LEBANESE UNIVERSITY
FACULTY OF ENGINEERING III
ELECTRICAL AND ELECTRONIC DEPARTMENT

Adaptive Learning Platform: Personalized Education with Local AI

Spring - SEM VIII

Dr. Mohammed Aoude

Presented By:

Ali Rahme 6425

Mostafa Kabalan 6183

Spring 2023-2024

Contents

Introduction

1.1 Problem Statement

Traditional educational approaches often lack personalization, leading to disengagement and varied learning outcomes among students. Generic curricula and limited one-on-one support fail to address individual learning paces, strengths, and weaknesses. Students struggle to find relevant resources for specific topics, and teachers face challenges in efficiently generating contextualized assessments and tracking individual progress. The reliance on cloud-based AI solutions also introduces concerns regarding data privacy and accessibility in offline or resource-constrained environments.

1.2 Project Objectives

The primary objectives of this project are:

1. To develop a web-based Adaptive Learning Platform capable of delivering personalized educational content and tools.
2. To integrate a local-first Large Language Model (LLM) to ensure offline functionality and enhanced data privacy.
3. To implement a Retrieval Augmented Generation (RAG) system to provide highly accurate, context-specific responses from a curated knowledge base.
4. To enable AI-powered quiz generation tailored to specific courses and lessons.
5. To provide personalized YouTube video recommendations based on student performance and learning needs.
6. To design a role-based user management system for distinct Student and Doctor functionalities.

1.3 Scope of the Project

This report covers the design and implementation of the core features mentioned in the objectives. It details the backend API development using Flask and MySQL, frontend user interfaces, and the integration of various AI components. While the project lays a strong foundation for an adaptive learning system, advanced features like comprehensive grading analytics, sophisticated learning path recommendations, and real-time LLM-based feedback on quiz answers are outlined as future work. The focus remains on demonstrating the viability and benefits of a locally-powered, personalized educational platform.

Background Technologies

2.1 Adaptive Learning Concepts

Adaptive learning is an educational method that uses technology to modify learning content and experiences based on an individual learner's needs. It aims to accelerate or slow down learning, provide remediation, or offer advanced challenges based on performance data and learning styles. Key principles include personalized pathways, immediate feedback, and continuous assessment.

2.2 Web Framework: Flask

Flask is a lightweight Python web framework that provides the tools and libraries necessary to build web applications. Its minimalism and flexibility make it suitable for rapid development and for building modular components. In this project, Flask serves as the backbone of our back-end, handling HTTP requests, routing, database interactions, and rendering dynamic HTML templates.

2.3 Database Management: MySQL

MySQL is an open-source relational database management system (RDBMS) widely used for web applications. It offers robust, scalable, and reliable data storage. For our project, MySQL stores all structured data, including user profiles, course information, quiz details, and chat histories. `Flask-MySQLdb` is used as the Python connector, configured with `DictCursor` for convenient dictionary-like access to query results.

2.4 Local Large Language Models (LLMs) & `llama.cpp`

Large Language Models (LLMs) are deep learning models trained on vast amounts of text data, capable of understanding, generating, and processing human language. For our project, running LLMs locally was a core constraint to ensure offline capability, privacy, and cost-effectiveness.

- `llama.cpp`: A C/C++ port of Facebook's LLaMA model that allows LLMs to run efficiently on CPUs, and optionally with GPU acceleration (though our project specifically targeted CPU-only). It uses GGUF (GGML Universal File Format) for optimized model weights.

- $\texttt{llama}_{c}pp-python : The Python bindings for \texttt{llama.cpp}, enabling seamless integration of local LLM inferenced$
  $following capabilities, making it suitable for both chatbot interactions and structured content generation (like quiz$

2.5 Embeddings (`sentence-transformers`) & Vector Databases (`ChromaDB`)

- **Embeddings:** Numerical representations of text that capture semantic meaning. Texts with similar meanings have embeddings that are close in vector space.
- `sentence-transformers`: A Python library for generating high-quality sentence embeddings. It provides access to a wide range of pre-trained models (e.g., `all-MiniLM-L6-v2`) specifically optimized for semantic similarity tasks, which is crucial for RAG.
- **Vector Databases:** Specialized databases designed to store and efficiently query vector embeddings based on similarity (e.g., cosine similarity).
- `ChromaDB`: An open-source, lightweight, and persistent vector database. It was chosen for its ease of local setup and strong integration with RAG pipelines.

