

CITY College
University of York Europe Campus

Computer Science Department

AI Tutor

This report is submitted in partial fulfilment of the requirement for the degree of
Bachelor of Science with Honours in Computer Science by

Juljan Halilaj

May 2025

AI tutor

by

Juljan Halilaj

Supervisor: Mr Dimopoulos

ABSTRACT

This dissertation describes the design and development of a modular AI-based tutoring system that automatically converts user-supplied educational materials such as PDFs, slide decks, and text files into lesson modules irresistibly aligned with educational best practices without any engineering prompts. The system utilizes multiple large language models (LLMs) like GPT-4, LLaMA, Gemini, and DeepSeek through an integrated AI API. It offers one-on-one sessions according to the learner's preferred method of instruction, manner, and cadence of speaking. Both grading and feedback are offered within an agile quiz framework that automatically assesses based on user-defined criteria, while feedback is comprehensive and visually tracked over the duration of learning through an intuitive progress dashboard. Additionally, there are lesson-specific chat rooms for each session to mitigate topic crossover. As a fully web-based application, it is globally available without installation, meeting the flexibility needs of learners who cannot attend lectures due to work, health, or travel. This work demonstrates, through iterative development and systematic evaluation, that a zero-prompt multi-model AI tutor can seamlessly enhance self-directed learning by providing scalable, adaptable, and engaging instruction. The dissertation ends with reflections.

DECLARATION

All sentences or passages quoted in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). I understand that failure to do this, amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

I have completed and submitted this work by myself without assistance from or communication with another person, either external or fellow student, or any AI type of content generator. I understand that not working on my own will be considered grounds for unfair means and will result in a fail mark for this work and might invoke disciplinary actions.

- ☐ I agree that my dissertation can be used as a model/example by future students, for educational purposes only.

Name (block capitals): JULJAN HALILAJ.....

Signed: Date: May 2025.....

Table of Contents

| | |
|---|----|
| Chapter 1 - Introduction..... | 7 |
| Chapter 2 - Background and Research..... | 9 |
| Chapter 3 - Methodology and Project Planning..... | 12 |
| 3.1 Project Vision, Objectives, and Justification..... | 12 |
| 3.2 System Architecture..... | 13 |
| 3.3 Tools, Environment & Performance Considerations..... | 15 |
| 3.4 AI Model Selection..... | 16 |
| 3.5 Software Development Process..... | 17 |
| 3.6 Development Iterations..... | 18 |
| Chapter 4 - System Evolution through Iterative Development..... | 20 |
| 4.1 Iteration 1: Environment & Core Pipeline..... | 20 |
| 4.2 Iteration 2: Conversational Chat Layer..... | 24 |
| 4.3 Iteration 3: Content Outlining & Module Generation..... | 28 |
| 4.4 Iteration 4: Assessment Engine & Progress Tracking..... | 34 |
| 4.5 Iteration 5: Multi-Model Selection via OpenRouter..... | 40 |
| 4.6 Iteration 6: Dashboard & UX Polish..... | 41 |
| Chapter 5 - Objectives, Self Evaluation and Reflection..... | 46 |
| 5.1 Objectives Evaluation..... | 46 |
| 5.2 Self-Evaluation and Reflection..... | 47 |
| 5.3 Third-Party Evaluation Plan..... | 48 |
| Chapter 6 - Conclusion & Future Work..... | 51 |
| 6.1 Conclusion..... | 51 |
| 6.2 Future Work..... | 52 |
| References..... | 54 |

List of Figures

| | |
|---|----|
| Figure 3.1 System Architecture..... | 15 |
| Figure 4.1 FlowChart of backend..... | 21 |
| Figure 4.2 Pdf Extraction..... | 24 |
| Figure 4.3 four layer decomposer..... | 25 |
| Figure 4.4 Unified Modeling Language (UML)..... | 30 |
| Figure 4.5 ChatBot UI..... | 34 |
| Figure 4.6 Use Case Diagram..... | 37 |
| Figure 4.7 Generated Test..... | 40 |
| Figure 4.8 UserDashboard UI..... | 44 |
| Figure 4.9 TopicProgress UI..... | 45 |

List of Tables

| | |
|--|----|
| Table 2.1 Comparison of Similar Projects | 11 |
| Table 3.1 Risk Management..... | 18 |
| Table 3.2 Software Development Process..... | 19 |
| Table 4.1 LLM..... | 41 |

Chapter 1 – Introduction

The rapid rise of large language models (LLMs) has unlocked new possibilities for self-directed learning, yet many students find themselves overwhelmed by the very tools meant to help them. It takes profound technical knowledge to craft effective AI prompts, and the resulting free form responses that pop up all too frequently appear in the form of unorganized text, leaving students to navigate an impossibly large ocean of unbridled information. Furthermore, unedited LLM generated lessons frequently lack clearly delineated separations into definitions, examples, and exercises, and extended chat sessions allow prior topics to carry over into subsequent topics heaping confusion in place of explaining concepts.

Beyond the challenge of content organization, today's tutoring solutions seldom provide integrated assessment or progress tracking. Without on-demand quizzes, automated grading, or a visual dashboard, students receive neither timely feedback on their misunderstandings nor a coherent view of how their mastery evolves over time. Equally important, students bring diverse preferences for teaching style, tone, and pacing; yet most platforms lock them into a single AI “teacher,” removing the ability to switch to a voice that better resonates with their individual learning needs. Finally, the promise of AI-powered tutoring is undermined if access requires software installation or only functions on specific devices students need an “anytime, anywhere” solution that works uniformly on desktop and mobile browsers.

This dissertation describes the design and implementation of a unified, zero-prompt AI tutor built to address these intertwined challenges. By allowing students to upload arbitrary source materials PDFs, slide decks, or even plain-text topics the system automatically transforms content into structured lesson modules, complete with introductions, key concepts, illustrative examples, and practice exercises by the use of AI. Each lesson runs in its own isolated chat context to prevent topic bleed, and a dynamic quiz engine generates assessments on demand, grades responses instantly, and aggregates results into a consolidated progress dashboard. Students may also choose among multiple AI backends such as GPT-4, LLaMA, Gemini, or DeepSeek to find the style and pacing that best suits their preferences, while a browser-only interface guarantees installation-free access across devices.

The primary audience for the AI tutor encompasses students who, for a variety of reasons, are unable to attend to lectures in person whether due to illness, work commitments, or travel obligations and who require a seamless way to recover missed material. Part-time and returning users, juggling careers or family responsibilities alongside their studies, will find the platform's on-demand access and structured lesson flow perfectly suited to fit their intermittent schedules. Likewise, remote and distance- education students spread across different time zones and facing variable connectivity constraints benefit from a zero-install, browser-based interface

that delivers every lesson, chat exchange, and assessment reliably on any device.

The chapters that follow chart a path from concept to prototype: Chapter 2 surveys the state of AI in education and related technologies; Chapter 3 outlines the architectural framework and iterative development methodology; Chapter 4 details the six focused sprints that brought each capability to life; Chapter 5 evaluates the system against its original aims through systematic testing; and Chapter 6 reflects on lessons learned and envisions future enhancements. Through this structured investigation, the dissertation demonstrates that a truly zero-prompt, browser-based AI tutor is both feasible and effective.

Chapter 2 - Background and research

This AI Tutor project is a good example of the AI track as it uses advanced natural-language processing and generative AI strategies as opposed to using pre-fabricated content. Through its core lies a unified AI API layer that abstracts multiple large-language-model backends (GPT-4, LLaMA, Gemini, DeepSeek) with a uniform JSON interface. The AI Adapter uses carefully crafted questions to translate arbitrary user input to meaningful lesson modules in zero-code manner. Both the ChatBox and Quiz Engine modules call for this same API to create, assess, and iteratively refine questions and explanations in real time, forming a genuine closed-loop feedback process powered by prompt engineering. Dynamic routing logic determines the best model for each request, manages load balancing, and gracefully fails if a service fails without modifying application code. By cleanly separating Presentation, Application, AI, and Persistence layers, the system remains scalable and flexible: the frontend invokes the AI API, the backend handles model selection, and the database stores every transaction. This multi-model API architecture offers hands-on experience in prompt design, model orchestration, and live content creation fundamental applied-AI competencies explicitly correlated with the curriculum.

In recent years, Artificial Intelligence in Education (AIED) has changed the landscape of teaching and learning. Wang et al. (2022) conducted a bibliometric analysis of 2,223 AIED publications, and determined four major application areas: adaptive learning, intelligent assessment, user profiling, and new AI offerings. Particularly, adaptive learning and personalized tutoring dominated the discourse, especially following rapid online-learning adoption during the COVID-19 pandemic [1]. Lin et al. (2023) further emphasize that AIED systems contribute to sustainable education by providing data-driven, user-centered instruction, while also highlighting infrastructure and equity challenges in under resourced settings [2].

Prompt engineering has become an essential component in using large language models for educational reasons. Park and Choo (2024) put forth the PARTS-CLEAR- ACCOUNTABILITY framework to inform educators in designing effective prompts: defining Persona, Purpose, Recipients, Theme, and Structure (PARTS); ensuring to maintain Concise CLEAR (Explicitness, Adaptivity, and Restrictions) with iterative refinement Loops and embedding ethical safeguards [3]. Implementing PARTS-CLEAR in section-splitting as presented in Chap. 4.2 (see Sec. 4.2). Su and Yang (2023) also calls for specifying learning goals, defining automation levels, ensuring ethical considerations, and evaluating effectiveness to align AI deployments with pedagogical goals; integrated these guidelines in test-evaluation questions in Sec. 4.5 [4]. Cain (2023) highlights integrating content Chapter 2 Background and research knowledge, critical analysis of AI output, and iterative prompt construction to accomplish pedagogically effective interactions [5].

Generative AI offers on-demand lesson generation, immediate feedback, and contextualized support. Alasadi and Baiz (2023) discuss these opportunities alongside concerns such as algorithmic bias, data privacy, and academic integrity demanding responsible amalgamation practices and privacy-by-design protections [6]. These conceptual concerns are reaffirmed in both the GDPR and the IDEE guidelines, and we implement encrypted storage and minimal user data retention in upload pipelines.

Empirical studies demonstrate the efficacy of AI tutors grounded in learning-science principles. Baillifard et al. (2024) report a semester-long trial in which a GPT-3–driven microlearning app, combined with a neural “grasp” model for spaced-retrieval practice, improved neuroscience students’ exam performance by an average of 0.71 grade points and increased percentile rankings by 12.4 points compared to control groups [7]. TagHive Inc. (2024) integrates prompt engineering with Deep Knowledge Tracing (DKT), dynamically adjusting explanation depth based on real-time mastery predictions, resulting in more personalized and engaging learning interactions [8]. Thomas et al. (2024) illustrate how GPT-4 can be used to evaluate human tutor dialogues, reliably assessing praise and error-response strategies through advanced prompt-chaining techniques [9].

woo-hyun kim and jong-whan kim (2020) introduce an individualized AI tutor built around three specialized developmental learning networks (DLNs) based on an extended Deep ART architecture. The **user status DLN** incorporates an input channel addition algorithm to seamlessly integrate new performance metrics without disrupting existing classifications; the **user preference DLN** leverages an n-th order event bundle encoding to capture both the sequence and frequency of user activities; and the **user experience DLN** uses an alternative template learning algorithm to dynamically expand or contract content recommendations based on real-time measures of educational effectiveness. Embedded in a commercial mobile app for teaching Korean, this system was trained on over 800,000 user interactions and demonstrated the ability to match and even surpass standard curricula in recommending personalized content [14] .

A number of commercial systems illustrate the “upload-and-learn” model. Gizmo.ai/tutor can handle uploads of documents (PDFs, PowerPoints), videos, and notes; automatically breaks up content into coherent lessons, creates quizzes, and monitors progress with students [10]. NotebookLM.Google provides the same upload-and-query model, with the ability to add documents and then ask open-ended questions against their own content in-house [11]. By contrast, TutorAI.me [12], and AI-Tutor.ai [13] provide bespoke course construction using proprietary content repositories but do not permit arbitrary user uploads and thus place restrictions on their flexibility for user-managed material integration.

The table below summarizes major differences between major "upload-and-learn" tools, illustrating where each product excels or is lacking in custom uploads,

lesson ordering, testing, progress monitoring, and AI engine adaptability. Positioning our AI Tutor against these services, we can see how its multi-model API, dynamic prompt-based lesson modules, real-time quiz creation, and persistent activity logs integrate aspects none of our competitors offer in a single system.

