BoilerTutors

Design Document - Team 7

Team Members: Alice Tang, Jane Billa, Jacob Mobley, Andrei Deaconescu, Gavin McCormack

Purpose

Many Purdue students struggle to find tutors who match their academic needs, such as specific course topics, availability and price. Likewise, student tutors lack a centralized way to find those who need help. Unlike general tutoring platforms, our project focuses exclusively on Purdue students and supports in-person meetings on campus, filling a gap since Purdue currently lacks a comprehensive tutor matching service. Our app will allow students to request help for Purdue classes and an AI-powered recommendation system will analyze their request and tutor profiles to suggest the best matches. This creates a more convenient and efficient way for Purdue students to give and receive academic help.

Functional Requirements

User Requirements

1. As a user, I want to be able to create an account and securely sign in using my Purdue email
2. As a user, I want to be able to modify my in-app profile (name, role, bio, preferences).
3. As a user, I want to be able to message other users within the app
4. As a user, I want to be able to receive notifications for important events (messages, bookings, updates, etc.)
5. As a user, I want to be able to easily contact an administrator as needed
6. As a user, I want to be able to log out and manage my account security settings
7. As a user, I want to delete or deactivate my account if I choose to leave the platform

Tutor Requirements

8. As a tutor, I want to be able to sign up and register as a tutor for any subject I am qualified to teach
9. As a tutor, I want to be able to receive and read my reviews/ratings
10. As a tutor, I want to only be able to receive reviews from those I have tutored (avoid fake reviews)
11. As a tutor, I want to select which Purdue classes I am interested in tutoring
12. As a tutor, I want to show the grade I earned in a class I offer tutoring for.
13. As a tutor, I want to indicate whether I have previously TA'ed a class I offer tutoring for
14. As a tutor, I want to indicate whether my sessions are online or in person
15. As a tutor, I want to specify the locations where I offer in-person tutoring
16. As a tutor, I want to be able to filter through students who are looking for online or in person sessions

17. As a tutor, I want a payment system for my services
18. As a tutor, I want a profile that shows my available tutoring times, reviews, and tutoring credentials
19. As a tutor, I want email notifications when someone purchases my tutoring services
20. As a tutor, I want to be able to set a cap on number of tutoring sessions that can be purchased from me in a certain time period (ex: 2 sessions per week)
21. As a tutor, I want students who purchase my tutoring services to automatically receive a message on the app from me
22. As a tutor, I want the option to decline a student without being required to provide a reason
23. As a tutor, I want to view a potential student’s availability schedule before accepting a session/match
24. As a tutor, I want to be able to view the materials my potential match has uploaded
25. As a tutor, I would like to set my own prices for each course/subject I tutor
26. As a tutor, I want to temporarily pause my availability/new matches without deleting my account
27. As a tutor, I want to see a summary of my upcoming and past tutoring sessions

Student Requirements

28. As a student, I want to be able to leave reviews for tutors I have worked with
29. As a student, I want to be able to view reviews and ratings of tutors
30. As a student, I want to be able to find tutors for specific Purdue classes
31. As a student, I want to view tutor credentials such as grades and TA experience
32. As a student, I want to indicate whether I prefer online or in-person tutoring
33. As a student, I want to quickly set a location for in-person tutoring sessions
34. As a student, I want to be able to set a price range or limit
35. As a student, I want to view and compare tutoring prices
36. As a student, I want to receive tutor recommendations that match my needs
37. As a student, I want a simple and secure system to purchase tutoring sessions
38. As a student, I want the option to cancel a tutoring session and receive a refund
39. As a student, I want to receive a refund if my tutor does not show up or misrepresents their credentials
40. As a student, I want optional email reminders for upcoming tutoring sessions
41. As a student, I want to have the option to contact my tutor outside of tutoring hours

42. As a student, I want to be able to add tutoring sessions to my personal calendar through the app
43. As a student, I want to be able to create and manage a schedule of availability
44. As a student, I want to be able to upload my materials/documents that I need help with
45. As a student, I want the option to submit tutoring reviews anonymously
46. As a student, I want to see a history of my past tutoring sessions and payments
47. As a student, I want to be able to report a tutor if there is an issue with a session

Administrator Requirements

48. As an administrator, I want to be able to ban users that violate our policies
49. As an administrator, I want to be able to send refunds for canceled tutoring sessions or accidental purchases.
50. As an administrator, I want to be able to remove inappropriate or fraudulent reviews
51. As an administrator, I want to be able to verify tutor credentials before allowing them to offer tutoring
52. As an administrator, I want to view platform activity logs for moderation purposes

Non-Functional Requirements

Security & Privacy

- The platform should protect user data and prevent unauthorized access. We will use role based access control so that tutors can only access tutor-related features and information, and students can only access student-related features. All users will be required to sign in using a Purdue email to make sure the platform is only used by Purdue students
- Any actions that involve sensitive data, such as viewing personal information or handling payments and refunds, will require proper authentication. Users will only be able to view and modify their own data, and tutors will only be able to see sensitive information of students that they are matched with

