Design Specification

1. Statement of Purpose

The Conductor app is a collaborative platform designed to help instructors, TAs, tutors, team leads, and students manage course-related activities like scheduling sessions, organizing groups, sharing materials, and tracking participation. The goal is to create a lightweight, user-friendly, and accessible system that is acceptable to all stakeholders. The Conductor app aims to reduce administrative overhead and improve transparency across teaching teams with the goal of allowing the class to run smoothly and giving staff more time to focus on teaching and mentorship.

2. User Personas

Instructor
The instructor oversees multiple classes and teams. They need an easy and structured way to track all course members, form groups efficiently, and monitor student progress. The system should simplify group creation, automate distribution and keep all records organized. The instructor also cares about the sustainability of the system and wants tools that are easy to maintain and extend in future quarters. They value simplicity, documentation, and minimal dependencies. Professor Powell has emphasized the importance of a lightweight and easy to deploy system with a simple click of a button.
Proposed Features
- Groups Manager: View and manage all groups and members
- Form Builder: Upload or edit initial student-group assignments
- Dashboard View: Basic overview showing number of groups, members, and attendance summaries
- Meeting Scheduler: For students to schedule meetings throughout the week or quarter

Teaching Assistant
Teaching assistants manage several student teams and need quick access to each group's information. They want to schedule meetings, record notes, and track student participation throughout the quarter. They also require a consistent evaluation process for fairness across TAs. The system should help them identify struggling groups, keep attendance logs and maintain records of interaction.
Proposed Features
- Group Overview Page: List of all assigned groups and their members
- Attendance Tracker: Record attendance and participation scores
- Group Notes: To record notes for each group
- Meeting Scheduler: For students to schedule meetings throughout the week or quarter

Tutor
Tutors provide on-demand help to students during labs and assignments. They need an organized queue system that manages student requests and maintains a FAQ that encourages self-service learning. Tutors should be able to promote common questions into the FAQ to reduce repeat issues and send feedback to instructors about confusing lab problems.
Proposed Features:
- Help Queue: Students submit help requests with their name and topic
- FAQ List: Simple searchable list of common issues

Team Lead
Team leaders are students who manage small projects groups. They need a central workspace to organize meetings, assign work, and monitor team performance. They rely on attendance tracking, sentiment notes, and workload summaries so everyone contributes fairly. They also need clear communication channels with TAs and professors to report concerns early. A dedicated dashboard could help them lead effectively while staying focused on coordination and team outcomes.
Proposed Features
- Team Dashboard: View team members and their attendance records
- Attendance Input: Mark attendance for each lecture or activity
- Private Notes: Field to record quick notes or flags about team dynamics

Student
Students work within teams and need transparency in how their participation and progress are tracked. They want to know expectations, share their availability, and record reflections about their experience. The system should help them stay informed, communicate their status, and feel fairly represented in evaluations.
Proposed Features
- Session View: See upcoming sessions and attendance history
- Status Update: Option to mark availability
- Private Journal: Simple text box to reflect on lecture material or team experience

3. Scope and Shaped Work

In Scope
- **User Authentication and Role-Based Access:** Secure login and dashboard access for instructors, teaching assistants, team leaders, tutors, and students, role management dashboard for administrators
- **Groups Management:** Ability for instructors and TAs to view, create, and manage student groups
- **Session and Attendance Tracking:** Team leaders and TAs can record attendance, while students can view their attendance history

- **Team Dashboards:** Basic pages displaying members, attendance, and meeting notes for each group
- **Database Integration:** Unified schema connecting users, groups, attendance, and journals
- **Testing and Deployment:** Unit tests for each feature and integration testing before the final demo
- **Chat feature:** Ability for instructors, TAs, tutors, and students to message each other
- **Class Discussion Board:** Post questions for students/instructors to answer. Instructors can also post announcements. (Imitating Piazza)

Out of Scope
- Automated grading or rubric generation
- Notification and reminder systems

4. Risks and Rabbit Holes

General

- Not following The Definition of Done
- Adding additional unnecessary features outside the planned/required scope
- Implementing too many basic features. Focus on quality rather than quantity
- Underestimating the time to implement the features (5 weeks)

Feature 1 - Manager / Authentication / Roles

- **Orphaned data**: An administrator deletes a role but forgets to remove all the corresponding rows in the **user_roles** table, leading to orphaned data..
- **Privilege escalation:**
    - A user finds a loophole (editing a URL or making a direct API call) to perform an Admin action.
    - A user holds two roles (Member and Admin), but the system only checks one of them, granting too much or too little access.
- **Token expiration/OAuth rabbit hole:** Google OAuth should be seamless and invisible to the user, which is a major potential rabbit hole.
    - When the access token stored in the user_auth table expires, it will cause the Google OAuth integration to break. Once an access_token expires, any API request the app sends (e.g., trying to pull the user's profile picture) will be rejected with a 401 error. For security reasons, these tokens have a short lifespan of 30 minutes to an hour. We can get around this by creating a

refresh_token and storing it in the user_auth table but that introduces a new security risk because the tokens may not expire for several years, which means we have to encrypt the refresh_token in the database.

