COMS 4020 – Computer Science Senior Project

Final Report – Spring 2025

Project Group: SD-14

Adam Hisel

Ryan Johnson

Zachary Mueterthies

Carter Parks

Morgan Prieskorn

Client: Learn-X

Elvis Kimara

Tadiwa Mbuwayesango

Thabang Pila

Instructor

Simanta Mitra

Teaching Assistants

Shaiqur Rahman

Yonas Sium

Chapter 1: Requirements

1.1 User Management

- The system shall allow users to register and log in using email and password.
- The system shall assign users a role of either "instructor" or "student" upon registration.
- The system shall restrict access to instructor-only features based on user roles.

1.2 Instructor Features

- The system shall allow instructors to create and manage courses.
- The system shall allow instructors to manage students
- The system shall allow instructors to upload PDF and audio files to each course.

1.3 Student Features

- The system shall allow students to join courses using an access code
- The system shall allow students to view course materials uploaded by instructors.
- The system shall generate a personalized version ("personalized_file") of uploaded content based on a student's persona.
- The system shall allow students to interact with a course-specific AI chatbot.
- The system shall prompt students to complete onboarding questions after their registration.
- The system shall store onboarding data such as learning goals, preferred topics, educational background, and interests.
- The system shall allow students to update their onboarding information at any time from their profile.

1.4 Chatbot Features

- The system shall allow the chatbot to respond to course-relevant questions using uploaded course materials.
- The system shall instruct the chatbot to deny answering questions unrelated to course content.
- The system shall display chat history per user and course module.

1.5 Content Personalization

- The system shall use AI (RAG and prompt engineering) to summarize and restructure uploaded materials into a personalized format.
- The system shall generate learning modules with chapter titles and detailed subsections for each uploaded file.

1.6 File Preview & Navigation

- The system shall allow users to preview PDF files within the browser.
- The system shall allow students to return to previously viewed or personalized files.

Chapter 2: System Design

2.1 High-Level Architecture

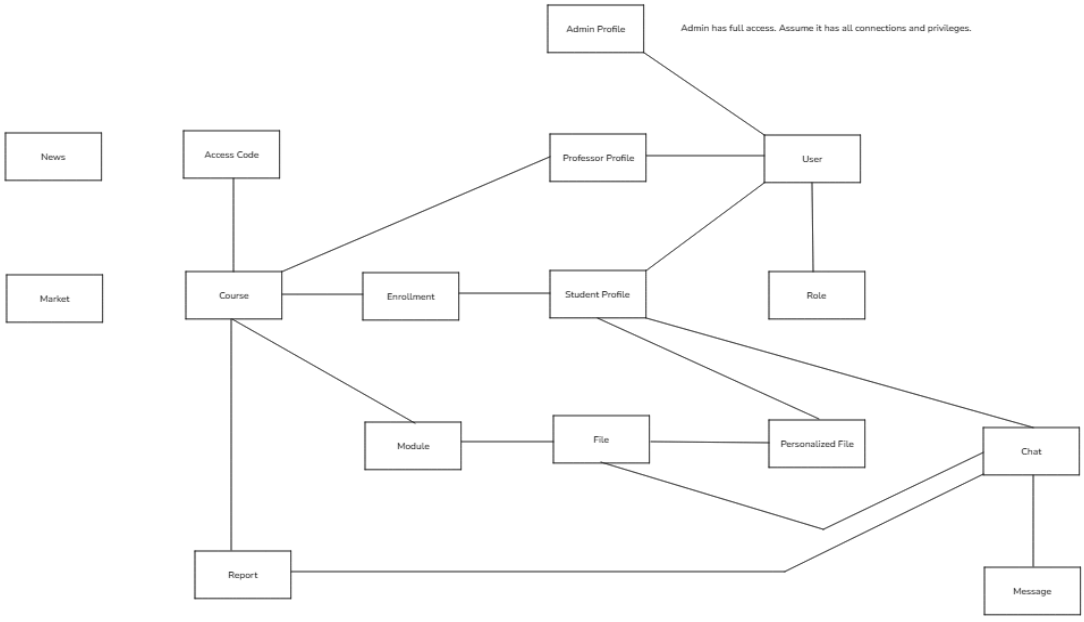

Figure 1: Entity-Relationship Diagram

This diagram illustrates the core data entities and their relationships within the system. It reflects the structure of our backend database and how user roles, courses, modules, and chatbot interactions are connected.

2.2 Technology Stack

- Next.js
- Python
- Flask
- Firebase
- PostreSQL
- Docker
- CSS
- OpenAI
- FAISS
- RAG
- Loveable
- V0
- Pgvector
- Github
- Gitlab
- Neon

2.3 API

Our website uses React for functionality, Next.js routing for page traversal, Client-side Web APIs for fetching, Server-side Flask APIs for returning, SQLAlchemy for database interaction, Firebase API for authentication, OpenAI API for prompting, and several more APIs for basic functionality.

