CourseSwap Project Proposal

Team #25
Jixin Li (jixinl2), Lewen Shen (lewens2), Yuyangsong Xie (yx40) and Yuncong Cao (yuncong6)

Pitch

Course registration and scheduling can be a frustrating experience for students, often leading to **exam conflicts, limited course availability, and the need for schedule adjustments**. Many students find themselves struggling to attend required tests due to overlapping exam times, while others miss out on desired courses because of seat limitations. Additionally, students may want to rearrange their schedules to better align with their academic goals.

CourseSwap is a streamlined platform designed to solve these challenges by allowing students to **exchange courses or exam slots** with peers who have matching needs. By leveraging an intuitive matching system and AI-powered recommendations, **CourseSwap ensures a seamless and reliable process for students to find swap partners efficiently**.

Functionality

1. Users can post information about give / demand / exchange course or exam slots on the app.
2. Users can browse and claim available exchange posts.
3. Users can see a profile of rating systems (5 stars) maximum for potential swap partners.
4. Users can confirm a swap process through a two-way claim system.
5. Users can see recommendations post on front page based on AI matching
6. Users can track swap histories.

Components

The project CourseSwap is a platform that helps students swap courses or exam times, combined with an AI recommendation system to optimize the matching effect. It is basically divided into four parts.

<u>Frontend:</u>
Purpose: Provide an interactive interface that allows users to publish, browse, exchange course information, and view AI recommendation results.
Language: Dart
Framework: Flutter (for iOS & Android)
Library: provider (state management) & dio (handling API requests)
How to test: we can choose unit tests to ensure that UI components (e.g. swap buttons, recommendation screens) render correctly. We can use integration test to ensure that API requests and backend interactions are correct, and that data is displayed correctly.
How to interact with other components: Get and submit exchange requests via the REST API. Listen to the recommendation matching results on the backend and dynamically update the UI.

<u>Backend:</u>
Purpose: Handles all business logic, including user exchange requests, data storage, matching

calculations, etc.
Language: Python
Framework: FastAPI (lightweight, high-performance web framework)
Library: SQLAlchemy (database operations) & Pydantic (Data Validation)
How to test:

Unit testing: Make sure the API interface returns the correct data (e.g. published posts, query matches).

Integration test: Test whether the CRUD operation of the database is working properly.

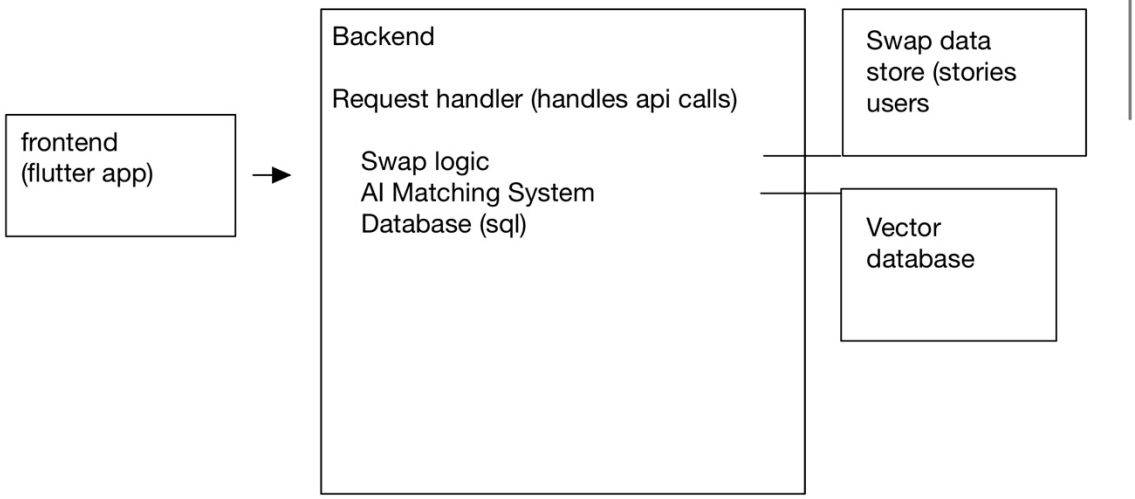
How to interact with other components: Provide a REST API for front-end calls (publish exchange information). Invoke the AI model to calculate the recommended match score and return the result.