- **Security**: If the access_token and refresh_token in user_auth are not encrypted, they can be stolen, giving an attacker permanent access to the user's Google account.
- **Database operation failure:** If a critical database operation fails, the corresponding activity log entry might also fail, creating a gap in the user's activity log.

Feature 2 - Class Directory

- Overly complex design: Adding unnecessary integrations or third-party tools may slow development and make the system harder to maintain over time.
- Integration challenges: Building the directory separately from other features could lead to inconsistencies when connecting shared APIs or UI components, requiring extra refactoring later.
- Accessibility and privacy risks: Missing or uneven attention to accessibility and data protection standards could cause compliance issues that need costly fixes near the end of the project.

Feature 3 - Attendance System:

- Data consistency: Attendance data not synced properly between frontend and backend
- Privacy and Access Control: Attendance data must only be visible to authorized roles
- Time zone: Class times stored or displayed inconsistently due to time zone issues
- Backend Reliability: Server outage or lost network during check-in window
- Possible future feature: GPS check-in

Feature 4 - Work Journal / Stand-up Tools:

- Low User Adoption: Overly complex or time-consuming input processes may discourage students from using the tool consistently
- About emotional sentiment: Improper visibility of emotional sentiments or blockers might discourage honest feedback from students
- Possible future feature: Provide a "private notes" option visible only to the author.
- Make the process easy to fill

5. Functional Requirements

Must be written in core platform tech for long term support

- Standard HTML
- Standard CSS
- Vanilla JS
- Web Components / HTMX / Framework choice
- Backend using NodeJS or PHP
- Express or Hono
- Postgres DB
- Authentication service (Auth.js)

6. Non-Functional Requirements

Accessibility and Internationalization

- Must be a11y friendly, especially for OSD students
- Must be internationalizable for ESL students

Performance and Scalability

- Must handle 500 - 1000 simultaneous users
- Must handle 10,000s of stored students over time
- Performance should be sufficient on a low-cost instance

Security and Privacy

- Must adhere to FERPA compliances and protect student data
- Must implement roles and permissions
- Must include logging and audit trails
- Must have security monitoring
- Must have UCSD authentication
- Must support SSL transmission and encryption at rest

Usability and Mobility

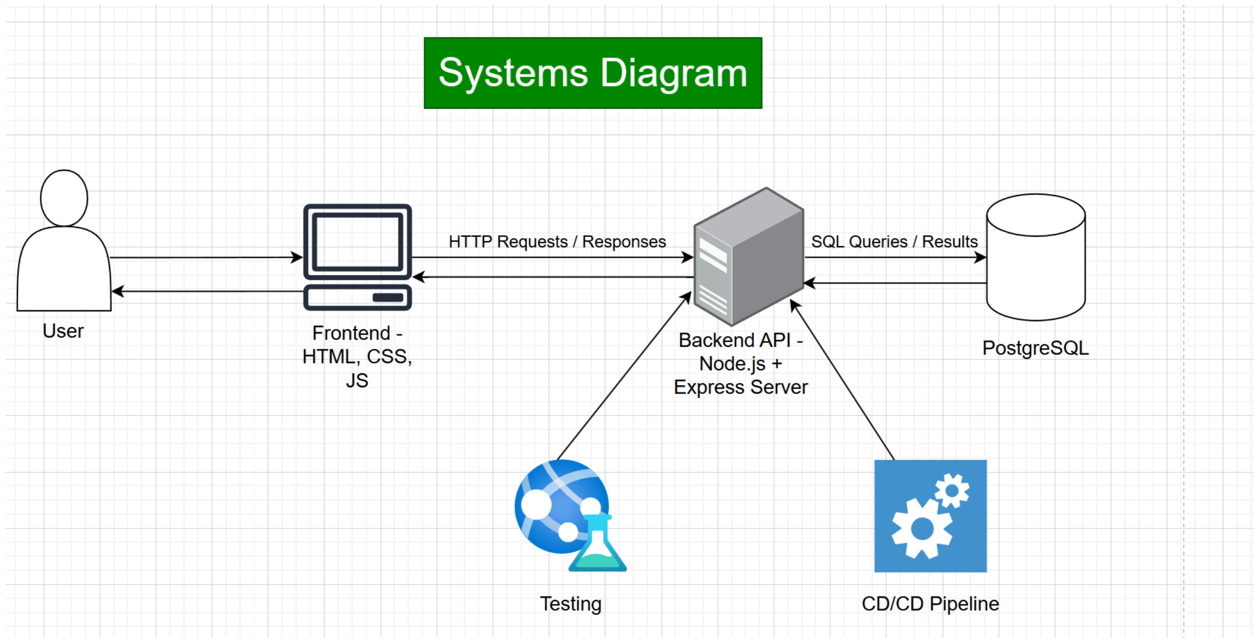
- Must be mobile friendly in both layout and performance