Usability

- The platform should be easy to use for both students and tutors. Common actions like finding a tutor, booking a session, uploading materials, and messaging should be straightforward and not require unnecessary steps. The interface should be clean and intuitive so that new users can quickly understand how to use the app without needing detailed instructions or a tutorial

- If something goes wrong, loading states and error messages should be clearly shown so users always know what is happening and are not left confused

Performance

- The system should respond quickly to user actions. Features such as searching for tutors, loading profiles, and sending messages should typically complete within a few seconds. If an action takes longer than expected, the app should show a loading indicator or error message instead of being frozen

Scalability & Maintainability

- The app should be able to scale as more users and tutoring sessions are added. All Purdue courses should be available as options so students can find tutoring help for any class they need
- Our code should be organized and documented so that it is easier to debug, maintain, and add new features in the future

Transactions & Reliability

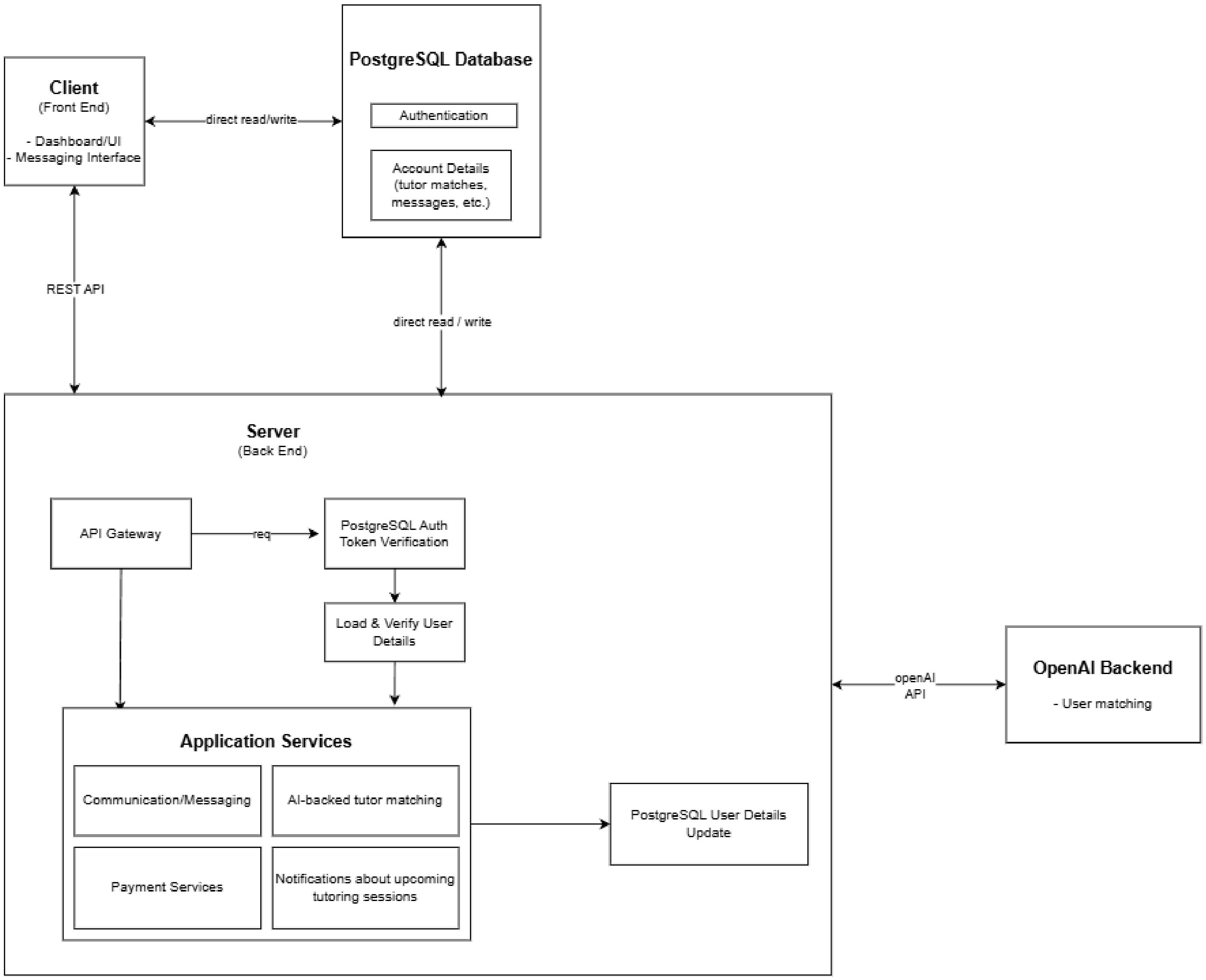
- The app should support a secure and reliable payment system for tutoring sessions. Payments should be processed safely, and refunds for cancellations or no shows should be easy to handle. The system should prevent issues such as double charges or failed transactions and ensure that users are clearly informed about what's happening with their payments

Design Outline

This project follows a client–server architecture and will consist of two main applications: the web client and the server-side backend(hosted on). The client is started as a web-based platform, built with React Native and TypeScript, and provides tutors and students a secure interface to find help for Purdue-specific courses. The backend services handle authentication, AI-drive smart selection, data storage, and secure transactions.

