



Design and Implementation of an Intelligent Chatbot Using Large Language Model for Consistent Learning Support

¹Buremoh, E. P., ²Olaoye, I.O.

¹Department of Computer Science and Engineering, University of Fairfax, Roanoke Virginia, USA.

² Department of Metallurgical and Materials Engineering, Federal University of Technology Akure, Nigeria

DOI: <https://doi.org/10.51584/IJRIAS.2026.11070200>

Received: 05 August 2026; Accepted: 10 August 2026; Published: 21 August 2026

ABSTRACT

This study developed an AI-enabled lecturer assistant system powered by Large Language Models (LLMs) to address challenges in the higher institutions, particularly high student-to-lecturer ratios and scheduling conflicts that hinder effective communication and timely academic support. An AI-driven chatbot platform was designed to provide continuous access to academic resources and personalized responses to student inquiries. Lecturers uploaded course materials, which the chatbot utilized through a retrieval-based model to generate context-aware explanations. MongoDB was employed to store interaction histories, enabling the system to adapt based on student feedback and learning patterns. The implemented system successfully delivered accurate, detailed responses aligned with academic content, improved accessibility to educational support, and enhanced learning efficiency through immediate feedback. It also reduced repetitive administrative tasks for lecturers, thereby allowing them to focus more on teaching and mentoring. The findings demonstrated the transformative potential of integrating advanced AI technologies into higher education, contributing to improved student engagement, equitable access to resources, and reduced instructional burdens. Overall, the study highlighted the societal impact of scalable, inclusive, and efficient AI-driven educational tools in fostering interactive and technologically adept learning environments.

Keywords: Chatbot, LLM, Learning, AI-driven, MongoDB

INTRODUCTION

In today's era of rapid technological advancement, the integration of digital tools into education has shifted from being optional to essential (Bates, 2019; Selwyn, 2016). This transformation is particularly important in addressing persistent challenges such as overcrowded classrooms and scheduling difficulties, which continue to undermine the delivery of quality education worldwide (UNESCO, 2018). Among the most influential innovations in this space is artificial intelligence (AI). Increasingly, AI technologies—especially chatbots—are recognized for their potential to strengthen communication between students and lecturers (Luckin et al., 2016; Winkler & Söllner, 2018). These advanced systems provide uninterrupted access to learning resources, academic support, and real-time interaction, all of which are vital for sustaining educational continuity and improving overall effectiveness (Holmes et al., 2019).

Developing an AI-powered chatbot requires careful design, rigorous testing, and ongoing optimization to guarantee accuracy and reliability. The primary objective is to strengthen student-teacher communication by offering timely and individualized academic assistance, even as student enrollment grows. Research highlights that chatbots are becoming more prevalent in higher education, where their adaptive intelligence can improve learning outcomes by tailoring interactions to each student's unique behaviors and preferences (Okonkwo & Ade-Ibijola, 2021; Woolf, 2010). Through natural language processing models such as GPT, these systems can summarize lecture notes, answer questions contextually, and interpret complex academic material, thereby enhancing how students engage with coursework (Brown et al., 2020).

The broader aim is to establish a learning environment that is dynamic, inclusive, and technologically advanced (UNESCO, 2020). Beyond addressing immediate challenges like limited lecturer availability, AI-driven chatbots can contribute to higher retention rates, greater student satisfaction, and improved academic performance (Zawacki-Richter et al., 2019). Successful implementation of such systems also provides a model for other institutions facing similar obstacles, demonstrating how AI can fundamentally reshape education delivery while promoting equity and accessibility (Holmes et al., 2019). Integrating chatbots into higher education is not simply a technological upgrade but a deliberate effort to prepare students for a future where digital literacy is indispensable (Selwyn, 2016). If effectively deployed, these systems could usher in a new era of AI-enhanced educational excellence, particularly in Nigerian universities and similar contexts where large class sizes and limited resources often hinder effective learning (Adebayo & Oyedele, 2020). While existing communication channels between students and lecturers provide some support, they remain inadequate for meeting diverse academic needs outside the classroom. Chatbots, by contrast, offer a flexible, context-aware solution that can deliver dependable assistance at scale (Okonkwo & Ade-Ibijola, 2021).

The central problem this research addresses is the lack of immediate, personalized support for students when lecturers are unavailable. High student-to-lecturer ratios and limited access to teachers outside scheduled lessons exacerbate this issue, leaving learners without sufficient guidance during critical study periods (UNESCO, 2018). To resolve this, the project proposes the design of a chatbot powered by advanced natural language processing techniques. Its role will be to respond to academic queries in real time, encourage effective communication, and provide tailored learning support (Brown et al., 2020). Ultimately, the system aims to enhance self-study skills, improve academic performance, and ensure consistent access to educational resources (Chen et al., 2020). The specific objectives of this project are: To design an AI-enabled chatbot capable of processing student inquiries and delivering real-time academic support; to implement the chatbot using advanced NLP techniques; and to evaluate the system's performance in real-world educational settings. The scope of the study focuses on developing an AI-powered lecturer assistant chatbot for higher education, leveraging large language models to support learning. While the project is initially limited to specific institutions, its framework can be expanded to other contexts in the future.

This study is important because it introduces an AI-driven chatbot platform with the potential to transform higher education. For students, it promises round-the-clock access to academic support, particularly during high-pressure periods such as examinations or project deadlines (Narciss, 2013). For lecturers, it reduces time spent on repetitive queries and administrative tasks, enabling them to devote more energy to direct instruction and personalized engagement (Holmes et al., 2019). For administrators, it offers insights into integrating advanced technologies for more efficient resource management and institutional modernization (Luckin et al., 2016). Developers in the educational technology sector can also benefit, using the findings as a foundation for creating scalable, innovative tools (Chen et al., 2020). At a societal level, the study contributes to building a more knowledgeable and skilled workforce by strengthening learning support systems (Zawacki-Richter et al., 2019). Moreover, it advances values of inclusivity, equity, and accessibility, ensuring that education remains responsive to diverse student needs (UNESCO, 2020).

Related Work

Machine Learning Techniques in Chatbots

Over the past ten years, artificial intelligence (AI) has evolved from an idea of the future to a practical asset affecting education across all levels (Luckin et al., 2016; Zawacki-Richter et al., 2019). The growing accessibility of digital learning tools, combined with the need for flexible educational access, has made AI especially pertinent in higher education institutions (Holmes et al., 2019). In simpler terms, AI systems are designed to replicate certain elements of human intelligence such as reasoning, decision-making, and problem-solving (Russell & Norvig, 2021). In the educational realm, these systems are being utilized to alleviate the routine burdens faced by instructors and provide students with a more tailored learning experience (Luckin et al., 2016).

One significant area where AI has made a notable difference is through the implementation of chatbots. A chatbot is an interactive program crafted to communicate with users in everyday language (Shawar & Atwell, 2007). In universities, chatbots serve roles as teaching aides, administrative assistants, and study companions (Woolf, 2010). For instance, a chatbot can assist a student in accessing information regarding exam schedules at any hour or delivering prompt feedback on straightforward assignments. This alleviates the workload on lecturers and staff while granting students quicker access to assistance (Okonkwo & Ade-Ibijola, 2021). Educational chatbots typically fall into two general categories. The first group is rule-based chatbots, which rely on established patterns and scripts. While they are straightforward to create, they are often restricted in their capabilities, as they struggle with unexpected or intricate inquiries (Shawar & Atwell, 2007). The second group consists of AI-driven chatbots, which utilize natural language processing (NLP) and machine learning techniques to comprehend questions, evolve over time, and offer more adaptable responses (Jurafsky & Martin, 2023). Contemporary systems, particularly those driven by extensive language models like GPT, are associated with this second category (Brown et al., 2020).

In practice, chatbots are being used in three primary domains of education: I. Student support services – addressing frequently asked questions regarding matters like registration and housing (Holmes et al., 2019). II. Academic tutoring – delivering explanations, supplementary exercises, and reminders beyond the classroom setting (Woolf, 2010). III.

Lecturer assistance – streamlining tasks such as grading multiple-choice tests, organizing attendance records, or summarizing discussion threads (Chen et al., 2020). Recent research indicates that students tend to experience greater satisfaction when chatbots are incorporated into their educational environment due to their ability to provide immediate answers and operate tirelessly (Okonkwo & Ade-Ibijola, 2021; Følstad & Brandtzaeg, 2017). This is especially beneficial in Nigerian universities, where large student populations and inadequate staffing create obstacles for timely academic support (Adebayo & Oyedele, 2020).

Supervised Learning

Supervised learning is among the most widely used techniques in developing chatbots, particularly for intent recognition and response generation (Goodfellow, Bengio, & Courville, 2016). This method involves training the model on a dataset that includes inputs (such as student questions) aligned with the correct outputs (like assigned intents or expected answers). In educational chatbots, supervised learning is often employed to create Natural Language Understanding (NLU) components (Chen et al., 2020).

For example, it can categorize questions into groups like “requesting lecture notes,” “inquiring about assignments,” or “seeking explanations of terms.” In addition to classification, supervised models can also be trained for named entity recognition (NER), enabling the chatbot to recognize essential terms such as course codes or instructor names (Jurafsky & Martin, 2023). The benefit of supervised learning lies in its high accuracy when given ample and appropriately labeled training data, though it depends heavily on dataset availability (Holmes et al., 2019).

Unsupervised Learning

Unsupervised learning operates without labeled data, detecting patterns and clusters within unprocessed queries (Aggarwal, 2018). In chatbot applications, it can automatically organize student queries into categories without prior labels, revealing prevalent concerns educators may not have foreseen (Chen et al., 2020). This is particularly beneficial for exploratory analysis and adapting to changing student demands.

Reinforcement Learning

Reinforcement learning (RL) draws inspiration from how humans and animals learn through rewards and punishments (Sutton & Barto, 2018). In chatbot systems, RL improves responses by engaging with users and receiving feedback. For educational chatbots, RL is effective in tutoring scenarios, where positive student feedback reinforces accurate explanations (Singh et al., 2002). RL’s adaptive capability makes it suitable for



personalized learning environments, though it requires careful reward design to avoid unintended behaviors (Holmes et al., 2019).

Deep Learning

Deep learning employs multi-layer neural networks to analyze complex data (LeCun, Bengio, & Hinton, 2015). It is highly effective for natural language tasks such as text classification, sentiment analysis, and dialogue generation. Educational chatbots increasingly utilize transformer-based frameworks like BERT, GPT, and T5 (Devlin et al., 2019; Brown et al., 2020; Raffel et al., 2020).

These models comprehend context, manage lengthy conversations, and generate coherent responses. Deep learning also enables advanced functionalities such as context management and sentiment analysis, though training and deployment are computationally expensive (Vaswani et al., 2017).

Information Retrieval Techniques

A crucial element of data management is how the chatbot accesses and delivers stored information. Educational chatbots typically depend on retrieval-based methods rather than generating answers entirely independently (Jurafsky & Martin, 2023). This approach helps ensure responses are factually accurate and based on content provided by lecturers.

