CollabTrack

Project: Design

COSC4320

Jonathan Onwusa, Uche Onwe, Michael Arodiogbu

Table of contents

Abstract (100–150 words)

CollabTrack is a lightweight organizer for individuals and small teams to plan projects, manage tasks, and finish work on time. The MVP delivers secure user accounts, project spaces (personal or shared), and task management with due dates, priority, and status. A project-level activity log and comments keep decisions close to the work. The system is intentionally simple—Node.js + Express with EJS views and SQLite—to maximize reliability and speed of delivery this term, while preserving a clean migration path to React/PostgreSQL later. Optional reminders and email notifications are scoped as stretch features. The prototype demonstrates end-to-end flows from sign-up to project completion and emphasizes a "today-first" dashboard that highlights due and at-risk items. CollabTrack targets students and small clubs needing a fast, low-friction way to coordinate and complete tasks.

Introduction (≈300–500 words)

The goal of CollabTrack is to provide a focused alternative to heavyweight project suites by emphasizing task capture, deadlines, and progress visibility. Our primary users are students and small teams who need to organize deliverables without complex configuration. After completing the specification phase, we refined scope to ensure a demonstrable MVP: secure authentication; project creation (personal or shared by invite); tasks with title, description, assignee, due date, priority, and status; and a simple comments/activity log to reduce context switching.

Key changes since specification:

- Clarified the stack to **Node.js + Express + EJS + SQLite** for speed and stability; added an explicit repository/DAO layer to enable future migration to PostgreSQL and/or a React SPA without rewriting core business logic.

- Tightened usability goals: (1) capture a task in under 10 seconds, (2) surface "today" and overdue items on the dashboard, (3) keep decisions with tasks via comments/activity, and (4)

provide low-friction collaboration through invite-only sharing.

- Defined boundaries for stretch items (email reminders, calendar export) to avoid feature creep.

- Added testing and formatting expectations (Jest, Supertest, ESLint, Prettier) to support team collaboration and maintainability.

From a process perspective we will develop in small vertical slices—e.g., “create task” includes route, validation, DB persistence, view updates, and an activity-log entry—so that each increment is demo-ready. Risks include scope creep, auth/security mistakes, and under-designed data relationships. We mitigate these by enforcing a minimal feature set, hashing passwords with bcrypt, isolating JWT handling, and documenting DB constraints and access patterns early in this design.

This document details the architecture, program units, database schema, and the behavior of non-trivial operations through sequence diagrams. It also outlines the user interface snapshots required for the demo, team contributions, and the updated glossary.

