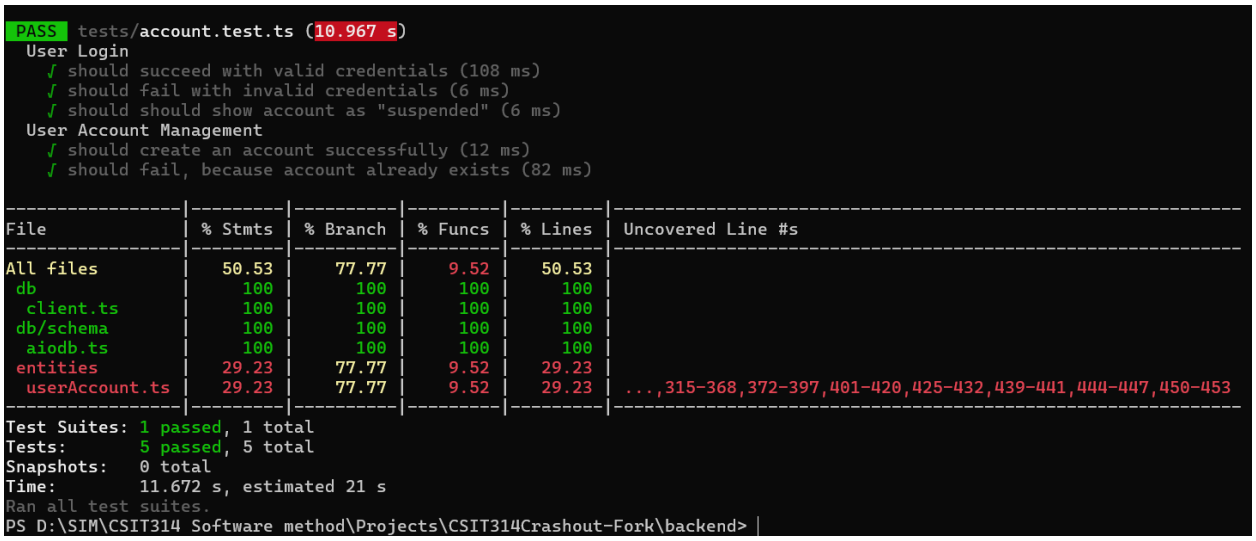
9.4. Platform Manager

Test Date: 07/11/2025				Written By: Dillon			
Actor: Platform Manager				Tested By: Zhan Qi			
USER STORY (#411): As a Platform Manager, I want to generate monthly reports so that I can view the monthly statistics of the volunteer services requests.							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-07	Viewing monthly statistics of volunteer service requested.	There are relevant data in the database in the date range of the current month.	1.Navigate to PM's Reports page. 2. Click "Generate Monthly Report".	System will display monthly report.	System will display monthly report.	PASS	
TC-07 {Alt1}	Viewing monthly statistics of volunteer service requested.	There are **no available** relevant data in database in the date range of the current month.	1.Navigate to PM's Reports page. 2. Click "Generate Monthly Report".	Error Message displayed: "No data"	Error Message displayed: "No data"	PASS	

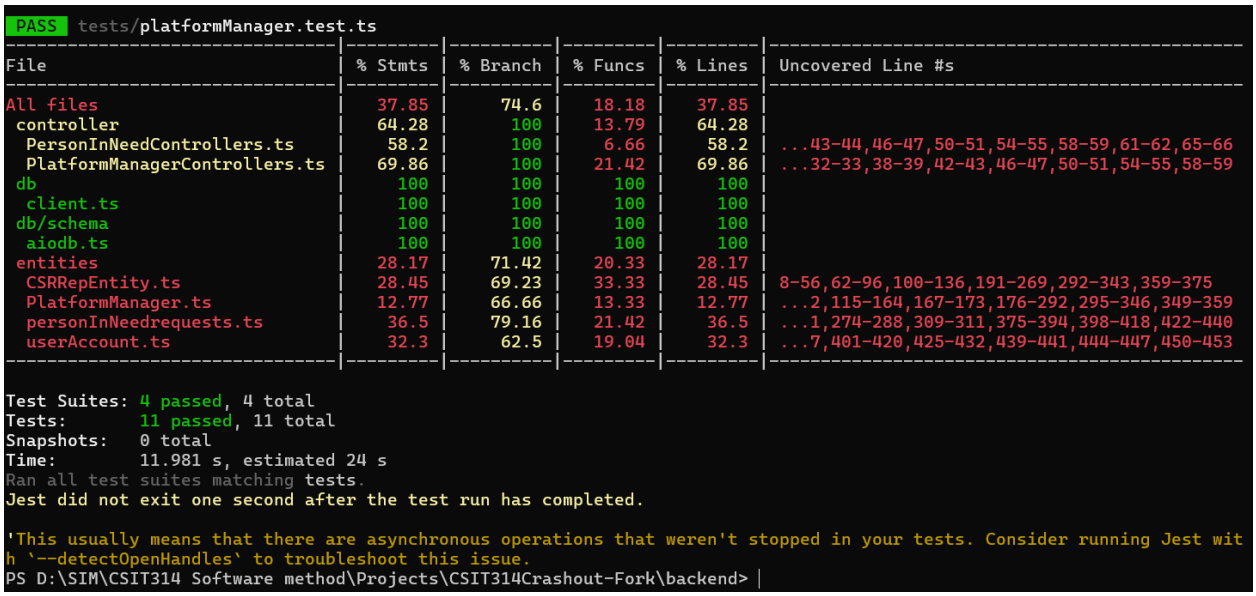
Test Date: 07/11/2025				Written By: Dillon			
Actor: Platform Manager (PM)				Tested By: Zhan Qi			
USER STORY (#413): As a Platform Manager, I want to send announcements to all users so that I can communicate important updates (e.g., maintenance, events).							
Test Case	Test Step Description	Test Data	Test Steps	Expected Results	Actual Results	Pass/Fail	Remarks
TC-08	Sending Announcement to all Users with content	"There is scheduled maintenance from the 1 Jan 2025 00:00hrs to 1 Jan 2025 02:00hrs".	1.Navigate to the PM's Announcement page. 2. Insert Test Data into the textbox. 3. Click "Send to all users".	Announcements are sent to all users.	Announcements are sent to all users.	PASS	
TC-08 {Alt1}	Sending Announcement to all Users without content	NIL	1.Navigate to PM's Announcement page. 2. Click "Send to all users".	Error Message displayed: "Message cannot be empty"	Error Message displayed: "Message cannot be empty"	PASS	