Improving Interactivity through User Interface Design

In the realm of educational chatbots, interactivity is crucial for fostering user adoption, especially given that many students may already be under academic pressure and should not encounter additional obstacles when seeking help (Følstad & Brandtzaeg, 2017). An effective interface should be user-friendly, responsive, and accessible across various devices (Shneiderman et al., 2016). As students frequently use mobile phones to access academic resources, mobility-friendly chatbot designs are becoming increasingly essential (Okonkwo & Ade-Ibijola, 2021).

Important principles for creating interactive educational chatbots include:

- **Clarity:** Information should be presented in a legible format, utilizing simple fonts, adequate spacing, and helpful visuals when needed
- **Multimodality:** If possible, the chatbot should accommodate not only text but also voice input and output, enhancing accessibility for visually impaired students (Shneiderman et al., 2016).
- **Personalization:** Interfaces can tailor themselves to individual users, such as addressing students by their names or displaying their recently accessed materials (Chen et al., 2020).
- **Feedback Mechanisms:** Students should have the ability to indicate whether the provided answer was useful, enhancing both usability and the chatbot's underlying AI framework (Følstad & Brandtzaeg, 2017).
- **Error Recovery:** The UI should assist users when the chatbot fails to understand a request, for example, by proposing alternative inquiries or displaying related information (Holmes et al., 2019).

Real-world examples can be seen in current platforms. For instance, chatbots incorporated into Learning Management Systems (LMS) frequently offer quick-access buttons for assignments, deadlines, and lecture notes (Woolf, 2010). Likewise, mobile chatbots might employ conversational bubbles resembling familiar interfaces like WhatsApp, making them less daunting for new users (Adebayo & Oyedele, 2020). The Simplified Chatbot User Interface Layout is shown in Figure 1.

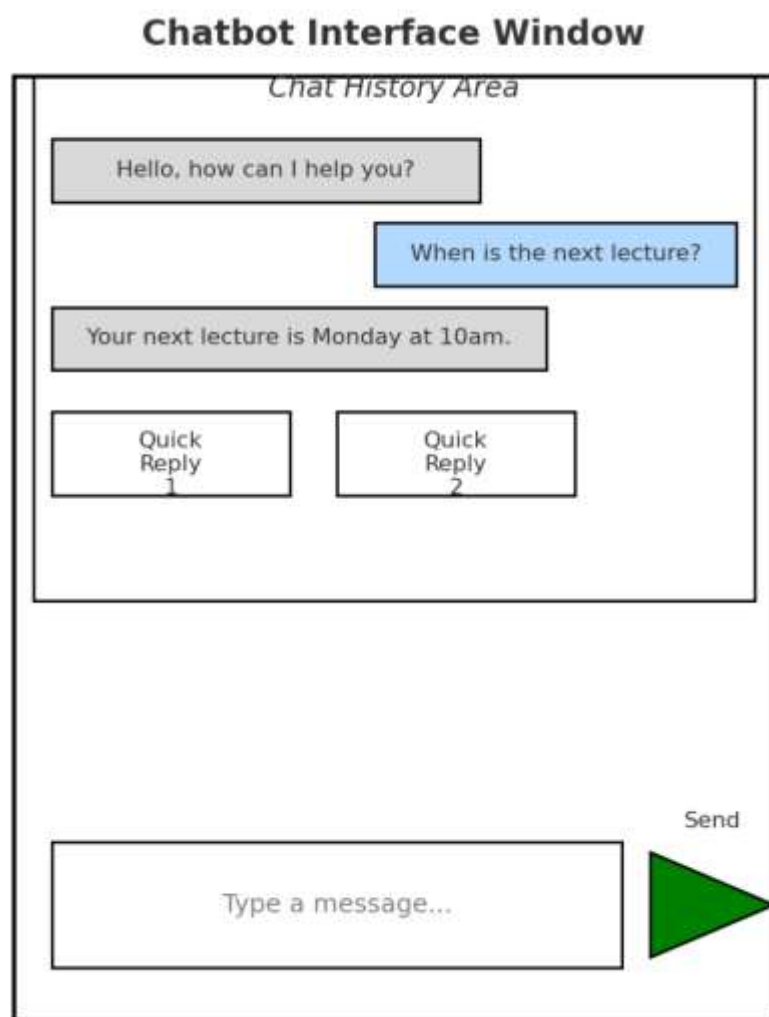


Figure 1: Simplified Chatbot User Interface Layout

AI-Driven Personalization Techniques One of the most significant advantages of incorporating artificial intelligence into education is the ability to provide personalized learning experiences (Chen et al., 2020; Zawacki-Richter et al., 2019). Unlike conventional classroom instruction, which often treats all students equally in terms of pace and teaching style, AI systems can customize content to suit the unique needs of each learner (Woolf, 2010). Chatbots that utilize personalization not only answer queries but also adjust to a student's individual learning style, progress, and challenges (Holmes et al., 2019). This enhances their effectiveness as virtual teaching assistants and keeps students engaged throughout their educational journey (Følstad & Brandtzaeg, 2017).

Adaptive Learning Algorithms Adaptive learning involves algorithms that modify the difficulty, pace, or type of content presented to a student based on their past performance (Brusilovsky & Millán, 2007). For instance, if a student frequently faces challenges with a specific topic like recursion in programming, the chatbot can identify this pattern and offer additional explanations, simpler examples, or extra practice tasks. On the other hand, if a student shows proficiency in a topic, the system can introduce more demanding problems to maintain their motivation.

Various methods are commonly employed in adaptive learning platforms:

- **Recommendation algorithms** propose relevant materials tailored to the learner's profile (Drachsler & Greller, 2016).
- **Knowledge tracing models** estimate how well a student has grasped a concept and predict their chances of correctly answering future questions (Piech et al., 2015).



- **Dynamic difficulty adjustment** alters the complexity of practice tasks based on learner performance, ensuring that the learner neither feels overwhelmed nor bored (Koedinger et al., 2013).

In higher education, adaptive learning has demonstrated improvements in retention rates and student satisfaction because it replicates the advantages of one-on-one tutoring on a larger scale (Chen et al., 2020). For universities in Nigeria, where the ratio of lecturers to students can be quite high, such customization could help address the disparity between limited human resources and varied learner requirements (Adebayo & Oyedele, 2020).

Feedback Mechanisms Feedback plays a crucial role in successful learning (Hattie & Timperley, 2007). In traditional settings, educators provide feedback through assignments, tests, and one-on-one discussions. However, this feedback can often be slow and inconsistent in larger classes. AI-powered chatbots have the capability to offer instant and ongoing feedback (Narciss, 2013).

Feedback mechanisms available in chatbots can manifest in several ways:

- **Corrective feedback** – When a student selects an incorrect answer, the chatbot instantly clarifies the error and offers hints for improvement (Shute, 2008).
- **Motivational feedback** – The chatbot can motivate learners through positive reinforcement (e.g., “Well done!” or “You’ve made progress since last time”) (Deci & Ryan, 2000).
- **Progress tracking** – By keeping a record of interactions, chatbots can deliver periodic reports to students, highlighting their strengths and areas for growth (Luckin et al., 2016).
- **Personal reminders** – Alerts about deadlines, assignments, or forthcoming tests help ensure that students stay organized (Okonkwo & Ade-Ibijola, 2021).

Research indicates that immediate and personalized feedback considerably enhances knowledge retention and boosts learner confidence (Narciss, 2013; Hattie & Timperley, 2007). Moreover, AI-driven feedback loops can also support educators by identifying common difficulties faced by students, allowing them to modify their teaching strategies (Chen et al., 2020). The integration of personalization through adaptive algorithms and real-time feedback transforms chatbots from mere question-and-answer tools into intelligent learning partners. By customizing support to fit the needs of each student, these techniques ensure that educational chatbots not only aid instructors but also significantly improve the overall learning experience in meaningful ways.

SYSTEM ANALYSIS AND DESIGN

System Analysis

System analysis consists of assessing the current methods of providing academic support, pinpointing their shortcomings, and detailing the enhancements offered by the suggested solution. This phase is crucial as it lays the groundwork for establishing the system requirements and design.

Existing System

In numerous Nigerian universities, the prevailing academic support framework is predominantly manual and relies on direct interactions between lecturers and students. Students generally depend on lectures, office consultations, notice boards, and departmental WhatsApp groups to gather information. Learning management systems (LMS), where they exist, are frequently underutilized or limited to the mere posting of lecture slides and course materials. When students seek clarification outside of class hours, they typically try to reach lecturers directly via email, social media, or personal messages. Often, lecturer responses are delayed due to their busy schedules and the high ratio of students to lecturers. While peer-to-peer communication can be beneficial, it is not consistently reliable as it may lack accuracy or informed insights.



Some faculties have made attempts to implement departmental portals or e-learning platforms to enhance communication. Nevertheless, these systems are generally static and offer minimal interactivity and personalization. Consequently, students continue to face challenges in obtaining timely and context-specific support.

Limitations of the Existing System

The current system has multiple drawbacks:

Delayed Responses

Students commonly experience waiting periods of hours or days for lecturers to reply to academic inquiries, which impedes self-study and preparation.

High Lecturer Workload

The substantial number of repetitive questions increases the burden on lecturers, diminishing their capacity for advanced teaching and research activities.

Lack of Personalization

The available systems do not cater to individual student requirements, learning rates, or understanding levels.

Limited Accessibility

Students are restricted to receiving help during set office hours, while digital platforms in use are either passive or poorly integrated with academic processes. Scalability Issues: As student enrollments rise, manual and semi-digital systems struggle to deliver timely support to all students.

Fragmented Communication Channels

Students and lecturers depend on various platforms (e.g., emails, WhatsApp, notice boards), resulting in inconsistent information dissemination.

These shortcomings underscore the pressing need for a more intelligent, responsive, and centralized system capable of ensuring consistent academic support.

Proposed System

The proposed system is a chatbot powered by AI, intended to assist lecturers by providing round-the-clock academic support through the use of large language model and natural language processing. In contrast to existing solutions, this centralized and interactive platform will cater to both students and Lecturers. It offers real-time engagement, enabling students to receive immediate and context-specific responses based on the uploaded course materials. Educators can access a specialized dashboard to upload content, and ensure the chatbot aligns with the educational curriculum. The system places a high priority on data protection, utilizing encryption and access controls, and is designed to integrate smoothly with current educational resources. In the end, the chatbot aims to alleviate some of the workload from lecturers by addressing routine inquiries, allowing them to concentrate on more complex academic responsibilities.

Features of the Proposed System

The suggested AI-driven lecturer assistant offers a unified text chat platform focused on courses for both students and instructors. Key features include:



24/7 Text Chat

Support Instant, conversational assistance regarding course topics, schedules, and assessments through an intuitive web interface (no voice input required).