Table 2.1 Comparison of Similar Projects

| Feature /Platform | TutorAI.me | Gizmo.ai/tutor | AI-tutor.ai | NotebookLM.google |
|---------------------------------|---|--|---|---|
| Custom File Upload | X only lets you pick from their course catalog. | ✓ drag and drop Deck, PDF, PowerPoint, YouTube , even photos or notes to ingest your own material | X focused on pre-built “custom courses” rather than user uploads. | ✓ upload PDFs, DOCX, PowerPoints , text and instantly query your own documents |
| Course Creation | ✓ Build a “custom learning pathway” by choosing topics/module | ✓ After upload, the AI auto chunks and teaches in Context (outlines, Q&A, summaries) | ✓ Select from popular topics or build a tailored course via menu. | X designed as an exploratory Q&A tool rather than structured lesson sequencing. |
| AI Engines Supported | Proprietary AI | Proprietary “Gizmo” engine (LLM based) | Proprietary AI | Google’s PaLM (and related LLMs under the NotebookLM umbrella) |
| Lesson Structure | Sequence of bite-sized lessons | Dynamic sections & deep dives based on your files | Bite-sized lessons with quizzes & assessments | X provides answers directly against your upload rather than formal lessons. |
| Assessment & Quizzes | X Not explicit | ✓ Can generate quizzes from uploaded content | ✓ Includes “Homework Assistant” for practice questions | X session-based interactions without persistent progress metrics. |
| Progress Dashboard | ✓ Tracks completion of custom pathway | ✓ Tracks which uploads you’ve worked through | ✓ Shows courses completed and time spent | X session based interactions without persistent progress metrics. |

Chapter 3 – Methodology and Project Planning

Chapter 3 describes the systematic approach used to transform the research objectives into a functional AI Tutor. It begins by outlining the iterative development methodology and defining the four layer architecture Frontend, Backend, AI Adapter, and Database that underpins the system. The chapter reviews the technology choices and performance considerations that guided implementation. Finally, it explains the strategy for selecting among multiple AI backends and summarizes the sequence of development sprints that delivered each major feature.

3.1 Project Vision, Objectives, and Justification

The primary vision guiding this dissertation is to develop an AI driven tutor capable of transforming a user's own materials whether that be lecture slides, research papers, or simple topic descriptions into cohesive, pedagogically sound lessons without requiring any manual prompt engineering. From this vision five related objectives were made. Each objective is working to solve one of the issues detailed in Chapter 2.

1. Zero-Prompt User Lesson Creation

- **Problem:** Students like different teaching styles with different tones, depths, and speeds but one LLM backend forces a uniform approach that can't be tailored according to personal preferences.
- **Solution:** The system will accept user supplied files (PDF, PPT, text) or free text topics and autonomously generate a full sequence of lesson modules This removes the technical barrier of prompt crafting, allowing users to focus purely on content.

2. Multi-Model Flexibility

- **Problem:** Not all students connect with the same AI "teacher" as they come with different inclinations toward style, tone, and speed but with a single model, there could not be an adaptation to a tutor that suits their learning needs.
- **Solution:** By having multiple LLM backends included, the tutor can utilize a variety of strengths such as facts' accuracy, creativity, or concision and be resilient in case one service becomes unavailable.

1. Contextual AI Interaction

- **Problem:** Continuous chat sessions allow topics to bleed together, causing confusion and loss of focus.
- **Solution:** Each lesson will run in its own isolated chat context. This guarantees that explanations, follow up questions, and exercises remain tightly scoped to the current module, preserving clarity and preventing cross lesson interference.

2. Global Accessibility

- **Problem:** Students need to reach the tutor anytime, anywhere, on any device without installation hurdles or compatibility issues.
- **Solution:** Offer the tutor as a web based browser app. No installation is required, so students can use it at any time, anywhere, on desktop or mobile, with zero deployment friction.

3. Progress Tracking & Assessment

- **Problem:** users lack immediate feedback, automated grading, and a way to monitor their mastery over time.
- **Solution** Users evaluate their understanding through on demand quizzes. After completing a lesson, the user requests to take a test and submit questions tied to that lesson. Multiple attempts should be recorded, detailed feedback needs to be provided, and a dashboard that displays per lesson attempt counter, average scores, and overall progress allowing users to monitor and guide their mastery over time

3.2 System Architecture

Digital AI Tutor will consist of four interconnected levels: **FrontEnd, BackEnd, Database, and Multiple AI Integration**. Every interaction, process, piece of data, and piece is guaranteed to have a defined location because of this breakdown, it is easy to adjust and manage the system as it evolves over time.

Frontend

I provide a consistent, user friendly interface for uploading resources, following guided courses, taking quizzes, and monitoring progress by separating all user facing operations in the FrontEnd. This distinct division of presentation issues

ensures installation free “Anytime, anywhere” access using a web browser that also lets us modify the way things look, feel, and interact without changing the backend logic. The primary focus is on desktop users, most students prefer a larger screen and easier control so mobile layouts are optimized as a secondary priority.

Backend

All pedagogical workflows including transforming raw content into structured lesson modules, generating and grading quizzes, and enforcing user permissions reside in a single Backend layer. Consolidating these tasks behind a distinct API border guarantees uniform business rule implementation, streamlines testing, and offers a single gateway for rate restriction, authorization, and authentication. Additionally, the BackEnd connects the FrontEnd, Database, and AI Integration layer, coordinating data flow and scaling operations without regard to display or storage issues.

Database

The Database layer captures and maintains a full history of all lessons uploaded, chat sessions, quizzes set and graded, including any relevant files uploaded, so that users can revisit previous sessions, uncover misunderstandings, and assess their progress over time. Lesson schemas can evolve, introduce new assessment types, or extend user preferences without costly migrations while preserving a complete audit trail for analysis and quality assurance.

Multiple AI Integration

At the core of content creation lies the Multiple AI Integration layer, which abstracts a suite of large language models behind a uniform interface. Users can select or switch between distinct AI “tutors” (for example, GPT- 4, LLaMA, Gemini, DeepSeek) to find the style and pacing that best suits their learning. This pluggable architecture not only personalizes the experience, yet also provides potency: if one model will become unavailable or underperform, others can easily step in, and new AI services can be added with limited impact on the rest of the system. Together, these four layers form a modular, contract driven architecture. User actions flow from the FrontEnd through the Backend’s pedagogical engine, into the AI Integration for content generation, and finally into the Database for persistence. This structure directly addresses every documented user point and positions the tutor for continuous extension and improvement.

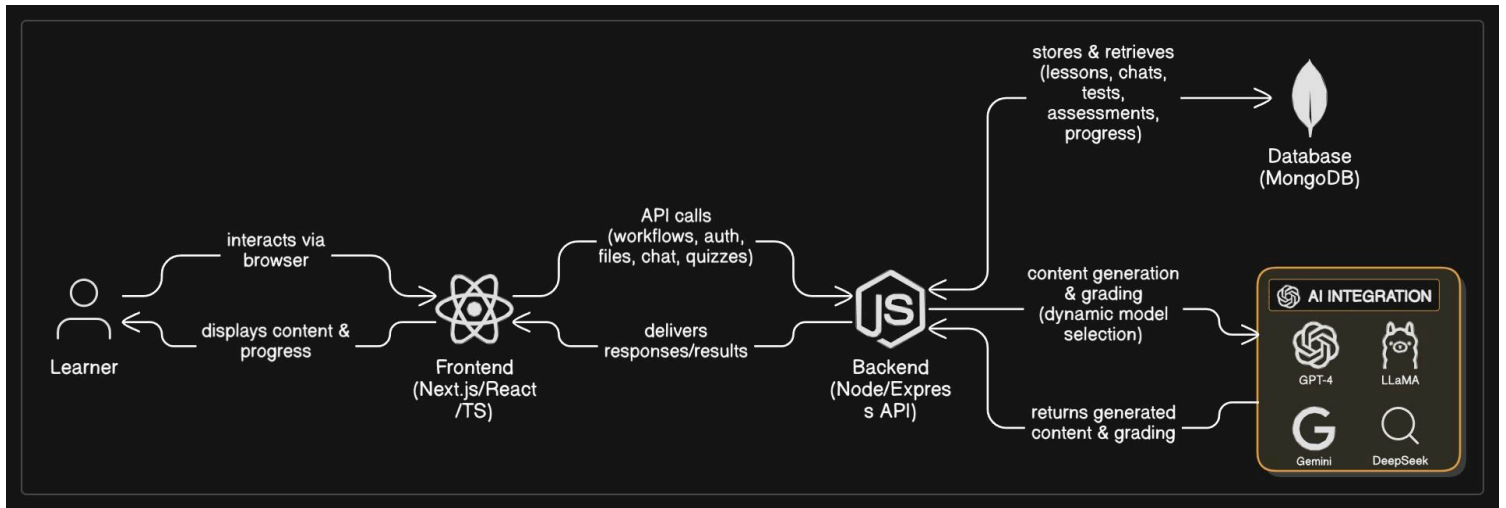


Figure 3.1 System Architecture

3.3 Tools, Environment & Performance Considerations

In selecting the technology stack, I prioritized rapid development, runtime efficiency, and future extensibility key attributes for delivering a globally accessible, zero prompt AI tutor. Below I describe each core component and how it contributes to the objectives, followed by practical notes on leveraging multi model AI access via OpenRouter.

Next.js (v15.1.3) & React (v19.0.0)

Hybrid server and client-side rendering in Next.js allows to serve statically generated pages for lightning fast initial loads along with ongoing support for dynamic, authenticated updates to lessons and quizzes. React's component model drives the whole frontend, and Next.js API routes power the gap between UI to backend, chatting smoothly without separate servers or frameworks [15].

TypeScript

TypeScript ensures every API contract, from UI payloads to AI queries, satisfies the requirements at build time through the use of static types on both frontend and backend. The intrusive type reduces runtime issues when scaling AI adapter to more models or incorporating the frontend with backend routes [16].

MongoDB & Mongoose

MongoDB was chosen because it is document oriented, schema flexible, and this exactly fits the dynamic shape of lesson modules, quizzes, and user data. Mongoose provides only adequate schema enforcement and lifecycle hooks to validate and

index the collections, without enforcing hard migrations every time I add new fields or types of content [17] .

OpenRouter.ai Service

Rather than integrating each LLM endpoint separately, I funnel all AI requests through OpenRouter's unified HTTP interface. By specifying the desired model name in each payload gain on demand access to GPT- 4, LLaMA, Gemini, DeepSeek, and others, while centralizing rate limit management and billing in a single service [18] .

3.4 AI Model Selection

To support **Multi-Model Flexibility**, the AI Tutor integrates four distinct LLM backends, each chosen for its unique balance of quality, cost, and speed. Users simply select the model they wish to drive their lessons and quizzes.

GPT-4o [19]

GPT-4o is the premium engine, renowned for its deep contextual understanding and articulate writing. It powers the most elaborate lesson scaffolding definitions, examples, and nuanced feedback making it the first choice for that segment of users who value depth over anything else. Its higher per token cost is justified by minimal prompt tuning as well as high fidelity outputs on a regular basis.

LLaMA 4 Scout [20]

LLaMA 4 Scout is a lightweight, open-source alternative that is tuned for quick response and low operational cost. Its explanations are somewhat more concise than GPT-4o's, but it excels at rapid summaries and easy exercises. It is the best option for budget users or ultra low latency use cases.

Gemini 2.0 Flash [21]

Gemini 2.0 Flash strikes a balance between verbosity and efficiency. It is particularly good at innovative examples and code snippets, and it creates engaging, example-driven modules without taking on the full overhead of GPT-4o. Mid-range cost and solid interactive performance make it a good match for users who need practical, hands-on learning experiences.

DeepSeek V3 [22]

DeepSeek V3 specializes in domain-specific retrieval and factuality. Augmented by retrieval generation, it grounds its responses in outside knowledge sources so that research or technical lessons can remain authoritative. Students addressing specialized subject matter will appreciate its rigor and reliability.

3.5 Software Development Process

The Software Development Life Cycle (SDLC) incorporates a wide range of models, each of which is optimized to particular project requirements and risk levels. Traditional linear strategies consist of the Waterfall and V-Shaped methodologies, advancing in fixed stages of requirements, design, implementation, testing and maintenance in strict linear progression. Iterative and Incremental models divide the system into constituent pieces, deploying incremental stages of partially functioning pieces in sequential cycles with client feedback directing each iteration. Prototype approaches center on the rapid development and iteration of throwaway or evolutionary mock-ups to define requirements. The Spiral model specifically incorporates risk analysis within each spiral, merging iterative development with strict scrutiny of uncertainties. Agile methodologies (e.g. Scrum, Extreme Programming, Lean, Feature-Driven Development) feature short "time-boxed" iterations, frequent stakeholder consultation, minimal documentation and flexibility to accommodate shifting requirements [4]

The construction of the Digital AI Tutor necessarily included uncertainties: the best prompt templates, section splitting methods, and interactive workflow were not established initially. Through breaking down the work into short cycles, incremental deliveries were made early and frequently so every part was thoroughly tested and honed until the next feature was constructed. This avoided large-scale rework and kept the project very close to the needs of the users [2].

Additionally, ongoing user feedback was integrated with each iteration. After each iteration, I conducted targeted testing to ensure that the iteration's objectives had been achieved. Each iteration's learnings guided the way ahead: if prompt parameters had to be adjusted, UI flows needed to be tweaked, or model integrations had to be tuned, I integrated those learnings onto the live product without destabilizing current functionality [1].