Research and Development Practices

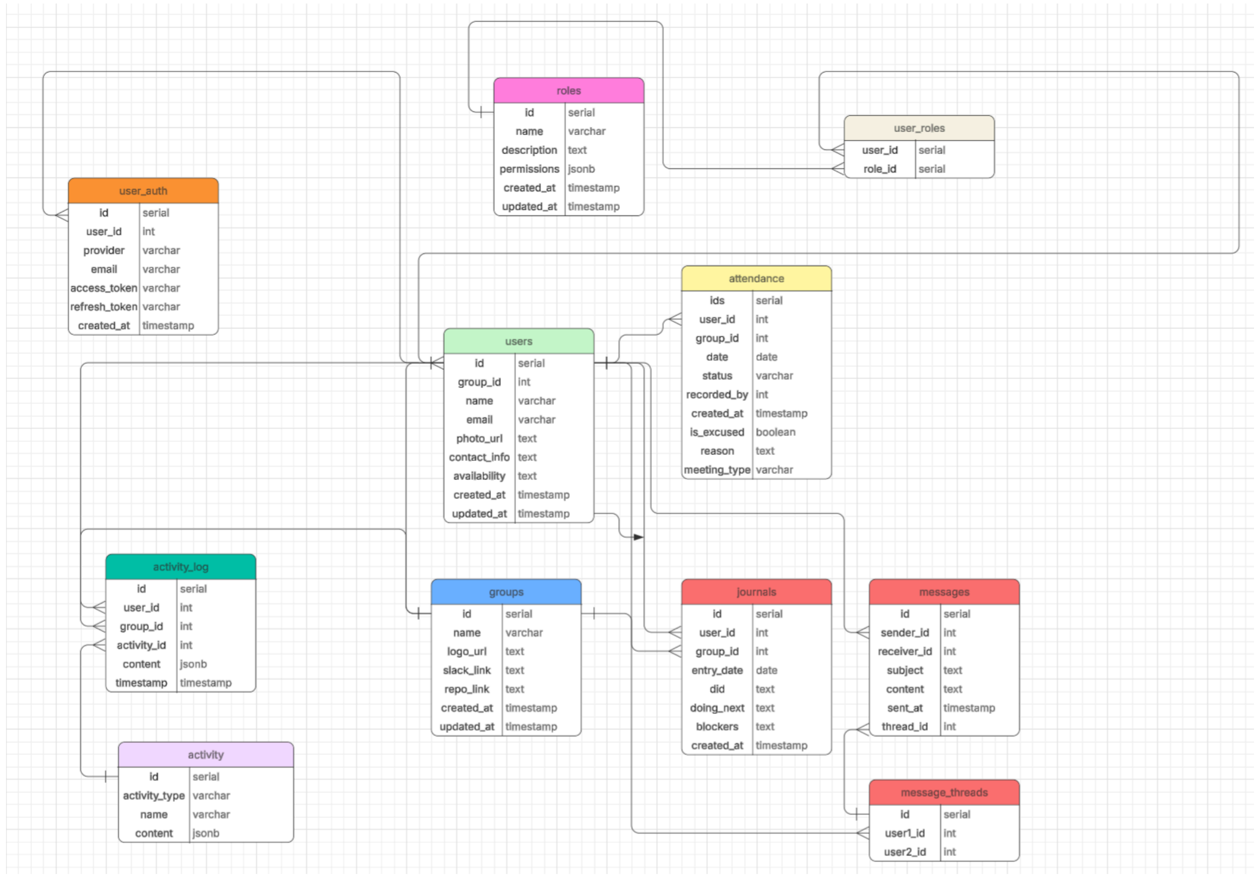
- Should explore working in the medium
- Should explore moldable and evolutionary development principles

7. System Overview Diagram

Systems Diagram



DB Mappings



roles	
id	serial
name	varchar
description	text
permissions	jsonb
created_at	timestamp
updated_at	timestamp

user_roles	
user_id	serial
role_id	serial

user_auth	
id	serial
user_id	int
provider	varchar
email	varchar
access_token	varchar
refresh_token	varchar
created_at	timestamp

attendance	
ids	serial
user_id	int
group_id	int
date	date
status	varchar
recorded_by	int
created_at	timestamp
is_excused	boolean
reason	text
meeting_type	varchar

users	
id	serial
group_id	int
name	varchar
email	varchar
photo_url	text
contact_info	text
availability	text
created_at	timestamp
updated_at	timestamp

activity_log	
id	serial
user_id	int
group_id	int
activity_id	int
content	jsonb
timestamp	timestamp

groups	
id	serial
name	varchar
logo_url	text
slack_link	text
repo_link	text
created_at	timestamp
updated_at	timestamp

