MoodleMate - UIR Connect

Project Technical Report

An AI Learning Companion for Students

Prepared by:
Fatima Zahra Agendouil
Aghrouch Badr
Akhridi Ahmed

Prepared for:
Universite Internationale de Rabat (UIR)

Date:
January 13, 2026

Contents

1 Introduction

MoodleMate is an advanced Retrieval-Augmented Generation (RAG) chatbot designed specifically to assist university students. It solves the problem of information overload by ingesting locally stored PDF documents (such as lecture notes, syllabi, and lab manuals) and using the **Llama 3** Large Language Model (via Groq) to answer student questions based strictly on the content of those documents.

The project features a modern, UIR connect-inspired user interface and a robust backend architecture, ensuring a seamless and secure learning experience.

2 Key Features

- **Modern UI:** A professional, university-branded interface featuring the official UIR Navy and Gold color palette, with a clean light-themed dashboard.
- **Multi-PDF Support:** Automatically ingests multiple PDF files from the root directory.
- **RAG Intelligence:** Uses vector embeddings to find exact context for answers, eliminating generic internet noise.
- **Source Citations:** Every answer includes clickable chips linking to specific pages in the source PDF.
- **PDF Deep Linking:** Clicking a citation opens the PDF in a new tab, scrolled to the exact page of the answer.
- **Persistent History:** Chat sessions and individual messages are saved in a local SQLite database for future reference.

3 AI & RAG Technology Deep Dive

This MVP relies on a sophisticated stack of AI technologies. Below is a detailed breakdown of each component:

1. Large Language Model (LLM)

Tool: Llama 3.1 8B (via Groq)
Concept: An LLM is a deep learning algorithm that can recognize, summarize, translate, and generate text based on knowledge gained from massive datasets.
Role in MoodleMate: It acts as the **'Reasoning Engine'**. It does not memorize your PDFs; instead, it reads the specific snippets we send and formulates a coherent, human-like answer based *only* on that content. We use Groq for ultra-fast inference.

2. Vector Embeddings

Tool: HuggingFace (all-MiniLM-L6-v2)
Concept: Embeddings convert text into long lists of numbers (vectors). Similar ideas end up with similar numbers, allowing computers to understand 'semantic meaning' rather than just matching keywords.
Role in MoodleMate: When a PDF is ingested, this model turns every paragraph into a vector. When a question is asked, it is also turned into a vector. We mathematically compare them to find the most relevant paragraphs.

3. Vector Database

Tool: ChromaDB
Concept: A specialized database optimized for storing and querying high-dimensional vectors.
Role in MoodleMate: ChromaDB acts as the **'Long-Term Memory'** of the application. It stores the embedded representations of your PDF files locally, allowing for near-instant search and retrieval.

4. RAG Architecture

Concept: Retrieval-Augmented Generation enhances LLMs by fetching relevant data from external sources before generating a response.
Role in MoodleMate:

1. User asks a question.
2. System searches ChromaDB for the Top 3 relevant PDF chunks.
3. System creates a prompt: *"Using these 3 chunks, answer the user's question"*.
4. Llama 3 generates the answer, providing accurate, sourced information without hallucinations.

5. Orchestration

Tool: LangChain
Concept: A framework for developing applications powered by language models.
Role in MoodleMate: LangChain provides the **'Glue'** code. It handles the chains of logic: loading the PDF $\rightarrow$ splitting text $\rightarrow$ embedding $\rightarrow$ storing $\rightarrow$ retrieving $\rightarrow$ prompting.