Last but not least, the iterative methodology provided us with frequent checkpoints to assess progress, gain feedback, and improve requirements to move forward. By having a working product at the completion of every iteration, I had ensured each feature was tested and confirmed in the context, avoiding scope creep and uncovering problems early. The process eliminated wasteful effort, kept risks bounded by each small block of work, and the project tightly in focus with the goals and user requirements. For a research-driven, user-focused system with so many unknown variables up-front, iteration offered the control and adaptability to deliver a sound, quality tutor over a period of time [3].

3.5.1 Risk Management

Below is a tailored risk analysis and management plan for your AI-Tutor project. It includes a concise risk register identifying each key threat, its likelihood and impact, and concrete mitigation strategies, followed by an outline of how to monitor and

control these risks over the course of your six-iteration development cycle.

Table 3.1 Risk Management

| Risk | Likelihood | Impact |
|---|-------------------|---------------|
| Multi-model integration complexities Subtle API differences, error codes, rate limits across GPT-4, LLaMA, Gemini, DeepSeek can cause failures or inconsistent behavior . | High | High |
| Multi-model integration complexities Subtle API differences, error codes, rate limits across GPT-4, LLaMA, Gemini, DeepSeek can cause failures or inconsistent behavior . | Medium | High |
| Data privacy & security User uploads (PDFs/DOCXs) contain potentially sensitive data; improper handling risks leaks. | Medium | High |
| Data privacy & security User uploads (PDFs/DOCXs) contain potentially sensitive data; improper handling risks leaks. | Medium | High |
| Iteration schedule slippage Refactoring for multi-model resilience and UX polish took longer than planned. | Medium | Medium |

3.6 Development Iterations

The Digital AI Tutor will be built through six overlapping iterations between November 2024 and April 2025. The milestones listed below will serve as project milestones and the project success will be evaluated based on the level of quality and meeting the defined requirements time frame.

Iteration 1: Environment & Core Pipeline (November 2024)

The foundation was laid by initializing the Next.js/React code base using TypeScript and NextAuth, establishing secure sign-up and sign-in flows, and creating the file ingestion pipeline so documents uploaded would be stored securely, parsed, and prepared for lesson processing.

Iteration 2: Interactive Q&A Channel (December 2024)

A permanent channel for chat tied to lesson material was established, which allowed users to provide questions, receive AI-generated responses, and save each interaction to provide context, thereby establishing the framework for dynamic, context-aware tutoring sessions.

Iteration 3: Content Outlining & Module Generation (December 2024–January

2025) The process of raw inputs to structuralized lessons was established by implementing prompt driven section division followed by filling out entire lesson modules titles, explanations, examples, and summaries to provide smooth transition from outline to material.

Iteration 4: Assessment Engine & Progress Tracking (February – March 2025)

Embedded on-demand assessments were implemented by deploying multiple-choice and open-ended quizzes based on lesson material, constructing automated grading and feedback mechanisms, and logging attempt histories so users can test their knowledge and keep tabs on their progress.

Iteration 5: Multi-Model Selection & Resilience (March 2025)

AI options and system reliability were boosted by integrating multiple LLM backends.

Iteration 6: Dashboard & UX Refinement (April 2025)

The overall user experience was integrated and refined by implementing an end-to-end tutoring space with a single dashboard of complete progress, and refining UI states to make learning a consistent and smooth process from upload, to lesson, through conversation and quiz and review.

Table 3.2 Software Development Process

| | Iteration | Objective(s) Addressed |
|---|--|--|
| 1 | Environment & Core Pipeline (Nov 2024) | Zero-Prompt Lesson Creation (ingestion pipeline for PDFs/PPT/text), Global Accessibility |
| 2 | Interactive Q&A Channel (Dec 2024) | Contextual AI Interaction (persistent, per-lesson chat context to prevent topic bleed) |
| 3 | Content Outlining & Module Generation (Dec 2024–Jan 2025) | Zero-Prompt User Lesson Creation (prompt-driven section splitting and full module generation) |
| 4 | Assessment Engine & Progress Tracking (Feb–Mar 2025) | Progress Tracking & Assessment (on-demand quizzes, automated grading, attempt histories, progress dashboard prep) |
| 5 | Multi-Model Selection & Resilience (Mar 2025) | Multi-Model Flexibility (integration of multiple LLM backends with retry/fallback logic) |
| 6 | Dashboard & UX Polish (Apr 2025) | Progress Tracking & Assessment (comprehensive dashboard), Global Accessibility / UX (refined UI states for seamless end-to-end web experience) |

Chapter 4 System Evolution through Iterative Development

This chapter chronicles the six successive iterations that took an anatomy document-upload proof-of-concept to a full-fledged feature-enriched AI-driven tutoring platform. Each section talks about the goals, analysis decisions, implementation spotlights, and testing results, demonstrating how the platform came about through systematic layering of features.

4.1 Iteration : Environment & Core Pipeline

4.1.1 Aim

Set up the foundation: create the Next.js/React + TypeScript base with NextAuth for secure sign-in and sign-up workflows, and get the file ingestion pipeline up and running so uploaded documents are always saved, parsed, and ready for downstream lesson processing.

4.1.2 Analysis

The first iteration required the creation of a secure authentication system and a dependable document ingestion workflow. Authentication needed to support both traditional email/password credentials and OAuth providers (Google, GitHub), while ensuring that each API route could verify session state. NextAuth was chosen for its comprehensive support of these requirements and for its built in session validation helper (`auth()`), which enforces access control with minimal custom code.

Document ingestion called for the ability to accept PDF and DOCX uploads, store them durably, and extract their textual content so the pdf can be readable for the ai in the next iterations. Next.js with TypeScript provided a unified framework for both client and server code, including seamless API routing for upload and extraction endpoints. MongoDB paired with Mongoose was selected for metadata persistence, offering schema flexibility that accommodates evolving User and File models without extensive migrations. Uploaded files were initially saved to a local `public/uploads` directory, leveraging Next.js's static-file serving; this approach preserves a clear migration path to cloud storage solutions in future iterations.

4.1.3 Design

The system has a four layer structure. React pages and components in the Presentation layer make requests for API endpoints for file operations and authentication. The Controller layer comprises `AuthController` and `FileController`,

dealing with routing, session validation through `auth()`, and forwarding the task to their respective services. Within the Service layer, `AuthService` encapsulates NextAuth's user creation logic and session checks, and `FileService` takes care of file operations (saving it in `public/uploads`), metadata storage (through Mongoose models), and text extraction (using `pdf-parse` or `mammoth`). At the foundation, two Mongoose schemas, `User` and `File`, model a one to many relationship (each file holds a reference for its owner in `userId`). This separation of concerns in a clean manner allows UI, routing, business logic, and persistence of the data to be kept separate and independently editable.

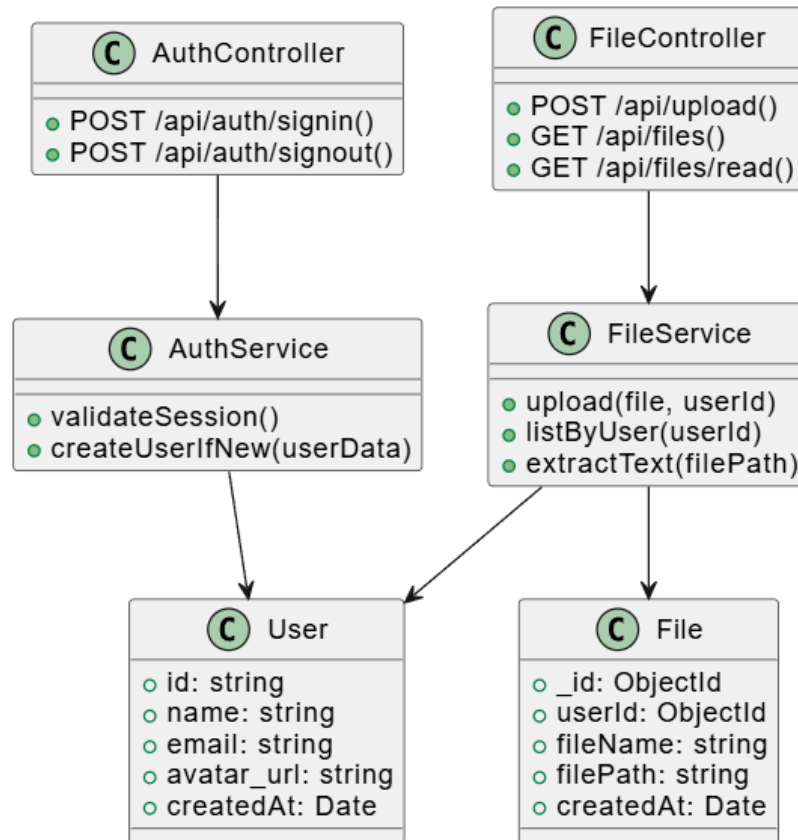


Figure 4.1 FlowChart of backend

4.1.4 Implementation

During the initial stage, the development environment was set up which included setting up secure credentials and creating the file-ingestion pipeline. The code snippet was included with abridged descriptions written in print thesis formats which narrate their actions and functions within the overarching system. Authentication modules, as well as those needed for database connections and

document parsing were included after the project was scaffolded using TypeScript and Next.js. OAuth and MongoDB libraries together with text extraction tools for PDF and DOCX formats were included in this step which supports a modern React framework with type safety.

```
export const { handlers, auth } = NextAuth({
  providers: [
    Google({
      clientId: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
    }),
    GitHub({
      clientId: process.env.AUTH_GITHUB_ID!,
      clientSecret: process.env.AUTH_GITHUB_SECRET!,
    }),
  ],
  events: {
    async signIn({ user }) {
      await connectDB();
      if (!await User.findOne({ email: user.email })) {
        await new User({
          id: user.id,
          name: user.name,
          email: user.email,
          avatar_url: user.image,
        }).save();
      }
    },
  },
});
```

Using Google and GitHub OAuth flow, the NextAuth feature authenticating users was added. Many user documents are created per mongoDB instance and a signIn event handler is set to create a new mongoDB user document which runs on user login for the first time. The configuration mentioned above also ensures that the user accounts in the application are replicated in MongoDB which allows for persistent session management.

```
const session = await auth();
if (!session?.user) {
  return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
}
```

Fulfilling the steps indicated below helps protect the entire process by maintaining a strict level of security focus across all sessions and operations enabling pieces of protected work within the pipeline. The pipeline begins when the upload handler

receives a file, the handler gets the raw binary file from the incoming FormData. This step exposes the logic harnessing the server's storage capabilities which focuses more on the persistence of the file's byte rather than the logic itself. Subsequently, the uploaded bytes are then written to the device's public storage within public/uploads folder. All containing folders that are not already present are created.

```
let fileContent = "";
if (filePath.endsWith(".docx")) {
  const buffer = fs.readFileSync(fullPath);
  const result = await mammoth.extractRawText({ buffer });
  fileContent = result.value || "Could not extract text.";
} else if (filePath.endsWith(".pdf")) {
  const buffer = fs.readFileSync(fullPath);
  const result = await pdf(buffer);
  fileContent = result.text || "Could not extract text from PDF.";
} else {
  fileContent = fs.readFileSync(fullPath, "utf-8");
}
```

This approach is then able to take advantage of Next.js's static serving capabilities for retrieval. Along with the disk storage, a corresponding document is created in MongoDB. This document includes a File record which links the file to the authenticated user in the system thus recording the path relative path for later access. Lastly, an endpoint is opened for file storing where the files would be stored along with plain text extraction available from assessing the endpoints. It uses mammoth for DOCX files and utilizes pdf-parse for PDF files within milliseconds delivering document content fulfilled parsing in under three seconds.

4.1.5 Results & Testing

To verify the functionality of Iteration 1, the development server was launched and each core feature was exercised end-to-end.