3. High-level and medium-level design

3.1 System-level architecture (pattern)

Pattern: Layered MVC on Express

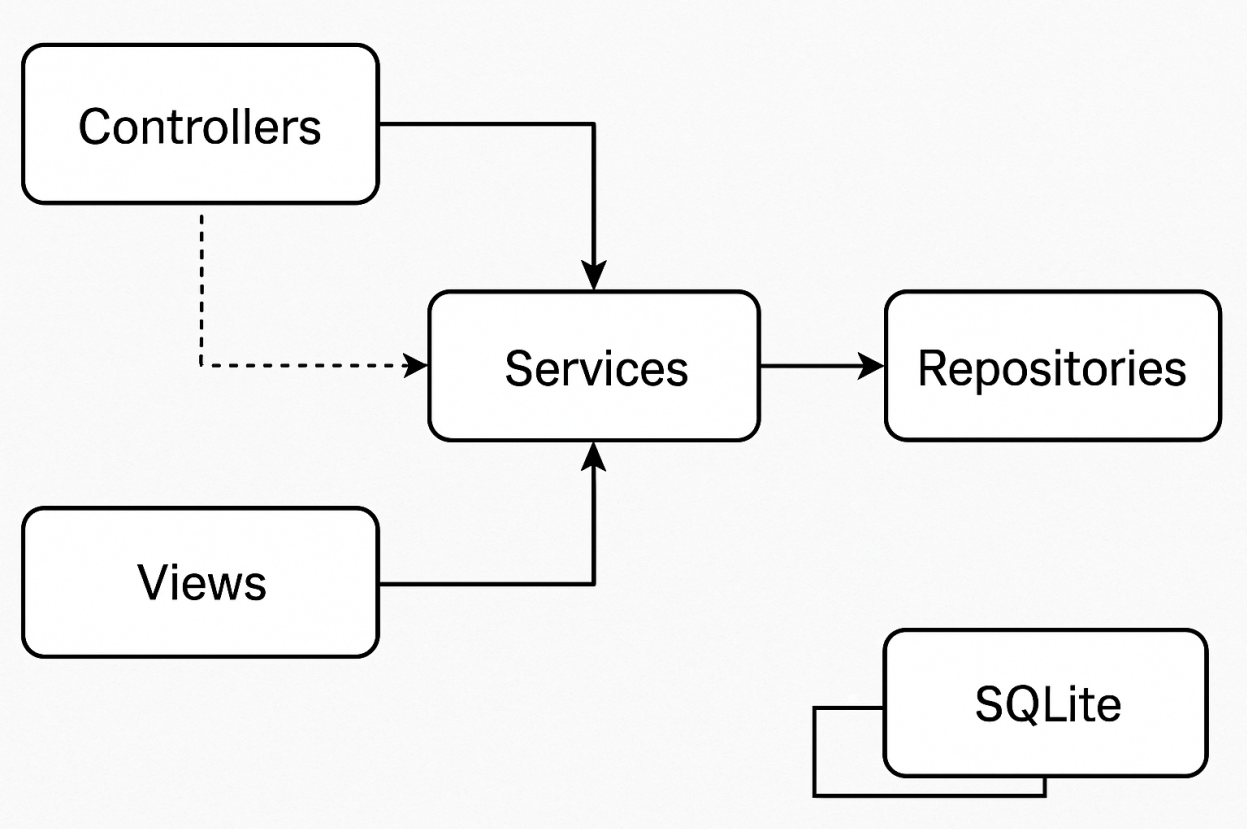
- **Presentation (Views):** EJS templates render server-side HTML for auth, dashboard, projects, and tasks.

- **Controllers (Routes):** Express route handlers validate input, orchestrate services, and choose views or JSON responses.

- **Services (Domain logic):** Auth, Project, Task, Comment, Reminder encapsulate business rules and permissions.

- **Data Access (Repository/DAO):** SQLite queries are centralized so the DB can be swapped later without changing services.

- **Infrastructure:** JWT auth middleware, error handling, logging, optional schedulers (node-cron) and email (Nodemailer).