Course-Focused Responses (RAG)

Responses are grounded in materials uploaded by lecturers (such as notes, slides, and past exam questions) via retrieval-augmented generation, which helps minimize inaccuracies and ensures citations are included.

Lecturer Dashboard

Facilitates the upload and management of course materials, the review of frequently asked questions, response corrections, and the publication of FAQs. It supports role-based access for lecturers and administrators.

Content Storage and Indexing

PDFs, documents, and slides are saved with metadata, and embeddings are created and indexed for semantic searching.

Security and Privacy Measures

Includes HTTPS/TLS protocols, JSON Web Token sessions, role-based access control, input sanitization, and anonymized logging; access to data is limited to the minimum necessary.

Cross-Platform Web Access

Features a responsive design compatible with smart phones, tablets, and desktops; progressive enhancement is provided for users on low-bandwidth connections.

System Requirements

System specifications outline the necessary conditions that the proposed AI-driven lecturer assistant chatbot must satisfy to meet its intended goals. These specifications are categorized into two types: functional specifications, detailing what the system should accomplish, and non-functional specifications, discussing the quality attributes and constraints that the system must adhere to.

Functional Requirements

Functional requirements detail the primary services and functionalities the system must deliver in order to achieve its goals:

User Authentication and Access Control

- The system will enable lecturers and administrators to establish secure accounts.
- Students will be able to interact with the chatbot without complicated registration, while role-based access will ensure that only lecturers can upload or manage educational materials.

Course Material Upload and Management

- Lecturers will have the capability to upload lecture notes, presentation slides, and additional resources to the system.
- The chatbot will automatically categorize these materials to provide contextually relevant replies.



Real-Time Question Answering

- The chatbot will permit students to pose academic questions in natural language.
- The system will utilize natural language processing (NLP) to interpret the questions and deliver context-centered answers.

Lecturer Dashboard

- Lecturers will have access to a dashboard that displays student interaction trends, commonly asked questions, and analyses of chatbot performance.
- The dashboard will also allow lecturers to modify or enhance chatbot responses when required.

Interaction Logging

- The system will maintain records of student-chatbot interactions for ongoing improvement and oversight.
- Logs will be anonymized to safeguard student identities.

Integration with Existing Platforms

- The chatbot will be implementable as a web application and accessible on mobile devices.
- It will integrate with institutional portals and, where feasible, learning management systems.

Non-Functional Requirements

Non-functional specifications delineate the essential qualities and constraints that ensure the system remains effective, dependable, and user-friendly:

I. Performance

- a. The system will be able to manage multiple simultaneous users without any decline in performance.
- b. Response time for queries will not exceed three seconds during typical server load.

II. Scalability

- a. The system will accommodate increasing numbers of students and lecturers without significant reconfiguration.
- b. Consideration will be given to cloud deployment for scalability beyond local institutional limitations.

III. Security and Privacy

- a. All exchanges between users and the chatbot will be encrypted using TLS/HTTPS.
- b. Access to sensitive information will be limited through role-based authentication.
- c. Interaction logs will be anonymized to preserve student confidentiality.

IV. Usability

- a. The interface will be user-friendly, requiring minimal training for students to navigate. The system will support both text input and voice input to enhance accessibility. Reliability
- b. The system will target a 99% uptime during academic terms.
- c. Automated failover systems will be implemented to ensure service continuity during server disruptions.

V. Maintainability

- a. The system will be modular, allowing separate components (e.g., chatbot engine, database, UI) to be updated independently of the whole system.
- b. Documentation will be available for future developers to improve or expand system features.

VI. Portability

- a. The system will be platform-agnostic and deployable on various operating systems (Windows, Linux, macOS).
- b. The chatbot will be accessible via web browsers, smartphones, and tablets.

VII. Compliance

- a. The system will comply with international standards like GDPR for data privacy and adhere to institutional IT policies.
- b. Ethical considerations will be taken into account to promote fairness, transparency, and inclusivity.

System Design

System design translates the requirements into technical structures and interaction models. This section details the architecture, core use cases, runtime message flows, the logical data model, and the main processing flow for the AI-enabled lecturer assistant chatbot.

System Architecture

The system architecture as shown in Figure 2, follows a layered, modular architecture to ensure scalability, maintainability, and security.

- **Client Layer (Students & Lecturers):** Web/mobile clients provide a conversational interface, authentication views, and lecturer dashboards. Clients communicate with the server via HTTPS using REST for CRUD actions and WebSocket for real-time chat updates.
- **Application Server (API):** The server exposes endpoints for authentication, chat sessions, content management, and analytics logging. It validates requests, enforces rate limits, and orchestrates calls to downstream services.

Service Layer:

- **Auth & RBAC:** Manages login, token issuance, and role-based authorization (student, lecturer, admin).
- **Conversation Orchestrator:** Maintains dialogue state, session context, and routing of prompts/responses between the LLM, retrieval service, and client.
- **Retrieval Service (RAG):** Builds/uses embeddings to fetch relevant course materials; returns top-k passages/snippets for grounding LLM responses.
- **Course Content Store:** Persists lecturer-uploaded materials (PDFs, slides, notes) and structured metadata.
- **Embedding Index (Vector Store):** Stores vector representations of content for fast semantic search.
- **LLM Provider:** Handles text generation and embeddings. It receives the user query plus retrieved context and returns grounded responses.
- **Analytics & Monitoring:** Collects interaction logs, feedback ratings, and performance metrics for evaluation and iterative improvement.
- **Security & Privacy:** TLS for transport security; JWT for sessions; RBAC for least-privilege access; input sanitization; anonymized logs for privacy.

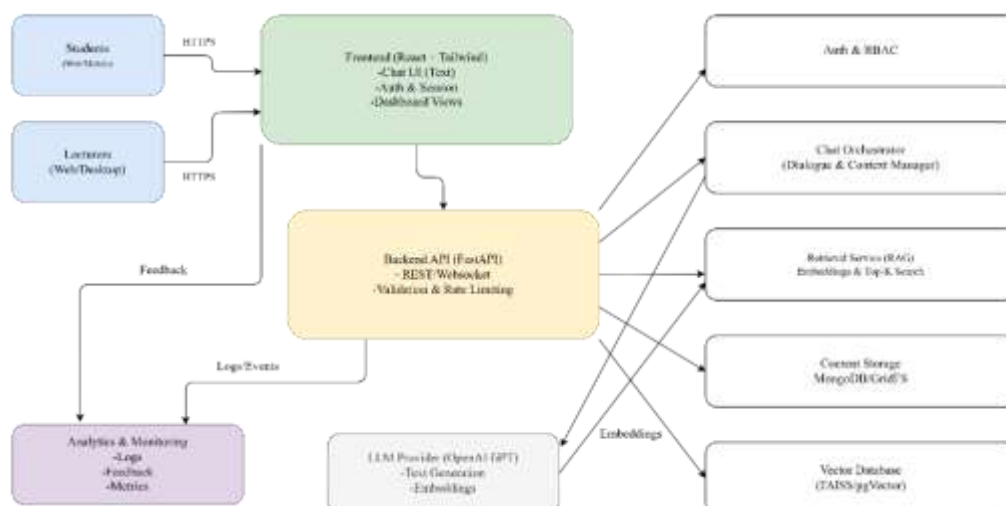


Figure 2: Diagram of the System Architecture



User Interface Components

The frontend interface is composed of the following major components:

1. Login and Registration Views
 - a. Secure login with email/password and session management.
 - b. Role-specific landing pages (student/lecturer/admin).
2. Student Chat Interface
 - a. Main chat window for submitting **text-based queries**.
 - b. Display of bot responses with clear formatting, citations, and expandable reference links.
 - c. Feedback controls (“Helpful / Not Helpful”).
 - d. Scrollable history of previous messages in the current session.
3. Lecturer Dashboard
 - a. Content management panel for uploading and managing course materials.
 - b. Overview of system interactions.
 - c. Analytics and feedback visualization (graphs, tables, charts).
4. Navigation Bar & Sidebar
 - a. Provides quick access to courses, profile settings, and support resources.
5. Accessibility Elements
 - a. ARIA labels for screen readers.
 - b. Clear, high-contrast themes.

The wireframe of the Frontend User Interface is shown in Figure 3.2

Responsive Design

To ensure accessibility across devices, the system employs a responsive design strategy:

1. Mobile Devices ($\leq 768\text{px}$)
 - a. Chat interface adapts into a single-column layout.
 - b. Sidebars and dashboards collapse into toggleable menus.
 - c. Optimized input box for smaller touchscreens.
2. Tablets (769–1024px)
 - a. Two-column layout: chat interface plus collapsible sidebar.
 - b. Lecturer dashboards show charts in stacked form for better readability.

3. Desktops and Large Screens ($\geq 1025\text{px}$)

- Full multi-panel interface with side navigation, central chat window, and lecturer analytics on the right.
- Richer dashboard views with real-time graphs and multi-course analytics.

4. Cross-Browser Compatibility

- Tested on Chrome, Firefox, Safari, and Edge.
- Graceful fallback for older browsers (reduced animations, simplified UI).

5. Performance Optimization

- Lazy loading of non-critical assets.
- Client-side caching of chat history and frequently used assets.
- Minimal external dependencies to reduce load times.

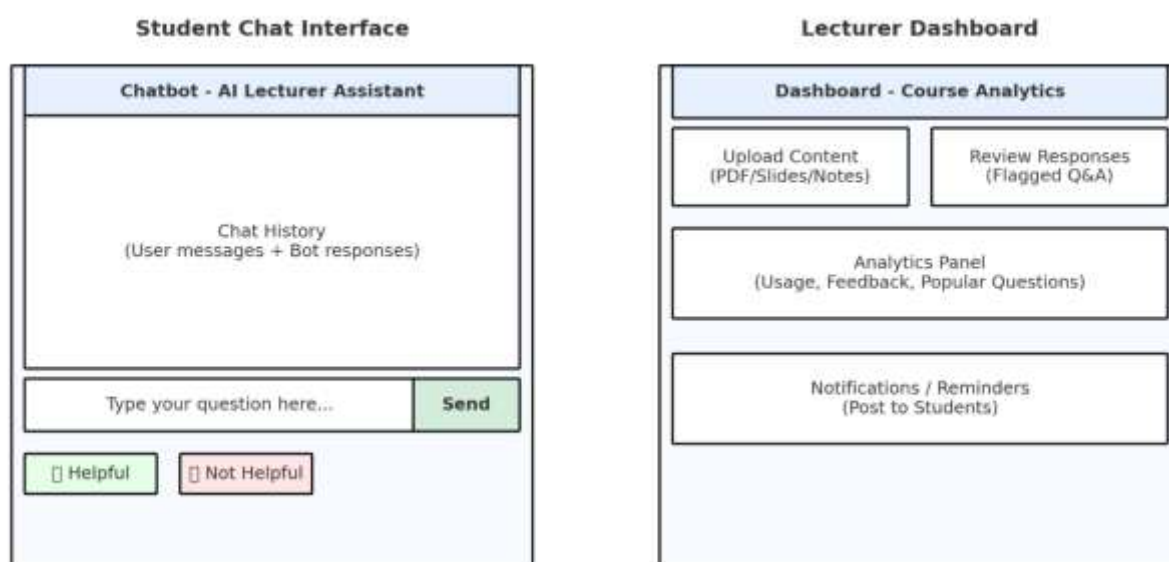


Figure 3: Wireframe of the Frontend User Interface

Backend Development

The backend is the core of the AI Lecturer Assistant system, responsible for handling authentication, routing requests, orchestrating interactions with the AI services, and managing persistent data. It exposes secure APIs that the frontend consumes, and it integrates with external services such as the LLM provider and analytics engine. Flowchart of the Backend is shown in Figure 3.3.