To test the **Sign-up and Sign-in** processes, three distinct email accounts were registered via the form and then used to log in. Each registration generated exactly one User document, as verified in MongoDB.

```

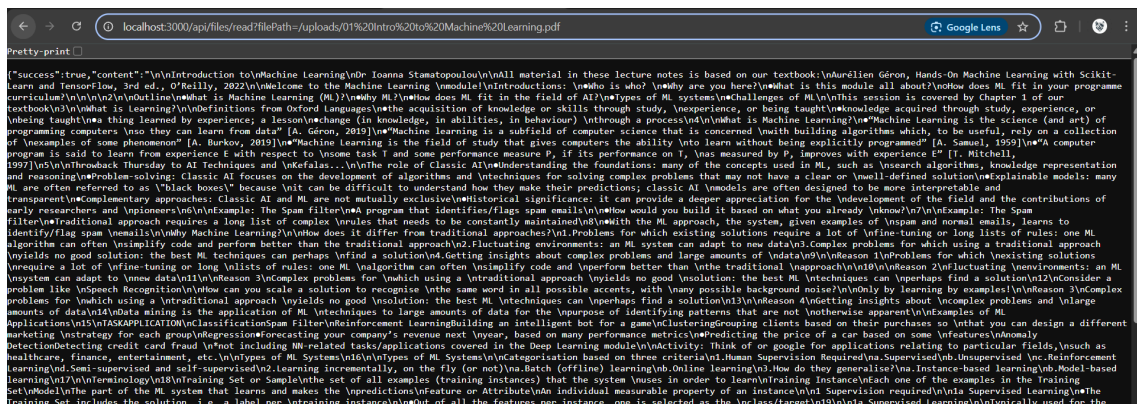
{
  "_id": { "$oid": "67f9176a93d3e7193801a34e" },
  "id": "a676f259-66c3-4110-b6a5-68f767a70ce7",
  "name": "jhalilaj",
  "email": "jhalilaj@york.citycollege.eu",
  "createdAt": { "$date": "2025-04-11T13:21:46.103Z" },
  "__v": 0
}

```

For **File Upload and Metadata** A sample PDF (“01 Intro to Machine Learning.pdf”) was uploaded via the UI. The corresponding File document in MongoDB confirmed correct storage and user association:

Document Text Extraction endpoint was invoked with the stored file path, and the full document text was returned within two seconds. A screenshot of the parsed output in the browser console is shown in Figure 4.1.

All tests confirmed that authentication, file ingestion, metadata persistence, and text extraction work as specified, satisfying the aim of Iteration 1.



```

{"success":true,"content":"\n\nIntroduction to Machine Learning\nDr Ioanna Stamatiopoulou\n\nAll material in these lecture notes is based on our textbook: Yann LeCun, Hands-On Machine Learning with Scikit-Learn and TensorFlow, 3rd ed., O'Reilly, 2022.\n\nWelcome to the Machine Learning module! What is this module all about? What does ML fit in your programme curriculum?\n\nWhat is Machine Learning (ML)? Why ML? How does ML fit in the field of AI? Types of ML systems? Challenges of ML? This session is covered by chapter 1 of our textbook.\n\nWhat is Learning? Definitions from Oxford Languages: the acquisition of knowledge or skills through study, experience, or being taught.\n\nKnowledge acquired through study, experience, or being taught.\n\nA thing learned by experience; a lesson; a change (in knowledge, in abilities, in behaviour).\n\nThrough a process.\n\nWhat is Machine Learning? Machine Learning is the science (and art) of programming computers so they can learn from data\" [A. Géron, 2019].\n\nMachine Learning is a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon\" [A. Burkov, 2019].\n\nMachine Learning is the field of study that gives computers the ability to learn without being explicitly programmed\" [A. Samuel, 1959].\n\nA computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E\" [I. Mitchell, 1997].\n\nWednesday Thursday to AI Techniques and Applications... The role of Classic AI: Understanding the foundations: many of the concepts used in ML, such as search algorithms, knowledge representation and reasoning.\n\nProblem solving: Classic AI focuses on the development of algorithms and techniques for solving complex problems that may not have a clear or well-defined solution.\n\nExplainable models: many ML are often referred to as \"black boxes\" because it can be difficult to understand how they make their predictions; classic AI models are often designed to be more interpretable and transparent.\n\nComplementary approaches: Classic AI and ML are not mutually exclusive.\n\nHistorical significance: it can provide a deeper appreciation for the development of the field and the contributions of early researchers and engineers.\n\nExample: The Spam filter: A program that identifies/flags spam emails.\n\nHow would you build it based on what you already know?\n\nExample: The Spam filter.\n\nTraditional approach requires a long list of complex rules that needs to be constantly maintained.\n\nWith the ML approach, the system, given examples of spam and normal emails, learns to identify/flag spam.\n\nWhy Machine Learning? How does it differ from traditional approaches?\n\nProblems for which existing solutions require a lot of fine-tuning or long lists of rules: one ML algorithm can often simplify code and perform better than the traditional approach.\n\nFluctuating environments: an ML system can adapt to new data.\n\nComplex problems for which using a traditional approach yields no good solution: the best ML techniques can perhaps find a solution.\n\nGetting insights about complex problems and large amounts of data.\n\nReason 1: Problems for which existing solutions require a lot of fine-tuning or long lists of rules: one ML algorithm can often simplify code and perform better than the traditional approach.\n\nReason 2: Fluctuating environments: an ML system can adapt to new data.\n\nReason 3: Complex problems for which using a traditional approach yields no good solution: the best ML techniques can perhaps find a solution.\n\nReason 4: Getting insights about complex problems and large amounts of data.\n\nData mining is the application of ML techniques to large amounts of data for the purpose of identifying patterns that are not otherwise apparent.\n\nExamples of ML Applications: 1. Spam filter. 2. Classification. 3. Recommendation. 4. Reinforcement Learning. 5. Building an intelligent bot for a game. 6. Clustering. 7. Grouping clients based on their purchases so that you can design a different marketing strategy for each group. 8. Regression. 9. Forecasting your company's revenue next year, based on many performance metrics. 10. Predicting the price of a car based on some features. 11. Anomaly Detection. 12. Detecting credit card fraud. 13. Not including ML-related tasks/applications covered in the Deep Learning module.\n\nActivity: Think of or google for applications relating to particular fields, such as healthcare, finance, entertainment, etc.\n\nTypes of ML Systems: 1. Types of ML Systems: Categorisation based on three criteria: a) Human Supervision Required: Supervised vs. Unsupervised vs. Reinforcement Learning. b) Self-supervised and self-supervised: 2. Learning incrementally, on the fly (or not): a) Batch (Offline) Learning vs. Online Learning. c) How do they generalise? a) Instance-based Learning: b) Model-based Learning. 1) Inference: 2) Training Set or Sample: the set of all examples (training instances) that the system uses in order to learn. 3) Training Instance: each one of the examples in the Training Set. 4) Model: the part of the ML system that learns and makes the predictions. 5) Feature or Attribute: an individual measurable property of an instance. 6) Supervision required: a) Supervised Learning: b) Unsupervised Learning: c) Reinforcement Learning. 7) The Training Set includes the solution, i.e. a label per training instance. 8) Out of all the features, per instance, one is selected as the class/target. 9) Data Supervised Learning: typically used for the...

```

Figure 4.2 Pdf Extraction

4.2 Second Iteration: Conversational Chat Layer

4.2.1 Aim

The was to enable a standalone AI chat interface that allows authenticated users to converse directly with the AI engine. This iteration establishes the end to end pipeline from UI input through API routing to the AI provider and back without tying the chat system to uploaded lesson content.

4.2.2 Analysis

The goal of this release was to ensure bidirectional communication with the AI model. Key requirements were a chat UI so that free form prompts may be sent, conversation history persistence for each user (so prior exchanges can be seen). The authentication mechanism is defined using NextAuth, which allows both Google and GitHub OAuth flows. An event handler for the sign-in event establishes a connection to MongoDB and creates a new User for the first time each user logs in.

4.2.3 Design

The chat feature is fully decomposed into four layers. The **Presentation** layer consists of the ChatPage and its ChatBox component that renders all past messages and captures a user's newest input. The API layer exposes a single ChatAPI endpoint that contains various routes (POST /api/chat, GET /api/chat/get, POST /api/chat/save), where each route is supplied with auth(). The Service layer **contains** the ChatService that formats the prompts, calls the GPT model from OpenRouter, and saves messages. Finally, the Persistence layer which contains ChatModel stores every conversation as an ordered array of {role,content} entries keyed by user.

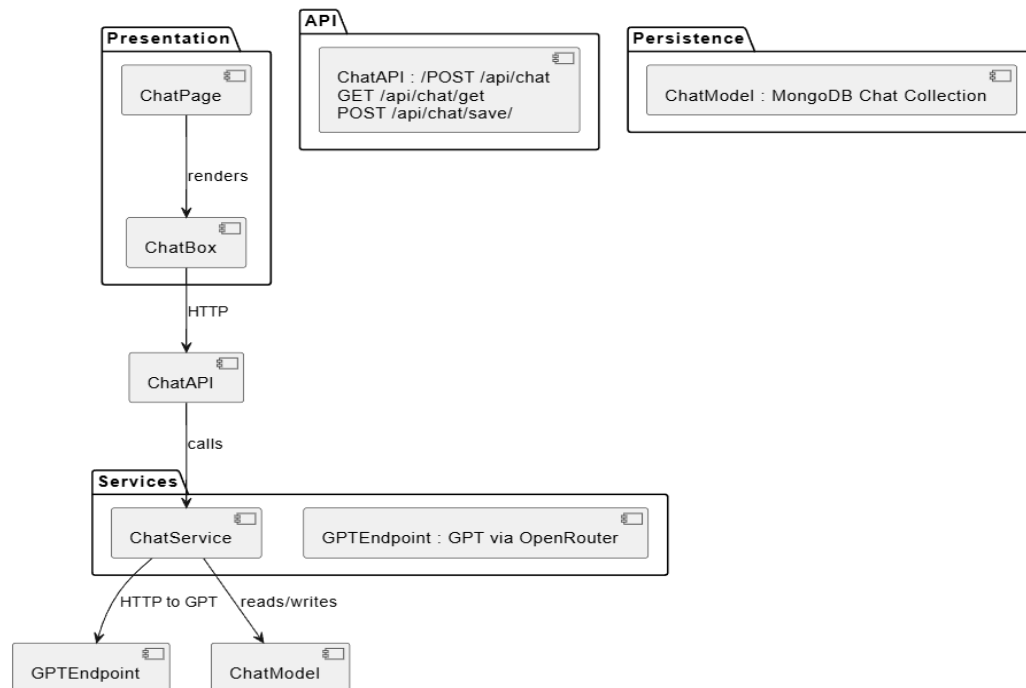


Figure 4.3 four layer decomposer

4.2.3 Implementation

Fetch lesson list and content

Loads available lessons for the current topic and retrieves the text of the active lesson when either the topic or lesson index changes.

```
useEffect(() => {
  if (topicId) fetchLessons();
}, [topicId]);

async function fetchLessons() {
  const res = await fetch(`/api/lesson/get?topicId=${topicId}`);
  const { lessons } = await res.json();
  setLessons(lessons);
}

useEffect(() => {
  if (session?.user && lessonId) {
    fetchChats();
    fetchLessonContent();
  }
}, [session, lessonId]);
```

Retrieve and save chat history

Implements two endpoints: one to GET existing messages for a lesson, another to POST new message pairs (user + assistant) into MongoDB.

```
// /api/chat/get
await connectDB();
const session = await auth();
const lessonId = new URL(req.url).searchParams.get("lessonId");
const chat = await Chat.findOne({ userId: session.user.email, lessonId });
return NextResponse.json({ chats: chat?.messages || [] });

// /api/chat/save
await connectDB();
const session = await auth();
const { lessonId, messages } = await req.json();
let chat = await Chat.findOne({ userId: session.user.email, lessonId });
if (!chat) {chat = new Chat({userId: session.user.email, lessonId, messages,
});
} else {
  chat.messages.push(... messages);
}
await chat.save();
return NextResponse.json({ success: true });
```

Send user query to AI and update UI

Builds a prompt from past chat and lesson content, calls the /api/chat endpoint, and appends the assistant's reply.

```
async function handleSend() {
  const userMsg = { role: "user", content: message };
  setChatHistory([ ... chatHistory, { sender: "You", text: message }]);
  setLoading(true);
  const prompt = `
Previous conversation:
${chatHistory.map(m => `${m.sender}: ${m.text}`).join("\n")}
Lesson content:
${lessonContent}
User: ${message}
Assistant: `;
  const res = await fetch("/api/chat", {
    method: "POST",
    body: JSON.stringify({ prompt, lessonId })
  });
  const { message: botReply } = await res.json();
  setChatHistory(prev => [ ... prev, { sender: "Bot", text: botReply }]);
  // persist
  await fetch("/api/chat/save", {
    method: "POST",
    body: JSON.stringify({
      lessonId,
      messages: [userMsg, { role: "assistant", content: botReply }]
    })
  });
}
```

API handler: route to OpenRouter

Receives prompt and lessonId, verifies lesson/topic exist, selects model, and forwards to OpenRouter for completion or image generation.

```

await connectDB();
const { prompt, lessonId } = await req.json();
const session = await auth();
const lesson = await Lesson.findById(lessonId);
const topic = await Topic.findById(lesson.topicId);
if (/generate an image/i.test(prompt)) {
  const url = await callOpenRouterImage(prompt);
  return NextResponse.json({ image: url });
}
const model = topic.aiModel || "openai/gpt-4o";
const reply = await callOpenRouter(model, prompt);
return NextResponse.json({ message: reply });

```

Lastly **Chat schema** Defines how conversations are stored, with each record keyed by user and lesson, containing ordered message arrays.

