Faculty of Informatics Engineering

Department of Software Engineering

AI Student Assistant

(Moein)

A junior project report - submitted to complete the requirements for obtaining a bachelor's degree in software engineering

Prepared by

MHD Zaher Krayem

MHD Yahia Abou Samra

MHD Awab Kheir

Othman Darkal

Supervised by

Dr. Riad Sonbol

Eng. Raghad Al Hossny

2025-202

SUPERVISION CERTIFICATION

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and gratitude to Dr. Riad Sonbol for his valuable guidance and continuous support throughout my graduation project. His insights and valuable feedback were a great help in accomplishing this work.

Thank you for your time and great effort

From Awab

ABSTRACT

In modern higher education environments, students face increasing volumes of academic content, making efficient content organization and revision essential for effective learning outcomes. To address this challenge, educational software systems must not only manage learning materials but also enhance student understanding through intelligent processing and continuous improvement. Such systems play a critical role in reducing cognitive load and supporting students' evolving academic needs.

In response to this need, this report presents an intelligent Student Course Management and AI Summarization System designed to assist university students in organizing courses and comprehending lecture materials more efficiently. The system enables students to create and manage courses, upload lecture files in supported formats (PDF, DOCX, PPTX), and automatically generate structured text summaries using artificial intelligence techniques. By analyzing lecture content, the system produces concise and organized summaries that can be stored, reviewed, and managed by students for future use.

Through the integration of modern web technologies, secure authentication mechanisms, and AI-based text processing, the proposed system enhances the learning experience by minimizing revision time, improving content accessibility, and providing a scalable foundation for future educational features.

ملخص

في بيئات التعليم العالي الحديثة، يواجه الطلاب كميات متزايدة من المحتوى الأكاديمي، مما يجعل تنظيم المحتوى ومراجعته بكفاءة أمرًا ضروريًا لتحقيق نتائج تعليمية فعّالة. ولمواجهة هذا التحدي، يجب ألا تقتصر أنظمة البرمجيات التعليمية على إدارة المواد التعليمية فحسب، بل يجب أن تُعزز أيضًا فهم الطلاب من خلال المعالجة الذكية والتحسين المستمر. وتلعب هذه الأنظمة دورًا حاسمًا في تخفيف العبء المعرفي ودعم الاحتياجات الأكاديمية المتطورة للطلاب.

استجابةً لهذه الحاجة، يقدم هذا التقرير نظامًا ذكيًا لإدارة المقررات الدراسية وتلخيصها باستخدام الذكاء الاصطناعي، مصممًا لمساعدة طلاب الجامعات على تنظيم مقرراتهم الدراسية وفهم مواد المحاضرات بكفاءة أكبر. يُمكّن النظام الطلاب من إنشاء المقررات الدراسية وإدارتها، وتحميل ملفات المحاضرات بالصيغ المدعومة (PDF، DOCX، PPTX)، وإنشاء ملخصات نصية منظمة تلقائيًا باستخدام تقنيات الذكاء الاصطناعي. من خلال تحليل محتوى المحاضرات ، يُنتج النظام ملخصات موجزة ومنظمة يمكن للطلاب حفظها ومراجعتها وإدارتها لاستخدامها لاحقًا.

من خلال دمج تقنيات الويب الحديثة وآليات المصادقة الآمنة ومعالجة النصوص القائمة على الذكاء الاصطناعي، يعزز النظام المقترح تجربة التعلم من خلال تقليل وقت المراجعة وتحسين إمكانية الوصول إلى المحتوى وتوفير أساس قابل للتطوير للميزات التعليمية المستقبلية.

TABLE OF CONTENT

List of abbreviations

Table 1 List of abbreviations

Abbreviation	Definition
API	Application Program Interface
AI	Artificial Intelligence
UML	Unified Modeling Language
NoSQL	Not only Structured Query Language.
SQL	Structured Query Language.
SOW	Statement of Work

Chapter1 Introduction

1. Introduction:

In this chapter we introduce the Student Course Management & AI Summarization System, describe the motivating problems that led to its development, and state the main objectives we intend to achieve. The system addresses the increasing volume of course materials and the time students spend reading and reviewing lectures. By combining course and lecture management with automated AI-based text summarization and assessment, the proposed system reduces revision time and improves learning effectiveness.