Server-Side Logic

The backend server acts as the **application hub**, providing REST and WebSocket endpoints for all client interactions. Its responsibilities include:

Authentication & Authorization

- Implements secure login and registration.



- Issues JWT tokens for session management.
- Enforces Role-Based Access Control (RBAC) to differentiate student, lecturer, and admin privileges.

Chat Orchestration

- Receives student queries via WebSocket/HTTP.
- Invokes the Conversation Orchestrator, which maintains session context and dialogue flow.
- Calls the Retrieval Service (RAG) to fetch top-k passages from the indexed course materials.
- Packages the query + retrieved context into a prompt for the LLM provider (OpenAI GPT).
- Returns the LLM-generated response, annotated with citations, back to the frontend.

Lecturer Operations

- Provides endpoints for uploading course content (PDF, slides, notes).
- Triggers preprocessing: extracting text, generating embeddings, updating the vector index.
- Supports dashboards for reviewing flagged responses, managing FAQs, and creating announcements.

Feedback Handling

- Stores student ratings (helpful/not helpful) for each chatbot response.
- Flags low-confidence interactions for lecturer review.
- Aggregates feedback into analytics reports.

Analytics Logging

- Captures usage statistics (queries per course, latency, feedback trends).
- Stores anonymized logs for later evaluation and improvement.

Error Handling & Recovery

- Gracefully manages unavailable services (e.g., if LLM provider is offline, fall back to FAQs).
- Provides clear error messages to users while logging technical details for developers.

Database Management

The database layer provides persistent storage for users, course materials, chat logs, embeddings, and analytics. It is designed for scalability and efficient retrieval.

User and Role Management

- Stores user profiles (ID, name, email, role).
- Maintains password hashes and session tokens.
- Links users to roles for RBAC enforcement.

Course and Content Storage

- Each course has associated content records (title, type, metadata).
- Uploaded files (PDFs, notes, slides) are stored in object storage (e.g., MongoDB).
- Metadata tables link files to courses, topics, and lecturers.

Embedding Index (Vector Database)

- Stores vector representations of course materials for semantic search.
- Supports top-k similarity queries using FAISS or pgVector.

- Regularly updated when new content is uploaded.

Chat Sessions and Messages

- Stores all conversations between students and the system.
- Each message record includes sender type (student/system/lecturer), timestamps, and feedback ratings.
- Supports context retrieval for ongoing sessions.

Feedback and Analytics

- Feedback table stores ratings and comments.
- Analytics events (latency, query type, session duration) are logged for performance reporting.

Data Integrity and Security

- Input validation and sanitization applied before storage.
- Enforces referential integrity across entities (users ↔ courses ↔ content ↔ sessions).
- Periodic backups ensure resilience against data loss.

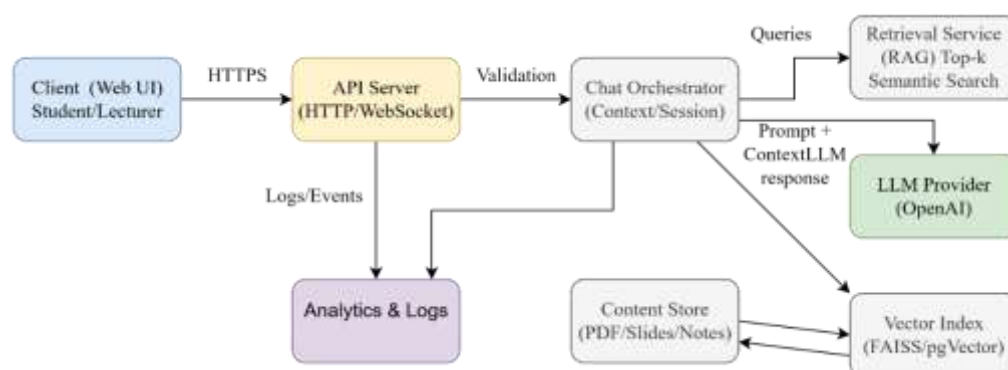


Figure 4: Flowchart of the Backend

AI Integration

AI integration is the intelligent core of the system, enabling the chatbot to provide accurate, clear, and course-specific responses.

Chatbot Training

- Pre-trained LLM: Uses a model ‘OpenAI GPT’ for natural language understanding and generation.
- Fine-tuning / Prompt Engineering: No custom training from scratch instead, structured prompts, role instructions, and few-shot examples are used to align responses with academic context.
- Continuous Improvement: System incorporates lecturer corrections and student feedback to refine FAQs and improve grounding over time.

Integration with Educational Content

- Content Upload: Lecturers upload PDFs, slides, and notes system extracts text and metadata.
- Embedding Generation: Text passages converted into vector embeddings stored in a vector database (FAISS/pgVector).
- Retrieval-Augmented Generation (RAG): When students ask questions, top-k relevant passages are retrieved and injected into the LLM prompt.
- Answer Composition: LLM produces responses grounded in the retrieved content, citing sources for transparency.

- Feedback Loop: Low-rated responses flagged, allowing lecturers to correct them and enrich the knowledge base.

User Case Diagram and Description

The user case diagram shown in Figure 5 is the system supports three actors (**Student**, **Lecturer**, **Admin**) and seven primary use cases. The diagram shows their interactions within the system boundary.

Student Use Cases

- Ask Question: Submits course-related queries through the chat interface.
- Get Recommendations: Receives suggested readings or materials based on activity.
- Rate Response: Provides feedback (helpful/not helpful) to improve quality.

Lecturer Use Cases

- Upload Content: Adds new course notes, slides, and references.
- View Analytics: Monitors course activity and student engagement trends.

Admin Use Cases

- Review Responses: Validates or corrects chatbot answers flagged as inaccurate.
- Manage Users: Creates, edits, or deactivates student/lecturer accounts.
- View Analytics: Accesses institution-level insights and performance logs.

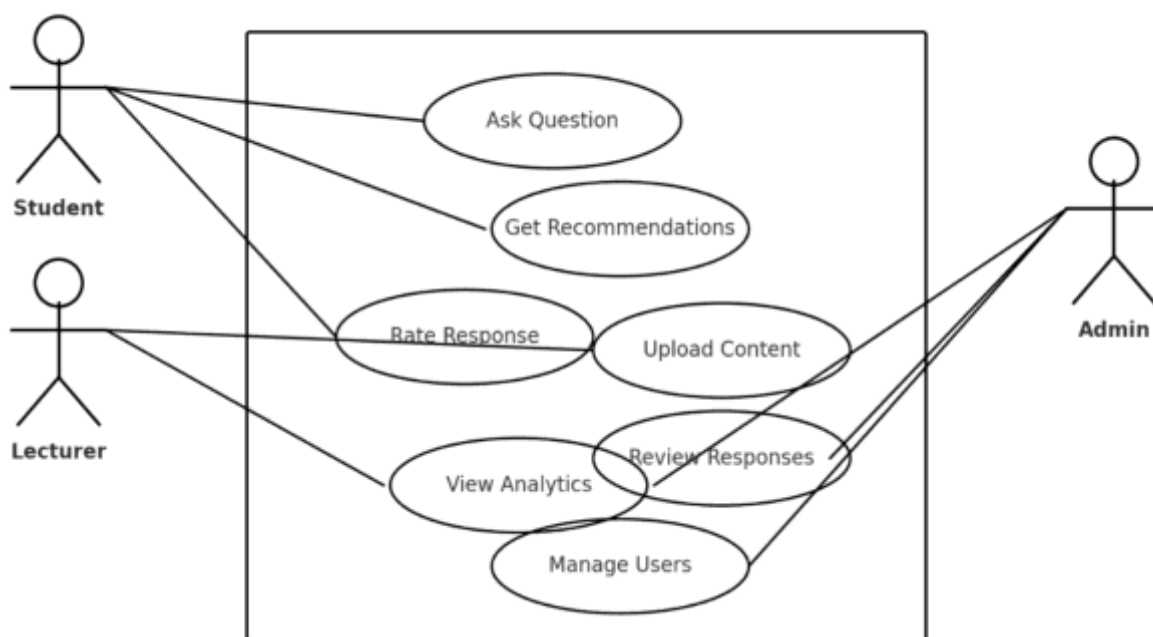


Figure 5: User case diagram

Sequence Diagram

The sequence diagram as shown in Figure 6 illustrates how different components of the system interact when a student submits a query. It captures the **request/response workflow** from the client through backend services to the LLM provider and back.

Scenario: Student asks a course-related question

- Student Client → API Server → Sends query request.
- API Server → Auth Service → Validates session token and role.
- API Server → Conversation Orchestrator → Forwards the message and context.
- Orchestrator → Retrieval Service → Generates embedding, performs semantic search.
- Retrieval Service → Content Store & Vector Index → Returns top-k relevant passages.
- Orchestrator → LLM Provider → Sends prompt (query + retrieved context).
- LLM Provider → Orchestrator → Returns grounded response.
- Orchestrator → API Server → Prepares final message.
- API Server → Student Client → Delivers answer.
- Student Client → API Server → Sends feedback.
- API Server → Analytics Service → Logs interaction, response time, feedback rating

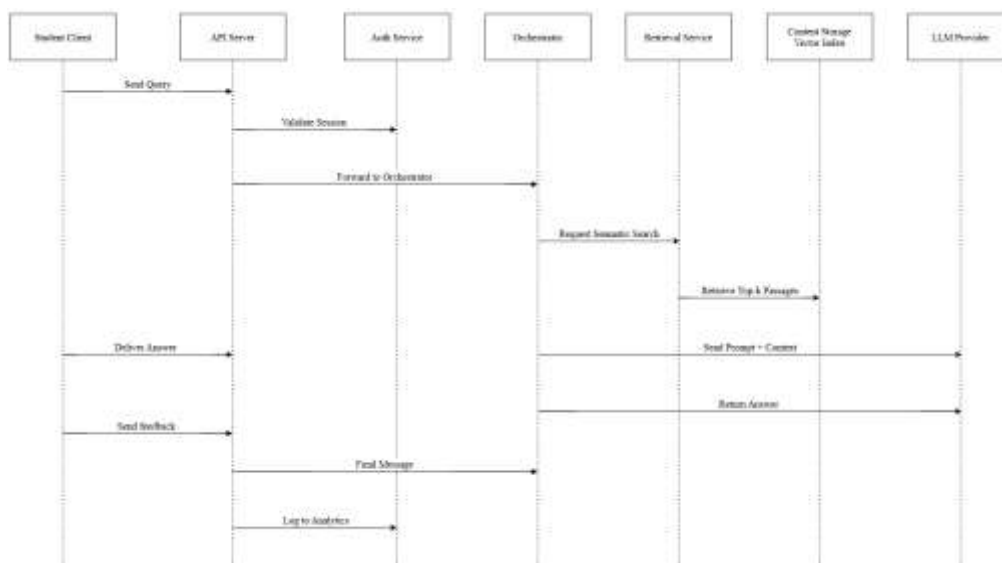


Figure 6 Sequence Diagram for the AI-enabled Chatbot

Entity Relationship Diagram (ERD)

The Entity Relationship Diagram of the Data Model as shown in Figure 7, defines the logical data model for the AI Lecturer Assistant system. It captures how users, courses, content, chat sessions, and analytics data relate.