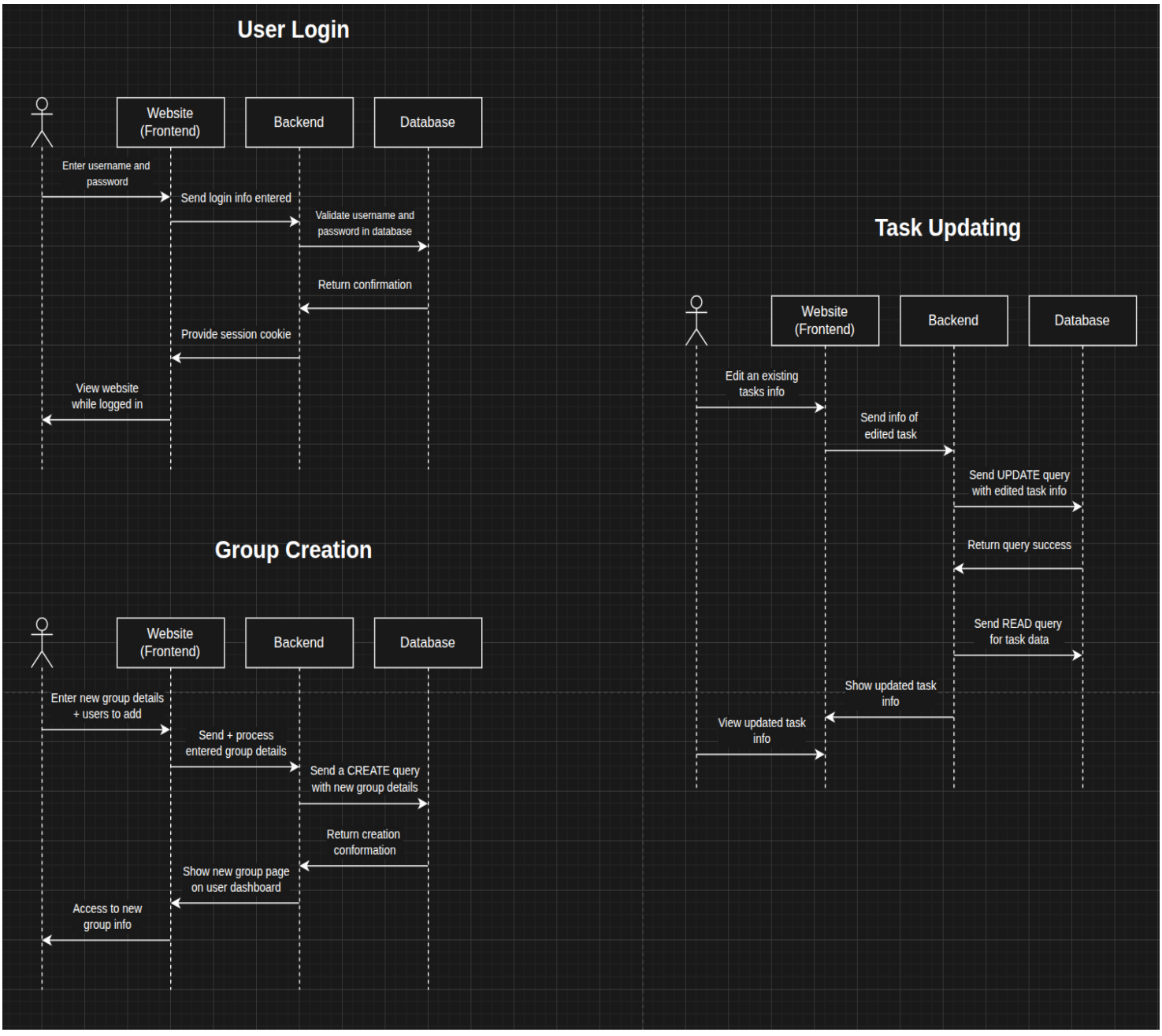
3.2 Program units (classes/modules)

Below are core units (at least 5 are required; we list 8). For each: purpose, key methods, notable constraints.

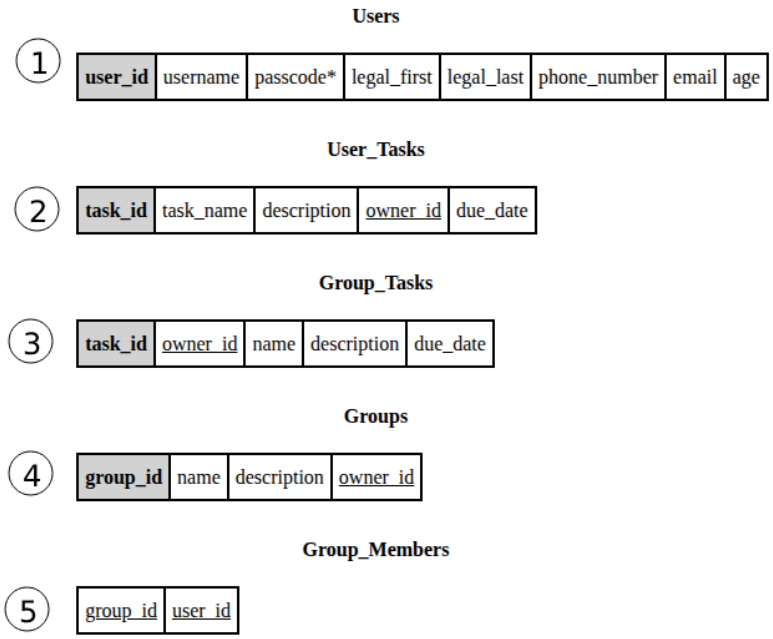
1. **AuthService**
    - **Purpose:** Registration, login, JWT issuance/verification, password hashing.
    - **Key methods:** register(dto), login(credentials), verifyToken(token).
    - **Constraints:** Passwords hashed with bcrypt; emails unique; tokens expire.
2. **UserRepository**
    - **Purpose:** CRUD for users.
    - **Key methods:** create, findByEmail, findById, update.
    - **Constraints:** email unique; created_at immutable.
3. **ProjectService**
    - **Purpose:** Create projects, manage membership, enforce access control.
    - **Key methods:** create, addMember(projectId, userId, role), listForUser.
    - **Constraints:** Owners can invite; members read/write; private by default.
4. **ProjectRepository**
    - **Purpose:** DB access for projects and membership.
    - **Key methods:** insertProject, insertMember, findById, findUserProjects.