9.5 Test Driven Plan

In this project, TDD is applied mainly to the **backend Node.js services** using **Jest**. The goal is to ensure that each core function (e.g. user registration, authentication, features) is **specified, tested, and verified automatically** as part of the CI/CD pipeline.

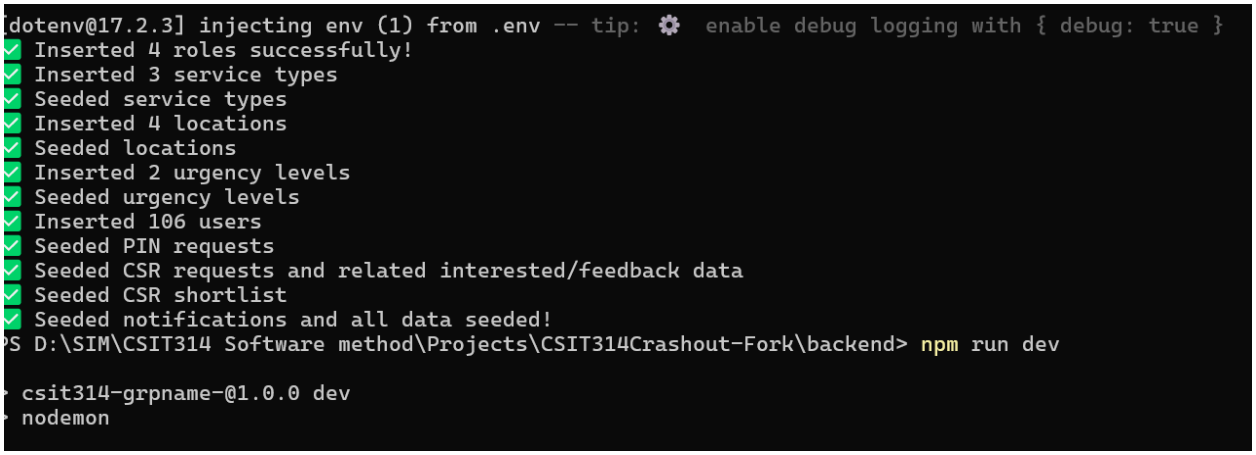


```
PASS tests/account.test.ts (10.967 s)
  User Login
    ✓ should succeed with valid credentials (108 ms)
    ✓ should fail with invalid credentials (6 ms)
    ✓ should should show account as "suspended" (6 ms)
  User Account Management
    ✓ should create an account successfully (12 ms)
    ✓ should fail, because account already exists (82 ms)

-----------------|---------|----------|---------|---------|-------------------------------------------------------------------
File             | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------|---------|----------|---------|---------|-------------------------------------------------------------------
All files        |   50.53 |    77.77 |    9.52 |   50.53 |
 db              |     100 |      100 |     100 |     100 |
  client.ts      |     100 |      100 |     100 |     100 |
 db/schema       |     100 |      100 |     100 |     100 |
  aiodb.ts       |     100 |      100 |     100 |     100 |
 entities        |   29.23 |    77.77 |    9.52 |   29.23 |
  userAccount.ts |   29.23 |    77.77 |    9.52 |   29.23 | ...,315-368,372-397,401-420,425-432,439-441,444-447,450-453
-----------------|---------|----------|---------|---------|-------------------------------------------------------------------
Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        11.672 s, estimated 21 s
Ran all test suites.
PS D:\SIM\CSIT314 Software method\Projects\CSIT314Crashout-Fork\backend>
```



```
PASS tests/platformManager.test.ts
--------------------------------|---------|----------|---------|---------|-----------------------------------------------
File                            | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------------|---------|----------|---------|---------|-----------------------------------------------
All files                       |   37.85 |     74.6 |   18.18 |   37.85 |
 controller                     |   64.28 |      100 |   13.79 |   64.28 |
  PersonInNeedControllers.ts    |    58.2 |      100 |    6.66 |    58.2 | ...43-44,46-47,50-51,54-55,58-59,61-62,65-66
  PlatformManagerControllers.ts |   69.86 |      100 |   21.42 |   69.86 | ...32-33,38-39,42-43,46-47,50-51,54-55,58-59
 db                             |     100 |      100 |     100 |     100 |
  client.ts                     |     100 |      100 |     100 |     100 |
 db/schema                      |     100 |      100 |     100 |     100 |
  aiodb.ts                      |     100 |      100 |     100 |     100 |
 entities                       |   28.17 |    71.42 |   20.33 |   28.17 |
  CSRRepEntity.ts               |   28.45 |    69.23 |   33.33 |   28.45 | 8-56,62-96,100-136,191-269,292-343,359-375
  PlatformManager.ts            |   12.77 |    66.66 |   13.33 |   12.77 | ...2,115-164,167-173,176-292,295-346,349-359
  personInNeedrequests.ts       |    36.5 |    79.16 |   21.42 |    36.5 | ...1,274-288,309-311,375-394,398-418,422-440
  userAccount.ts                |    32.3 |     62.5 |   19.04 |    32.3 | ...7,401-420,425-432,439-441,444-447,450-453
--------------------------------|---------|----------|---------|---------|-----------------------------------------------

Test Suites: 4 passed, 4 total
Tests:       11 passed, 11 total
Snapshots:   0 total
Time:        11.981 s, estimated 24 s
Ran all test suites matching tests.
Jest did not exit one second after the test run has completed.

'This usually means that there are asynchronous operations that weren't stopped in your tests. Consider running Jest wit
h `--detectOpenHandles` to troubleshoot this issue.
PS D:\SIM\CSIT314 Software method\Projects\CSIT314Crashout-Fork\backend>
```

9.6 Test Data Generation



To support system testing and validation, **mock data** was generated using the **Faker.js** library. Faker.js was utilized to automatically create realistic yet non-sensitive data for all entities within the system, including users, requests, and notifications.