2.3.1 React

Provides built-in interfaces for networking (fetch), cookie handling, and other browser capabilities used when calling endpoints or manipulating storage.

2.3.2 Next.js

Framework hooks and utilities (e.g. useRouter, file-based API routes) that handle client-side routing, server-side rendering, and lightweight serverless functions.

2.3.3 Client-Side Web

Provides built-in interfaces for networking (fetch), cookie handling, and other browser capabilities used when calling endpoints or manipulating storage.

2.3.4 Firebase Auth (Client and Admin)

Firebase Web SDK methods for email/password sign-in, ID token retrieval, and authentication flows in the browser. Server-side Firebase methods (verify_id_token, create_session_cookie, verify_session_cookie) for authenticating requests and managing sessions.

2.3.5 Sonner/Toast

Utility functions for displaying success and error toasts in response to application events.

2.3.6 Flask

Route decorators, request parsing (request.json, request.files), and response helpers (jsonify, status codes) that define and handle HTTP endpoints.

2.3.7 CORS

Configuration calls to enable cross-origin requests and credential support in Flask.

2.3.8 SQLAlchemy

Low-level query constructors (select(), func, text) for composing expressive SQL and vector similarity searches. Session methods (add, commit, refresh, execute, bulk_save_objects) and model-instance operations used to create, update, and delete records.

2.3.9 OpenAI

Client library calls (chat.completions.create) to generate AI-driven responses and personalized content.

2.3.10 FAISS and Embedding

FAISS functions and custom helpers (rebuild_file_index, openai_embed_text) for building, loading, and querying vector indices.

2.3.11 PSL

Modules (os, uuid, tempfile, shutil, json, datetime) and load_dotenv for environment configuration, temporary file management, JSON parsing, and timestamp handling.

2.3.12 Werkzeug Security

The generate_password_hash utility for secure password hashing prior to database storage.

2.4 Security and Data Management

All authentication, authorization, and data storage are performed from the backend using Firebase Auth and PostgreSQL.

2.4.1 Authentication

- Client-side sign-in is handled via Firebase Auth (email/password), which returns an ID token.
- The backend's /sessionLogin endpoint verifies that ID token with the Firebase Admin SDK and issues a secure, HTTP-only session cookie.
- Subsequent requests read and verify that session cookie (auth.verify_session_cookie), rejecting any missing or invalid sessions (401).

2.4.2 Authorization

- A small middleware layer (verify_role) extracts the Firebase UID from the session, looks up the corresponding user record in PostgreSQL, and fetches their Role.
- Routes are grouped by required role—student or instructor or admin—so that unauthorized users receive a 403 before any business logic runs.
- Within instructor-only endpoints, ownership checks ensure that the instructor's user ID matches the resource's instructor_id.

2.4.3 Data Storage

- Persistent data lives in PostgreSQL, accessed via SQLAlchemy ORM (models defined in src.db.schema) and Session objects.
- CRUD logic is encapsulated in src.db.queries, using ORM methods and occasional raw SQL/text queries (e.g. vector similarity via ORDER BY embedding <-> :query_vec).
- FAISS indexes and pickled embeddings are stored as byte arrays in dedicated table columns, rebuilt on file uploads.
- Environment configuration (database URL, Firebase key path, OpenAI API key) is loaded via python-dotenv.

Chapter 3: Work Done by Each Team Member

3.1 Member 1 – Adam Hisel

- **Technologies Learned:** PostgreSQL, OpenAI, Docker, Next.js, Flask
- **Contributions:**
    - Developed and refined core **professor-side functionality**, including course and module creation, file upload interfaces, and instructor dashboards.
    - Led the design and implementation of the **student learning experience**, creating intuitive pages for file viewing, personalization access, and chatbot interaction.
    - Built and styled the **"Learn" page**, ensuring responsive design, clean layout, and seamless navigation between modules and personalized files.
    - Implemented an **audio transcriber**, enabling instructors to upload audio files and convert them into structured text for personalized learning experiences.
- **Challenges Overcome:**
    - Learning OpenAI prompting
    - Redesigning frontend aspects late in the semester

3.2 Member 2 – Ryan Johnson

- **Technologies Learned:** PostgreSQL, Docker, Flask, Next.js, Firebase
- **Contributions:**
    - Developed website backend (CRUD API).
    - Designed and created databases for PostgreSQL and Firebase Auth.
    - Developed basic analytics and report creation system using OpenAI prompting.
    - Implemented multi-user security in the backend working with Firebase and PostgreSQL.

- **Challenges Overcome:**
    - Performed multiple backend and database redesigns due to changing expectations.
    - Transferred initial website backend to Docker from Next.js.
    - Designed multi-user and multi-database security in the backend.