2.6 External APIs: YouTube Data API

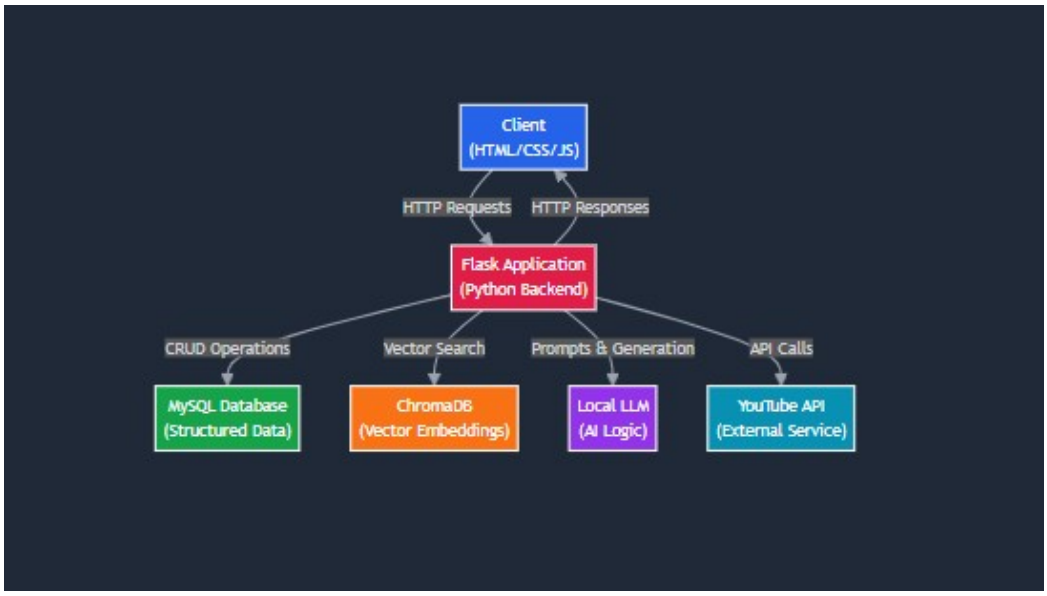
To provide personalized video recommendations, the project integrates with the YouTube Data API v3. This API allows programmatic searching for YouTube videos based on specific queries, retrieving video titles, IDs, and thumbnail URLs. An API key obtained from the Google Cloud Console is required for access, with proper restrictions for security.

System Design

3.1 Overall Architecture

The system follows a typical client-server web application architecture with a three-tier logical separation:

- **Frontend (Client-Side):** HTML, CSS, and JavaScript responsible for the user interface, user input, displaying information, and making asynchronous requests to the backend.
- **Backend (Server-Side):** Python Flask application handling business logic, user authentication, data management, and AI component orchestration.
- **Database Layer:** MySQL for structured data (users, quizzes, courses) and ChromaDB for vector embeddings (knowledge base).