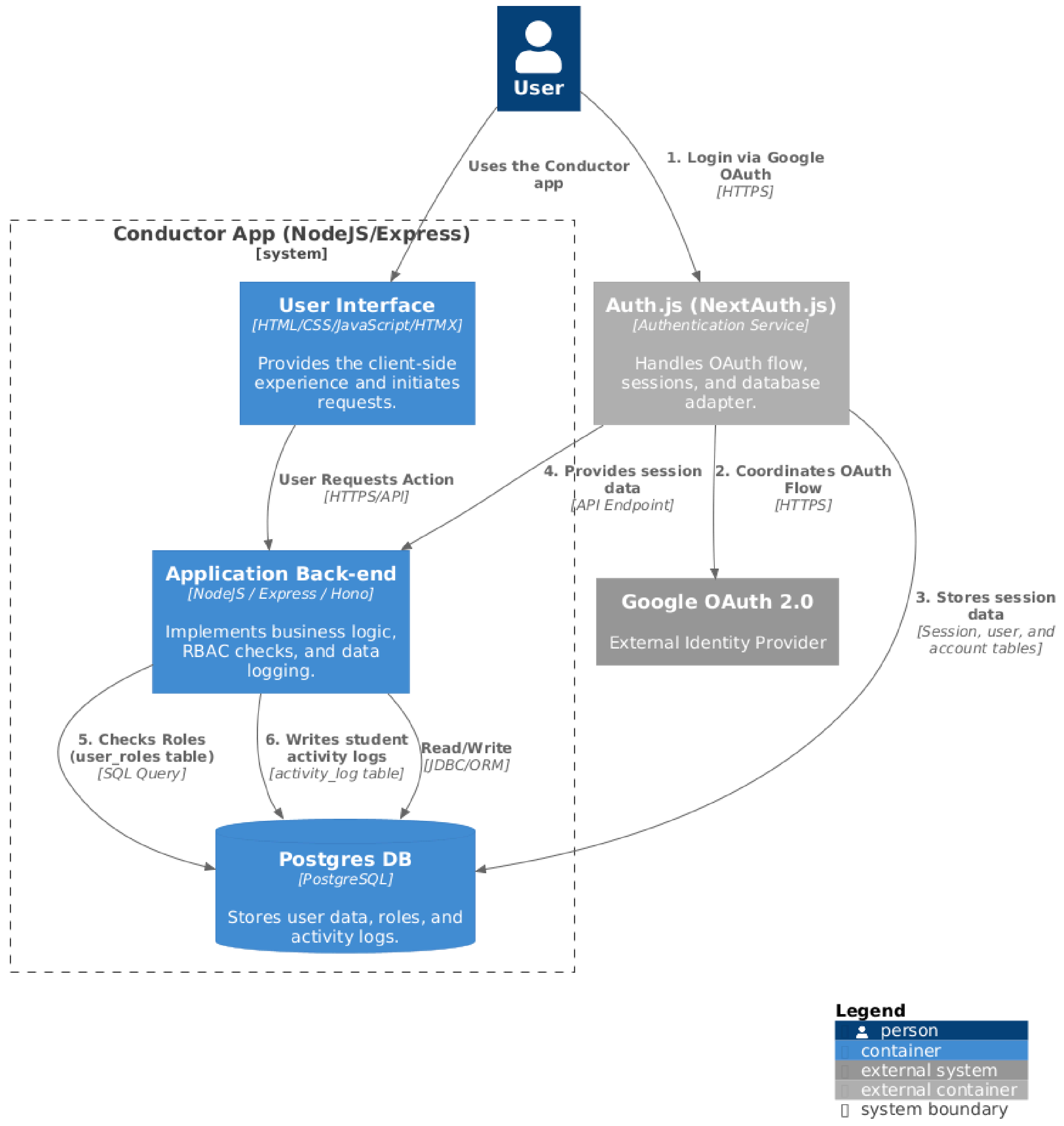
journals	
id	serial
user_id	int
group_id	int
entry_date	date
did	text
doing_next	text
blockers	text
created_at	timestamp

messages	
id	serial
sender_id	int
receiver_id	int
subject	text
content	text
sent_at	timestamp
thread_id	int