High-Level Overview of Client-Server

The project follows a client-server architecture, with a 3rd-party PostgreSQL database for storing user credentials and state (banned/not banned/etc.) and a connection with the OpenAI server through the OpenAI API to leverage the AI-powered tutor-student matchings.
We will create a lightweight frontend that displays the UI to the user, while the backend will handle communications, displaying to the frontend based on user state (student/tutor/admin), AI matching, and database verification.



1. **Client**
    a. The client provides the user with a UI interface through which they can communicate with the backend
    b. Since the frontend is lightweight, it will not have much functionality beside displaying the contents pushed by the backend
    c. The frontend will also handle pushing authentication data to the PostgreSQL database
2. **Server/API Layer**
    a. The server will perform most of the functionality of the app
    b. It will handle communication between peers, the API connection with OpenAI for the AI-backed matching, and scheduling
    c. It will also instruct the front end on what to display based on the database
3. **Database (PostgreSQL)**
    a. The database stores all user credentials and details
    b. The backend retrieves this information from the database and sends it to the frontend for display
    c. This will also contain information about the user’s banned state (e.g. if the user is banned, the frontend will display a “banned” message and block all usage of the app for that particular user)
4. **AI Driven Matching**
    a. The backend will retrieve user information from the database from both the students and the tutors (information consisting of their skills, their needs, preferences, etc.)
    b. The backend then pushes the information through the OpenAI API to perform the AI-backed matchings, read the matchings, and update the database with the matches.
5. **Scheduling**
    a. The scheduling will be performed by the backend based on declared availability times by the students and the tutors
    b. The updated schedules will be pushed back to the database

Description of Backend, Frontend, Webpage Interactions

Frontend

The frontend will be developed using React Native. The user types their login credentials in the application, which sends these directly to the database for validation. Then, the backend also verifies the authentication token to access the database and the user details. The backend then loads all the user details and pushes the relevant details to the frontend for UI display.

Web app

The web application will serve as the main access point for users to access the platform. It will handle the entire UI and display all relevant user interface activities and information. The page will be rendered through React Native which receives data from the backend about the user status.

Backend

The backend will handle almost all processes of the app. It will fetch user data, perform processing such as scheduling and message updates, update the frontend, and communicate with the OpenAI backend for the AI tutor-student matches. The backend will be designed via FastAPI and Python.

Database

The PostgreSQL database will store all details about all users (tutors, students, admins). The backend has full access to the database as long as the user auth token is validated, and can perform read and write operations on the database.

OpenAI Connection

The backend will utilize all relevant details for matching of students and tutors. This ranges from classes, availability, GPA, tutor skill level, goals, etc. When a student clicks the “Match” button, the backend will collect all relevant details about the student and all the tutors that match the student’s availability, and push them into an OpenAI API pipeline. After the AI processing, the backend collects the matches performed and updates the matches dictionary in the database for the student and the tutor. They will then be put in contact through a messaging app.

Design Issues

Functional Issues

1. How will new users sign into the system (Purdue-only access)?
    a. Option 1: Allow any email to sign up/login (no restrictions)
    b. Option 2: Allow sign up/login only if the email domain is @purdue.edu (email + password)
    c. Option 3: Require Purdue Google SSO (OAuth) only, restricted to @purdue.edu

Choice: Option 2
Justification: Restricting accounts to @purdue.edu ensures BoilerTutors is limited to the Purdue community and reduces spam/fake accounts. Using email + password keeps onboarding simple and avoids OAuth setup complexity while still enforcing the Purdue only requirement. The backend (FastAPI) will validate the domain during registration and login, and user accounts will be stored in PostgreSQL with a unique constraint on email to prevent duplicates.

---

2. How will tutoring sessions be scheduled?
    a. Option 1: Tutors manually coordinate sessions outside the app
    b. Option 2: Real time in-app scheduling with availability calendars
    c. Option 3: Hybrid approach with availability calendars and manual confirmation

Choice: Option 3
Justification: A hybrid approach allows students to request sessions based on tutor availability while giving tutors final control to confirm or adjust sessions. This prevents scheduling conflicts and accommodates last minute changes. Availability and session data will be stored in PostgreSQL, while FastAPI enforces scheduling constraints and conflict checks.

---

3. How will tutor-student communication be handled?
    a. Option 1: External communication only (email or text)
    b. Option 2: In-app messaging only
    c. Option 3: In-app messaging with optional email notifications

Choice: Option 3
Justification: In-app messaging keeps conversations organized and tied to tutoring sessions, while email notifications ensure users do not miss important updates when they are offline. Messages will be stored in PostgreSQL, and FastAPI will manage message delivery and notification triggers.