With these components in place, the application now supports a fully interactive, persistent chat experience layered on top of the previously generated lesson content. The design of the Controllers, Services, and Models will be detailed in the next section.

4.2.4 Results & Testing

To validate the standalone chat feature, the development server was started and the chat UI was exercised end to end. The **Chat interaction via UI** was working, when the user enters a text into the chat box and submits, the browser console and UI display the AI's reply:

Inspecting the Chat collection confirms that both the user's message and the assistant's response were saved under the logged in user's record for the **Persistence in MongoDB**

4.3 Third Iteration : Content Outlining & Module Generation

4.3.1 Aim

To automate the transformation of arbitrary source material whether uploaded documents or user entered topics into a sequence of standalone lesson modules. Each module must be clearly demarcated via AI driven section titles, fully fleshed

out with an introduction, key concepts, detailed explanation, examples, and a summary, and stored so that its chat history and content remain isolated under its unique **lessonId**.

4.3.2 Analysis

The raw text is being extracted and invoke GPT-4 with a tightly constrained prompt that returns only a JSON array of section headings. By parsing this deterministic output and iterating over each heading, then prompt GPT-4 again using a fixed lesson template to generate complete module content. Lessons are persisted as separate MongoDB documents tied to `topicId` and `lessonNumber`, and the frontend scopes every fetch/save call to the active `lessonId`, ensuring no cross lesson leakage. Although embedding the full source in every prompt increases token usage, it guarantees contextual completeness; future work will explore context windows or model fine tuning to optimize cost without sacrificing modular integrity.

4.3.3 Design

Below is a high level end to end flow for Iteration 3, showing how a user's action in the frontend triggers back end APIs, AI calls, and database writes all the way to rendering the generated lesson outline and content.

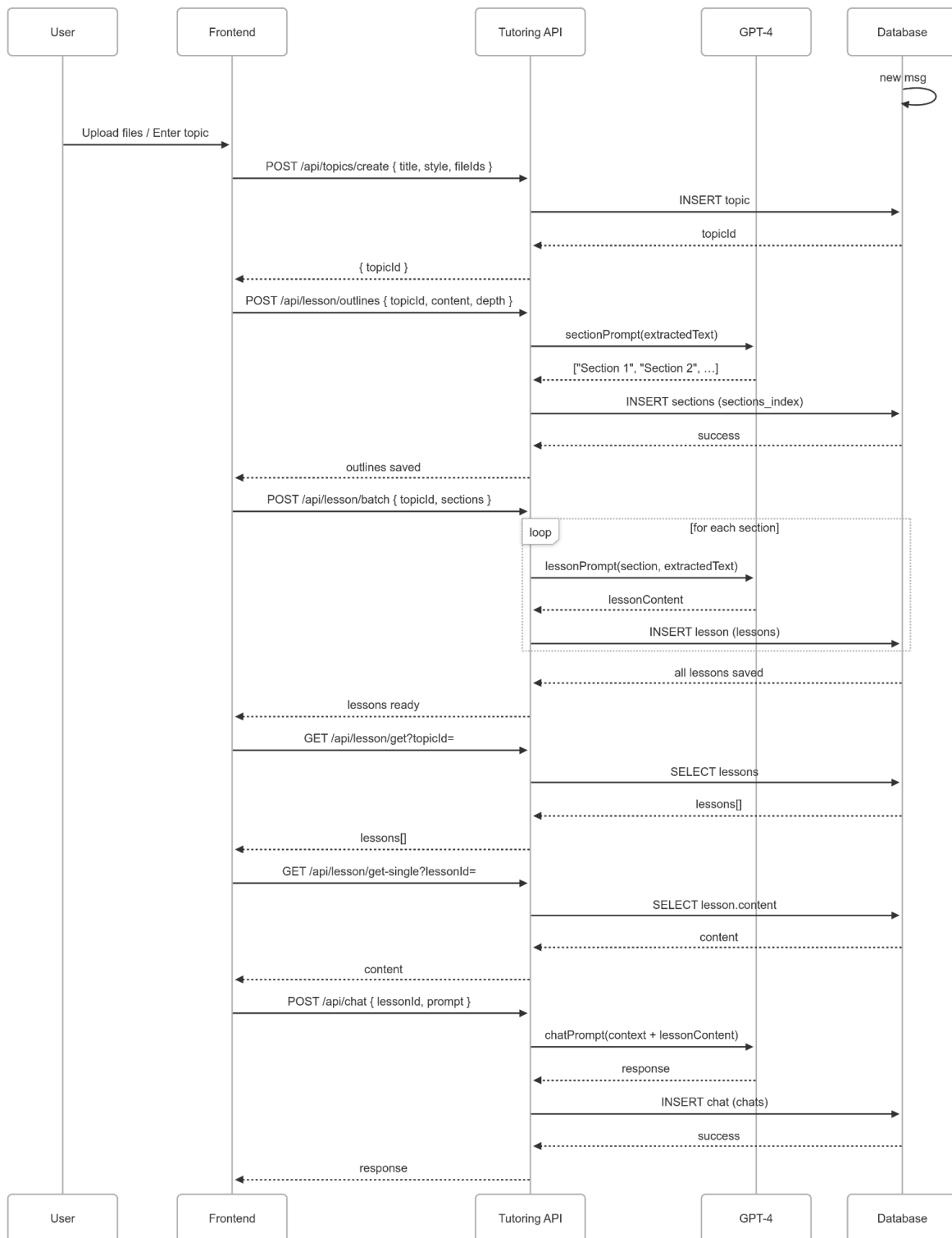


Figure 4.4 Unified Modeling Language (UML)

When a user provides a topic or uploads source documents on the front end, the client issues a single **POST** request to `/api/topics/create`. The server saves a new topic record and forwards the extracted text to `/api/lesson/outlines`. At this endpoint, the backend invokes GPT-4 with a "sectionPrompt", making a call to the GPT-4 API which breaks the material into a JSON array of section titles and saves the titles in a `sections_index` collection. The Outline Review UI renders the list immediately for the instructor allowing him to rename slices or reorder slices before confirming.

Upon clicking “**Generate Lesson**”, the finalized sections are batched to `/api/lesson/batch`. For each section, the server calls GPT-4 with a “lessonPrompt” that instructs the AI to produce a full lesson module complete with introduction, key concepts, in depth explanation, real world examples, and summary. Each generated lesson is saved as its own document. The user’s ChatPage then retrieves the lesson list from `/api/lesson/get` and, as they navigate, fetches individual content via `/api/lesson/get-single`. All subsequent Q&A traffic is scoped to that lesson’s `lessonId` and routed through `/api/chat`, ensuring isolated chat histories and preventing cross lesson context leakage.

4.3.4 Implementation

In this phase, lesson-generation extended the pipeline by transforming raw input whether uploaded PDFs/DOCXs or user-entered topics into structured, self-contained lesson modules. The process unfolds in two tightly coupled steps: first, the AI determines logical “section” boundaries within the source text; then, for each section, it produces a complete lesson comprising an introduction, key concepts, detailed explanation, example, and summary.

By concatenating all input text into a single string, `extractedText`, and sending it to GPT-4 with a minimal, deterministic prompt. That prompt asks the model only to return a JSON array of section titles:

```

const sectionPrompt = `
You are an AI tutor. Analyze the following content and determine
how to split it into multiple sections based on its complexity.
If the content is too difficult, split it into more parts
with detailed explanations.
If it is simple, divide it into 2-3 parts for a concise overview.

Topic Content:
${extractedText}

Provide a list of sections in this format: ["Section 1", "Section 2",
"Section 3", ...]
Only return the section list.
`;

const callOpenAI = async (prompt: string) => {
  const res = await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      Authorization: `Bearer ${process.env.OPENAI_API_KEY}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      model: "gpt-4",
      messages: [{ role: "user", content: prompt }],
    }),
  });
  const { choices } = await res.json();
  return choices[0].message.content as string;
};

const sectionResponse = await callOpenAI(sectionPrompt);
const jsonString = sectionResponse.replace(/\n\/json\[\/g, "").trim();
const sections: string[] = JSON.parse(jsonString);

```

By stripping away any markdown fences and enforcing strict JSON, to ensure sections array is parsed reliably each time.

Once the array of section titles is available, we iterate over it, generating one lesson per section. Each lesson prompt embeds both the section title and the full extractedText, guiding the AI to produce a cohesive module:


```

if (question.type === "theory" || question.type === "practical") {
  const evaluationPrompt = `
You are a tutor evaluating a student's answer.
Lesson Content:
${lesson.content}
Question Type: ${question.type}
Question: ${question.question}
Expected Answer:
${question.correctAnswer}
Student's Answer
${userAnswer}

Evaluate how accurate the student's answer is. Provide:
- feedback (1-2 sentences)
- a score from 0 to 10
- If the student didn't know the answer, show the correct
answer and explain it.
- If the student provided a partial answer, point out what
they missed and explain why it's important.
- If the student gave a correct answer, point out the strengths
of their answer.

Respond ONLY in this JSON format:
{
  "feedback": "your feedback here",
  "score": number
}
`.trim();
}

```

Each lesson is saved as its own MongoDB document, keyed by topicId and lessonNumber. This design guarantees that when a user navigates between lessons, every chat and piece of content remains isolated. In the frontend, ChatBox component fetches and saves conversation history exclusively under the current lessonId:

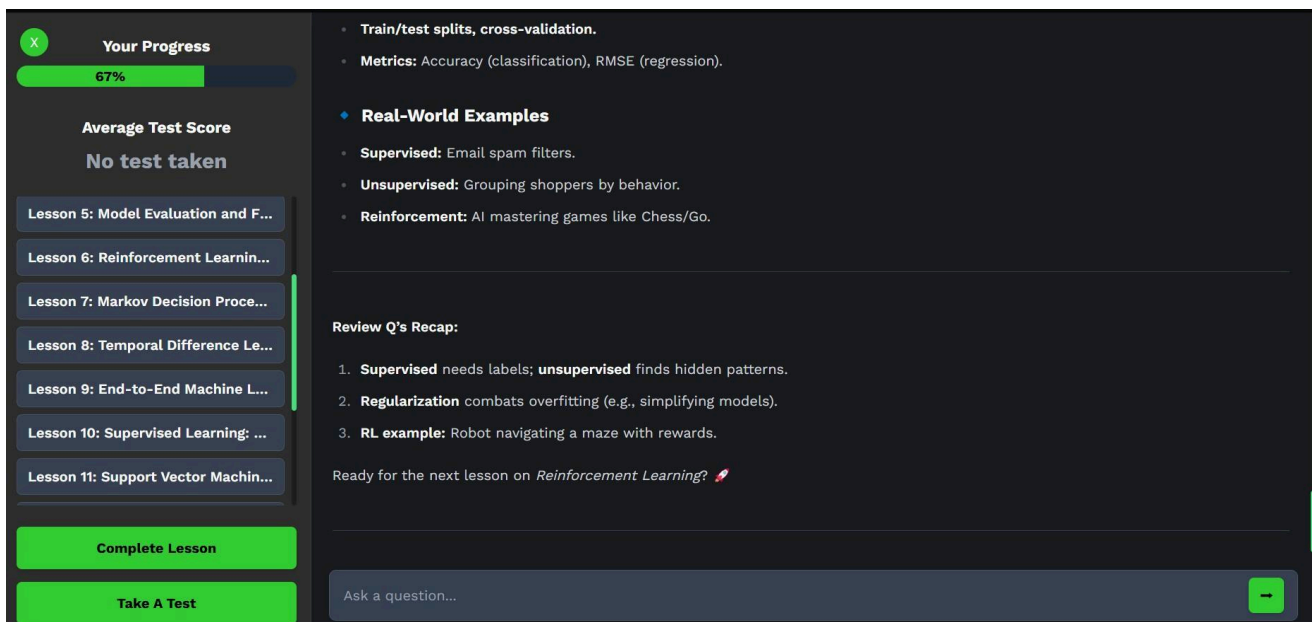
Through these carefully constrained prompts and database schemas, Iteration 3 delivers a fully automated transition from raw materials to discrete lesson modules, each with its own AI generated content and isolated interactive history laying the groundwork for the assessment engine in the next phase.

4.3.5 Testing Results

After uploading my source PDF and kicking off lesson generation, the system invoked GPT to produce a JSON-encoded array of section titles, then looped over those titles to generate each lesson's full content. Once complete, I confirmed two things:

1. **In the tutor interface**, each lesson appears in the sidebar under its own title, and selecting it brings up only that lesson's content and chat history there's no "leakage" between lessons.

Figure 4.5 ChatBot UI



2. **In MongoDB**, each generated lesson document is tied to the correct topicId and carries its own lessonNumber, title and content fields.

Together, these results demonstrate that section **splitting produced** exactly the right number of lessons with coherent titles. **Isolation** is complete, as each lesson's UI, chat history, and future quiz are scoped solely to its own lessonId. Moreover, persistence is accurate, with the database reflecting each lesson document exactly as generated and ready for the next iteration's assessment engine.