The successful data seeding process was verified through console output as shown in Figure, confirming that all records were inserted and seeded properly.

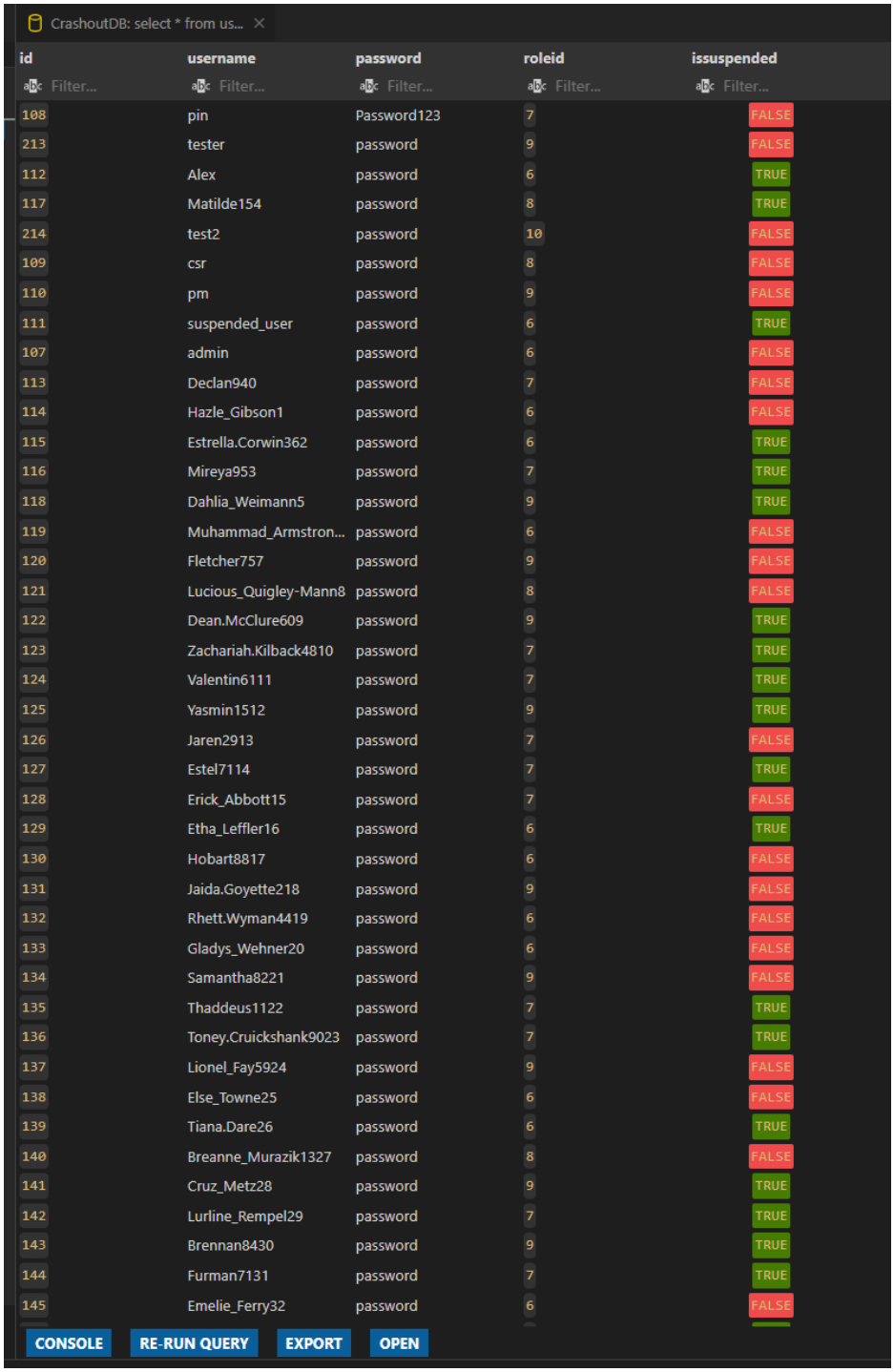
This dataset allowed developers to test authentication, role-based access control, and data relationships under realistic conditions.

The automated data seeding process ensured that all dependent entities—such as **roles**, **locations**, **service types**, and **urgency levels** were also pre-populated before inserting the user data.

The system inserted and seeded multiple entities, including:

- **4 roles** (for access control)
- **3 service types** and **4 locations**
- **106 users**
- Associated **PIN requests**, **CSR requests**, **shortlists**, and **notifications**

Faker.js generated randomized yet structured user information such as names, email addresses, phone numbers, and assigned roles.
This ensured that the database contained diverse and realistic test data for verifying all backend functions — including authentication, request handling, and notification delivery.