---

4. How will tutoring subjects and tutor qualifications be represented?
    a. Option 1: Hard-code supported subjects in the frontend
    b. Option 2: Store subjects and qualifications as flexible database records
    c. Option 3: Separate subject categories with custom schemas per subject

Choice: Option 2
Justification: Storing subjects and tutor qualifications in PostgreSQL allows easy expansion without frontend redeployment. This supports future growth (new classes or departments) while keeping queries efficient. The flexible schema enables filtering tutors by course, experience, or rating.

---

5. How will payments be handled?
    a. Option 1: No in-app payments; payment handled externally
    b. Option 2: Full in-app payment processing
    c. Option 3: Session tracking only, with future payment integration

Choice: Option 3
Justification: For the initial version, tracking sessions and hours without handling payments reduces complexity and security risk. The database schema will be designed to support future integration with payment providers if we have time, without requiring major refactoring.

Non-Functional Issues

1. How will the system ensure data privacy and access control?
    a. Option 1: All users can view all tutor and student data
    b. Option 2: Role-based access control (student vs. tutor vs. administrator)
    c. Option 3: Role-based access control with per-resource permissions

Choice: Option 3
Justification: Students should only see their own session data, while tutors should only access sessions they are assigned to. Admin functionality will require broader access. FastAPI will enforce role-based permissions, and PostgreSQL constraints ensure data isolation. This design protects user privacy and aligns with university data expectations.

---

2. How will the system scale with increased usage?
    a. Option 1: Single monolithic backend with no scaling strategy
    b. Option 2: Horizontally scalable FastAPI backend with a centralized database
    c. Option 3: Microservices for each major feature

Choice: Option 2
Justification: FastAPI supports horizontal scaling behind a load balancer, which is sufficient for expected usage. PostgreSQL provides strong consistency for scheduling and messaging data. A microservices architecture would add unnecessary complexity for the current scope.

---

3. How will errors and failures be handled?
    a. Option 1: Generic error messages only
    b. Option 2: Structured API errors with user-friendly messages
    c. Option 3: Full retry queues and offline write handling

Choice: Option 2
Justification: FastAPI will return structured error responses, allowing the frontend to display clear and actionable messages (ex: scheduling conflicts or invalid requests). Full offline support is outside the project scope and would add significant complexity.