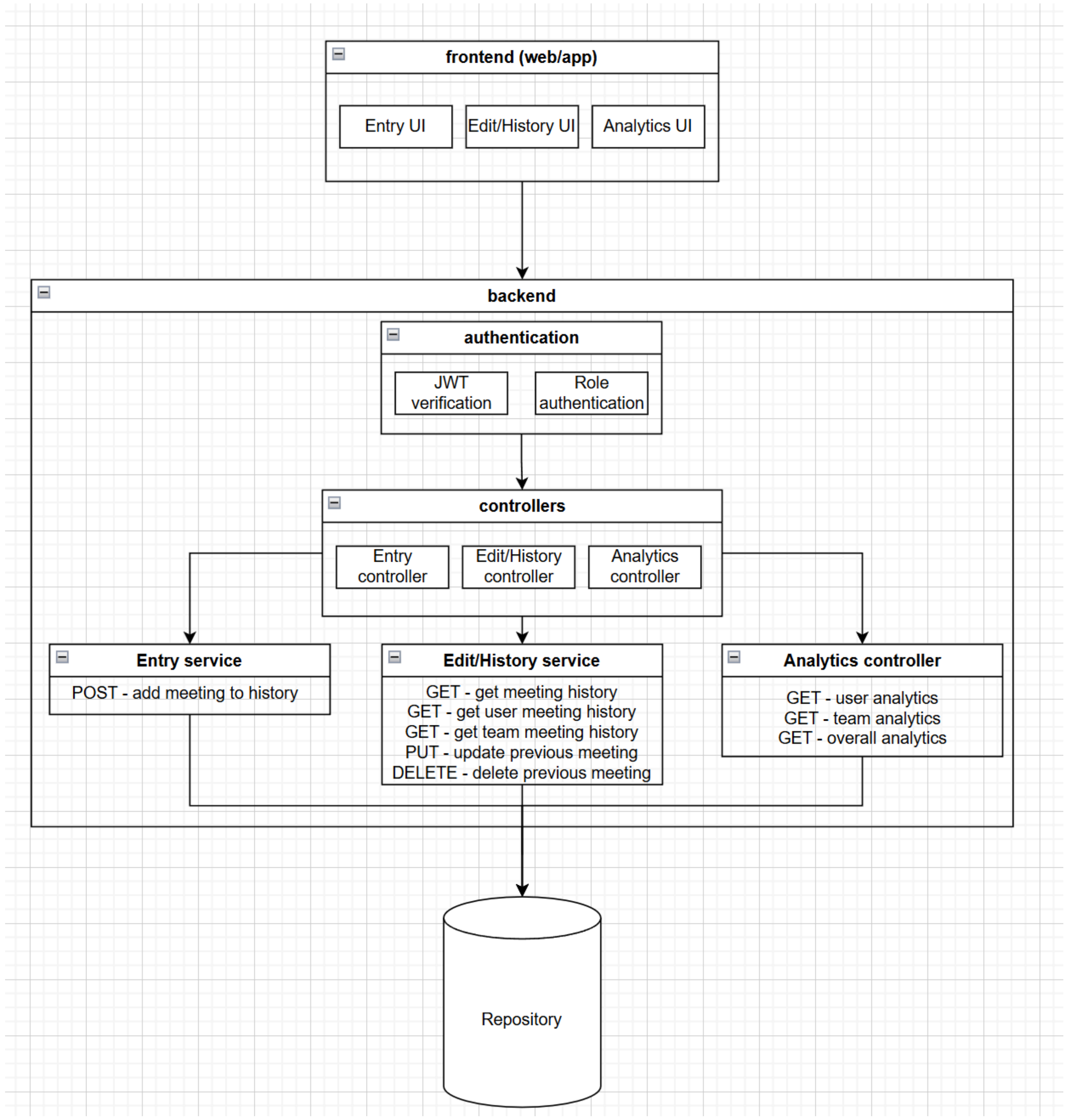
activity	
id	serial
activity_type	varchar
name	varchar
content	jsonb

message_threads	
id	serial
user1_id	int
user2_id	int

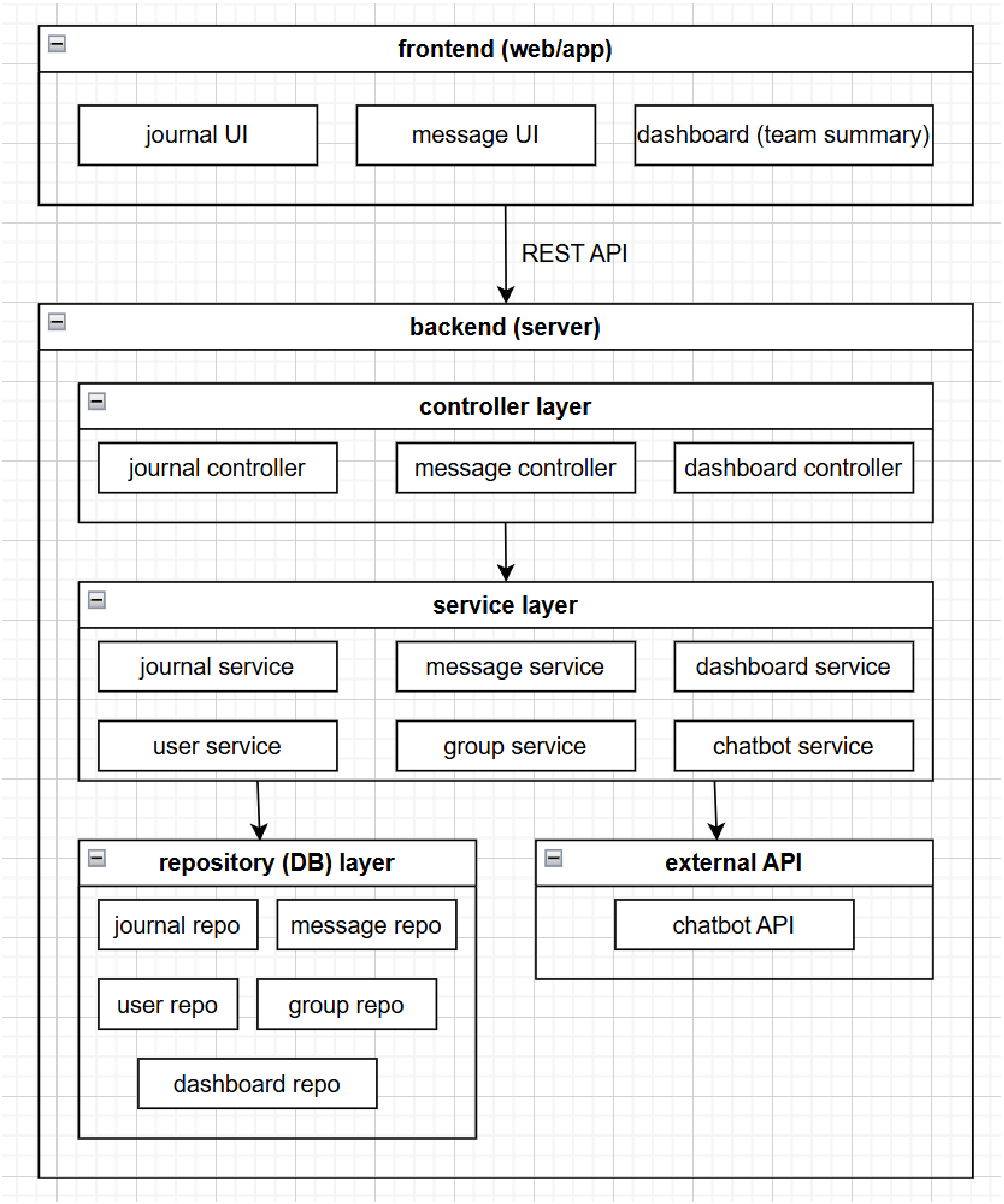
Feature 1: Manager / Authentication / Roles


