# Pitch Check-in [ Team 05 ]

## Statement of Purpose

Our website will be an online hub for Professors and Teachers to organize their courses. Our mission is to make it easy for everyone involved in the course to track their progress and have a centralized place for course information. We aim to create a secure environment where all students feel comfortable and welcome.

## Features

1. **User Manager / Authentication / Roles**
    a. Google Oauth
    b. Limited Login Attempts
    c. Login Sessions
    d. Roles
        i. Professor
        ii. TA
        iii. Tutor
        iv. Team Leader
        v. Student (Base permissions)
    e. Professor can update user permissions if needed, but we will have a default that will work most of the time
    f. Each role can elevate permissions of other users, as long as the elevated permissions is less than theirs (i.e. a TA can elevate a Student to a Team Leader)
2. **Class Directory**
    a. After login, users will be able to sign up for a class with a code.
    b. Users will be able to query the class directory, depending on their role.
3. **Attendance System**
    a. Students will enter an attendance code.
    b. Professors and TAs can create meetings
4. **Work Journal**
    a. Plaintext documents that can be organized with folders.

## User Personas

### Professor

- As a Professor, I want to avoid tedious workflows. I want creating classes and adding students to be as simple as possible. I do not want to have to manually configure roles

and, instead, would like sane defaults that I will likely not need to update, but can if I need to. Similarly, I want to delegate tasks, like team creating, to the TAs.
- As a Professor, I want to be able to quickly get an overview of my classes. I want to be able to track metrics, like attendance and participation trends, for my students and TAs.
- As a Professor, I want to be able to quickly switch between my active classes.
- As a Professor, I want to be able to track student attendances during my lectures.
- As a Professor, I want to be able to create announcements that everyone in my class can see.
- As a Professor, I want to follow security and accessibility standards, in order to comply with the University's rules and create a safe environment for all students.

## TA

- As a TA, I want to make the Professor's job as easy as possible.
- As a TA, I want to make announcements to all of the teams that I oversee, both individually and combined.
- As a TA, I want to have access to data that allows me to grade projects easier. I want to be able to see individual student activity and group contribution statistics.
- As a TA, I want a place where I can organize my thoughts, and keep track of the course. I want to be able to keep track of observations I make, and be able to easily refer back to them.

## Team Leader

- As a Student Lead, I want to be able to get a better understanding of everyone on my team. I want to have access to data that helps me know what to improve, so I can make more informed decisions, instead of guessing. Furthermore, I want to be able to see the work completed and in-progress for each student on my team, so I can allocate tasks to spread the workflow evenly across the team.
- As a  Student Lead, I want to be able to have more data available that helps me schedule meetings and assign tasks.
- As a Student, I want to be able to make announcements to my team.

## Student

- As a Student, I want to have a centralized place to find everything I need: announcements, important links, and peer availability.
- As a Student, I want a place where I can track and organize my progress during the course. I want to be able to get credit for my hard work, so I want my TA and Student Lead to be able to see my journal entries.
- As a Student, I want to be able to track my progress and involvement in the course relative to my peers.
- As a Student, I want to know my data is secure.

# User Flows

- Student:
    b. A UCSD student can login with the OAuth using their UCSD email address. A non-UCSD student can login with a magic link provided by the backend, assuming the student is verified as a temporary UCSD student.
    c. Once the student has logged in, there will be an option to enroll into a course. This will be done by entering a code shared by the professor/TA with the students.
    d. After enrolling, the relevant courses will appear in the application. On selecting a course, the student will be able to see the course materials, his team, his statistics and other relevant information.
    e. The student can also view his team profile, the professors profile and the TA's profile.
    f. At any given point the student can also write information into his work journal via a collapsible sidebar to the right of the screen. The visibility of his notes can be toggled via a selection.
5. TA's and Tutors:
    a. The TA's will initially login and enroll as a student, and the professor will assign them special privileges once they are enrolled in a course.
    b. As a TA, they can see the attendance, performance, and team performance of all the students in the class. They can assign student team leads.
    c. The TA's can make notes of the student performance or the student participation, via the Work Journal.
6. Professors:
    a. The professor will login through the OAuth using their official UCSD email address.
    b. Once the professor has logged in, they can create a course. Once the course has been created, students and TA's can join that course.
    c. The professor will have a choice to appoint TA's for the course. The professor can see the attendance, performance, team performance of all the students as well as the TAs in the class.
    d. The professor can make notes of the student or TAs performance or the student participation, via the Work Journal.
7. Team Leads:
    a. The team lead will be assigned by the TA's. The team leads will have all the functionalities of a student along with more information about the team members and team performance.
    b. They can also schedule meetings or set up one on one discussions with members of their teams.