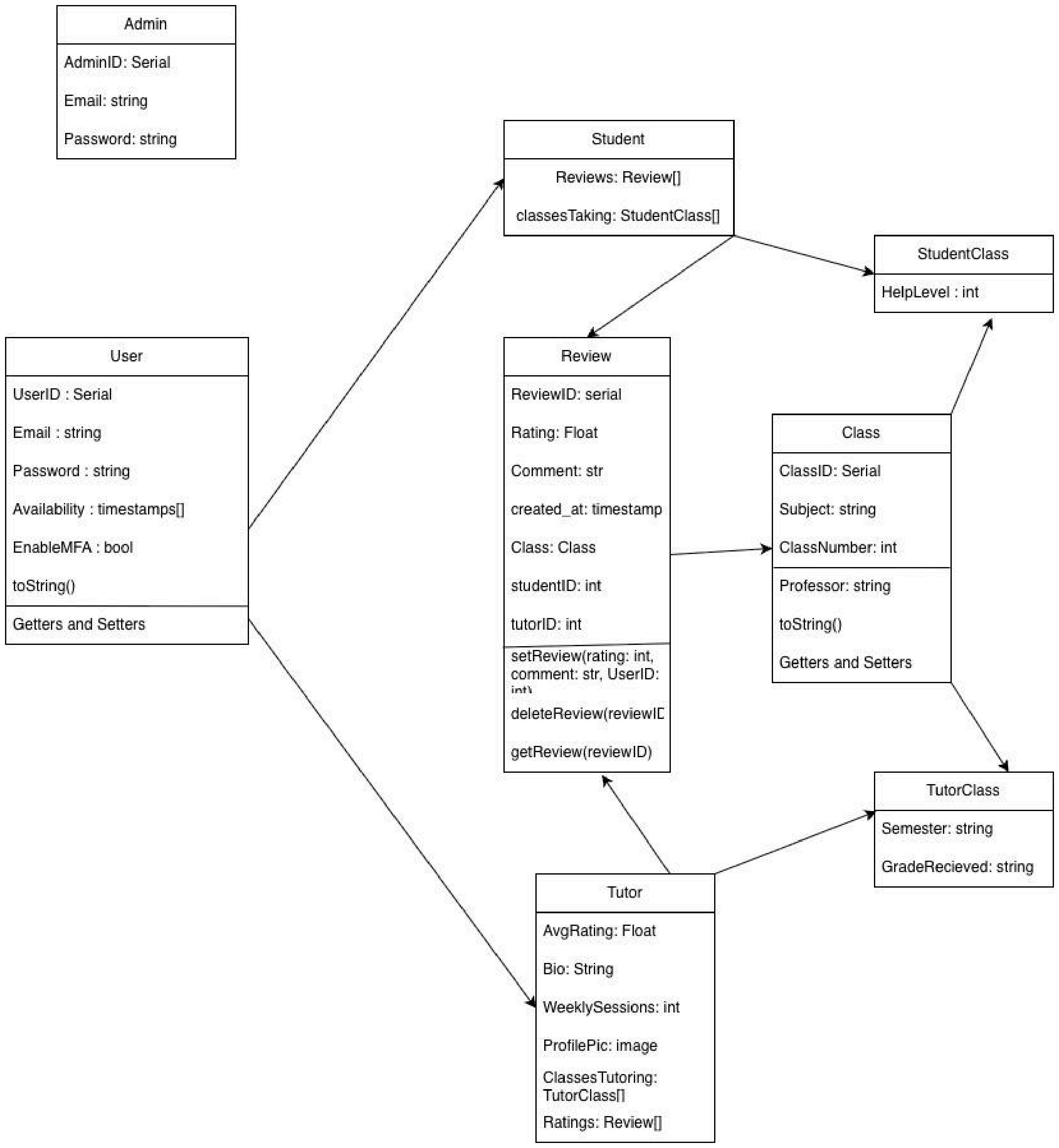
---

4. How will the system support future feature expansion?
    a. Option 1: Hardcode logic directly into frontend and backend
    b. Option 2: Modular backend with clearly separated services
    c. Option 3: Separate applications per feature

Choice: Option 2
Justification: A modular FastAPI backend allows adding features such as ratings, reviews, or payments without impacting existing functionality. PostgreSQL schemas can be extended incrementally, ensuring long-term maintainability.

Design Details

Class Design



Description of Classes and Their Interactions

User

- This represents the basic user of the application
- It is inherited by both the **Student** and the **Tutor** classes
- It allows the user to perform basic actions like enable MFA or set their availability, and also contains details that are shared by students and tutors

Admins

- Are a standalone class; do not extend **User**
- They can manage the platform through their own interface, including banning users and forcing tutor cancellations for any reason as needed
- They can also issue refunds
- They can see messages left by **Users** in the “Report” tab

Class

- This is the basic object representing an academic class
- It is a superclass for **TutorClass** and **StudentClass**, which in turn represent details about classes from a student and a tutor’s perspective

Student

- This class extends **User**
- They have the same attributes as **User**, but also including **Reviews** left by that student and **ClassesTaking**, which marks the **Classes** the student is currently in

StudentClass

- This is a subclass of **Class** which is used for student-tutor matching
- For each **Class** that the **Student** is in, this class includes the **HelpLevel**, which marks the level of help that the **Student** needs in that specific **Class**

Tutor

- This class extends **User**
- They have attributes added on top of user, including their **AvgRating**, **Bio** **WeeklySessions**, **ProfilePic**, **ClassesTutoring** and **Ratings**
- This is how **Students** will be able to recognize the **Tutor**’s skill level, rating, and if they are comfortable with them

TutorClass

- This is a subclass of **Class** which is used for student-tutor matching

- For each **Class** that the **Tutor** teaches, this includes the **Semester** when the tutor took the class, and the **GradeReceived** (the grade the tutor received in that class when they took it)