AI Matching System

Purpose: Calculate which exchange posts best match the needs of current users and improve matching efficiency.
Language: Python
Model: OpenAI Embeddings (Semantic Embedding)
Faiss / Milvus (vector database)
Library: sentence-transformers (handling text embeddings) & FAISS (High Efficiency Vector Search)
How to test:
We use unit test to make sure that the similarity calculation logic is correct (e.g., calculating the match between two posts). We can use regression test to make sure that the recommendation results are still reasonable after the algorithm is upgraded.
How to interact with other components: Listen to the exchange requests submitted by users and calculate the matching score, then returns the best matching exchange post to the backend API.

Database

Purpose: Store user information, exchange history, match data, etc.
Type: PostgreSQL / MongoDB
Library: SQLAlchemy (ORM framework) & Redis (caching mechanism)
How to test:
We can use integration testing to ensure that data is stored and queried correctly without loss or error.



Continuous Integration

Test Library:

Backend: **pytest**
Frontend: **flutter_test**

Style Guide:

Python: **PEP 8**, check with **black** and **flake8**
Dart: **Dart Official Style Guide**, check with **dart format** and **dart analyze**

Test Coverage:

Python Backend: use the **coverage.py** library to calculate test coverage
Dart Frontend: use **flutter test --coverage command** to generate test coverage reports and **lcov** tool to visualize the reports

Pull Requests:

will be reviewed weekly by at least one team member, with GitHub's auto-assign feature ensuring fair distribution. To avoid merge conflicts, we'll use "Rebase and Merge" and ensure all branches are synced with the main branch before development.

Weekly Planning

Week # (Date)	Tasks
Week 1 (2/24-2/28)	Set up GitHub repository and project structure. Design database schema for user data, swap posts, and match history.
Week 2 (3/3-3/7)	Develop basic UI components (e.g., login, registration, and home page) using Flutter. Implement state management with the provider library.
Week 3 (3/10-3/14)	Set up FastAPI backend with basic CRUD operations for swap posts. Integrate PostgreSQL database using SQLAlchemy.
(3/17 - 3/21)	
Week 4 (3/24-3/28)	Implement frontend features for posting and browsing swap requests. Develop backend APIs to handle swap post creation, retrieval, and updates.
Week 5 (3/31-4/4)	Set up OpenAI embeddings for semantic matching of swap posts. Integrate FAISS for efficient vector search and matching.
Week 6 (4/7-4/11)	Develop backend logic to calculate match scores using AI. Display recommended swap posts on the frontend.
Week 7 (4/14-4/18)	Implement two-way swap confirmation logic in the backend. Add a 5-star rating system for swap partners.
Week 8 (4/21-4/25)	Develop a feature to track swap history for users. Implement real-time notifications for new matches and updates.
Week 9 (4/28-5/2)	Conduct unit and integration tests for all components. Fix bugs and optimize performance.
Week 10 (5/5-5/9)	Complete documentation and prepare a project demo. Deploy the platform and conduct final testing.

Potential Risks

1. Backend-to-Frontend Integration Challenges:
   We may experience difficulties when calling backend methods from the frontend, as this is a new area for our team. If such issues arise, our plan is to isolate the problem through controlled testing and then consult with our mentor, who has relevant experience. We expect that resolving this could delay our project by roughly 1–2 days, though it should not derail our overall progress.
2. AI Recommendation System Performance:
   Our AI matching feature might not perform as accurately as we hope due to limited training data or unforeseen algorithmic challenges. If this risk materializes, we will consider either reducing the scope of the AI functionality or pivoting to a different approach. This could result in a delay of approximately 1–2 weeks, and we will adjust our schedule by reassigning tasks to ensure that other components of the project remain on track.
3. Scalability and Concurrency Issues:
   The real-time exchange mechanism and two-way confirmation process might face performance

problems under high load conditions. In such cases, we plan to conduct thorough stress testing and implement necessary optimizations—such as load balancing or improved queue management—to address these issues. We estimate that this may add up to one additional week to our timeline, and we will update our project schedule accordingly.

Teamwork

To minimize friction and ensure consistency across our development efforts, our team will adopt a standardized Docker environment. This will allow us to maintain uniform package managers and development tools on every team member's machine, reducing the classic "it works on my machine" problem while still allowing everyone to use their preferred code editors.

With a team of four, we have decided to split into two sub-teams of two members each. One sub-team (for example, Jixin and Lewen) will focus on frontend development—designing and implementing the user interface, including the course swap listings and the two-way confirmation process. The other sub-team (for instance, Yuyangsong and Yuncong) will handle backend development, which involves building the API, managing the database, and implementing the core swap logic.