Entities and Key Attributes

- User → user_id, name, email, role
- Course → course_id, title, code, department
- Enrollment → enrollment_id, user_id, course_id, role_in_course
- Content → content_id, course_id, title, type, uploaded_by
- Embedding → embedding_id, content_id, vector_ref
- ChatSession → session_id, user_id, course_id, started_at
- Message → message_id, session_id, sender_type, text, created_at
- Feedback → feedback_id, message_id, rating, comment
- FAQ → faq_id, course_id, question, answer, updated_by
- AnalyticsEvent → event_id, user_id, session_id, event_type, created_at

Relationships

- User ↔ Course → Many-to-many via Enrollment.
- Course 1-* Content; Content 1-* Embedding.
- User 1-* Chat Session; Chat Session 1-* Message.
- Message 1-* Feedback.
- Course 1-* FAQ.
- User 1-* Analytics Event.

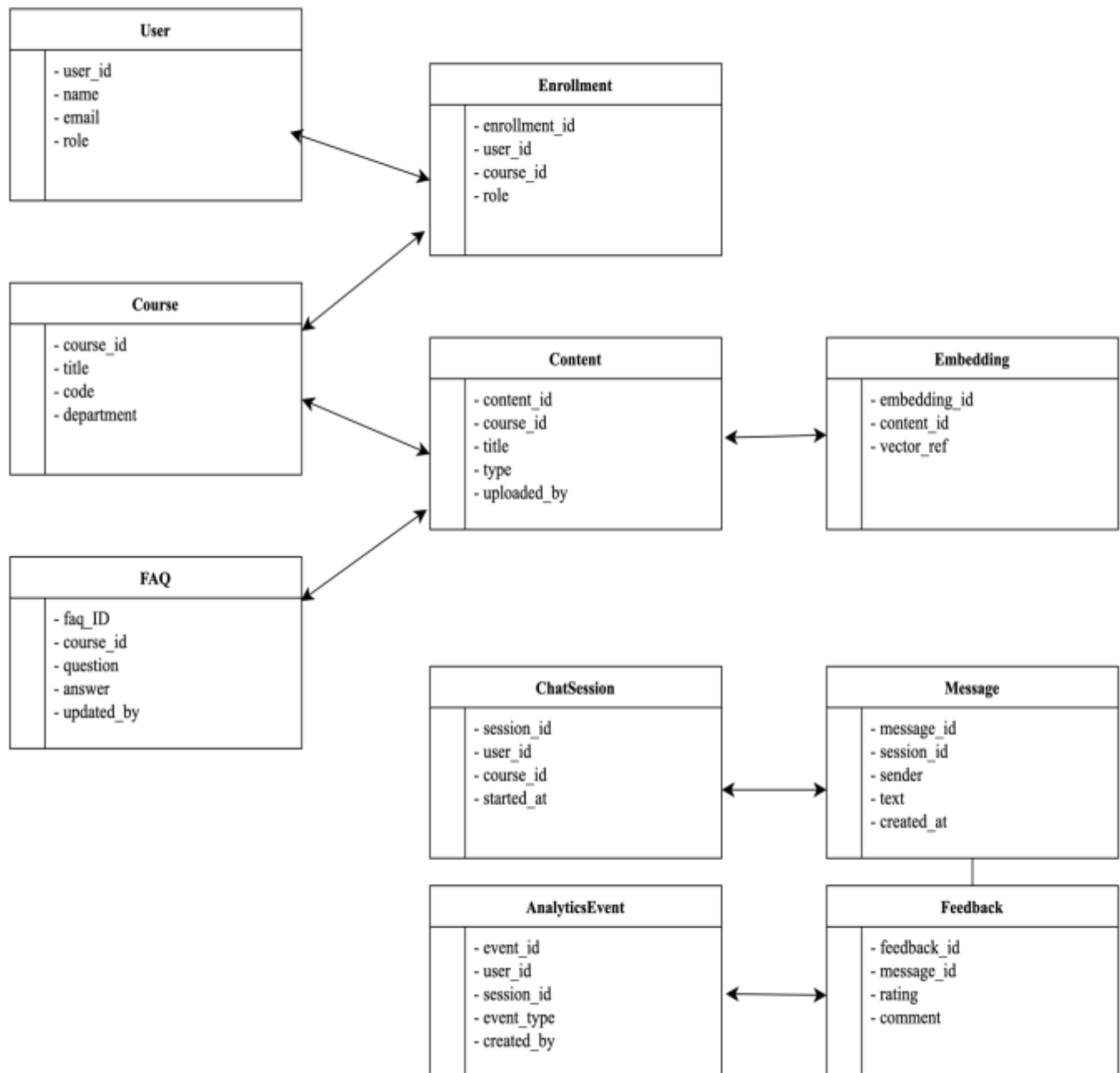


Figure 7 Entity Relationship Diagram of the Data Model

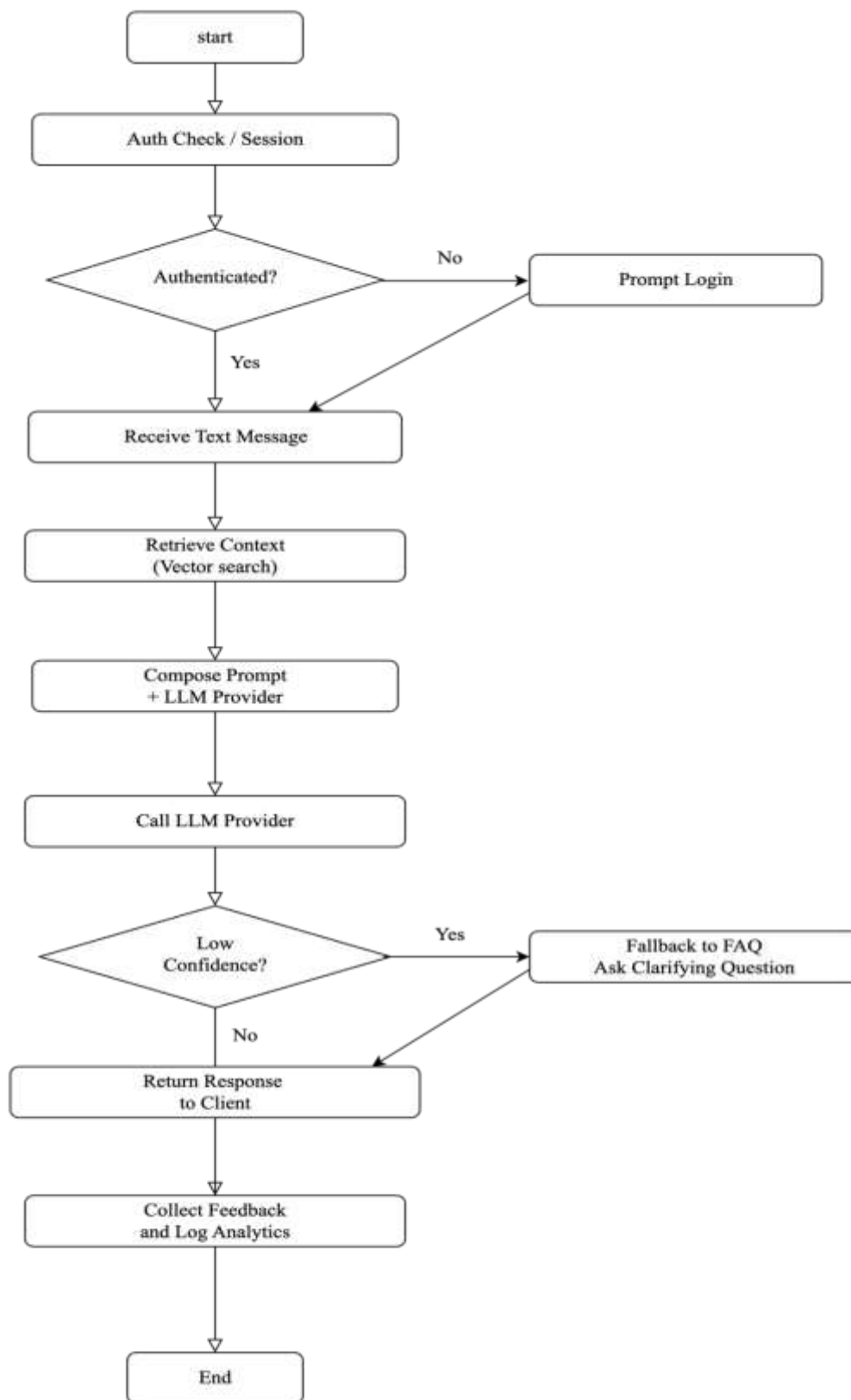


Figure 8: Flowchart of the System

Flowchart of the System

The Flowchart of the System as shown in Figure 8, outlines the step-by-step logic of the AI Lecturer Assistant chatbot:



- Authentication → System checks user session unauthenticated users are prompted to log in.
- Message Input → Student sends a text query through the chat interface.
- Context Retrieval → Relevant course content is fetched from the vector index.
- Prompt Composition → Query and retrieved passages are combined into an enriched prompt.
- LLM Processing → The prompt is sent to the LLM provider to generate a response.
- Decision Check → If response confidence is low, fallback to FAQs or clarifying questions.
- Response Delivery → Final answer returned to the student's client.
- Feedback & Logging → User feedback collected and analytics updated for monitoring.

RESULTS AND DISCUSSION

Implementation Environment

The system requirements consist of two sections: hardware requirements and software requirements.

Hardware Requirements

Development Machines:

Apple MacBook Air with 8 GB Unified Memory and 512 GB SSD. This was the main device used for coding, testing, and integration due to its optimized macOS ecosystem.

Windows 11 Laptop powered by AMD Ryzen 7 6800H, 64 GB RAM, 2 TB SSD storage, and NVIDIA GeForce RTX 3070 Ti GPU. This system was used to handle resource-intensive tasks such as embedding generation, Docker container management, and stress testing.