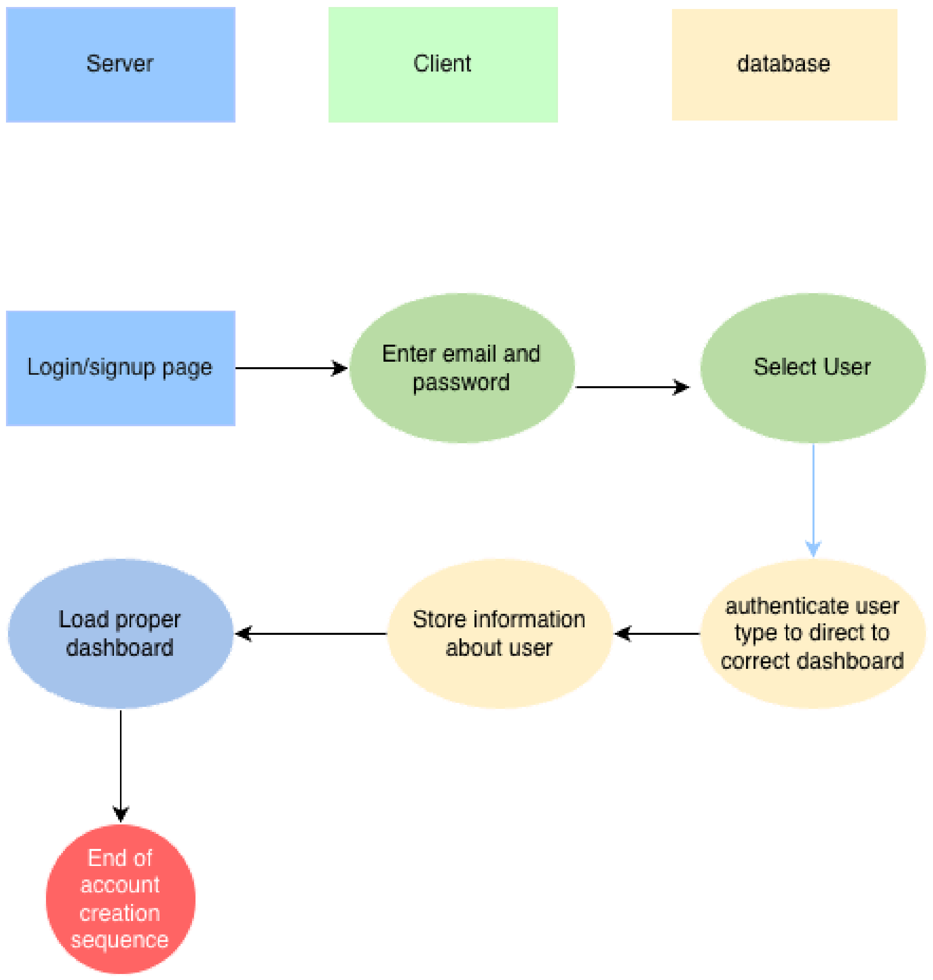
Review

- This represents the reviews that a **Student** may leave a **Tutor**
- This includes a **Rating** out of 5 stars, a **Comment** about the tutor's performance, **created_at** to store when the review was created, and the **class** for which the review was created

Sequence Diagrams of Events

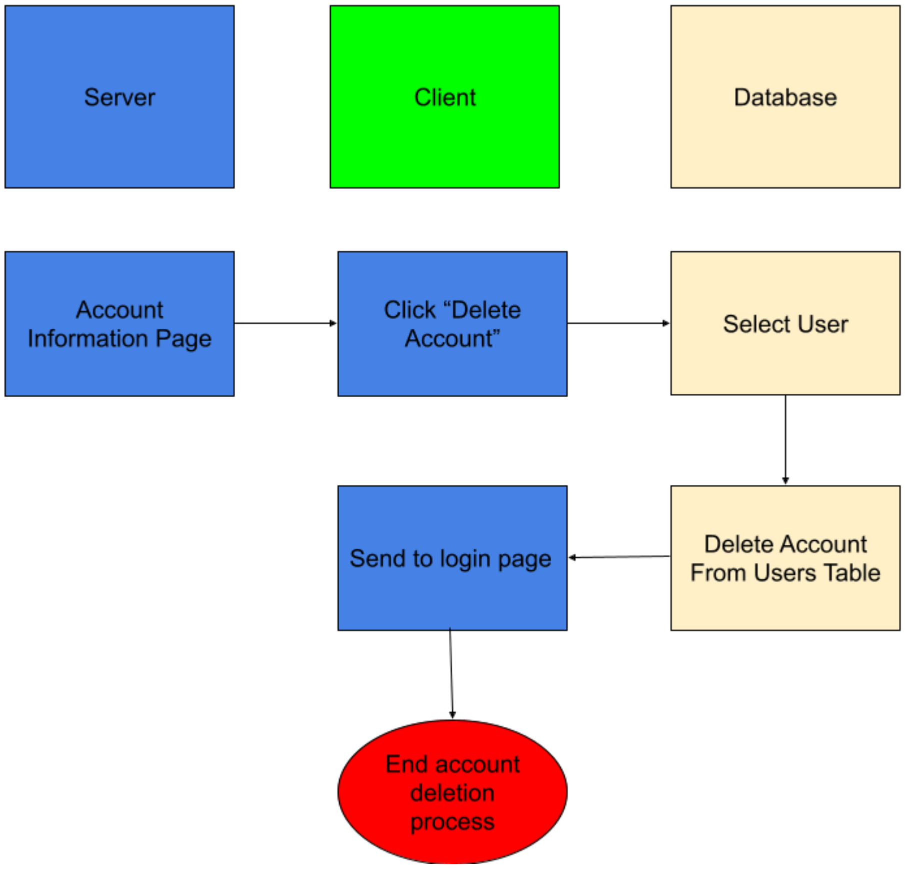
Account Creation

The user starts on the login page and must enter an email and password to create an account. They then must select a user type (tutor or student) to be directed to the proper page. The user is authenticated to make sure the proper information is displayed on their dashboard, and the information about the user is stored in the database to ensure they can login again. Then, the proper dashboard is loaded onto the user's screen.


Server
Client
database
Login/signup page
Enter email and password
Select User
authenticate user type to direct to correct dashboard
Store information about user
Load proper dashboard
End of account creation sequence