User
Uses the Conductor app
1. Login via Google OAuth
[HTTPS]
Conductor App (NodeJS/Express)
[system]
User Interface
[HTML/CSS/JavaScript/HTMX]
Provides the client-side experience and initiates requests.
Auth.js (NextAuth.js)
[Authentication Service]
Handles OAuth flow, sessions, and database adapter.
User Requests Action
[HTTPS/API]
4. Provides session data
[API Endpoint]
2. Coordinates OAuth Flow
[HTTPS]
Application Back-end
[NodeJS / Express / Hono]
Implements business logic, RBAC checks, and data logging.
Google OAuth 2.0
External Identity Provider
3. Stores session data
[Session, user, and account tables]
5. Checks Roles (user_roles table)
[SQL Query]
6. Writes student activity logs
[activity_log table]
Read/Write
[JDBC/ORM]
Postgres DB
[PostgreSQL]
Stores user data, roles, and activity logs.
Legend
person
container
external system
external container
system boundary

Feature 2 & 3: Directory & Attendance System

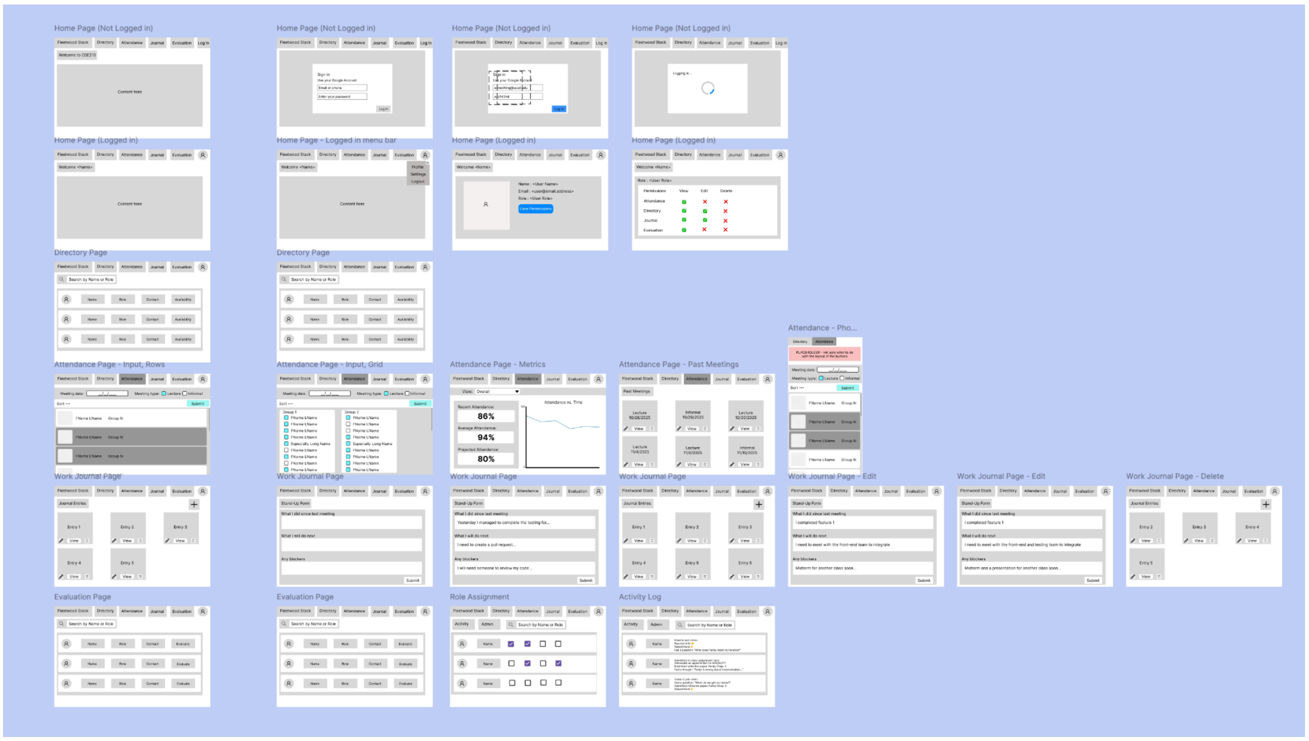


Feature 4: Work Journal / Stand Up Tool



8. Wireframes

Figma



9. Tools and Technologies

Frontend

- HTML and CSS
- Vanilla JavaScript
- Responsive design to support desktop and mobile
- ARIA and semantic markup for accessibility