3.3 Member 3 – Morgan Prieskorn

- **Technologies Learned:** Next.js, PostgreSQL, Docker, OpenAI, Flask

- **Contributions:** Designed frontend dashboard, implemented file upload and preview
    - Designed the Landing Page (twice)
    - Created onboarding questions and implemented onboarding page
    - Designed settings page. Enabled users to change their email, password, and onboarding answers.
    - Created the courses page, enabling students to join courses and view the courses they are in
    - Collaborated with Adam on the student dashboard, connected it to the backend

- **Challenges Overcome:**
    - Gettings a consistent UI/redesigning the UI
    - Changing the frontend to match the new backend
    - Creating the landing page design and coming up with ideas for what the product does/how to sell the product

3.4 Member 4 – Carter Parks

- **Technologies Learned:** Next.js, OpenAI, PGVector, PostgreSQL, Flask, Docker
- **Contributions:**
    - Implemented frontend and backend for **personalized chatbot**

    - Implemented a vector database to store course content chunks for relevancy retrieval
    - Designed and built the original **student dashboard**
    - Styled the learning content, chatbot, and dashboard components
- **Challenges Overcome:**
    - Performed multiple frontend redesigns to meet changing expectations.
    - Became an expert on **PGVector** and **OpenAI** prompting, both new technologies for me this semester

3.5 Member 5 – Zachary Mueterthies

- **Technologies Learned:** Next.js, PostgreSQL, FAISS, RAG, prompt engineering, Docker, Flask
- **Contributions:**
    - Designed the original landing page w/ routing
    - Wrote Docker compose file to simplify running the container
    - Rewrote FAISS scripts provided by clients to meet their requirements
    - Implemented RAG
    - Designed structured prompts to ensure consistent output for course content generation
    - Utilizing BOTH course content and user persona when generating modules
    - Implemented endpoints, ensuring compatibility with the frontend and FAISS methods.
    - Storing PDF content on the DB instead of locally
- **Challenges Overcome:**
    - Sudden changes of direction from the clients, requiring a lot of rewritten code
    - Structuring prompts so OpenAI follows as many of the instructions as possible
    - Understanding FAISS & RAG, which were new concepts to me

Chapter 4: Results Achieved

4.1 Completed Features

- User Management
- Professor Functionality

    - **Creating Courses:** Professors are able to create a course with details like course number and description. They can create modules in a course, and upload files to modules.
    - **Reports:** Analytics on student interaction and chatbot usage are gathered.
- Student Functionality
    - **Onboarding Experience**: Upon creating an account, student users are given a short onboarding quiz where Learn-X gathers basic information about the student's preferred learning style and interests.
    - **Course Organization**: Students are able to join a professor's course by access code, then view all course modules and content.
    - **Learning Experience:** Each student in a course gets their own personalized copy of each file. This file contains all content in the original file, but examples are given based on the user's interests gathered in the onboarding quiz. Students can also interact with an AI-chatbot which has access to their learning persona and the professor's course content.

4.2 Project Impact

- The final product met the goals multiple times, but the goals kept changing on us.
- The product started as a personalized learning platform that users would sign up for. We had that almost completed, but then they pivoted to Learn-X
- That had multiple types of users and we had to redesign the whole project to work with that
- We now have it personalizing files. Professors can upload files from their lectures and students can view them and personalize them
- The students take an onboarding quiz and their answers are used to personalize their files
- This was the main goal the clients wanted from us.
- We were able to redo the project and create all the main functionalities of the product
- All the main goals were fulfilled:
    - Landing page - sells the project to potential clients, giving information about what it can do to help students learn
    - Onboarding page - takes information from the students to personalize their files
    - Student dashboard - shows courses and modules and files for students. Can access personalized files from here
    - Professor dashboard - can create courses and add modules and files

    - Courses page - can join courses as a student
    - Settings page - can change user information
    - Learn page - the personalized file and the chatbot
- These were the main functionalities that they wanted
- Next Steps:
    - Implement analytics on the frontend
    - Deploy the project
    - Add deleting modules, files, etc.
    - Improve the personalization to make it sound more natural
    - Improve the length of the personalized file
    - Add privacy information and other legal information
    - Add extraneous information, contact page, etc.

4.3 Lessons Learned

- The main thing we learned was to be flexible and to meet changing requirements
- We learned to assign work to each other's strengths and work together collaboratively to complete the required elements
- We learned to move on from old code and reuse what we could
- We learned pivot and go from a single user type application to a multiple user type application
- We learned to use many new tools and languages, like Next.js, openAI, Flask, etc.
- We learned to manage girl conflicts.
- We learned to prioritize critical tasks and order tasks so that what one person was working on couldn't be worked on until someone else finished their task
- We learned to pivot quickly. What the client wanted done one week could end up scrapped the next week (e.g the stock market functionality)