Graphics Processing:

A laptop with a dedicated Nvidia/ AMD/ Intel GPU can provide sufficient power for model testing and high-volume retrieval operations.

Testing Devices:

Both desktop browser environments (on macOS and Windows) were used to test the web application. Additional testing was conducted using mobile browsers (iOS Safari, Android Chrome) to ensure responsive design and accessibility.

Software Requirements

Operating Systems:

- macOS Ventura
- Windows 11 Pro

Programming Tools:

- Python 3.10 (backend, AI integration)
- Node.js (supporting frontend React environment)

Frameworks and Libraries:

- FastAPI (backend API development)
- ReactJS with Tailwind CSS (frontend user interface)
- MongoDB with GridFS (database and file storage)
- FAISS (vector index for semantic search)

AI Services:

OpenAI GPT API (language generation and embeddings)

Development and Deployment Tools:

- Visual Studio Code (IDE for coding and debugging)
- Postman (API endpoint testing)
- GitHub (version control)
- Docker & MongoDB Compass (containerization and deployment)

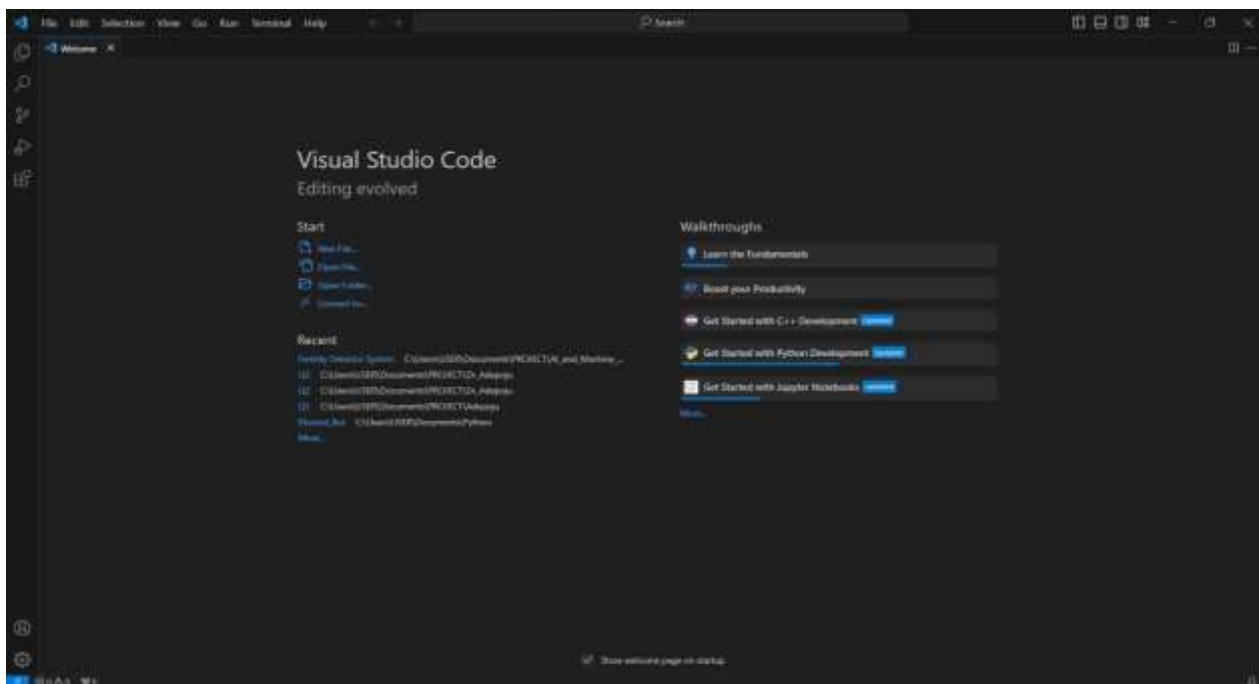


Figure 9: Visual Studio Code IDE

System Implementation

The implementation phase centered on turning the design specifications into a working system. The system was built using a modular approach. The frontend, backend, and AI integration were developed separately before combining them into a single platform.

Frontend Implementation

The AI Lecturer Assistant system's frontend was developed using ReactJS as the main framework. React was chosen because it has a modular component-based structure. This allows the interface to be divided into reusable elements like message windows, input fields, buttons, and dashboard panels. This approach simplifies maintenance and makes it easier to improve the system in the future. For styling and responsiveness, the project included Tailwind CSS. Tailwind provides utility-first classes that help with quick UI development and ensure the application stays consistent across different devices and screen sizes. This was especially important since the system was designed to be accessible on both desktop browsers and mobile devices.

The frontend comprises of two main interfaces: Student chat Interface and Lecturer Dashboard Interface



Student chat Interface

The student interface features a clear layout. On the left, there is a course selector and a panel for conversation history. The central area displays messages, and there is an input field with a Send button at the bottom. The design adopts a modern blue theme. This setup allows students to easily ask questions about their courses and access past conversations quickly.

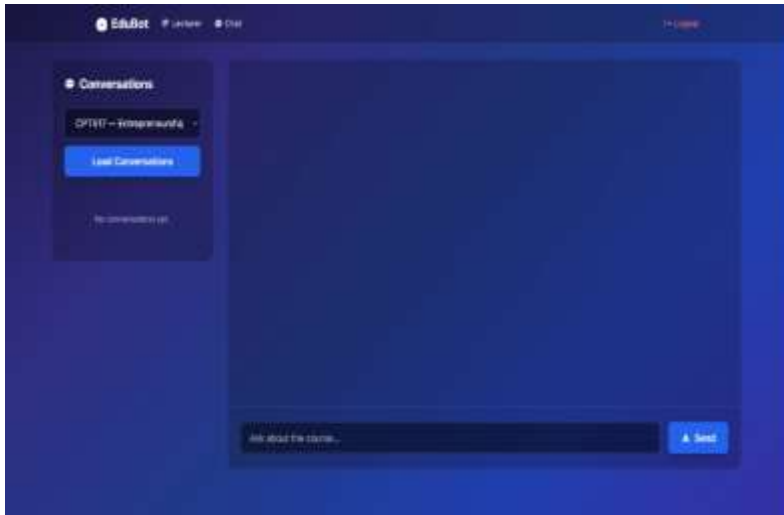


Figure 10: Student Chat Interface

Lecturer Dashboard Interface

The lecturer dashboard allows users to create courses by entering details like course code, title, lecturer name, and an optional description. It also has an upload section where lecturers can attach lecture notes, slides, or other materials for students to access. A list of available courses appears on the left panel, making it easy for lecturers to manage their uploaded content. The interface features a simple layout with clear forms and buttons for easy interaction.

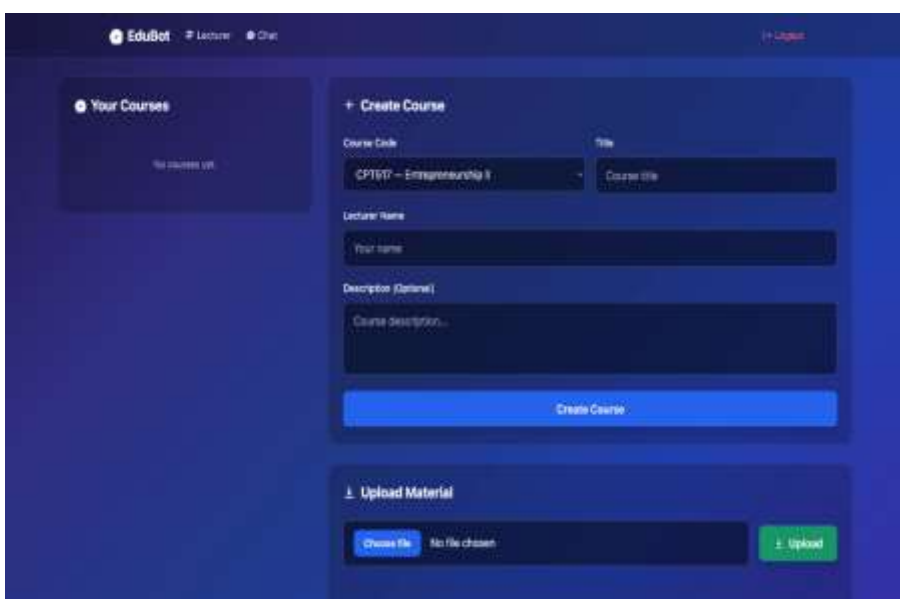


Figure 11: Lecturer Dashboard

Backend and AI Implementation

The AI Lecturer Assistant system's backend was built with FastAPI, a lightweight Python framework that is great for efficient APIs. Postman API was used to test API endpoints to ensure proper health checks and API structure as shown in figure 4.4 below.