## Risks

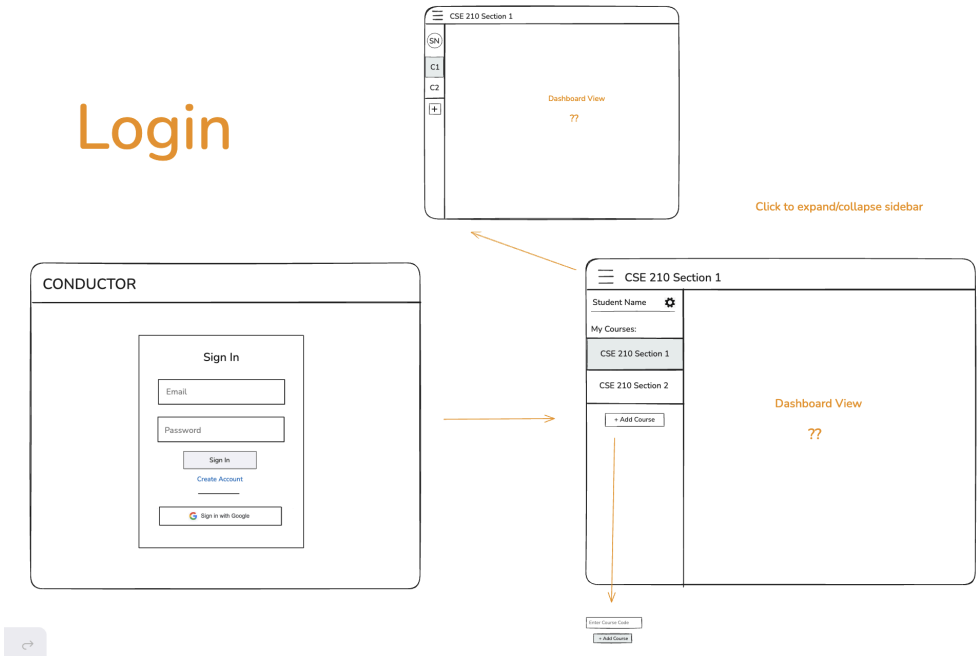
- Time limit: Only five weeks
- Security: Minimal experience in security, and we need to mitigate a lot of different attack vectors, like prompt injection. Interfacing with UCSD SSO may be a challenge.
- Inexperienced with tech stack and frameworks.

## Rabbit Holes (Deferred from MVP)

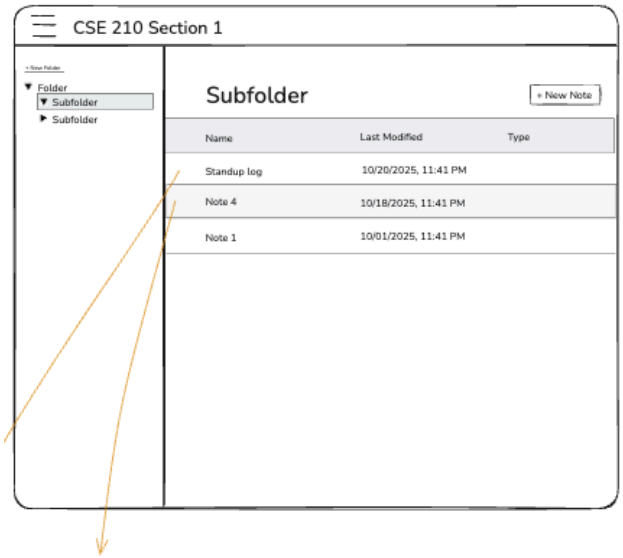
- Shared edits in work journal + forum/Q&A: A journal editable by multiple users with history plus a separate, forum-style Q&A system is too heavy for the database and moderation needs, not essential for MVP.
- Comprehensive course calendar: A unified calendar for assignments, lectures, and office hours (with reminders/sync) is out of scope for now. We will consider a simple schedule and .ics export later if time permits.
- TA participation tracking tool: Nice-to-have but extra work for the current timeline, launch MVP with attendance only and revisit lightweight participation tracking post-MVP.
- Grades and assignment views: Full gradebook/assignment display is a larger scope. Given the limited timeline, these features are intentionally deferred to keep focus on core flows (enrollment, rosters/teams, announcements, attendance, exports).