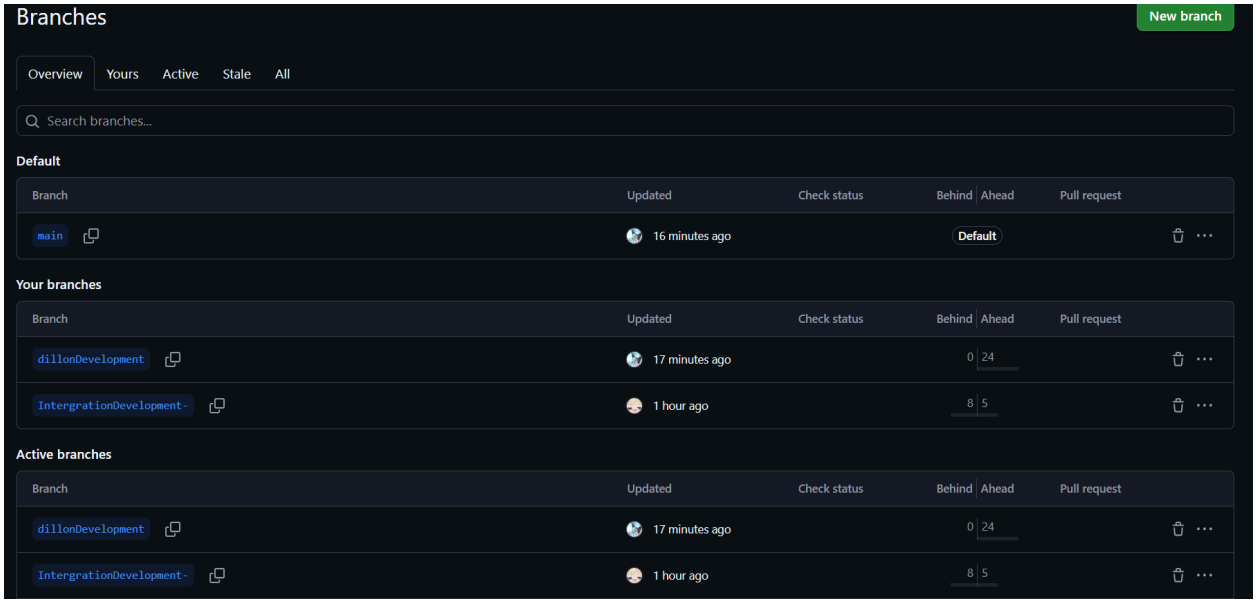
CrashoutDB: select * from us...

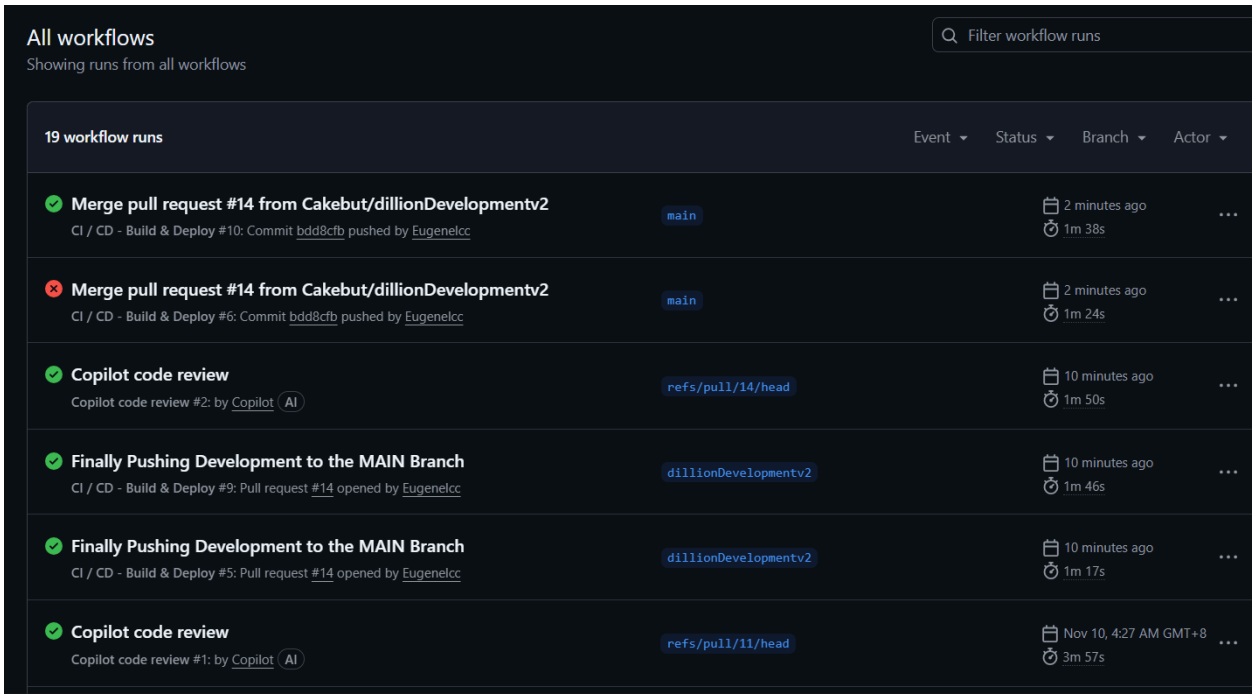
id	username	password	roleid	issuspended
Filter...	Filter...	Filter...	Filter...	Filter...
108	pin	Password123	7	FALSE
213	tester	password	9	FALSE
112	Alex	password	6	TRUE
117	Matilde154	password	8	TRUE
214	test2	password	10	FALSE
109	csr	password	8	FALSE
110	pm	password	9	FALSE
111	suspended_user	password	6	TRUE
107	admin	password	6	FALSE
113	Declan940	password	7	FALSE
114	Hazle_Gibson1	password	6	FALSE
115	Estrella.Corwin362	password	6	TRUE
116	Mireya953	password	7	TRUE
118	Dahlia_Weimann5	password	9	TRUE
119	Muhammad_Armstron...	password	6	FALSE
120	Fletcher757	password	9	FALSE
121	Lucious_Quigley-Mann8	password	8	FALSE
122	Dean.McClure609	password	9	TRUE
123	Zachariah.Kilback4810	password	7	TRUE
124	Valentin6111	password	7	TRUE
125	Yasmin1512	password	9	TRUE
126	Jaren2913	password	7	FALSE
127	Estel7114	password	7	TRUE
128	Erick_Abbott15	password	7	FALSE
129	Etha_Leffler16	password	6	TRUE
130	Hobart8817	password	6	FALSE
131	Jaida.Goyette218	password	9	FALSE
132	Rhett.Wyman4419	password	6	FALSE
133	Gladys_Wehner20	password	6	FALSE
134	Samantha8221	password	9	FALSE
135	Thaddeus1122	password	7	TRUE
136	Toney.Cruickshank9023	password	7	TRUE
137	Lionel_Fay5924	password	9	FALSE
138	Else_Towne25	password	6	FALSE
139	Tiana.Dare26	password	6	TRUE
140	Breanne_Murazik1327	password	8	FALSE
141	Cruz_Metz28	password	9	TRUE
142	Lurline_Rempel29	password	7	TRUE
143	Brennan8430	password	9	TRUE
144	Furman7131	password	7	TRUE
145	Emelie_Ferry32	password	6	FALSE