We will also give an overview of the report structure so the reader can easily follow the subsequent analysis, design, and implementation chapters.

2. Problem Definition:

University students often face large quantities of lecture materials (slides, handouts, notes) across many courses. Manually organizing, reading, and extracting key ideas from these documents is time-consuming and inefficient. Students commonly struggle to:

- Quickly grasp the core concepts of long lecture documents.
- Find and revisit essential points across multiple courses and lecture files.
- Self-assess understanding with rapid, relevant practice questions derived from course content.

From a system-administration perspective, institutions and course owners also need a reliable way to host and manage course content, control access, and support future enhancements (scalability, security, maintainability).

Existing learning management workflows may lack integrated, automated support for summarization and formative assessment. Without automated summarization and question generation, students spend excessive time on review; instructors and admins lack tools for scalable content processing. Furthermore, manual processing is inconsistent and does not scale when course material volume grows.

Therefore, there is a clear need for a robust web-based system that simultaneously:

- Provides course and lecture management (create/delete courses, upload lecture files).
- Automatically analyzes uploaded lecture files (PDF, DOCX, PPTX) and produces concise, structured text summaries.
- Generates multiple-choice questions from lecture text and automatically grades student attempts.
- Secures user accounts and course data, and supports a maintainable, scalable architecture for future extensions.

Such a system will enhance student learning efficiency, improve content accessibility, and offer a practical platform demonstrating modern web and AI techniques.

3. Project objectives:

The principal objective of this project is to design and implement a secure, scalable, and user-friendly web application that helps students organize course materials and study more effectively through AI-generated text summaries and automated multiple-choice assessment.

To fulfil this objective, the system will:

- Allow users (Students and Admin) to register, authenticate, and manage accounts using JWT-based security.

- Enable Students to create courses and upload lecture files in supported formats (PDF, DOCX, PPTX).

- Process uploaded lecture files asynchronously using NLP models (via Hugging Face) to generate concise, well-structured text summaries (headings, bullet points, key concepts).

- Persist original files and generated summaries for fast retrieval and searching.

- Automatically generate multiple-choice questions from lecture content (MCQs only) and perform automatic grading after student submission, providing immediate feedback.

- Use a microservices architecture with an API Gateway to centralize authentication and route requests, improving modularity and scalability.

- Employ the chosen technology stack (React + Vite frontend, Python/Django backend services, MongoDB for document/AI artifacts, MySQL for relational data, Docker for containerized deployment, and GitHub for source/version control).

- Provide a foundation for future features: full-text search, PDF export, sharing summaries, and support for additional media types (audio/video) via speech-to-text.

These objectives aim to reduce students' revision time, increase comprehension and retention, and demonstrate a production-oriented integration of web engineering practices with AI-driven text processing.

4. Report organization:

The report will be organized in the following chapters:

- Chapter 1: introduction.
- Chapter 2: Basic concepts and literature review.
- Chapter 3: project management.
- Chapter 4: high level System analysis and Design
- Chapter 5: Account Service
- Chapter 6: Course and Lectute Service
- Chapter 7: AI Service
- Chapter 8: Conclusion.

Chapter2 Basic Concepts and Literature Review

1. Introduction:

In this chapter we present the technical and theoretical building blocks used in the Student Course Management & AI Summarization System, and survey related systems and research that informed our design choices. The goal is to give the reader the background necessary to understand why particular algorithms, architectures, and technologies were chosen, and how past work in learning platforms, text summarization, and question generation relates to this project.

2. Basic concepts:

Maintenance in Software Engineering:
Maintenance in software engineering is essential to keep the system functional, secure, and efficient over time. For the Student Course Management & AI Summarization System, maintenance includes updating AI models, fixing bugs in upload/extraction pipelines, adapting to new file-format edge cases, and improving service interfaces. Regular maintenance prolongs system life, ensures reliable summaries and assessments, and preserves user trust.