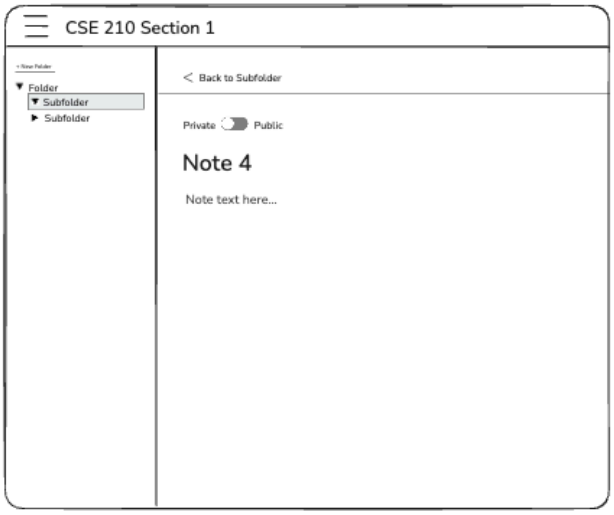
## Wireframes

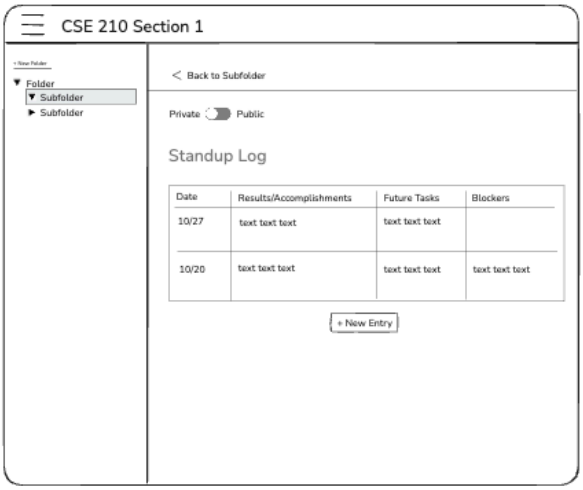
Wireframe for login screen and course enrollment:



Work journal wireframe showing file browser layout, a standard plaintext note format, and a note with a dedicated standup log template. Also shows how users can toggle the visibility of notes.