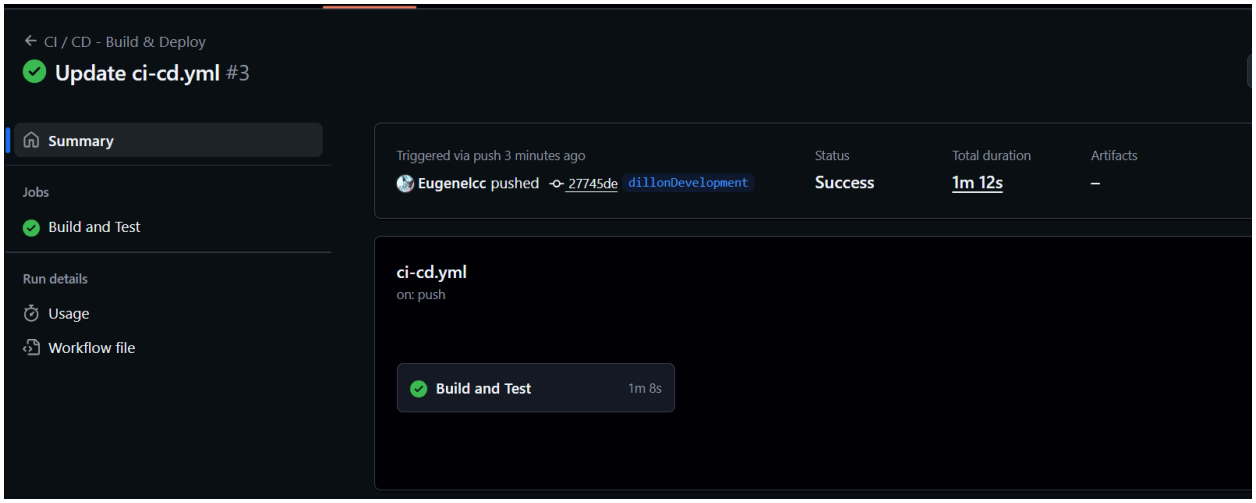
CONSOLE RE-RUN QUERY EXPORT OPEN

10. CI/CD

Gitlab.com was used for CI/CD. Development team used Gitlab git repo to exercise version control. Any approved push made to the main branch would trigger the pipeline to start. This fulfills CI. At the end of the pipeline, the app would be automatically deployed to dockerhub. This fulfills the CD.