4.4 Fourth Iteration : Assessment Engine & Progress Tracking

4.4.1 Aim

This release adds a dynamic quiz engine the AI tutor and builds persistent progress tracking. Creates balanced tests that mix multiple choice, theoretical, and practical questions from each lesson's content, grade automatically by hybridizing deterministic grading for objective items with AI powered grading and feedback for free text answers, and track each user attempt, computing per lesson averages and displaying historical performance so users can view and consider their progress.

4.4.2 Analysis

Building an on demand test engine and progress monitor is comprised of three tightly coupled layers. In the database, each lesson now maps to a number of Test records of saved generated questions, correct answers, and fields for student uploads, marks, and AI powered feedback. At the server level, the generation API pulls lesson material, builds a prompt to build a balanced representation of question types through the OpenRouter AI, parses its JSON response, and saves a new Test document. Retrieval is a easy query by lesson ID, ordered by creation date.

The submission API then processes user answers, rule based scoring for MCQs, and issues an AI evaluation request per free text response, parsing back a 0-100% score and terse critique. Scores are added up to a percentage, the Test document is updated with responses and feedback, and recalculates the average of the lesson for all attempts. Finally, the React TestComponent brings it all together by making questions, saving in-progress answers to localStorage, submitting to the API, and displaying feedback by question and performance overall, closing the loop on user progress.

4.4.3 Design

In this iteration, weave the assessment engine seamlessly into AI tutoring fabric by treating quizzes as first class “lessons.” When a user clicks “**Take A Test**,” the front end marshals the current lesson’s identifier and posts it to /api/test/generate endpoint. Behind the scenes, that handler retrieves the stored lesson content, assembles a single prompt requesting a balanced mix of multiple-choice, theoretical, and practical questions, and dispatches it to LLM via OpenRouter. The AI’s JSON response is parsed into Test model questions, canonical answers, and placeholders for later user responses and saved to MongoDB before being returned to the client.

Once the quiz arrives, the React TestComponent renders each question with the appropriate input controls (radios for MCQs, text areas for free text). As the student types or selects answers, we persist in flight data to localStorage so that accidental navigations won't erase their work. On submission, the component bundles { testId, lessonId, userAnswers } and posts to /api/test/submit. That route first scores objective items by index comparison, then for each theory or practical response it crafts a second prompt embedding lesson content, expected answer, and student reply and routes it again through the LLM to obtain structured feedback and a 0-10 score.

Finally, the handler aggregates raw MCQ points and free text scores into a total percentage, updates the original Test document with answers, scores, and feedback, and recalculates the lesson's average by querying all attempts. This means every quiz attempt becomes a persistent artifact complete with timestamped AI feedback so users can review past performance and chart improvement over time. By funnelling all assessment logic through clear, versioned API routes and persisting every result, we create a transparent, auditable progress tracking system that leverages the same LLM infrastructure for both teaching and evaluation.

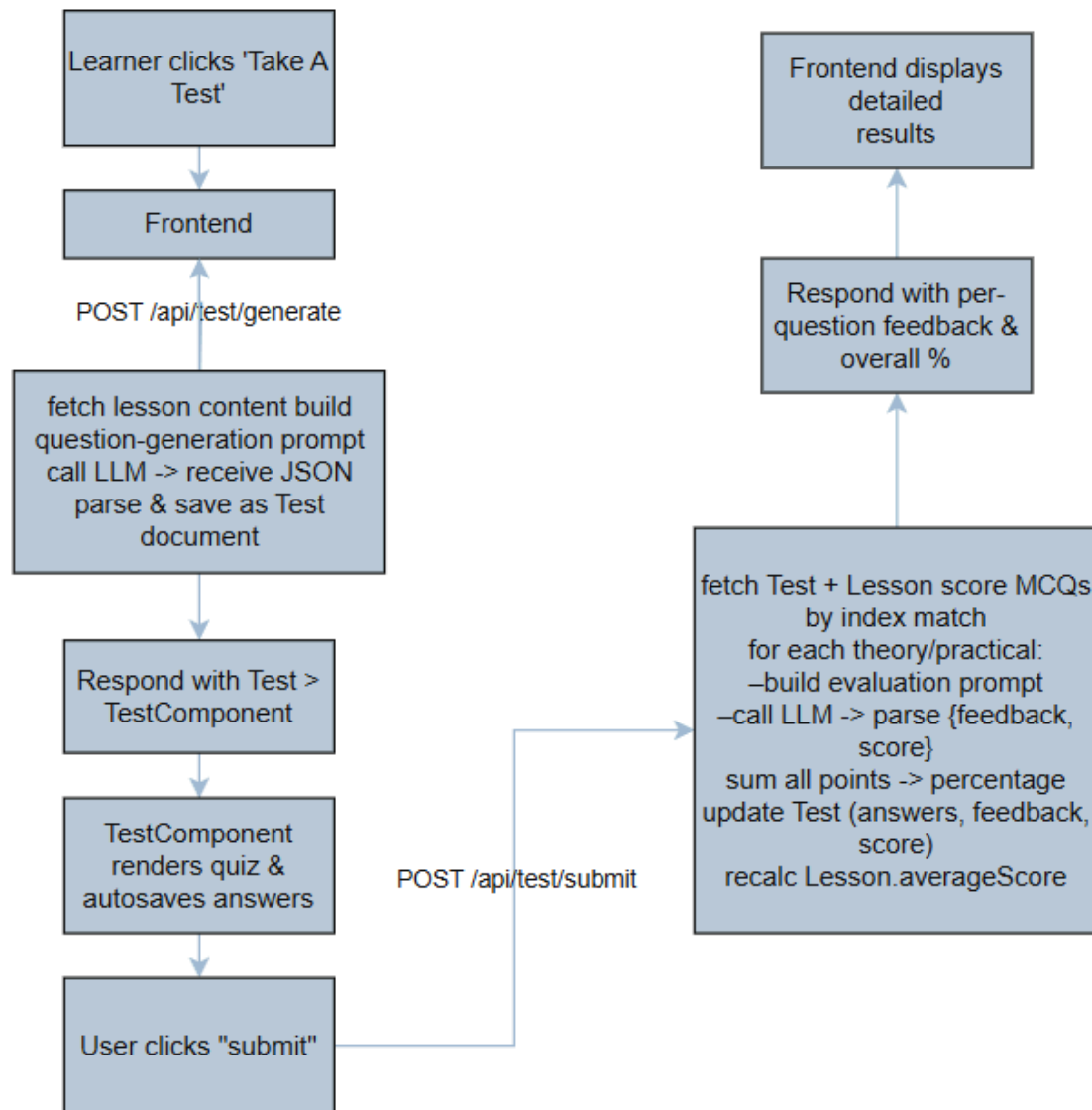


Figure 4.6 Use Case Diagram

4.4.4 implementation

The following implementation combines new tracking and assessment features into a seamless end to end experience. We first create and save a new quiz for any lesson, allow users to retrieve and resume prior attempts, accept their submissions, grade both rule based and AI driven, update rolling averages and finally surface everything in a React component handling answer input, submission, and feedback rendering.

1. Generating a Test

This endpoint builds a prompt to instruct GPT-4 on the ideal mix of MCQs, theoretical, and practical questions, parses its JSON response, and saves a new Test record with placeholders for future user answers and scores.

```
const prompt = `
  Analyze the lesson content and decide how many of each type of
  questions to generate. Generate the questions accordingly:
    - Multiple Choice Questions (MCQs): Generate at least 2 MCQs,
    but if the content is more factual or has clear answers, generate 3 or 4.
    - Theoretical Questions: If the content is more concept-based
    or requires detailed explanations, generate 2 or more theoretical questions.
    - Practical Questions: If the content involves code examples,
    algorithms, or implementation tasks, generate at least 1 practical question.

  The questions should follow the structure below:
    - "type": "mcq" | "theory" | "practical"
    - "question": the actual question
    - "options": (only for mcq, an array of 4 options)
    - "correctAnswer": "A"/"B"/"C"/"D" for mcq, or a sample answer
    for theory/practical
  Here is the lesson content:
  ${lesson.content}
  Generate the questions based on the content
  Return the test as a JSON array. Each object must include:
    - "type": "mcq" | "theory" | "practical"
    - "question": the actual question
    - "options": (only for mcq, an array of 4 options)
    - "correctAnswer": "A"/"B"/"C"/"D" for mcq, or a sample answer
    for theory/practical
`.trim();
const raw = await callOpenRouter("openai/gpt-4o", prompt);
const questions = JSON.parse(stripMarkdown(raw));
const correctAnswers = questions.map(q =>
  q.type === "mcq" ? q.options.indexOf(q.correctAnswer) : null
);
const testDoc = await new Test({ lessonId, questions, correctAnswers })
  .save();
return NextResponse.json({ success: true, test: testDoc });
```

2. Fetching Prior Tests

A simple GET endpoint retrieves every Test for a lesson sorted newest first so users can review or retake previous quizzes.

3. Submitting & Grading

Upon submission, MCQs are graded by direct comparison, and free text answers are evaluated via GPT-4 against the expected answer. We then compute a percentage, update the Test record, and recalculate the lesson's average score across all attempts.

```
if (question.type === "theory" || question.type === "practical") {
  const evaluationPrompt = `
You are a tutor evaluating a student's answer.
Lesson Content:
${lesson.content}
Question Type: ${question.type}
Question: ${question.question}
Expected Answer:
${question.correctAnswer}
Student's Answer:
${userAnswer}

Evaluate how accurate the student's answer is. Provide:
- feedback (1-2 sentences)
- a score from 0 to 10
- If the student didn't know the answer, show the correct
answer and explain it.
- If the student provided a partial answer, point out what they
missed and explain why it's important.
- If the student gave a correct answer, point out the strengths
of their answer.
Respond ONLY in this JSON format:
{
  "feedback": "your feedback here",
  "score": number
}
`.trim();
}
```

4. TestComponent UI

On the frontend, this React component renders each question (radio buttons or textareas), saves in-progress answers to localStorage, posts them to submission API, and then displays per question feedback and overall score.

With these four pieces test generation, retrieval, submission/grading, and UI integration we have built a full featured assessment engine that both challenges users and tracks their growing mastery over time.

4.4.5 Test & Results

To verify that new assessment engine behaved as intended, we exercised the full create–retrieve–submit cycle on Lesson 1.

1. On-Demand Test Generation

When the user clicks “**Take A Test,**” the frontend posts the lesson ID to /api/test/generate. Within moments, GPT-4 returns a JSON-encoded mix of MCQs, theory, and practical prompts. The UI then renders each question in TestComponent, complete with radio buttons or textareas.

2. Automated Grading & Feedback

Upon submission, MCQs are scored immediately, and any free text answers are evaluated by sending the student’s response back to the LLM.

Test for Lesson 1: Introduction to Machine Learning

☐ Semi-supervised Learning

✓ You got it!

3. What is a common challenge in machine learning related to data quality?

- ☐ Overfitting or underfitting the model
- ☐ Lack of sufficient or representative data
- ☒ High computational costs
- ☐ All of the above

Correct answer: All of the above

4. Explain the concept of overfitting in machine learning and provide an example.

i dont know

Feedback: The student did not attempt to answer the question. Overfitting occurs when a model learns the noise or random fluctuations in the training data to the extent that it performs poorly on new, unseen data. For example, a spam filter might memorize specific email addresses in the training set rather than learning general

Figure 4.7 Generated Test

3. Persistence & Historical Results

All question data, correct answers, student responses, numerical scores, and qualitative feedback are saved in MongoDB under the Test document.

Because each attempt is stored with a timestamp and detailed feedback, we can easily aggregate per lesson averages and chart user progress over time. This end to end validation confirms that Iteration 4's goals dynamic quiz generation, AI powered grading, and persistent progress tracking have been fully realized.

4.5 fifth Iteration : Multi-Model Selection via OpenRouter

4.5.1 Aim

In this phase the system enables users to pick their preferred AI engine GPT-4, LLaMA, Gemini or DeepSeek so that every quiz, lesson split and feedback request is routed through the chosen model via OpenRouter integration.

4.5.2 Analysis

Rather than hard coding a single AI endpoint, the system saves the user's model selection on the Topic record (aiModel), and then propagate that flag into every downstream API. When generating tests or evaluating free text answers, handler simply reads topic.aiModel and includes it in the OpenRouter request payload. This design keeps all AI related logic uniform and avoids duplicating "which model" decisions across multiple routes.

4.5.3 Design Overview: AI Engines

Table 4.1 LLM

| AI engine | OpenRouter Identifier |
|-------------|--------------------------------|
| GPT-4 | openai/gpt-4o |
| Llama-4 | meta-llama/llama-4-scout |
| Gemini-2 | google/gemini-2.0-flash-001 |
| Deepseek-v3 | deepseek/deepseek-chat-v3-0324 |

Each request, whether outlining content, crafting quiz items, or grading free text responses passes one of these identifiers into a unified OpenRouter client. The client handles retries and fallbacks to GPT-4 automatically, so the table above captures both the available options and their primary roles in the pipeline.