4 UI DEMONSTRATION

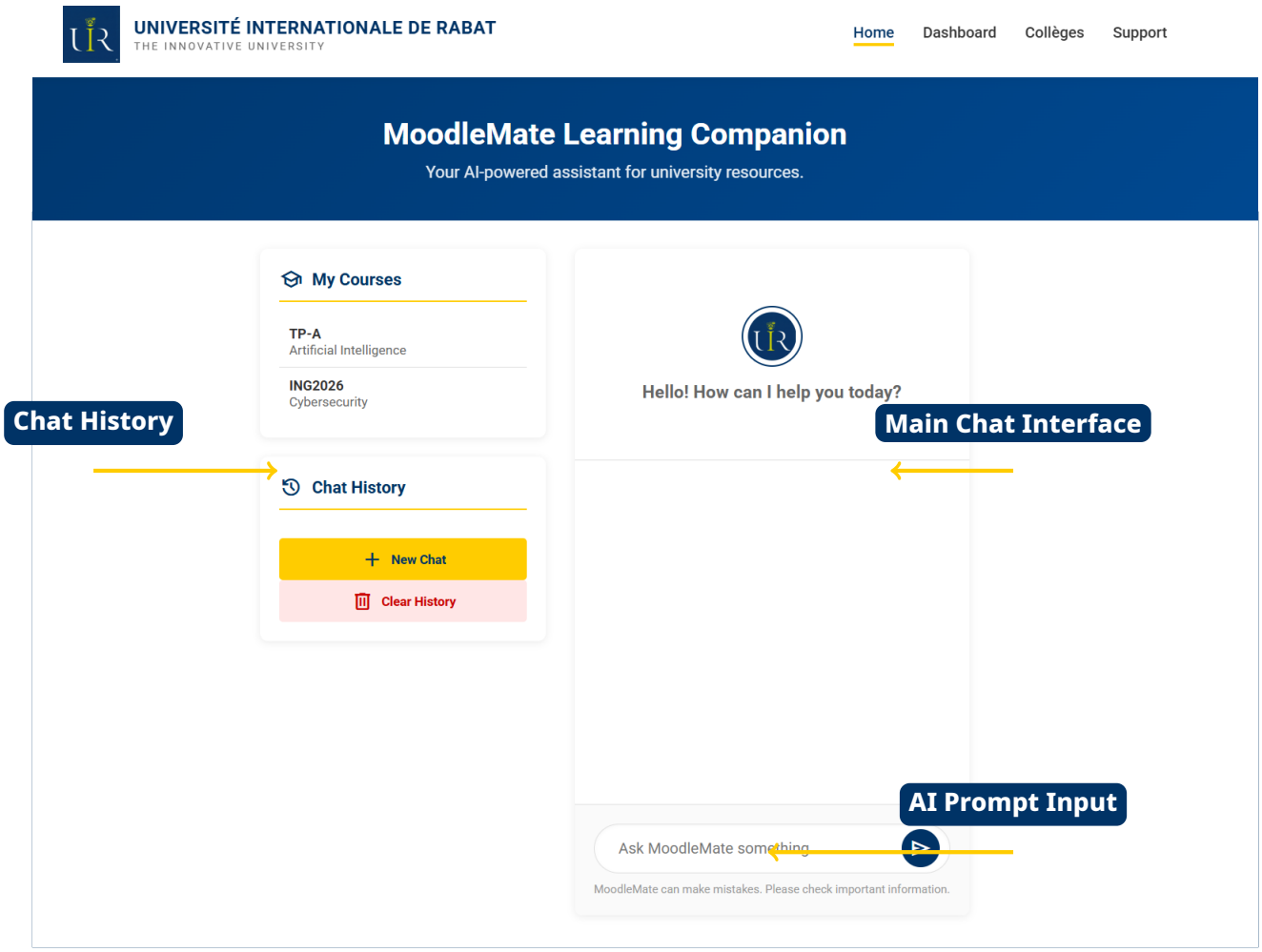

Figure 1: The MoodleMate User Interface with Key Components Highlighted

5 Technical Architecture

5.1 Tech Stack Summary

- **Back-End Framework:** FastAPI (High performance Python web framework)
- **Database:** SQLite (For Chat History persistence)
- **Frontend:** Vanilla HTML/JS with Gemini-inspired CSS
- **API Key Management:** python-dotenv

5.2 Project File Roles

A breakdown of the project structure and the specific responsibility of each file:

1. app.py (The Backend)

Role: The main application server.
Function: Runs the Uvicorn/FastAPI web server, handles API requests, manages connections to Groq and ChromaDB, and handles SQLite database operations.