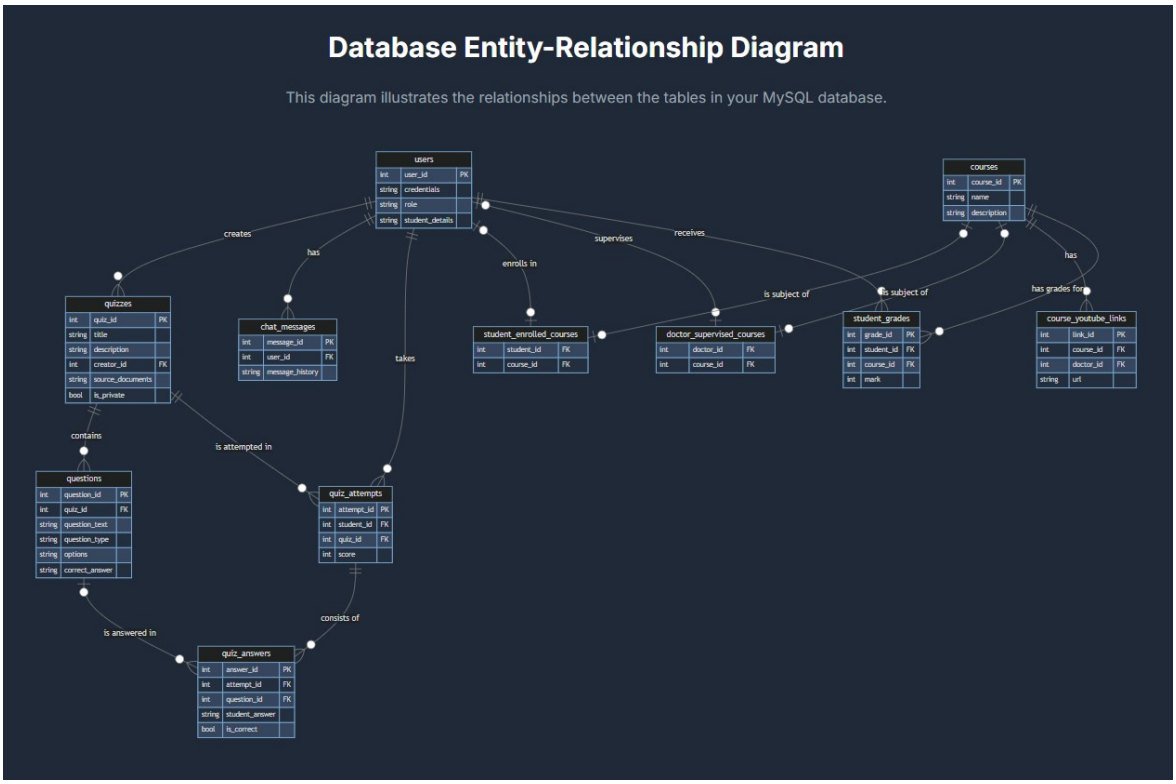
Overall System Architecture Diagram

3.2 Database Schema

The MySQL database stores critical user, course, and quiz-related data. The schema is designed to support personalization and persistence.

- `users`: Stores user profiles, including authentication credentials, role (student/doctor), and student-specific details (`year_of_study`, `major`).
- `courses`: Defines the academic courses available in the system.

- `quizzes`: Stores metadata for generated quizzes, including title, description, creator, source documents (if RAG-based), and privacy status (`is_private`).
- `questions`: Stores individual questions, linked to quizzes, with question text, type, options (for MCQ), and correct answer.
- `student_grades`: Tracks student performance by recording exam marks for specific chapters/courses.
- `student_enrolled_courses`: An association table linking students to multiple courses.
- `doctor_supervised_courses`: An association table linking doctors to courses they teach.
- `course_youtube_links`: Stores YouTube links added by doctors, associated with specific courses.
- `chat_messages`: Stores persistent chat history for each user.
- `quiz_attempts`: Records each time a student takes a quiz, including their score.
- `quiz_answers`: Stores individual student answers for each question within a quiz attempt, along with correctness.



Simplified Entity-Relationship Diagram (ERD)

3.3 Module Breakdown & Responsibilities

The Python Flask application is organized into several modules for clarity and maintainability:

- `app.py`: The main Flask application instance; handles configuration loading, database initialization, table creation, LLM/RAG component loading, and blueprint registration.

- `config.py`: Stores all application-wide configuration variables (database credentials, API keys, file paths, LLM parameters).
- `database.py`: Manages the MySQL database connection object.
- `models.py`: Defines Python classes that map to database tables (`User`, `Quiz`, `Question`, etc.), encapsulating all database interaction logic (CRUD methods).
- `routes.py`: Contains all Flask route definitions (`@main_bp.route`), handling HTTP requests, rendering templates, processing forms, and orchestrating calls to other modules.
- `rag_utils.py`: Encapsulates RAG-specific functionalities, including document loading (PDF, DOCX, TXT), text splitting, embedding generation, ingestion into ChromaDB, and context retrieval.
- `youtube_utils.py`: Handles all YouTube Data API interactions, including searching for videos and orchestrating LLM-powered recommendations based on student data.

Implementation Details

4.1 User Authentication & Role Management

The system provides a secure user authentication flow. User roles (Student/Doctor) dictate access permissions and features.

- **Registration:** Users provide name, email, password (hashed using `Werkzeug.security`), and select their role. Students provide additional details like year of study and major, and select 5 enrolled courses. Doctors select at least 3 supervised courses.
- **Login/Logout:** Standard session-based authentication. Users are redirected to role-specific starting points (chatbot for students, document ingestion for doctors) upon successful login.
- **Profile Management:** Users can view and update their personal information and associated courses.
- **Doctor Onboarding:** After initial doctor registration, the system redirects to a dedicated page (`doctor_initial_setup.html`) where doctors are prompted to upload at least one learning document for each supervised course. These documents are immediately ingested into the RAG knowledge base.