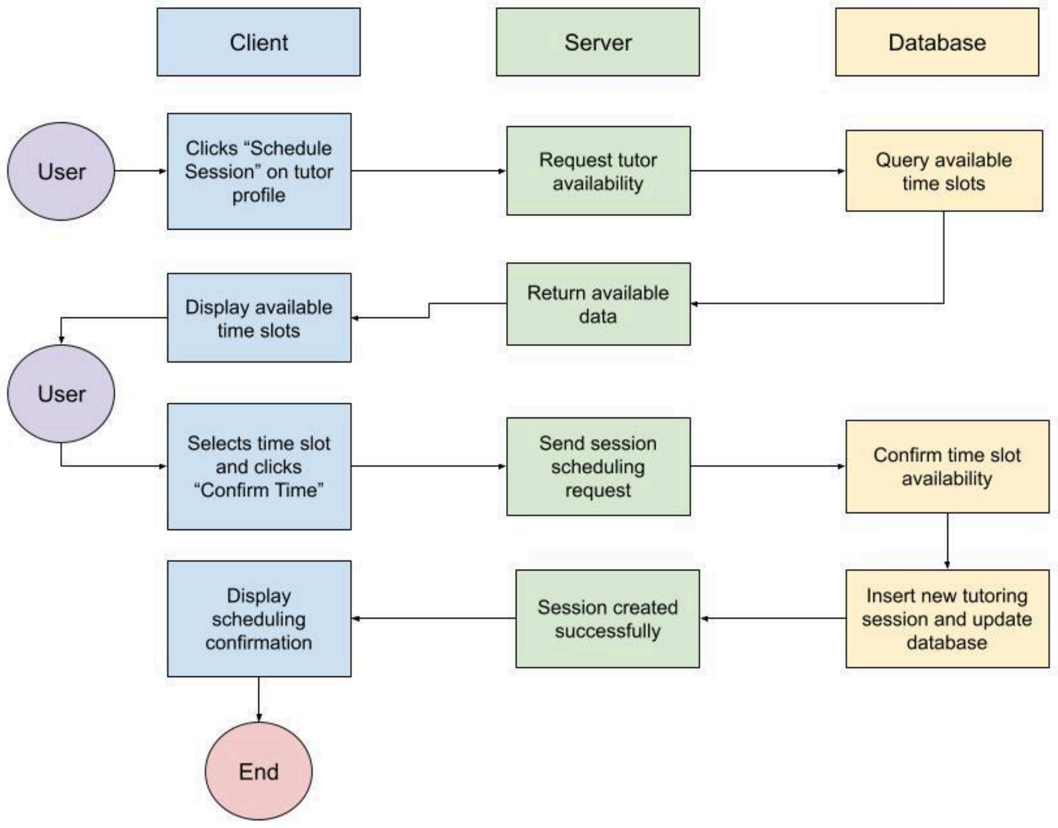
Account Deletion

The user starts on their Account Information page and clicks “Delete Account.” The client sends a delete request to the server, which identifies the currently authenticated user. The server then queries the database to select that user and permanently removes their record from the Users table. After the deletion succeeds, the server returns a success response and the client redirects the user back to the login page, ending the account deletion process.



Scheduling a Session

The user clicks the "Schedule Session" button on a tutor's profile in the client, which sends a request to the server to load the tutor's availability. The server queries the database for the tutor's available time slots and sends the availability data back to the client, where it is displayed to the user. The user then selects a date and time and clicks Confirm Schedule, which sends the scheduling request from the client to the server. The server validates the request by checking the database to ensure the selected time is still available, then creates a new tutoring session and updates the database with the session information. After the database confirms the update, the server sends a success response back to the client. The client displays a confirmation message to the user and loads the updated scheduled sessions view.

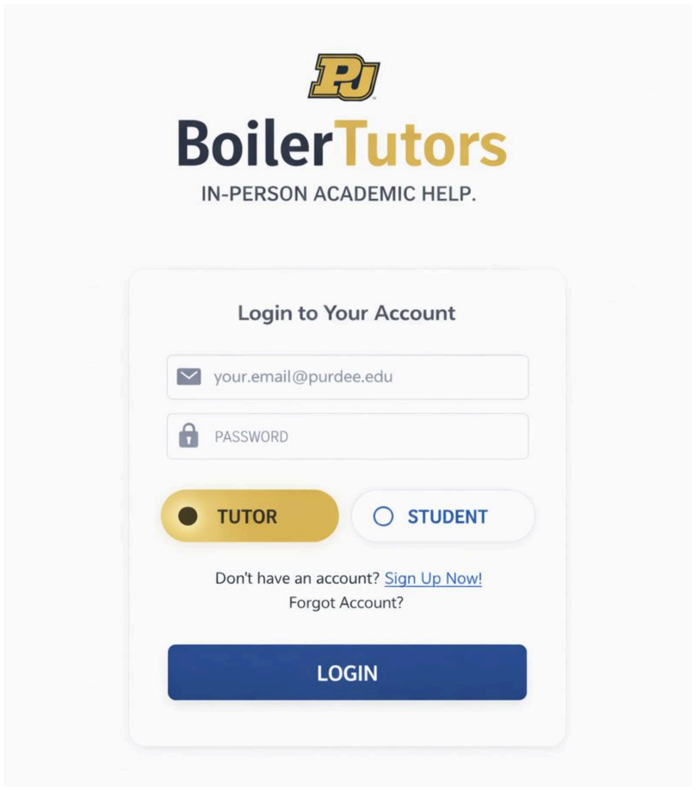


Client
Server
Database
User
Clicks "Schedule Session" on tutor profile
Request tutor availability
Query available time slots
Return available data
Display available time slots
User
Selects time slot and clicks "Confirm Time"
Send session scheduling request
Confirm time slot availability
Insert new tutoring session and update database
Session created successfully
Display scheduling confirmation
End