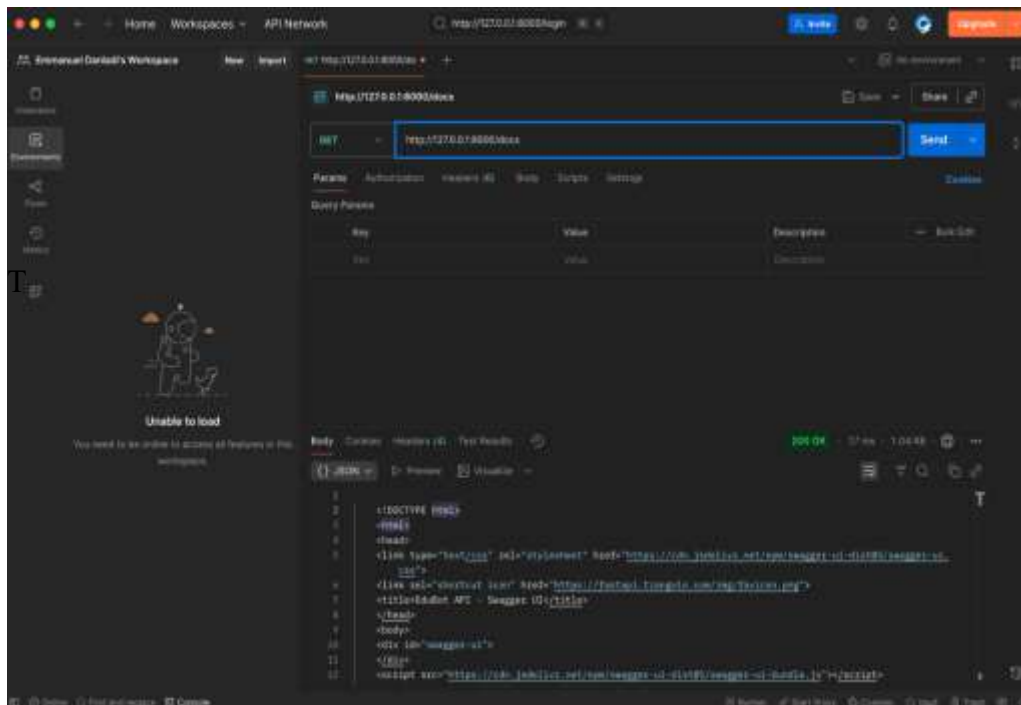


Figure 12: Postman testing API endpoints

This backend handles user login, session management, student questions, course content retrieval, and sends chatbot responses to the frontend.

```
async def chat(payload: ChatIn, user = SecurityOptional(current_user)):
    """
    This function handles the user's chat request. It takes a payload containing the user's ID, the course ID, and the message. It then checks if the user is logged in. If not, it returns a 401 status code. If the user is logged in, it checks if the user is a student. If not, it returns a 403 status code. If the user is a student, it checks if the user is enrolled in the course. If not, it returns a 404 status code. If the user is enrolled in the course, it checks if the user has already asked the question. If not, it adds the question to the database and returns the AI-generated response.
    """
    # Check if user is logged in
    if not user:
        raise HTTPException(status_code=401, detail="User not logged in")

    # Check if user is a student
    if not user.is_student:
        raise HTTPException(status_code=403, detail="Only students can ask questions")

    # Check if user is enrolled in the course
    if not user.enrolled_courses.contains(course_id):
        raise HTTPException(status_code=404, detail="User not enrolled in this course")

    # Check if user has already asked the question
    if not await db.query(ChatHistory).filter(ChatHistory.user_id == user.id, ChatHistory.course_id == course_id, ChatHistory.message == payload.message).exists():
        # Add the question to the database
        chat_history = ChatHistory(user_id=user.id, course_id=course_id, message=payload.message)
        db.add(chat_history)
        db.commit()

    # Generate the AI response
    chunks = await retrieve_top_k(payload.course_id, payload.message, k=5)
    answer = await answer_with_context(payload.message, chunks)

    # Return the AI response
    return ChatOut(
        conversation_id=chat_history.id,
        user_id=user.id,
        sender="bot",
        time=datetime.now(),
        retrieval_contexts=[chat_history.message] * len(chunks)
    )

from fastapi.responses import StreamingResponse
from modeline import Detokenize

@router.post("/chat")
async def chat_stream(payload: ChatIn, user = SecurityOptional(current_user)):
    """
    This function handles the user's chat request. It takes a payload containing the user's ID, the course ID, and the message. It then checks if the user is logged in. If not, it returns a 401 status code. If the user is logged in, it checks if the user is a student. If not, it returns a 403 status code. If the user is a student, it checks if the user is enrolled in the course. If not, it returns a 404 status code. If the user is enrolled in the course, it checks if the user has already asked the question. If not, it adds the question to the database and returns the AI-generated response.
    """
    # Check if user is logged in
    if not user:
        raise HTTPException(status_code=401, detail="User not logged in")

    # Check if user is a student
    if not user.is_student:
        raise HTTPException(status_code=403, detail="Only students can ask questions")

    # Check if user is enrolled in the course
    if not user.enrolled_courses.contains(course_id):
        raise HTTPException(status_code=404, detail="User not enrolled in this course")

    # Check if user has already asked the question
    if not await db.query(ChatHistory).filter(ChatHistory.user_id == user.id, ChatHistory.course_id == course_id, ChatHistory.message == payload.message).exists():
        # Add the question to the database
        chat_history = ChatHistory(user_id=user.id, course_id=course_id, message=payload.message)
        db.add(chat_history)
        db.commit()

    # Generate the AI response
    chunks = await retrieve_top_k(payload.course_id, payload.message, k=5)
    answer = await answer_with_context(payload.message, chunks)

    # Return the AI response as a streaming response
    return StreamingResponse(Detokenize(answer).iter_chunks(), media_type="text/event-stream")
```

Figure 13: Backend Code From chat.py

Figure 13 shows the Backend code snippet from chat.py enumerating how student queries are processed. The function handles user messages, stores them in the database, retrieves relevant content, and generates AI-powered responses that are returned to the frontend.

For the database, MongoDB was used to store user data, course information, and uploaded lecture materials. Large files like PDFs and lecture slides were managed using GridFS, which allows for effective file handling. To enable semantic search, the system includes a FAISS vector database to index the embeddings of course materials for quick access.

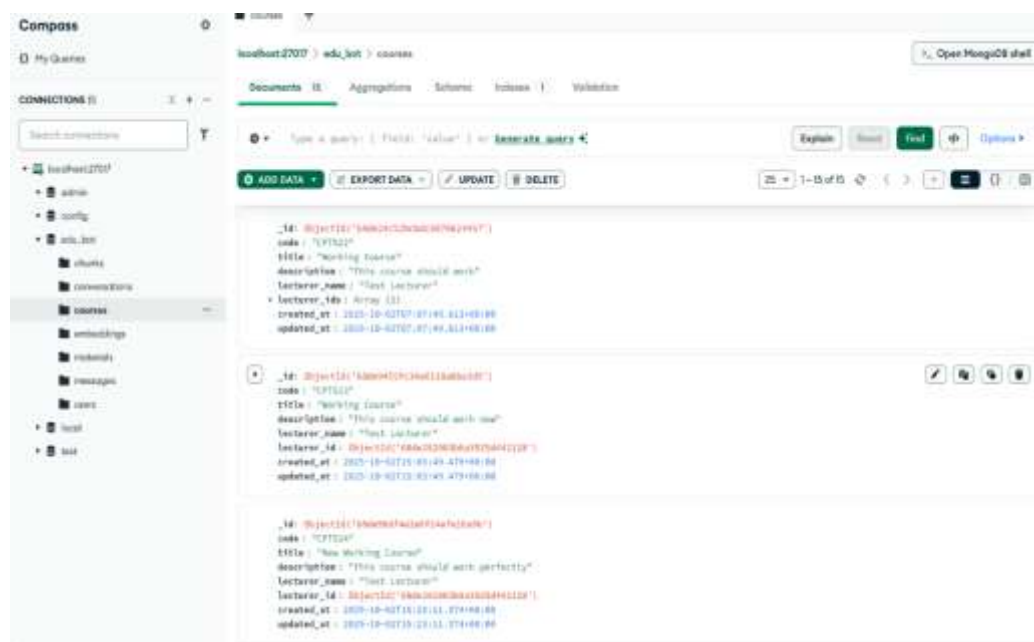


Figure 14: MongoDB Compass showing course collections

The AI integration follows a Retrieval-Augmented Generation (RAG) method:

- When a student asks a question, the backend turns it into an embedding through the OpenAI API.
- This embedding is matched with existing vectors in FAISS to find the most relevant content.
- The retrieved passages are combined into a contextual prompt and sent to the OpenAI GPT model.
- GPT generates a response, which is then sent back to the student through the frontend interface. For security, JWT authentication was used for session management, and RBAC (Role-Based Access Control) was implemented to distinguish between students, lecturers, and administrators.

```
1 # app/services/rag.py
import numpy as np
from bson import ObjectId
from app.db import db
from app.services.llm import embed_text, generate_answer

async def retrieve_top_k(course_id: str, query: str, k: int = 5):
    qv = np.array(embed_text(query)).astype(float)
    cursor = db.embeddings.find({"course_id": course_id})
    scored = []
    async for e in cursor:
        v = np.array(e["vector"]).astype(float)
        denom = (np.linalg.norm(v) * np.linalg.norm(qv)) + 1e-6
        score = float(np.dot(v, qv) / denom)
        scored.append((score, e["chunk_id"]))
    top = sorted(scored, key=lambda x: x[0], reverse=True)[:k]
    chunk_ids = [cid for _, cid in top]
    chunks = []
    for cid in chunk_ids:
        c = await db.chunks.find_one({"_id": ObjectId(cid)})
        if c: chunks.append(c)
    return chunks

async def answer_with_context(user_message: str, chunks: List[dict]) -> str:
    context = "\n\n---\n\n".join(c["text"] for c in chunks)
    system = (
        "You are a lecturer assistant. Use ONLY the provided course context."
        "If the answer is not clearly present, say you are unsure and suggest checking the relevant material."
        "Be concise, cite which parts you used by describing the topic/section if visible in the context."
    )
    user = f"Context:\n{context}\n\nQuestion: {user_message}\n\nAnswer concisely but clearly!"
    return generate_answer(system, user)
```

Figure 15: Backend code from rag.py implementing Retrieval Augmented Generation pipeline

Following the launch of the AI Lecturer Assistant, I evaluated whether it met the set goals and functioned well in real-world situations. The evaluation process had three phases: unit testing, integration testing, and user acceptance testing.

Each part of the system, including authentication, course creation, file uploads, and query handling, was tested separately. For example, I checked API endpoints with Postman to ensure that valid inputs returned the right responses and that errors were handled correctly for invalid inputs.