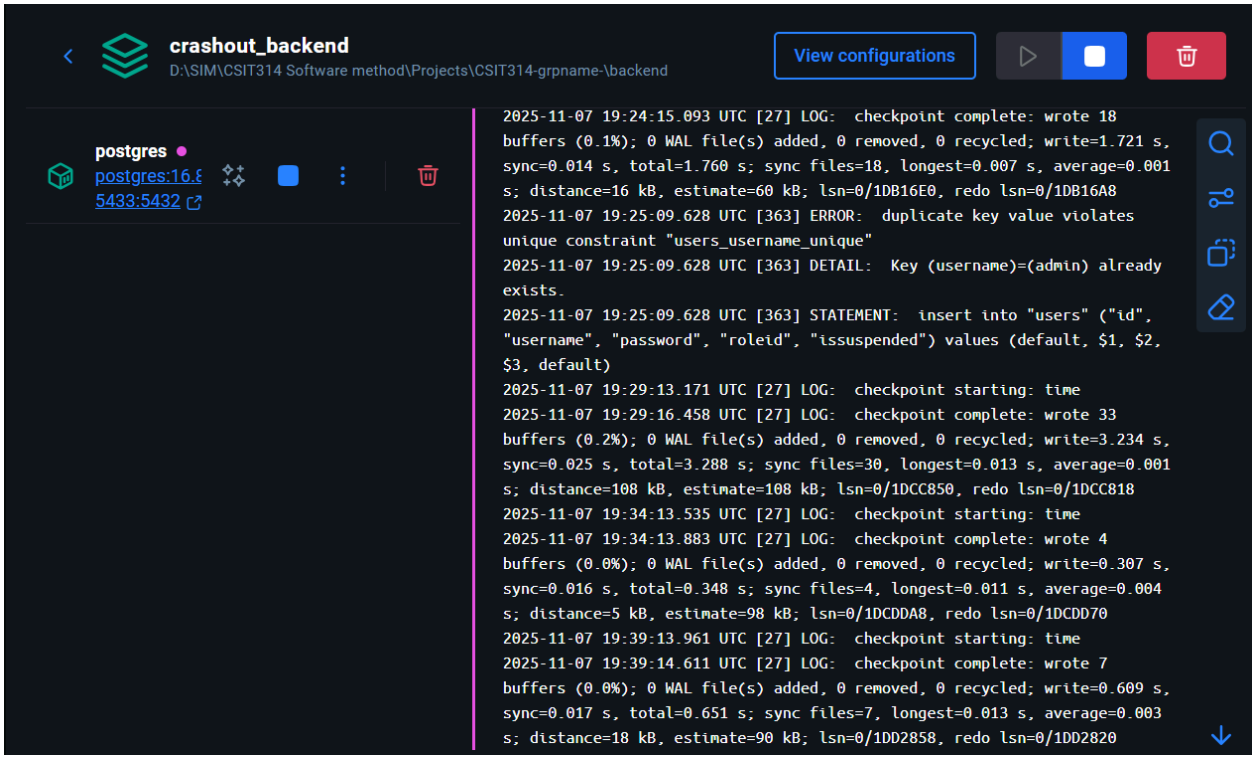




For CI/CD, we are utilizing GitHub for its comprehensive support for version control, collaboration, and automated CI/CD workflows. Within our project, GitHub allows developers to store code, manage branches, and initiate pull requests, all of which streamline team collaboration. Using GitHub Actions, we have set up automated workflows for continuous integration, which runs tests and enforces code quality checks each time new code is pushed or a pull request is made.

In terms of continuous deployment, GitHub Actions also supports deploying code changes to staging or production environments seamlessly, ensuring our cleaning services system remains up to date with minimal manual intervention. Additionally, GitHub's integration capabilities allow us to connect with monitoring tools, providing real-time insights into

application health and performance, which contributes to a smooth and reliable development cycle.



The PostgreSQL database is build and deployed into the docker container Within the pipeline, Docker ensures that each environment (testing, staging, or production) runs an identical and isolated PostgreSQL instance removing the need for manual setup.

Commits

main ▾ | All users ▾ | All time ▾

Commits on Nov 7, 2025

Merge pull request #4 from Cakebut/dillonDevelopment Eugenelcc authored 3 hours ago	Verified	6bda86a
Fix PM Page Cakebut committed 3 hours ago		02d2843
Pushh the UserAdminController Eugenelcc committed 3 hours ago		8f53a38
Fix the userAdmin Eugenelcc committed 3 hours ago		6b9f225
Remove the Cancel Button in CSR Offers Cakebut committed 5 hours ago		1ec2b29

Commits on Nov 6, 2025

Final fixes ver 1 Cakebut committed 5 hours ago		00a6f73
Merge pull request #3 from Eugenelcc/dillonDevelopment Eugenelcc authored 7 hours ago	Verified	08527e3
Push test Eugenelcc committed 7 hours ago		6b5fd66
PUSH test Eugenelcc committed 7 hours ago		e1907e3
update pm Eugenelcc committed 10 hours ago		21d5061
update seed and delete data Eugenelcc committed 15 hours ago		efb2609
Display page ui Cakebut committed 15 hours ago		16872c7
update UserAdminController Eugenelcc committed 16 hours ago		05c4fa2
Merge branch 'dillonDevelopment' of https://github.com/Cakebut/CSIT314Crashout into dillonDevelopment Eugenelcc committed yesterday		3564a84
Improve ProtectedRoute and addRequestTable to the DeleteData Eugenelcc committed yesterday		fcfea34
Feedback Feature Done Cakebut committed yesterday		bcdf50b
add protected role Eugenelcc committed yesterday		d20e91a
added delete password reqest btn Eugenelcc committed yesterday		2b89b55
Added Reset password features Eugenelcc committed yesterday		0e30b42