4.2 Core Chatbot Functionality

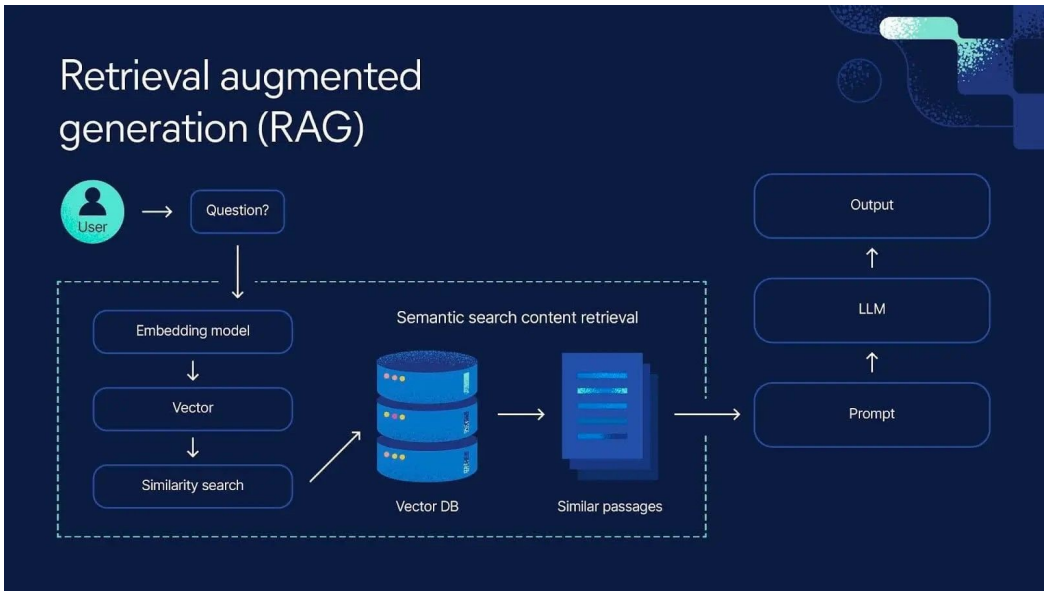
The chatbot is a central interaction point, offering dynamic responses.

- **Local LLM Integration:** The chatbot directly interfaces with a locally loaded Mistral-7B model (via `llama_cpp-python`). This ensures offline capability and data privacy.
- **Streaming Responses:** To improve user experience given CPU-only inference, the chatbot utilizes server-sent events to stream LLM responses word-by-word to the frontend.
- **Persistent Chat History:** All user and assistant messages are saved to the `chat_messages` table in MySQL, allowing conversations to persist across sessions and logins. Users can also clear their history.
- **Markdown Rendering:** Chatbot responses are formatted using Markdown, rendered on the frontend by `marked.js`, for improved readability (e.g., bold text, lists, code blocks).
- **Image Input (OCR):** The chatbot accepts image uploads. `EasyOCR` is used to extract text from these images, which is then sent as context to the LLM for question answering.
- **Document Input:** Supports text extraction from TXT, PDF (placeholder for full text), and DOCX (placeholder for full text) files, integrating content into the LLM's prompt.

4.3 Retrieval Augmented Generation (RAG) System

The RAG system enhances the LLM's ability to provide accurate and contextualized answers from specific learning materials.

- **Knowledge Base Ingestion:**
  - Doctors upload course-specific documents (PDF, TXT, DOCX) via dedicated pages (`ingest_documents.html`, `doctor_initial_setup.html`).
  - Documents are parsed (using `PyPDF2`, `python-docx` for initial text extraction).
  - Text is split into smaller, overlapping chunks.
  - `sentence-transformers` (e.g., `all-MiniLM-L6-v2`) converts these chunks into embeddings.
  - Embeddings are stored in `ChromaDB` with metadata including `source_filename` and `course_id`.
- **Context Retrieval & Augmentation:**
  - When a student asks a question in the chatbot, the query is embedded.
  - `ChromaDB` is queried for the top K (e.g., 3) most relevant document chunks, filtered specifically by the student's enrolled courses.
  - These retrieved chunks are added to the LLM's prompt as specific context, instructing the LLM to answer based solely on this information.
  - If no relevant context is found, the LLM falls back to its general knowledge.
- **Document Management:** Doctors can view and delete documents from the knowledge base, which removes both the file from disk and its embeddings from ChromaDB.



Overall System Architecture Diagram

4.4 Adaptive Assessment: Quiz Management

The platform allows for dynamic quiz generation and tracking.