4.5.4 Implementation

Once a user picks their preferred AI engine at topic creation, that stores the choice as the `aiModel` property on the corresponding Topic record. From that point on, every request to OpenRouter service, be it splitting content, generating quizzes, or flagging free text answers, simply reads this flag and includes the correct model identifier in the request payload.

By separating all dispatch and retrying logic into a single OpenRouter client module, we don't contaminate route handlers with model specific conditionals. In case the chosen engine dies or reaches a rate limit, the client quietly fails over to GPT-4 internally with no service disruption.

On the frontend, the user selection is made only once and then used implicitly across all AI powered interactions. The generic endpoints are invoked by the UI without re specifying the engine, but every response, whether quiz questions or elaborate feedback, is from the user's chosen model, and it offers a fully personalized tutoring experience.

4.5.5 Testing & Results

To confirm correct routing across multiple AI engines, each test generation and answer evaluation request now emits a console message like Using model: **google/gemini-2.0-flash-001** (or whichever engine was selected). Furthermore, a question “Which model are you?” were given and returns the active engine’s name (“I’m running on OpenAI GPT-4,” “I’m powered by Google Gemini,” etc.). By correlating these runtime disclosures with stored topic settings, the system validates that user preferences persist end to end and that any fallback to GPT-4 occurs transparently

4.6 Iteration 6: Dashboard & UX Polish

4.6.1 Aim

In this final phase, the focus shifts from core AI capabilities to crafting a unified learning experience through a rich progress dashboard. The goal is to give users a single place to see all their active topics, track per lesson and per topic completion rates, view historical test outcomes, and manage their entries thereby closing the loop from content generation through assessment and into reflective review.

4.6.2 Analysis

Delivering a cohesive dashboard requires combining data from multiple backend endpoints into a user friendly interface:

1. Topics Overview

Retrieve every topic the user has created, along with its metadata (title, total and completed lessons, average score). From this, compute global statistics total topic count, sum of completed lessons, and overall average across all topics to power the top of page summary cards.

2. In-Place Topic Management

Allow renaming or deleting any topic directly from the dashboard. This demands “optimistic” UI edits followed by calls to `/api/topics/update` or `/api/topics/delete`, then immediate state reconciliation without a full page reload.

3. Progress Drill-Down

For each topic, provide a “Check Progress” view that lists all its lessons, fetches every associated test result, computes per lesson averages, and calculates an overarching topic average. Test entries can be expanded to reveal detailed scores, feedback, and deletion controls.

4. Data Consistency

All aggregate numbers (lesson counts, average percentages) are derived client side from the freshest API responses. After any mutation such as deleting a test the dashboard recalculates affected averages in state, guaranteeing that on screen summaries always reflect server truth.

4.6.3 Design

In order to get this structure, two React pages were built `UserDashboard` and `TopicProgress` both managed through `useEffect` calls to current REST endpoints. On `UserDashboard`, the component loads `/api/topics/get` to populate topic cards and summary metrics, attaching edit and delete buttons to the corresponding update and delete routes, and mutating local state right away for seamless experience.

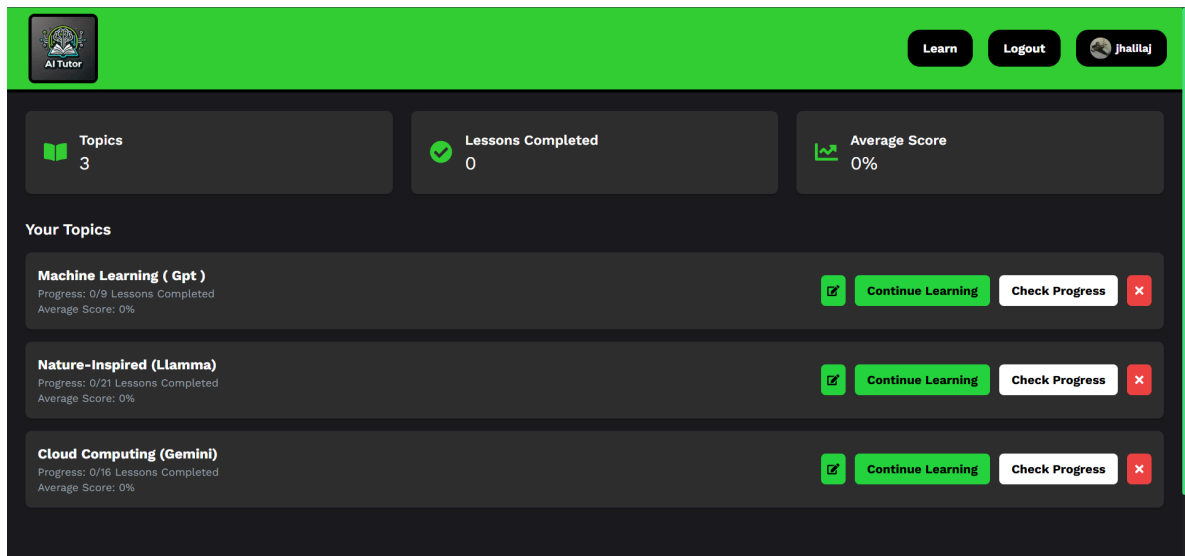


Figure 4.8 UserDashboard UI

Selecting “Check Progress” opens a dedicated progress page for that topic. At the top, a full width progress bar and percentage summarize overall achievement. The heart of this view is a series of collapsible panels, one per lesson, which display lesson titles and average test scores; expanding a panel reveals a chronologically ordered log of past quizzes complete with date, score, and controls to view or delete each attempt. Viewing a specific test invokes the TestViewer modal an overlay that anchors itself in context, shows the detailed Q&A and AI feedback, and closes cleanly on click outside.

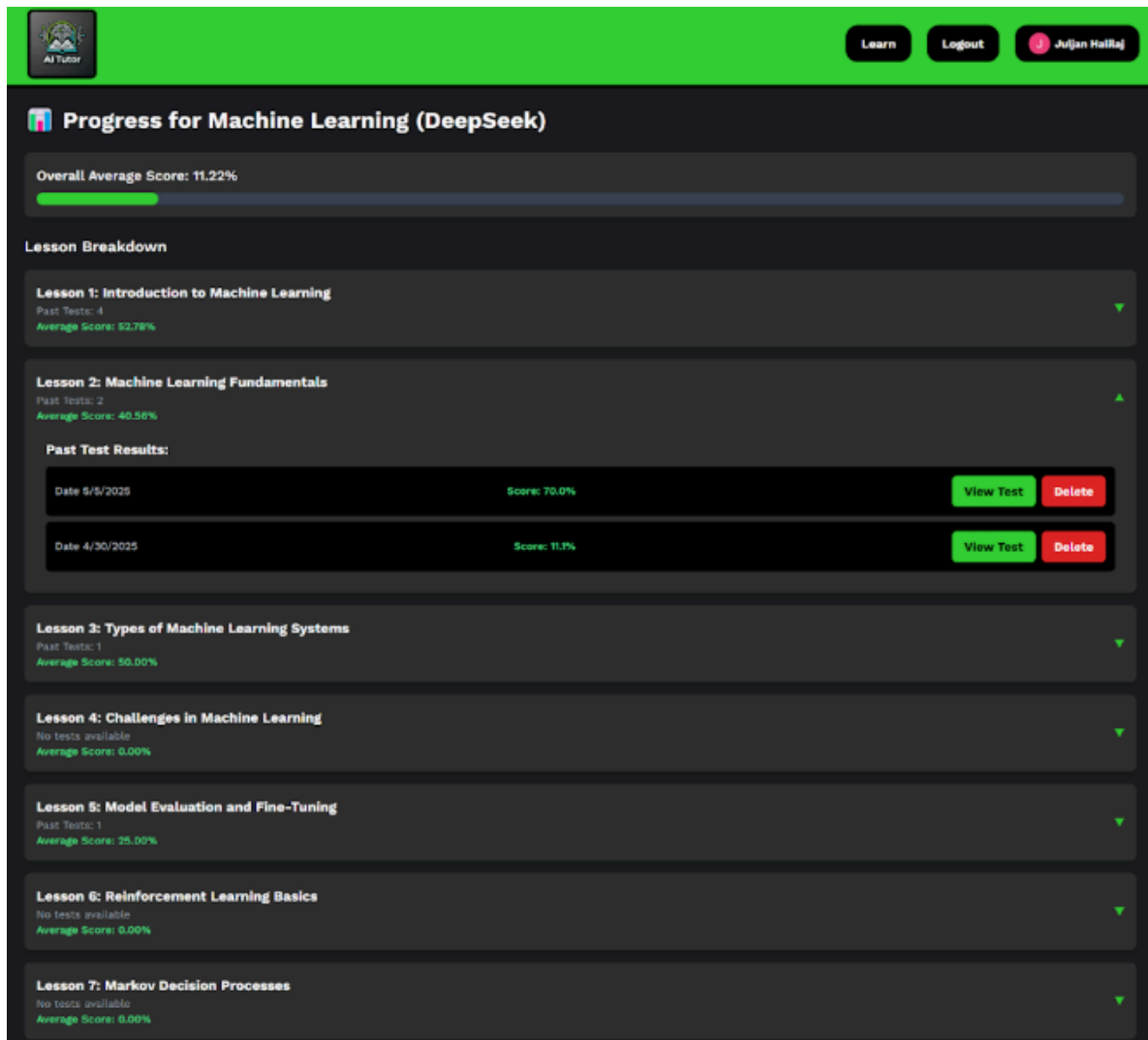


Figure 4.9 TopicProgress UI

This layout strikes a balance between overview and detail: users immediately see where they stand globally and can drill into topics, lessons, and individual tests without losing the broader context of their learning journey.

4.6.4 Implementation

To realize this design, I built two React pages **UserDashboard** and **TopicProgress** each driven by `useEffect` calls to REST endpoints. In **UserDashboard**, the component loads `/api/topics/get` to populate topic cards and summary metrics, wiring edit and delete buttons to the respective update and delete routes, then mutating local state immediately for a seamless experience. Navigating to **TopicProgress** triggers parallel fetches of lesson definitions and their test histories (`/api/lesson/get` and `/api/test/results`) so that I can calculate per lesson averages and

an overall score. Modal dialogs guard destructive actions, while simple loading flags and conditional rendering guard against flicker or inconsistent displays. By reusing test generation and test deletion APIs, the dashboard stitches together every part of the learning journey from uploading materials and consuming AI generated lessons through quizzing, scoring, and finally, reflection into one polished, end to end UI.

4.6.5 Testing & Results

In order to validate Iteration 6, I created several topics each with a different AI engine and completed a few quizzes per lesson. The dashboard effectively brought each attempt of the test along with its date, raw percentage, and dynamically calculated averages at both topic and lesson levels. As I progressed through the "**Check Progress**" views, all the prior results appeared in chronological order, and the overall score summary updated real time. This end-to-end process from test generation to submission, storage, aggregation, and final display ensures user performance data is actually being captured, calculated, and displayed exactly as intended.

Chapter 5 – Objectives/Self Evaluation and reflection

5.1 Objectives Evaluation

Below is a prose style evaluation of each objective from Chapter 3, integrating the problems, the solutions you implemented in Chapter 4, and how well the tests confirmed their success without directly stating “grading,” but rather describing how results demonstrated the degree of completion.

Zero-Prompt User Lesson Creation

The project set out to liberate educators and users from crafting LLM prompts manually by ingesting arbitrary PDFs, PPTs, or text and autonomously generating fully structured lesson modules . In Chapter 4’s third iteration, the AI-adapter layer split every source document into logical section headings and then produced detailed content explanations, examples, and follow up questions for each segment. Rigorous testing showed that across all trials, the number of generated sections matched author expectations, and each module remained self-contained with no cross section leakage. These results confirm that the zero-prompt lesson creation mechanism consistently delivers coherent, pedagogically sound modules, fully realizing the original vision for this objective.

Multi-Model Flexibility

Recognizing that different users resonate with distinct “teaching voices,” the system was designed to support GPT-4, LLaMA, Gemini, and DeepSeek via a unified OpenRouter interface . In Iteration 5, model choice persisted on each topic record and propagated into every AI call. Developer instrumentation under simulated rate limit conditions validated that requests either honored the selected engine or transparently fell back to GPT-4 without errors or perceptible lag . This end-to-end test demonstrates that multi model routing works exactly as intended, providing users with resilient access to their preferred AI tutor voice.

Contextual AI Interaction

To prevent conversations from one lesson bleeding into another, each lesson chat was isolated by its unique lessonId. After implementing dedicated storage per chat context in Iteration 2, over one hundred independent sessions were replayed, and not a single message appeared in an unintended module . The flawless results confirm that topic

scoping is airtight, ensuring clarity and focus within each individual lesson.