Backend

- Node.js for RESTful APIs and integrates well with JavaScript on the frontend
- Express.JS for managing the server and API
- PostgreSQL for database storage
- Auth.js for authentication and session management
- Encryption and SSL for secure data transmission

Analytics

- Mixpanel to collect real-time usage data

Version Control and Collaboration

- GitHub Repo
- Protected branches and pull requests require two approvals before merge
- GitHub Projects and Issues for task tracking and sprint management
- Slack for daily communication
- Zoom for online meetings

Design and Prototyping

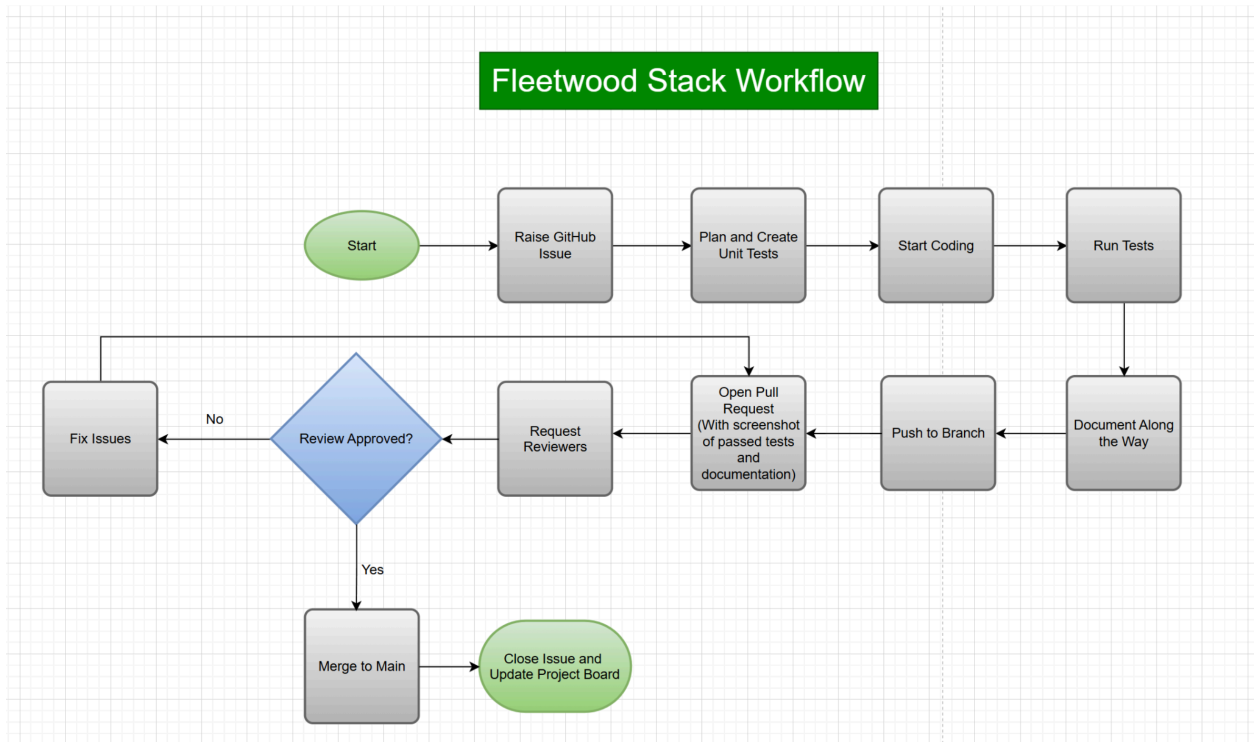
- Figma for wireframes and UI layout
- Lucidchart for database mappings, schema, and system diagrams

Testing and Deployment

- Docker for easy development environments

10. Workflow

Draw.io



11. Team Roles

Feature 1. Manager / Authentication / Roles
Team: Austin, Siri, Patryk, Isheta

Feature 2. Class Directory
Team: Alex, Akshay, Adam

Feature 3. Attendance System
Team: Jared, Junjie, Tej

Feature 4. Work Journal / Stand-Up Tools
Team: Tongke, Lei, Melvyn

Testing
Team: Austin, Akshay, Patryk

12. Software Engineering Practices

- **Version Control:** All code is managed on GitHub with branches for each feature team. Pull requests must be reviewed and approved by a different team before merging into main. All test cases must pass prior to merge.
- **Communication and Collaboration:** Slack will be used for daily communication with daily stand ups summarizing work completed, upcoming tasks, and any blockers. Zoom will be used for group meetings once Slack pro expires.
- **Documentation:** Design document, meeting notes, and other related material will be stored in the repository as the single source of truth.
- **Code Quality:** Code follows consistent formatting (variable naming camel case, etc) with clear, meaningful comments describing the code. No one line code.
- **Code Reviews:** All pull requests must be reviewed by at least two other members from different teams. Reviews will check for functionality, readability, and following the Definition of Done.
- **Definition of Done:** A feature is complete when the following are met:
    - Code implemented

    - Tests written and passing
    - Code reviewed and approved
    - Documentation updated
    - Merged into main branch
- **GitHub Issues and Project Boards:** Each task or feature begins with a GitHub issue containing a short description, assigned members, and relevant labels. Issues are tracked using GitHub Projects under columns
    - Todo
    - In Progress
    - In Review
    - Done
- **Testing:** Each feature team will write unit tests for their component before merging. A separate integration testing team verifies that features work correctly together. Automated tests will be run through GitHub Actions where possible and all test cases must pass before a pull request is approved.