5. **TaskService**
    - **Purpose:** CRUD tasks; status transitions; due-date and priority rules.
    - **Key methods:** create, update, changeStatus, assign.
    - **Constraints:** due_date optional; status in {todo, doing, done}; permission checks per membership.
6. **TaskRepository**
    - **Purpose:** DB access for tasks.
    - **Key methods:** insert, update, findById, listByProject.
7. **CommentService**
    - **Purpose:** Add and fetch task/project comments; append activity log entries.
    - **Key methods:** addComment, listForTask.
    - **Constraints:** Sanitization; references valid task/project; retains author.
8. **ActivityLogRepository**
    - **Purpose:** Append/read immutable activity entries.
    - **Key methods:** append(event), listForProject.
    - **Constraints:** Append-only; timestamps required; event types enumerated.

Sequence Diagrams



Database Tables



1. **User Table**

- *(passcode will be stored as a hash + salted value)

- Primary Keys is “user_id”

2. **User_Tasks**

- Holds tasks that belong to a specific user

- Primary Key is “task_id”

- “owner_id” is a foreign key (Value will be equal to a user_id) value

3. **Group_Tasks**

- Holds tasks that belong to groups

- Primary Key is “task_id”

- “owner_id” is a foreign key (Value will be equal to a group_id value)

4. **Groups**

- Holds list of existing groups for the application

- Primary Key is “group_id”

- “owner_id” is a foreign key (Value will be equal to a user_Id value)

5. **Group_Members**

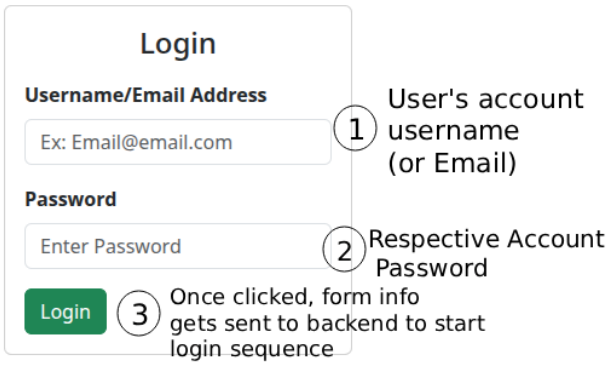
- Bridge table

- ‘group_id’ and ‘user_id’ are foreign keys

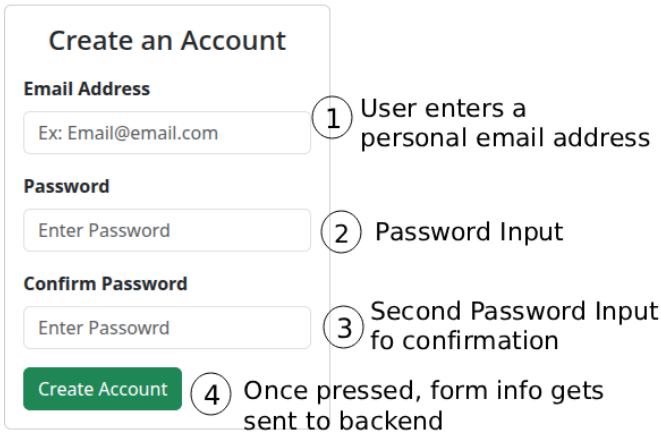
User Interface

(UI images are shrunken to fit inside design document)