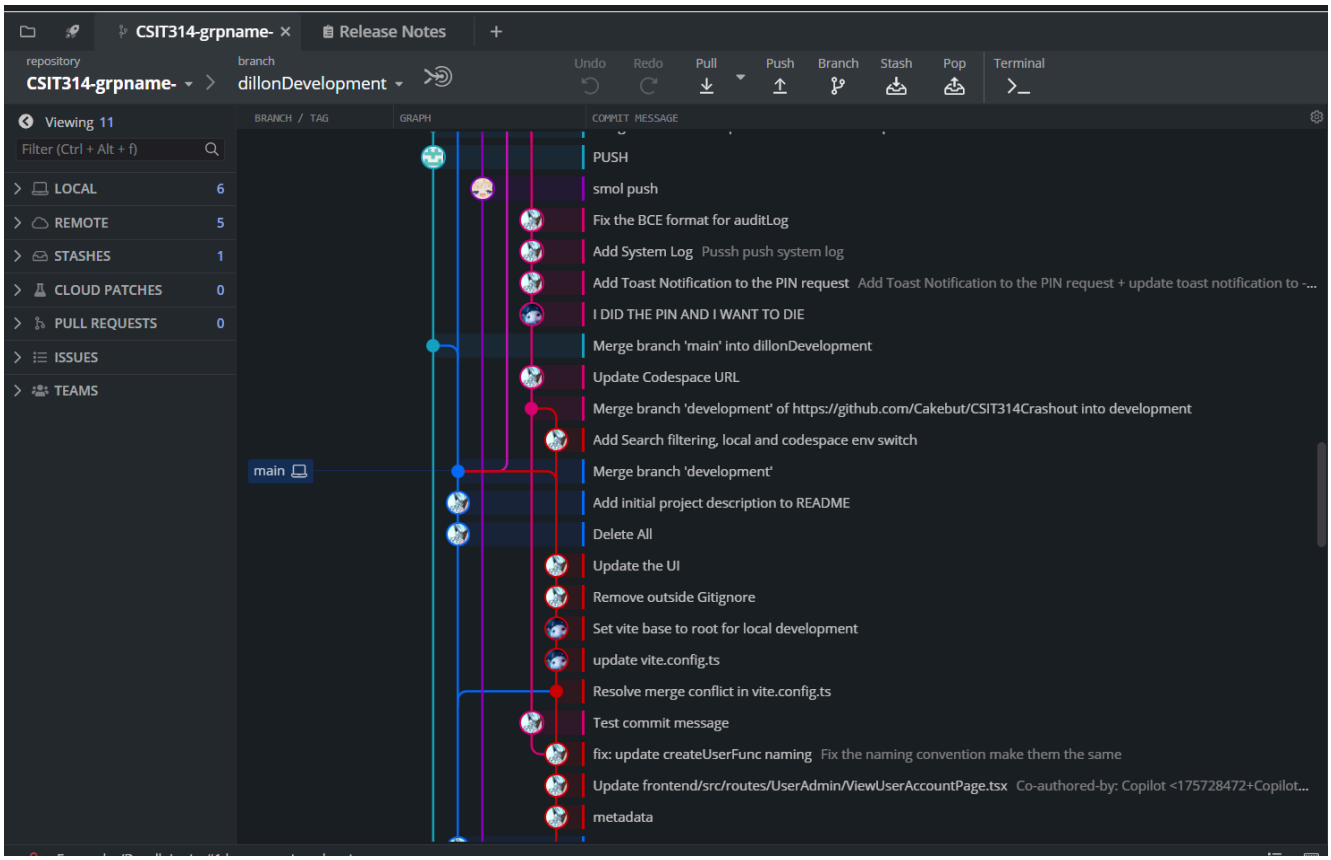
Commits on Nov 5, 2025

add stuff Eugenelcc committed yesterday	fe223e7
Merge branch 'dillonDevelopment' of https://github.com/Cakebut/CSIT314Crashout into dillonDevelopment Cakebut committed yesterday	8fe58ac
Feedback Feature Done Cakebut committed yesterday	8e67fd3
add ForgetPassword Page Eugenelcc committed yesterday	e9635a4
Update placement Eugenelcc committed yesterday	ecbccdd
Fix Past Download History Cakebut committed yesterday	d84d724
update exportName Eugenelcc committed yesterday	6c0d253
change filename UserAdminSystemLogPage Eugenelcc committed yesterday	1a9dac3
Create HandleExportUserDataclick () for user data export Eugenelcc committed yesterday	7868437
Completed Button Cakebut committed yesterday	1f0fc5c
Clean up the exportUserAccountController() Eugenelcc committed yesterday	e1e15d9
Add class AuditLogs in /entities/auditLog Eugenelcc committed yesterday	1b6d895
FIXED INTERACTION FOR CSR AND PIN Cakebut committed yesterday	c20d8e6
PIN Accept and Reject Offers Cakebut committed 2 days ago	1b81425

Commits on Nov 4, 2025

PUSH TEST.js Eugenelcc committed 2 days ago	95921c7
Updated View Count & Shortlist Count Cakebut committed 3 days ago	c3dc672

Previous Next >



The screenshot above displays the commit history in our GitHub repository, showcasing the version control process in action. This commit history offers a transparent view into each code update and refinement, reinforcing both team collaboration and code accountability throughout development. It shows what was changed, down to the number of lines that was added/removed.

11. Data-Driven Development Plan

11.1. The Model Requirement

After analysing the functionality and features, we have pinpointed an opportunity we could improve the experience of our users (Person in Need (PIN) and CSR) such as implementing artificial intelligence or machine learning technology into our system.

1. Recommendation Trend:
   We could use a machine learning algorithm to recommend the top most suitable PIN Request to our CSR Representative based on their past service history, location and how frequently the CSR is matched to that particular pin.

2. Category Trend:
   The System could implement machine learning to analyse and predict a service category popularity by tracking the amount of view frequency for the specific category(such as medical transport, household help and so on). Enabling a data driven resource allocation and targeted CSR recruitment in high demand areas

11.2. Data Collection

In order to build these models, we would require the following dataset:

- PIN Request details :
  1. service type, location, urgency level, request description,
  2. views and shortlist count
  3. Date Request created and status update
- CSR Request details
  1. CSR profile(service category offered, availability, rating and feedback), and service completion record
  2. Service completion records (total completed)
  3. Performance ratings and feedback from PINs

These datasets are crucial for accurately capturing the user's preference and performance on the platform. Additional user actions such as request views and shortlisting behavior can also be tracked in order to strengthen the data model and reveal deeper matching patterns

11.3. Data Cleaning

In order to make sure that our platform has reliable and meaningful recommendations for users, inaccurate data must be identified and removed, it includes

1. Removal of duplicate, noisy, inconsistent records or outliers
2. Handling of missing data through deletion / estimation via interpolation
3. Correction of spelling mistakes, data formatting
4. Handling outliers (Unusually high view counts,suspicious rating patterns)

A clean data reduces noise during training, leading to a quicker convergence of the machine learning algorithms and improved recommendation precision.