- **Quiz Generation (Doctor-Initiated):**
  - Doctors can access a dedicated page (`generate_quiz.html`) to generate quizzes.
  - Inputs include a specific supervised course, lesson name, quiz title, number of questions (e.g., 3-5), and question type (Multiple Choice, True/False, Short Answer).
  - The LLM (Mistral) generates questions based on the input, optionally leveraging RAG context from documents associated with the chosen course.
  - Generated quizzes are stored in the `quizzes` and `questions` tables. Doctor-generated quizzes are public by default.
- **Quiz Generation (Student-Initiated):**
  - Students can initiate generation of a 3-question MCQ quiz on any of their enrolled courses and a specified lesson/topic (`student_generate_quiz_form.html`).
  - These quizzes are saved as **private** quizzes, visible only to the generating student.
  - After generation, the student is immediately redirected to take the newly generated quiz.
- **Quiz Taking & Grading:**
  - Students access quizzes via a list of public quizzes or their own generated quizzes.
  - The `take_quiz.html` template presents questions as an interactive form.
  - Upon submission, the backend grades the quiz (exact match for MCQ/True/False/Short Answer).
  - Quiz attempts (`quiz_attempts` table) and individual answers (`quiz_answers` table) are saved to the database, tracking student performance.
  - A results page (`quiz_results.html`) displays the score, percentage, and feedback for each question (student's answer vs. correct answer).

4.5 Personalized Video Recommendations

Students receive tailored YouTube video suggestions based on their academic context.

- **Student Input:** Students provide their enrolled course, a specific chapter/topic they are struggling with, and their recent exam mark for that topic. This data is saved to the `student_grades` table.
- **LLM-Powered Query Generation:** The Mistral LLM analyzes the student's input (including their mark and course context) to formulate an optimal, concise Youtube query (e.g., "Calculus I integral tutorials for beginners").
- **YouTube Data API Integration:** The system uses the Google API Client library to search YouTube with the LLM-generated query, retrieving video titles, URLs, and thumbnails.
- **Visual Display:** Recommendations are presented in an appealing, responsive card layout with video thumbnails for easy Browse.

4.6 UI/UX Enhancements

Throughout development, emphasis was placed on creating an intuitive and engaging user experience.

- **Consistent Layout:** A global pure CSS two-column layout with a fixed left sidebar for navigation on logged-in pages.
- **Centralized Forms:** Login, registration, and other standalone forms are prominently centered on their respective pages.
- **Streaming Chatbot:** Provides immediate, interactive responses.
- **Markdown Rendering:** Enhances readability of AI-generated content.
- **Responsive Design:** Adapts layouts for various screen sizes (implicitly handled by flexible CSS units and media queries).
- **Clear Feedback:** Consistent use of Flask flash messages for user notifications.

Testing Evaluation

5.1 Functional Testing

The application underwent iterative functional testing throughout its development. Each major feature was tested to ensure it met its objectives:

- **User Authentication:** Successful registration, login, logout, and profile updates for both Student and Doctor roles. Verification of role-specific access controls.
- **Database Persistence:** Confirmation that user data, course associations, chat messages, quizzes, questions, quiz attempts, and answers are correctly stored and retrieved from MySQL.
- **LLM Chatbot:** Correct initiation of the Mistral model, streaming responses, and accurate retrieval of RAG context based on user queries and enrolled courses.
- **Document Ingestion:** Successful upload, processing (text extraction, chunking, embedding), and storage in ChromaDB for PDF, TXT, and DOCX files. Verified deletion of documents from both file system and ChromaDB.
- **Quiz Generation:** Doctors successfully generating quizzes based on supervised courses and lessons. Students successfully generating private quizzes. Verification of LLM's adherence to question formatting and type (MCQ, True/False, Short Answer).
- **Quiz Taking Grading:** Students able to take quizzes, submit answers, and receive basic automated grading and feedback on a results page.
- **Video Recommendations:** Students successfully receiving personalized YouTube video links based on input course, chapter, and grade. Verified correct YouTube API interaction.
- **UI/UX Elements:** Functionality of clear chat, dynamic form fields, and sidebar navigation confirmed across different pages.
- **Specific Test Cases (Examples for your report - adapt these):**
  - Register Student (Email: `student@example.com`, Major: CS, 5+ courses). Verify profile.
  - Register Doctor (Email: `doctor@example.com`, 3+ courses). Complete initial document upload for at least one course. Verify document appears on Ingest page.
  - Chat as Student: "What is photosynthesis?" (if doc ingested) → Contextual answer. "Can you find me a video on Python loops?" → YouTube links appear.
  - Doctor: Generate MCQ quiz on "Calculus I" / "Derivatives" (3 questions). Verify quiz appears on "Available Quizzes".
  - Student: Take generated quiz. Answer correctly/incorrectly. Verify score and per-question feedback on results page.