2. ingest.py (The Data Loader)

Role: The data preparation script.
Function: Runs once to "teach" the AI. It reads PDFs, chunks them, vectorizes them, and saves them to the `chroma_db` directory.

3. static/script.js (The Frontend Logic)

Role: Client-side interactivity.
Function: Handles user clicks, sends messages to the backend, updates the UI, and manages sidebar history.

4. chat.db

Role: The History Database.
Function: A local SQL file storing conversation sessions and messages.

5. chroma_db/ (Directory)

Role: The Knowledge Base.
Function: Stores the vector embeddings of the PDF contents.

6 Detailed Code Explanation

6.1 PART 1: app.py (The Backend)

1. **Imports (Lines 1-12):** We import `FastAPI` for the web server and `LangChain` components to connect the LLM (ChatGroq), Embeddings, and Vector Stores.
2. **Configuration:** `load_dotenv()` is used to read the secret API key. We define constants for paths and model names ('llama-3.1-8b-instant').
3. **Database Setup (init_db):** This function checks if 'chat.db' exists. If not, it creates two tables:
   - `sessions`: Stores ID and Title of each chat.
   - `messages`: Stores individual texts (User vs. Assistant).
4. **RAG Chain Setup (get_rag_chain):** This is the "Brain" of the app. It loads Chroma, initializes ChatGroq, creates a "Retriever", and defines the System Prompt ("You are an AI tutor... only use context..."). It chains these together so that data flows from Retriever to Prompt to LLM.
5. **Chat Endpoint (POST /chat):**
   - Creates a new Session ID if needed.
   - Calls `rag_chain.invoke(message)`.
   - Checks which source documents were used (Filename + Page #).

- Saves the transaction to SQLite.
- Returns the response and sources to the frontend.

6. **PDF Serving:** A secure endpoint `GET /pdfs/filename` that ensures users can only open local files intended for the course, returning them with the correct MIME type.

6.2 PART 2: ingest.py (The Data Loader)

1. **Cleaning:** It runs `shutil.rmtree(CHROMA_PATH)` to delete the old database. This ensures the knowledge base is always fresh and doesn't contain deleted files.
2. **Loading:** Uses `PyPDFLoader` to read every PDF in the root directory page-by-page.
3. **Splitting:** Uses `RecursiveCharacterTextSplitter` to break text into 1000-character chunks with a 200-character overlap. This overlap is crucial so that context isn't lost at the edges of a chunk.
4. **Embedding & Saving:** The heavy lifter `Chroma.from_documents()` converts chunks to vectors using the HuggingFace model and saves them to disk.

7 How MoodleMate Works: The Workflow

7.1 1. Data Ingestion

The process begins with `ingest.py`. The application scans the directory for files like "Lecture_2.pdf". It splits the text into manageable chunks and converts them into numerical vectors ("embeddings"). These are stored in `ChromaDB`, enabling semantic search.

7.2 2. The Backend Server

When `python app.py` runs, the FastAPI server starts. It establishes connections to the ChromaDB (for knowledge), the Groq API (for intelligence), and the SQLite database (for memory).

7.3 3. The Chat Experience

1. The user interacts with the system through a clean, university-branded chat interface designed for clarity.
2. The backend searches ChromaDB for the top 3 relevant text chunks.
3. It constructs a prompt: *"Here is the context: [Chunks]. Answer: [Question]"*
4. The Llama 3 AI generates a response based **only** on those chunks.
5. If successful, the app displays the answer with "Sources" (e.g., *Lecture_2.pdf (Page 5)*).

8 Conclusion

MoodleMate represents a state-of-the-art implementation of local RAG for education. By combining the speed of Groq's inference engine, the semantic accuracy of Vector Search, and a user-friendly design, it provides students with a powerful tool to navigate complex university materials efficiently and accurately.