11.4. Data Labelling

User interactions and outcomes should be labeled to help the system learn matching patterns

- Positive Match Label:
    1. "Successful match" - CSR offered help, PIN accepted, service completed with high rating (4-5 stars)
    2. "Repeat collaboration" - CSR and PIN successfully matched multiple times
    3. "High-demand category" - Service category with consistently high view frequency and completion rates
- Negative Match Label:
    1. "Offer rejected" - PIN declined the CSR's volunteer offer
    2. “Low satisfaction" - Service completed with low rating (1-2 stars)
    3. "Low-demand category" - Service category with declining views or poor completion rates
- Trend Label:
    1. "Rising trend" - Category showing increasing view frequency over time
    2. "Declining trend" - Category showing decreasing interest or engagement

These labels would enable the supervised learning algorithm to distinguish between successful and unsuccessful matches,as well as identify category trends for strategic planning

11.5. Feature Engineering

To ensure that raw data is transformed into meaningful inputs for the recommendation model, we identify key relevant attributes

CSR Feature:

- Average performance rating across all completed services
- The location of the PIN request
- The service category of the pin request

PIN Request Feature:

- Request engagement score (combination of views and shortlists)
- Urgency level indicator
- Date of when the PIN request was created
- Category popularity score

Category Trend Feature:

- The rate of the changes in the view frequency
- Location demand distribution (The demand of the category on a specific location)
- Cross-category correlation (relationships between related service types)

Feature engineering assists in reducing dimensionality by combining related attributes. For example, total request views and total shortlists can be merged into a single engagement score to avoid duplication. Service category matches between PIN needs and CSR specialization can be calculated as compatibility scores ranging from 0 to 1. Category trend features can be aggregated into a single "demand forecast score" predicting future popularity.

11.6. Model Training

In order to prepare and train the model to make an accurate recommendations and prediction, the labeled dataset is used for the system to learn patterns in successful matches and urgency status:

PIN Request Recommendation Model:

- Train using collaborative filtering or content-based filtering algorithms
- Learn which types of PIN requests each CSR is most likely to successfully complete based on historical data
- Optimize for match success rate and service completion

Category Trend Prediction Model:

- Train using time-series analysis or regression algorithms to forecast future category demand
- Learn seasonal patterns, day-of-week effects, and long-term trends in service category popularity
- Optimize for match success rate, service completion rate, and user satisfaction scores

Training Process:

1. Split dataset into training (70%), validation (15%), and test (15%) sets
2. Use cross-validation to prevent overfitting
3. Apply hyperparameter tuning to optimize model performance
4. Iterate on feature selection based on validation results

The trained model will output:

- Ranked list of recommended PIN requests for each CSR Representative with confidence scores
- Category demand forecasts for the next 7, 30, and 90 days
- Alert notifications for rapidly rising or declining category trends

11.7. Model Evaluation

Over time, it is crucial that the model is able to measure how well the trained models performed in recommending the suitable PIN request to the respective CSR Representatives. Using test datasets(unseen dataset), the models should be evaluated on:

PIN Request Recommendation Metrics:

- Precision@K: Accuracy of top K recommendations (e.g., how many of the top 5 recommended requests are actually accepted)
- Recall: Percentage of successful matches correctly identified by the model
- F1-Score: Harmonic mean of precision and recall, balancing both metrics
- AUC-ROC: Area under the receiver operating characteristic curve, measuring classification effectiveness
- Mean Average Precision (MAP): Overall ranking quality of recommendations

Category Metrics:

- Mean Absolute Error (MAE): Average deviation between predicted and actual view counts
- Root Mean Squared Error (RMSE): Standard deviation of prediction errors
- Trend Direction Accuracy: Percentage of correctly predicted rising/declining trends
- Lead Time: How far in advance the model can accurately predict trend changes

Business Metrics:

- Recommendation acceptance rate (percentage of recommended requests that CSRs accept)
- Reduction in time-to-match (how quickly PINs receive offers after posting)
- User satisfaction improvement (The changes in the average rating after the model has been deployed)
- CSR Representative efficiency

The models should predict optimal matches and accurate category trends while minimizing false recommendations and prediction errors.

11.8. Model Deployment

Once the model have passed evaluation criteria , they can be deployed into production environments:

Deployment Architecture:

- Expose the models through RESTful APIs
- Integrate with the CSR Representative dashboard to display personalized recommendations

User Interface Integration For the CSR:

- Display "Recommended for You" section on dashboard
- Show compatibility scores and match reasons ("High success rate in your area")
- Highlight trending service categories relevant to their expertise

11.9. Model Monitoring

In order to monitor the deployed models to ensure they continue to perform accurately , we can track key performance

Performance Tracking:

- The recommendation acceptance rate
- The Average matching type
- The CSR rating score
- Category prediction accuracy

12.Ethical Considerations

12.1 Process

12.1.1 Accountability

To ensure accountability through to the development of our project, Our team recognises the responsibilities associated with assigned tasks and acknowledges them made by each of the team members. We are committed in identifying and addressing potential risk that could occur during the development phase of the project

12.1.2 Fairness

The principle of fairness is upheld by ensuring that each team member has equal treatment. We have ensured that all our members are present for all the meetings that we have conducted. When some of the team members were absent, we updated the absent team-members with the discussions that we had via our meeting minutes. We have taken meeting minutes for all our meetings, so that everyone is updated with what's going on.

Once the tasks were delegated to all the team members, we set appropriate deadlines for ourselves, after discussing with one another thoroughly during the meetings, so that we get to review our taskings by the end of the deadline. To ensure that every team member is updated on the progress of the project, we have created a group chat so that each team member is updated on the tasks that are being done, the tasks that require attention, as well as the tasks that need to be done.

All of these have been evident in our task delegation that we have included at the beginning of our report.

12.2 Product

12.2.1 Privacy & Security

Our product ensures that each role that is assigned to each of the user can only Access their respective function (for example PIN can only access the pin dashboard but not the Admin dashboard). We also ensure that the data of our user (PIN, CSR) is protected from unauthorized access which is achieved by anonymising sensitive data

12.2.1 Reliability and Safety

Quality Assurancy: We ensure that our platform is up to standard instead of it focusing on having a bunch of functionality, we emphasis on quality, consistency, and rigorous training,integration testing to enhance our user experience

Error Prevention and User Safety: The platform incorporates robust validation, error handling, and user-friendly prompts to prevent actions that could compromise data integrity or system stability.

12.2.3 Transparency

The Users have the right to understand what data is being collected and how it is used. To ensure transparency, the platform provides a clear and accessible privacy policy that explains data collection practices, usage purposes, storage duration, and user rights regarding their personal information.