5.2 Performance Observations

Performance was a key consideration due to the CPU-only constraint.

- **LLM Inference:** The Mistral-7B (Q2_K) model, while functional on CPU, exhibits noticeable latency.
  - Prompt evaluation: Typically takes 80-120 ms per token, leading to several seconds for initial prompt processing.
  - Generation: Around 2-5 tokens per second.
  - Result: Chatbot responses stream, making them feel interactive. Full quiz generation can take 1-3 minutes for 3-5 questions.
- **OCR (EasyOCR):** Text extraction from images is performant for clear, printed text, typically taking a few seconds per image. Performance degrades with image quality and text density.
- **Database Operations:** MySQL and ChromaDB operations are generally fast for the current scale of data. LLM/RAG initialization at app startup (loading models, ChromaDB) can take 1-2 minutes.

5.3 Limitations & Challenges Faced

The development process encountered several challenges inherent in local AI integration and web development:

- **CPU-Only LLM Performance:** The primary limitation for inference speed. This was mitigated by choosing highly quantized models and implementing streaming.
- **Database Schema Mismatches:** Recurring `MySQLdb.OperationalError: Unknown column` issues due to iterative schema changes. Resolved by strict table dropping/recreation and explicit error handling in `app.py`.
- **Flask Session vs. Streaming Context:** `RuntimeError` when modifying session during streaming. Fixed by externalizing session updates for bot responses.
- **LLM Output Parsing Brittleness:** Initial string-based parsing for quizzes and YouTube queries was highly sensitive to LLM output variations. Resolved by using flexible Regular Expressions (`re` module) and extensive prompt engineering to guide the LLM into consistent formats.
- **External API Integration:** Troubleshooting `NameError` for YouTube API (due to persistent incorrect URL string), and ensuring correct API key configuration.
- **Frontend Development with Custom CSS:** Significant effort required for manual CSS layout (sidebar, centering forms, cards) after abandoning Bootstrap, to achieve desired responsiveness and aesthetics.
- **Jinja2 Template Quirks:** Debugging `UndefinedError: loop.parent` in nested template loops, resolved by careful index variable passing.

Conclusion Future Work

6.1 Project Summary

This project successfully designed and implemented an Adaptive Learning Platform that leverages local AI to provide personalized educational tools. The platform offers secure user management, an intelligent chatbot augmented with RAG from a custom knowledge base, dynamic quiz generation and assessment, and personalized video recommendations. All core AI functionalities operate offline, adhering to principles of privacy and accessibility.

6.2 Achievements

- Developed a fully functional web application demonstrating core adaptive learning principles.
- Successfully integrated and orchestrated multiple local AI components: `llama_cpp-python` (LLM), `sentence-transformers` (embeddings), `ChromaDB` (vector database), and `EasyOCR` (image text extraction).
- Implemented robust data persistence for chat history, quizzes, and student performance using MySQL.
- Created intuitive, role-specific user interfaces and navigation flows.
- Overcame complex challenges related to AI integration, database management, and frontend/backend synchronization, providing valuable lessons in full-stack development.

6.3 Future Enhancements

The current platform provides a strong foundation. Potential future enhancements include:

- **Advanced Quiz Grading:** Implementing LLM-based grading for short answers to provide richer, more qualitative feedback.
- **Learning Path Recommendations:** Developing an AI module to suggest personalized learning paths or next topics based on student quiz performance and chat history.
- **AI-Powered Feedback on Answers:** Providing dynamic, LLM-generated explanations for why a student's answer was correct or incorrect.
- **Comprehensive Doctor Tools:** Expanding the Doctor dashboard with tools to manage courses, add/edit learning materials directly, and view student progress analytics.
- **Collaboration Features:** Allowing doctors to share resources or collaborate on quiz creation.

- **User Interface Refinement:** Adopting a modern CSS framework (e.g., Tailwind CSS) for a more professional and streamlined visual design, or engaging a UI/UX designer.
- **Performance Optimization:** Further optimizing LLM inference (e.g., exploring quantizations, specific CPU flags, or eventually GPU offloading if hardware allows) and document processing.