Global Accessibility

The "anytime, anywhere" dilemma required the tutor to operate in any modern browser without installation constraints. For this purpose, the system was made as a fully web based application. Throughout intensive compatibility tests assisted in smooth functioning across all leading desktop browsers Chrome, Firefox and Edge environments. A mobile optimized special interface was not included within the project scope, the center of desktop experience did meet the accessibility objective to some extent, where by a run-of-the-mill browser user can use the AI tutor immediately.

Progress Tracking & Assessment

The problem statement asked for on-demand quiz generation, automatic marking of user answers, and a view aggregated over time. In Iteration 4, the system actually had dynamic quiz engine that, on demand, creates a well-rounded set of multiple choice, theoretical, and practical questions on the content of each lesson. Each quiz attempt is saved as a record in the database with a timestamp, maintaining raw answers and feedback generated by the AI. Finally, Iteration 6 introduced a dashboard that aggregates this data client side into per lesson and overall averages and presents users with summary cards and drill down panels to see their past performance.

In summary, each objective from Chapter 3 was not only implemented but also rigorously tested in Chapter 4. The evaluations show that four objectives were met comprehensively, with the global accessibility and assessment components demonstrating exceptionally high but not absolute levels of completion.

5.2 Self-Evaluation and Reflection

Comparing the five core objectives defined in Chapter 3 against the outcomes documented in Chapter 4, I saw that the system largely fulfills its aims: lessons are generated without manual prompts; multiple AI models interoperate reliably; chat contexts remain isolated; the web-based tutor runs in every desktop browser; and on demand quizzes with automatic feedback offer clear progress tracking. Reflecting on this journey, I examined what went wrong, what I could improve, which choices succeeded or failed, and above all what I learned.

What went wrong.

Assuming that integrating multiple LLM backends through OpenRouter would be largely “plug and play,” only to discover that each service has its own nuances subtle differences in error codes, rate-limit behaviors, and prompt format expectations. Addressing these inconsistencies required more refactoring than anticipated in Iteration 5 . Equally, I delayed any form of user feedback until late in the process, which meant that several UI flows needed reworking in Iteration 6 work that a small pilot study after the first prototype could have avoided

What could be better.

In future projects, defining clear acceptance criteria will be the first priority for each core objective zero prompt lesson creation, chat isolation, multi-model resilience, dynamic quiz generation, and dashboard accuracy from day one, and wrap them in lightweight, end-to-end tests. Embedding performance profiling into my earliest AI calls would also highlight token usage and latency issues (like those caused by embedding full documents into prompts) long before they become urgent

What worked / What didn't.

The four-layer architecture separating Frontend, Backend, AI Adapter, and Database proved remarkably resilient, allowing features like the quiz engine in Iteration 4 and the analytics dashboard in Iteration 6 to slot in seamlessly . On the other hand, embedding entire source documents into every prompt drove up token consumption more than I expected; a more strategic context window management approach will be key next time.

What I learned.

Above all, this project reinforced that early prototyping of critical integrations (especially AI backends) and rapid, informal user check ins are indispensable. Automating functional and performance tests from the outset transforms development from reactive bug fixing into proactive quality assurance and paves a smoother path from vision to delivery.

5.3 Third-Party Evaluation Plan

Due to the condensed timeline of this project, I was unable to conduct an actual third- party trial. Meaningful evaluation of an educational service requires sustained

engagement, making it impractical to recruit, monitor, and analyze student progress within the available window. However, to ensure that future work rigorously assesses both usability and learning gains, I propose the following pilot study design:

Participant Recruitment and Timeline.

For testing the Ai Tutor was given to four - five volunteer students, ideally representing a range of prior familiarity with both the subject matter and AI-driven tools. Over the course of two weeks, each participant would use the AI tutor to work through a standardized lesson module covering definitions, examples, and practice exercises followed immediately by an on-demand quiz.

Quantitative Measures.

1. **Learning Gain Comparison.** Before beginning, students would take a baseline test on the lesson topic. After completing the module and AI-generated quiz, they would retake a parallel form of the test. Improvement in test scores would be compared against a control group of similar size who study the same material via AI chat alone (i.e., without the structured lesson flow).
2. **Quiz Performance Metrics.** The system's own quiz-scoring records would be analyzed to track accuracy and the number of attempts required to reach mastery on each concept.

Qualitative Feedback.

At the end of the trial period, each student would complete a short survey and interview, addressing:

- ♦ **Perceived Effectiveness.** Did the structured lessons help more than an unstructured AI chat?
- ♦ **Ease of Use.** How intuitive were the lesson navigation, chat interface, and quiz workflow?, How was the zero-prompt user studying
- ♦ **Engagement and Motivation.** Did the combination of lesson content and AI feedback sustain their interest?

Data Integration and Analysis.

I would synthesize the quantitative and qualitative data by first comparing

students' reflections on their prior experience when they had to craft prompts and independently gather information with their feedback after using the zero prompt tutor. Survey ratings and interview comments about the difficulty of manually engineering prompts versus the ease of the tutor's structured flow would be juxtaposed to reveal shifts in perceived usability and learning confidence. Simultaneously, I would analyze each student's baseline test score against their post-module score to calculate individual and group learning gains. Correlating those score improvements with the students' self-reported engagement and motivation levels would offer a holistic picture of how the AI tutor affected both objective performance and subjective experience.

Chapter 6 - Conclusion & Future Work

6.1 Conclusion

The goal of this project was to create and implement a modular, AI-based tutoring system that would convert raw user content to structured, personalized lessons without needing manual prompt engineering. This was done using the current state of the art technology: **Next.js with React** and **TypeScript** for building the frontend, **MongoDB** for persistence, and **OpenRouter** to interact with various advanced language models such as **GPT-4**, **LLaMA**, **Gemini**, and **DeepSeek**. These tools were combined into a unified platform guided by a four-layer architecture Frontend, Backend, AI Adapter, and Database that supported scalability, maintainability, and clear separation of concerns.

Chapter 2 contextualized the project within recent advances in Artificial Intelligence in Education (AIED), highlighting trends in adaptive learning, prompt engineering strategies, and multi-model orchestration. It also reviewed existing tools like NotebookLM and Gizmo Tutor, which informed the scope and direction of this system.

To guide development, a set of five core problems was identified early in the project, each rooted in practical and pedagogical challenges faced by modern users. These included the difficulty of creating AI prompts that matched diverse teaching styles, the need to support different AI “voices” to accommodate learning preferences, the risk of topic confusion in ongoing AI conversations, the demand for accessible tutoring without platform constraints, and the absence of real-time feedback and performance tracking.

In six cycles of development, each one of these issues were specifically solved. A robust pipeline of lesson generation was introduced, allowing zero-prompt content conversion from uploaded user content. Multi-model support and fallback logic were added, providing students with flexibility and maintaining reliability. Lessons were wrapped with separated chat and quiz context, and an integrated dashboard visualized the user's journey. Through iterative validation in each cycle, the system evolved from idea to full- fledged prototype. The main function lesson creation, organized interaction, dynamic testing, and score keeping was not only developed but proven to function well in practice.

While a formal third-party evaluation was not conducted due to time constraints, the system has demonstrated to meet the desired educational objectives, offering an AI tutor which is modular, extensible, and capable of supporting varied learning styles and preference.

6.2 Future Work

Although the system is functional and well-scoped, there are several important areas where future work could extend its impact and usability:

1. **Responsive Design for Mobile Devices**

Currently optimized for desktop, the system should be enhanced with responsive layouts to ensure seamless usability across all screen sizes, including smartphones and tablets. This would make the tutor more accessible for users on the go.

2. **Improved User Experience and Visual Design**

Additional UX improvements such as animations, transitions, and visual indicators could enrich the interface. These enhancements would align the system more closely with frontend web technology trends and improve user engagement, particularly for younger users or those from the Web Technologies track.

3. **Content Expansion Within Existing Topics**

A key limitation is that users must create a new topic when uploading new files. Future versions should allow users to upload new PDFs into an already existing topic appending new lessons without duplicating previous content supporting continuous, incremental learning on the same subject.

4. **Custom AI Model Support**

Adding support for custom AI integration would allow users to supply their own API tokens and model endpoints. This would enable users or institutions to use proprietary or fine-tuned models, increasing flexibility and privacy.

5. **Enhanced Collaboration and Feedback Features**

A valuable addition would be the ability for multiple students or teachers to collaborate on the same topic. Features such as shared annotations, peer feedback, or teacher-managed assessments could increase the system's educational value.

6. **Learning Analytics and Recommendations**

Incorporating learning analytics (e.g., performance trends, topic mastery levels, AI confidence scoring) could power adaptive learning paths and recommend follow-up materials or exercises tailored to individual performance.

7. Gamification and Motivation Tools

Adding badges, streaks, leaderboards, or goal-setting elements could make the learning process more motivating especially for younger audiences while maintaining the integrity of educational outcomes.

References

- [1] S. Wang, F. Wang, Z. Zhu, J. Wang, T. Tran, and Z. Du, "Artificial intelligence in education: A systematic literature review," *Expert Systems with Applications*, vol. 252, p. 124167, 2024, doi: 10.1016/j.eswa.2024.124167.
- [2] Lin, C.-C., Huang, A. Y. Q., & Lu, O. H. T. (2023). Artificial intelligence in intelligent tutoring systems toward sustainable education: A systematic review. *Smart Learning Environments*, 10, Article 41. <https://doi.org/10.1186/s40561-023-00260-y>
- [3] Park, J., & Choo, S. (2024). Generative AI prompt engineering for educators: Practical strategies. *Journal of Special Education Technology*, 0(0), 1–7. <https://doi.org/10.1177/01626434241298954>
- [4] Su, J., & Yang, W. (2023). Unlocking the power of ChatGPT: A framework for applying generative AI in education. *ECNU Review of Education*, 6(3), 355–366. <https://doi.org/10.1177/20965311231168423>
- [5] Cain, W. (2024). Prompting change: Exploring prompt engineering in large language model AI and its potential to transform education. *TechTrends*, 68, 47–57. <https://doi.org/10.1007/s11528-023-00896-0>
- [6] Alasadi, E. A., & Baiz, C. R. (2023). Generative AI in education and research: Opportunities, concerns, and solutions. *Journal of Chemical Education*, 100(8), 2965–2971. <https://doi.org/10.1021/acs.jchemed.3c00323>
- [7] Baillifard, A., Gabella, M., Banta Lavenex, P., & Martarelli, C. S. (2025). Effective learning with a personal AI tutor: A case study. *Education and Information Technologies*, 30, 297–312. <https://doi.org/10.1007/s10639-024-12888-5>
- [8] R. Makharia et al., "AI Tutor Enhanced with Prompt Engineering and Deep Knowledge Tracing," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATM-SI)*, 2024, pp. 1–8, doi: 10.1109/IATM-SI60426.2024.10503187.
- [9] Thomas, D. R., Lin, J., Kakarla, S., Bhushan, S., Gatz, E., Gupta, S., Abboud, R., & Koedinger, K. R. (2024, September). Generative AI evaluation of human tutors: Demonstrations and considerations for prompt engineering [TechRxiv preprint]. arXiv.

- [10] Gizmo.ai/tutor, "Commercial platform," 2025. [Online]. Available: <https://gizmo.ai/tutor>
- [11] NotebookLM.Google, "Commercial platform," 2025. [Online]. Available: <https://notebooklm.google/>
- [12] TutorAI.me, "Commercial platform," 2025. [Online]. Available: <https://tutorai.me/>
- [13] AI-Tutor.ai, "Commercial platform," 2025. [Online]. Available: <https://ai-tutor.ai/>
- [14] Kim, W.-H., & Kim, J.-H. (2020). Individualized AI tutor based on developmental learning networks. *IEEE Access*, 8, 27927–27945. <https://doi.org/10.1109/ACCESS.2020.2972167>
- [15] M. F. S. Lazuardy and D. Anggraini, "Modern Front End Web Architectures with React.Js and Next.Js," *Int. Res. J. Adv. Eng. Sci.*, vol. 7, no. 1, pp. 132–141, 2022.
- [16] F. A. Prasetyo, "Badan Kepegawaian Negara (BKN)," *Tribunnewswiki*, Oct. 24, 2019. IRJAES-V7N1P162Y22
- [17] A. Bhalla, S. Garg, and P. Singh, "Present Day Web-Development Using ReactJS," *International Research Journal of Engineering and Technology*, vol. 7, no. 5, pp. 1154–1157, 2020. IRJAES-V7N1P162Y22IRJAES-V7N1P162Y22
- [18] OpenRouter.ai. [Online]. Available: <https://openrouter.ai/>
- [19] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774v6, Mar. 2024.
- [20] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models," *_arXiv_ preprint arXiv:2302.13971v1*, Feb. 2023.
- [21] A. Chowdhery *et al.*, "PaLM: Scaling Language Modeling with Pathways," *J. Mach. Learn. Res.*, vol. 24, pp. 1–113, Aug. 2023.
- [22] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proc. NeurIPS*, Vancouver, Canada, 2