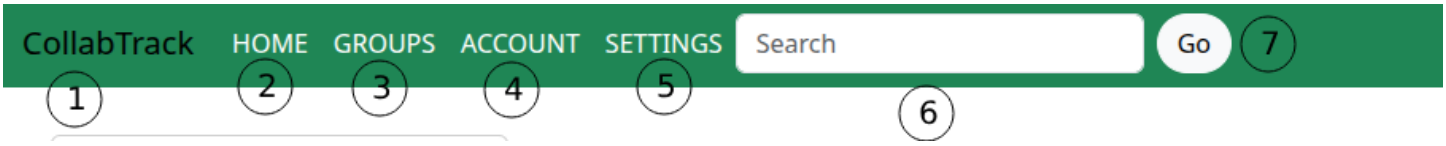
Login Page (Card)



Account Creation Page (Card)

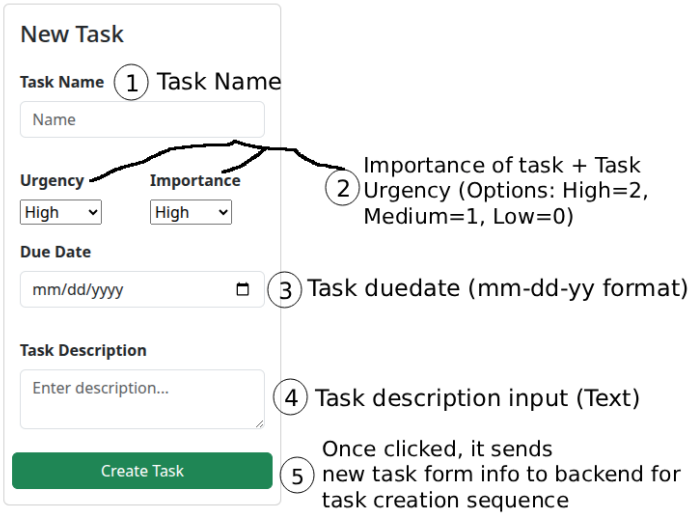


NavBar

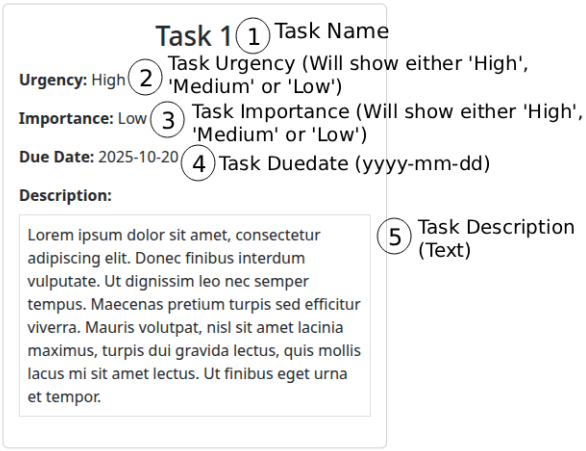


1. CollabTrack Site Name (This is also a button that takes user to the HOME UI page)
2. Once clicked takes user to HOME UI page where users can see their task calendar
3. Once clicked takes user to GROUPS UI page where users can see info on groups they are in
4. Once clicked takes user to ACCOUNT UI page where users can see their account info
5. Once clicked takes user to SETTINGS UI page where users can change various preferences
6. Search bar (takes in text) that user input usernames/account info to look for other accounts
7. Once clicked takes user to search results page based on search query

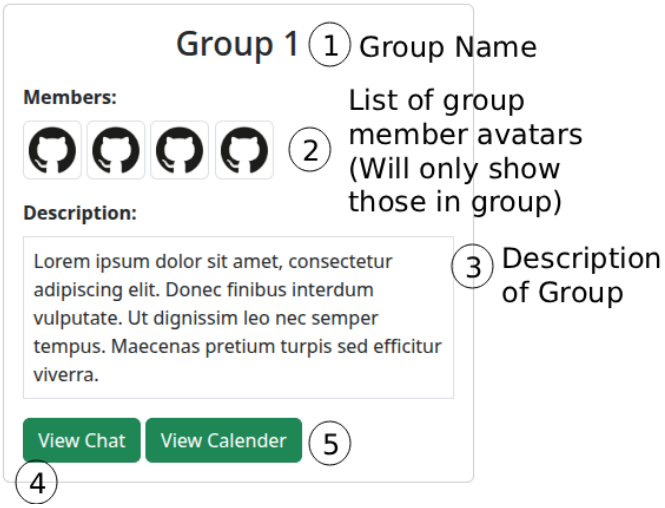
Task Creation Page (Card)