13. General Timeline - Project Roadmap

Week	Milestone	Dates for Completion (Thursday of each week)
5	Planning - Design - Setup	Oct 30
6	Backend - Foundation for each Feature Frontend - HTML + Views for each Feature	Nov 6
7	Write Unit Tests and Documentation (Complete unfinished tasks)	Nov 13 **Midpoint Presentation**
8	Work on API routes, connect frontend with backend	Nov 20
9	Integration Testing & UI Polishing	Nov 27 **Deploy**
10	Revisit, Finalize and Work on Presentation	Dec 4 **Code freeze**
11	Prepare Presentation	Dec 11 **Final Presentation**

14. Splitting Tasks

Feature 1 - Manager / Authentication / Roles

Team: Austin, Siri, Patryk, Isheta

Backend Tasks

- Design user and roles tables in PostgreSQL (`users`, `roles`, `user_roles`, `activity_logs`).
- Set up **Auth.js** with Google OAuth 2.0 integration.
- Implement `/login`, `/logout`, and `/session` routes.
- Add middleware for **role-based access control (RBAC)**.
- Store and encrypt tokens (access and refresh).
- Create API endpoint for managing users and assigning roles.
- Write unit tests for authentication and role validation.

Frontend Tasks

- Build **Login Page** (Google OAuth button + redirect).
- Create **Role Management Dashboard** for Admins.
- Add conditional navigation for different roles (Instructor, TA, Team Lead, Student).
- Implement session persistence and logout flow.
- Add UI alerts for failed login or token expiration.

Testing Tasks

- Test successful login flow (mock Google token).
- Test role-based page restrictions.

- Check invalid tokens, expired sessions, and logout behavior.
- Verify correct activity logging in DB.

Feature 2 - Class Directory

Team: Alex, Akshay, Adam

Backend Tasks

- Create database tables for **teams, members, and user details**.
- Write API endpoint `/api/class-directory` to fetch all users with role and team info.
- Add search and filter parameters to API (role, availability, group).
- Ensure endpoint is role-secured (only visible to authorized users).
- Write unit tests for all query functions.

Frontend Tasks

- Build **Directory Page** displaying all members in a grid or list.
- Add **Search Bar** and **Filter Dropdowns** (role, team, availability).
- Add sorting by name or role.
- Make profile cards clickable (expand with details).
- Connect frontend with backend API via fetch or HTMX.

Testing Tasks

- Test directory fetch with dummy data.
- Verify filters and search functionality.
- Check layout responsiveness and accessibility.
- Integration test: backend + frontend combined view.

Feature 3 - Attendance System

Team: Jared, Junjie, Tej

Backend Tasks

- Create attendance table linked to `users` and `sessions`.
- Implement API routes `/attendance/mark`, `/attendance/view`, `/attendance/summary`.
- Ensure only authorized users (TAs, Team Leads) can mark attendance.
- Handle timezone consistency in records.
- Write backend tests for insert, update, and retrieval.

Frontend Tasks

- Build **Attendance Page** with check-in button and summary table.
- Add date picker and session selector.
- Show attendance percentage by user/team.
- Add color indicators (green = present, red = absent).
- Integrate data from backend and refresh dynamically.

Testing Tasks

- Test data consistency across frontend and backend.
- Verify permissions (students can view but not edit).
- Check for missed or duplicate entries.
- Test network failures and retries.

Feature 4 - Work Journal / Stand-Up Tools

Team: Tongke, Lei, Melvyn

Backend Tasks

- Create `journals` table linked to user and team IDs.
- Implement routes `/journal/create`, `/journal/update`, `/journal/view`.
- Add filtering by date and team.
- Write endpoints for exporting weekly reports (CSV or JSON).
- Validate data (entry length, timestamps).

Frontend Tasks

- Design **Journal Entry Form** with text box and date selector.
- Build **Team Summary View** that aggregates entries by member.
- Add filters for “This Week”, “This Month”, or “By Team”.
- Create export button for journal reports.
- Ensure clean and accessible UI for frequent use.

Testing Tasks

- Test journal submission and editing flow.
- Verify correct linking of entries to users.
- Test export functionality.
- Ensure private notes are only visible to the author.

Shared Tasks (All Teams)

- Merge shared schema updates through pull requests.

- Keep documentation updated (`/docs/schema.md`, `/docs/api.md`).
- Maintain consistent naming conventions and endpoint patterns.
- Use CI/CD checks (linting, test coverage, build verification).
- Perform weekly cross-team review to check integration points.