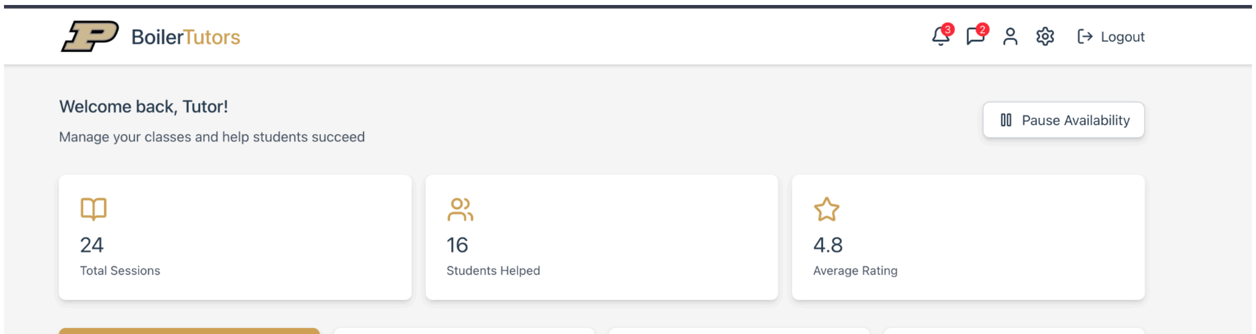
UI Mockups

This section provides preliminary interface mockups to the design of key pages in the application. The mockups are not final but serve as visual guides for payout, navigation, and functionality.

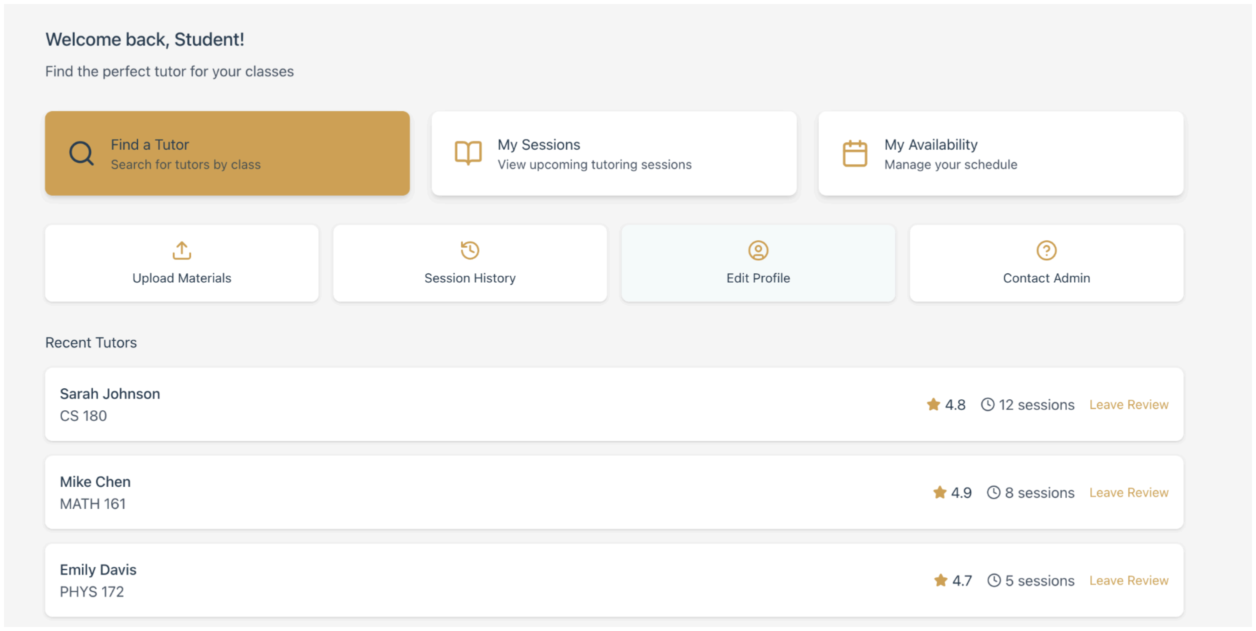
a. Login page: Simple login screen with options for username/password, and whether a user wants to login as a tutor or student.



b. Tutor Dashboard: This page shows tutors how many sessions, students, and ratings they have. They are able to add new classes, view their schedules, see their students, and the classes they are approved to teach.



c. Student Dashboard: Students are able to search for tutors, view their upcoming schedules, update their availability for tutors to view, upload specific materials for tutors to view, see their recent tutors and sessions, and leave ratings.



Welcome back, Student!

Find the perfect tutor for your classes

Find a Tutor
Search for tutors by class

My Sessions
View upcoming tutoring sessions

My Availability
Manage your schedule

Upload Materials

Session History

Edit Profile

Contact Admin

Recent Tutors

Sarah Johnson CS 180	★ 4.8	12 sessions	Leave Review
Mike Chen MATH 161	★ 4.9	8 sessions	Leave Review
Emily Davis PHYS 172	★ 4.7	5 sessions	Leave Review