Task Info (Card)

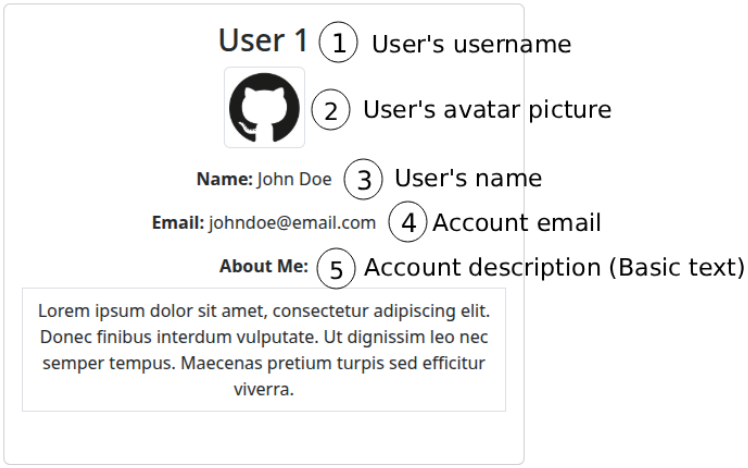


Group Info (Card)

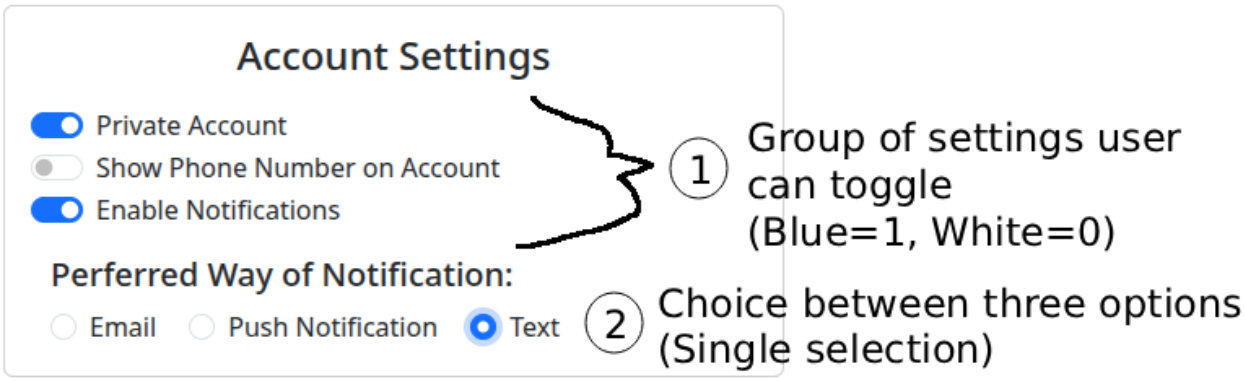


4. Once the ‘View Chat’ button is clicked, it takes user to group chat room UI page
5. Once the ‘View Calendar’ button is clicked, it takes user to group calendar UI page

User Account Info (Card)



Account Settings (Card)
(More settings to potentially be added in the future)



Member Contributions

Jonathan – Sequence Diagrams, Database Tables, UI Design and Snapshots. Worked on front-end and database design.

Michael - Built the frontend with EJS, HTML, and CSS; designed login, dashboard, and project/task pages; ensured usability and connected views to backend routes.

Uche - Built the backend logic with Node.js, and Front end UI design with bootstrap and Ejs. My part to work on is providing the snapshots of the UI.

Glossary Updates

No updates were made to the glossary.