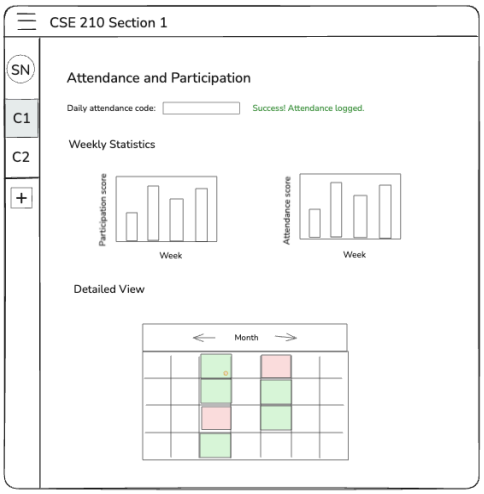


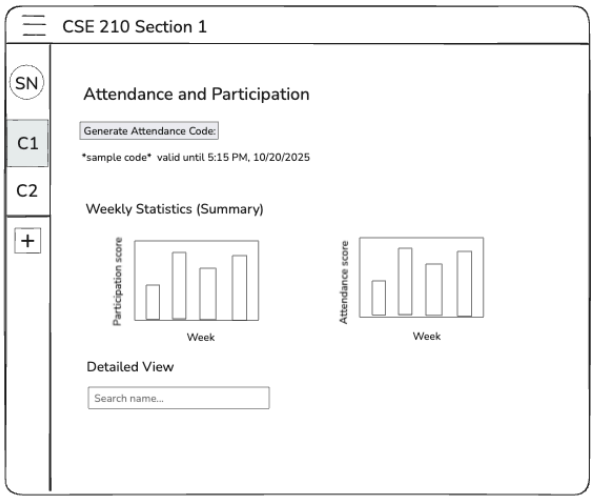




Below left: Student View of Attendance/Participation dashboard and detailed view. Contains an entry box for the daily lecture attendance code, with an example success message. Shows summary statistics over time for attendance and participation, using bar plots. Below, the detailed view provides a day-to-day look at attendance and participation stats.

Below right: Professor view of attendance/participation dashboard. Detailed view can be accessed by searching a specific member of the course.




# Technologies

**Languages**

JavaScript:

We will use JavaScript as our primary backend language. It is a core component of the web technology stack and integrates seamlessly with our frontend. JavaScript will be used to implement HTTP endpoints, business logic, and service integrations in the backend.

To maintain reliability and consistency in a dynamic language, we will adopt strict ESLint rules, JSDoc annotations, and runtime schema validation.

SQL:

SQL will be used for defining relational schemas and querying our PostgreSQL database. It enables us to perform structured queries, joins, and transactions to manage data integrity.

**Framework and Runtime**

Runtime – Node.js:

Node.js provides a fast, event-driven, non-blocking runtime that is ideal for building high-concurrency, I/O-heavy backend applications. It allows us to handle multiple client requests efficiently while maintaining low resource usage.

Framework – Fastify:

Fastify is a lightweight, high-performance web framework that runs on Node.js. It handles frontend–backend communication by receiving HTTP requests, processing business logic and database operations, and returning structured JSON responses.

We chose Fastify for its strong performance, built-in schema validation, and plugin ecosystem, which helps ensure a clean and maintainable backend architecture.

Authentication:

Fastify Plugins (@fastify/oauth2, @fastify/secure-session):

We plan to use Google OAuth together with our university's SSO (OIDC) for user login. Both will be implemented using the official @fastify/oauth2 plugin, which handles the authorization flow and securely exchanges tokens.
User sessions will be managed with @fastify/secure-session, storing encrypted cookies for a simple and secure login state.
These plugins are easy to integrate, well-documented, and provide built-in security features like PKCE and HTTPS-only cookies. If we encounter performance or compatibility issues, we can switch to JWT-based tokens for stateless authentication later.

**Database**
PostgreSQL:
A robust database is essential for our Conductor application. We chose PostgreSQL as our relational database management system, as required by the project's design document. PostgreSQL is an industry-grade, open-source Object-Relational Database Management System (ORDBMS) that supports both relational and semi-structured data through JSONB.
Pros:

- Through exploring database indexing principles and schema design, we can optimize our data access patterns based on fundamental database concepts.
- PostgreSQL provides ACID transactions, write-ahead logging (WAL), and point-in-time recovery (PITR), ensuring durability and consistency of our data. Even in the case of server failure, the system can recover from the WAL logs.
- It supports JSONB, allowing us to store semi-structured data efficiently. This is ideal for our work journal feature, where user notes or journal entries can be stored as text in JSONB fields.