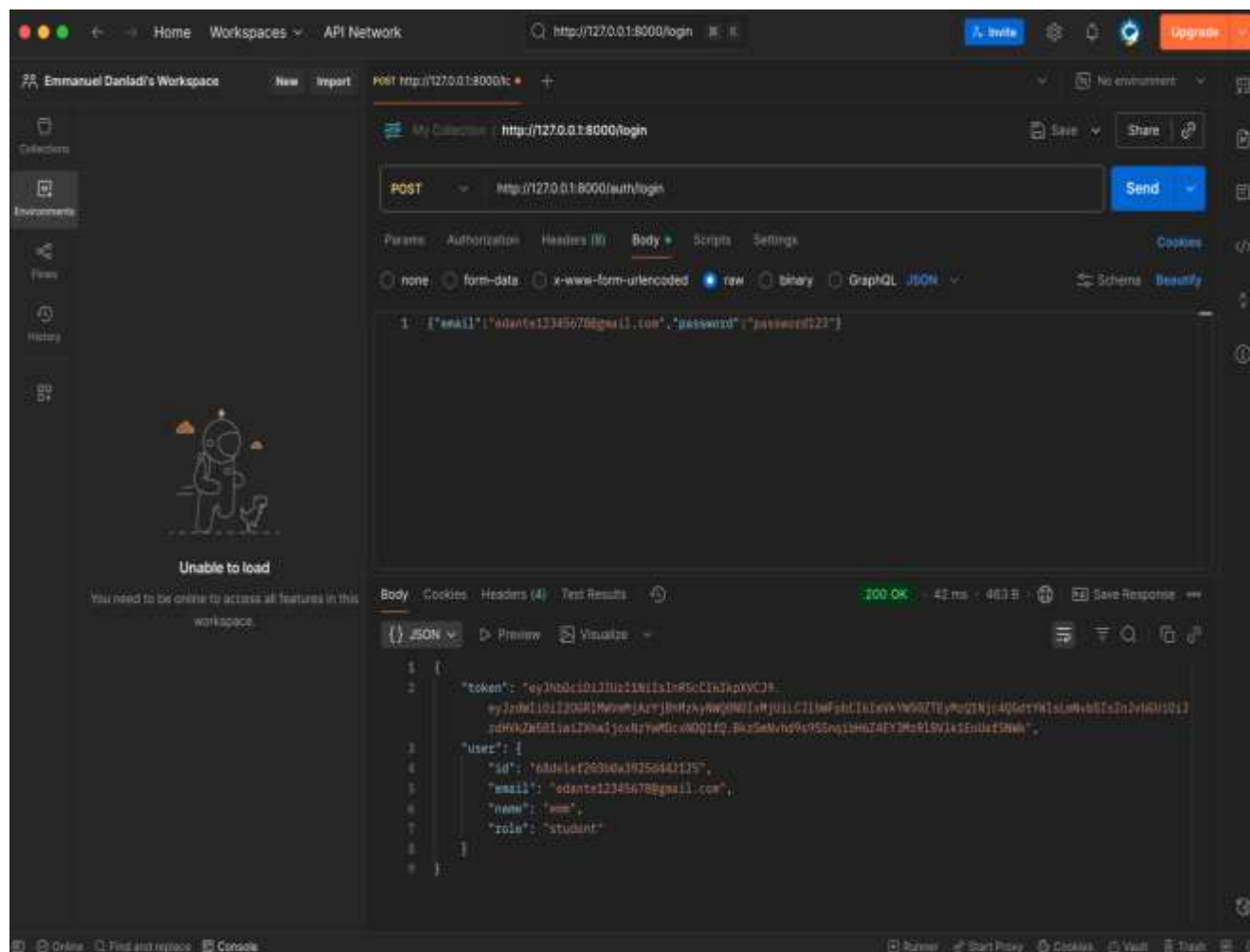


Figure 16: Postman test results for login endpoint

Figure 4.8 shows the postman results for the login endpoint where the JSON contains the token for the registered user and the user details containing the users “id”, “email”, “name” and “role”.

Integration Testing:

The entire process involving the frontend, backend, and AI service was assessed to ensure data was transmitted correctly across all layers. This included checking that a student's inquiry moved from the chat interface to the backend, was processed by the retrieval pipeline, and that a suitable AI-generated response appeared on the interface as shown in Figure 17.

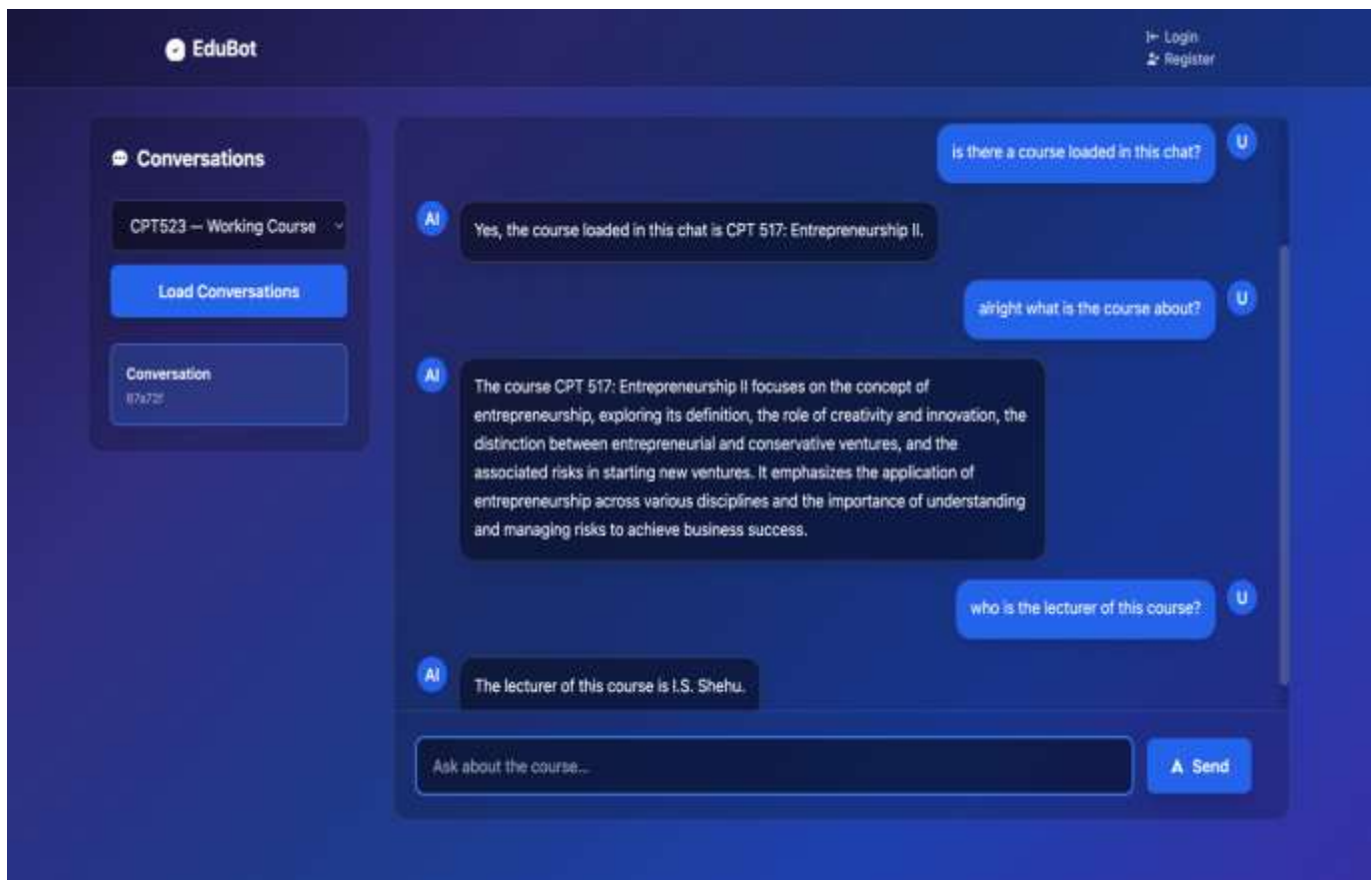


Figure 17: Chat interface displaying student query and the AI response

User Acceptance Testing (UAT)

A selected group of testers, made up of both students and lecturers, interacted with the system to evaluate its usability and relevance. They looked at features like asking questions about courses, uploading lecture materials, and accessing past conversations. Feedback from these participants indicated that the system was easy to use, and the chatbot's responses were contextually appropriate and helpful for academic purposes.

In summary, the assessment showed that the AI Lecturer Assistant chatbot was effective, responsive, and capable of supporting students and lecturers in their academic communications. The development of the AI Lecturer Assistant chatbot was covered. It described the hardware and software setup used during the development process, followed by a practical overview of both the frontend and backend components. The use of the AI model through retrieval-augmented generation was also highlighted. Finally, the system went through unit, integration, and user acceptance testing, which confirmed that it meets its intended functional goals and works reliably.

CONCLUSION

The findings show that the research goals were met. The system includes a student chat interface for course-related questions, a lecturer dashboard for managing learning materials, and a backend integrated with AI that uses retrieval-augmented generation to offer context-aware responses. Using FastAPI, MongoDB, FAISS, and the OpenAI GPT API was effective for building a scalable, responsive, and reliable system. Evaluation results confirmed that the system could return relevant answers, store and retrieve conversations, and let lecturers manage content easily. The user interface was straightforward and intuitive, allowing both students and lecturers to use the system without any technical problems. The project demonstrates the potential of using artificial intelligence in higher education. It shows that AI-driven assistants can effectively support human lecturers by closing communication gaps and providing students with ongoing access to academic guidance and resources.

REFERENCES

1. Adebayo, O., & Oyedele, A. (2020). Challenges of higher education in Nigeria: Implications for quality learning. *African Journal of Education and Technology*, 10(2), 45–56.
2. Aggarwal, C. C. (2018). *Machine learning for text*. Springer.
3. Bates, T. (2019). *Teaching in a digital age: Guidelines for designing teaching and learning*. Tony Bates Associates Ltd.
4. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
5. Brusilovsky, P., & Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems. In *The adaptive web* (pp. 3–53). Springer.
6. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 4171–4186.
7. Drachsler, H., & Greller, W. (2016). Privacy and analytics: It's a DELICATE issue. *Learning Analytics Review*, 1, 1–9.
8. Følstad, A., & Brandtzaeg, P. B. (2017). Chatbots and the new world of HCI. *Interactions*, 24(4), 38–43.
9. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
10. Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1), 81–112.
11. Holmes, W., Bialik, M., & Fadel, C. (2019). *Artificial intelligence in education: Promises and implications for teaching and learning*. Center for Curriculum Redesign.
12. Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed.). Draft.
13. Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2013). The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 37(4), 757–798.
14. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
15. Luckin, R., Holmes, W., Griffiths, M., & Forcier, L. B. (2016). *Intelligence unleashed: An argument for AI in education*. Pearson.
16. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.
17. Mendoza, J., et al. (2020). AI-enabled chatbots in higher education: Enhancing student support. *Journal of Educational Technology Development and Exchange*, 13(1), 45–60.
18. Narciss, S. (2013). Designing and evaluating feedback for learning. In *The Cambridge handbook of multimedia learning* (pp. 337–352). Cambridge University Press.
19. Nielsen, J. (1994). *Usability engineering*. Morgan Kaufmann.
20. Okonkwo, C. W., & Ade-Ibijola, A. (2021). Chatbots applications in education: A systematic review. *Computers and Education: Artificial Intelligence*, 2, 100017.
21. Piech, C., Bassen, J., Huang, J., Ganguli, S., & Guibas, L. (2015). Deep knowledge tracing. *Advances in Neural Information Processing Systems*, 28, 505–513.
22. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1–67.
23. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT networks. *Proceedings of EMNLP*, 3982–3992.
24. Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th ed.). Pearson.
25. Selwyn, N. (2016). *Education and technology: Key issues and debates*. Bloomsbury Publishing.
26. Shawar, B. A., & Atwell, E. (2007). Chatbots: Are they really useful? *LDV Forum*, 22(1), 29–49.
27. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., & Diakopoulos, N. (2016). *Designing the user interface: Strategies for effective human-computer interaction* (6th ed.). Pearson.
28. Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1), 153–189.
29. Singh, S., Kearns, M., & Mansour, Y. (2002). Nash convergence of gradient dynamics in general-sum games. *Proceedings of UAI*, 541–548.



30. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press.
31. UNESCO. (2018). Global education monitoring report 2018: Meeting commitments. UNESCO Publishing.
32. UNESCO. (2020). Global education monitoring report 2020: Inclusion and education. UNESCO Publishing.
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008.
34. Winkler, R., & Söllner, M. (2018). Unleashing the potential of chatbots in education: A state-of-the-art analysis. *Proceedings of the Academy of Management Annual Meeting*.
35. Woolf, B. P. (2010). Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning. Morgan Kaufmann.
36. Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education. *International Journal of Educational Technology in Higher Education*, 16(1), 39.