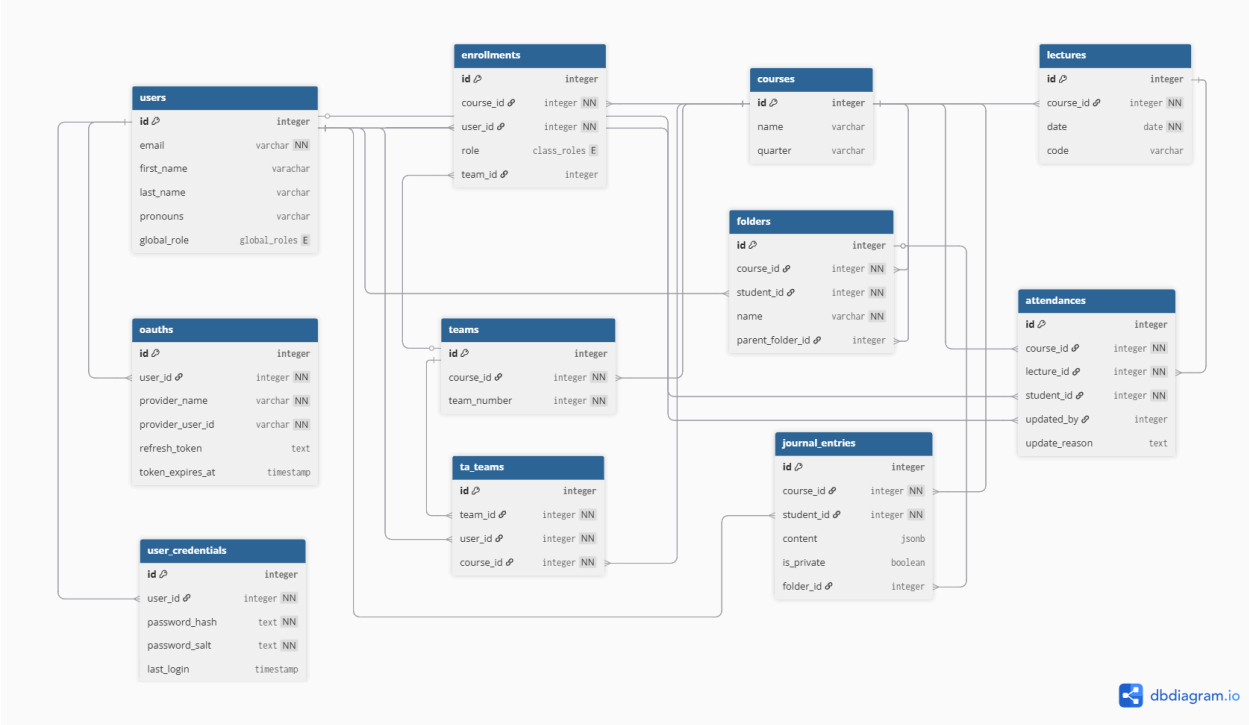
Cons:

- As mentioned in the design document, the system must handle potentially high concurrency — up to 10,000 students simultaneously.
- PostgreSQL alone may not efficiently handle that level of concurrent connections, as maintaining thousands of open connections can degrade performance.
- In future iterations, we plan to add a buffer or caching layer (such as Redis or a connection pooler like PgBouncer) between our backend and the database.
- For our MVP, however, we will use PostgreSQL alone, focusing on delivering a functional prototype with correct backend logic before optimizing for scale.

**ORM and Query Layer**
Prisma:
Prisma is a modern Object-Relational Mapping (ORM) tool that provides a clean and type-safe interface for interacting with PostgreSQL using JavaScript. It allows us to define data models in a schema file, automatically generate query functions, and manage migrations consistently across environments. However, in some extreme cases, SQL queries generated by Prisma might not be as efficient as the human-written ones.
We will assess Prisma based on its ease of integration with Node.js, query performance on large datasets, developer productivity, and its ability to maintain schema consistency during

migrations. If Prisma's abstraction introduces performance overhead in production testing, we may supplement it with optimized raw SQL queries for critical database operations.

# Database Schema




Enums -
Global_roles - "admin", "professor", "regular" (default)
Class_roles - "professor", "ta", "tutor", team_lead", "student"

# Security

## Data Storage/Transfer

- Send data with HTTPS
- Encrypt database information, with smth like (AES-256)
- Role-base permissions and data reads/writes
- Stuff like user passwords, ID, names, other identifiable stuff are fully private and inaccessible for everyone, for FERPA. Do this with either encryption layer.

## Deployment:

Ongoing discussion whether the application shall be deployed in the cloud? (Iaas or Paas)