▪ Requirements engineering:
Requirements engineering is the disciplined process of eliciting, documenting, and maintaining system requirements. For this project, it involves gathering needs from students and admins (features like course creation, file upload, summary generation, and automatic MCQ grading) through interviews, surveys, and prototyping. Clear elicitation ensures the delivered application matches user expectations and supports future extension (search, export, multimedia).

▪ Student feedback and usage telemetry:
Student feedback and usage telemetry provide direct insight into how the system is used and where it fails to meet learning needs. Collecting feedback (ratings on summaries, report of poor-quality MCQs) and telemetry (upload patterns, processing times) helps prioritize improvements, tune summarization models, and enhance UX for faster revision and higher satisfaction.

▪ Text summarization (extractive vs. abstractive):
Text summarization creates a concise representation of source content. Extractive summarization selects and concatenates key sentences from the lecture, preserving factual phrasing; abstractive summarization generates new, concise text that may

rephrase concepts for clarity. The system targets high-quality, text-only summaries that balance faithfulness and readability for student review.

▪ Question generation and automatic grading (MCQs):
Question generation produces multiple-choice questions from lecture text, including the correct answer and plausible distractors. Limiting output to MCQs simplifies automatic grading: once the student submits answers, the system compares them to stored keys and returns an immediate score and feedback, supporting rapid formative assessment.

▪ Semantic similarity and embeddings:
Semantic similarity measures how close two pieces of text are in meaning. Embedding models convert sentences or paragraphs into vectors so the system can detect duplicate points, cluster related concepts, and select distractors for MCQs. Semantic techniques improve summary coherence and reduce redundancy across generated outputs.

▪ File formats and text extraction:
Lecture materials are accepted in PDF, DOCX, and PPTX formats. Reliable text extraction is a prerequisite: parsing slide text, body paragraphs, and handling layout artifacts ensures the NLP pipeline receives clean input. Robust extraction also prepares the system for future OCR or multimedia processing.

▪ Asynchronous processing and job queues:
Summarization and question generation can be time-consuming; therefore, processing runs asynchronously. The system enqueues jobs when a student uploads a lecture, processes them in background workers, and stores results for later retrieval. This pattern preserves frontend responsiveness and enables scalable AI workloads.

▪ Storage choices: MongoDB vs. MySQL:
A hybrid storage strategy fits the project needs: MySQL stores relational data (users, courses, enrollments), while MongoDB holds variable AI artifacts (extracted text, summaries, generated MCQs, embeddings). This separation optimizes query performance for structured operations and flexible retrieval of document-like AI outputs.

3. Literature review:

StudyFetch

StudyFetch is one of the prominent educational platforms that utilizes artificial intelligence to enhance the learning experience for university students. The platform aims to convert study materials into interactive tools such as flashcards, quizzes, and summaries, helping improve comprehension and reduce study time.

Core Features:

- Converts study files (PDFs, presentations, notes) into automatic study tools.
- Summarizes lectures using AI techniques.
- Includes a smart assistant named **Spark.E** that provides instant answers, essay reviews, and progress tracking.
- Supports audio and video content by allowing lecture recordings to be transformed into study-ready summaries.

Relevance to the Project:

StudyFetch demonstrates how AI technologies can be used to improve learning, especially through automatic summarization and personalized study tools. However,

Chapter 2 – Basic concepts and literature review

this project aims to offer a more tailored experience for Arabic-speaking students by supporting Arabic audio content and providing a simplified interface suited to local educational environments.

Research Gap:

Although StudyFetch supports Arabic in its interface and some content, its core AI features—such as summarization and the Spark.E assistant—are still primarily designed for English-language input, which limits their effectiveness in Arabic educational contexts. This project seeks to bridge that gap by developing a smart system specifically for Arabic-speaking students, with expanded support for Arabic audio and text content, and a fully integrated, user-friendly learning experience.

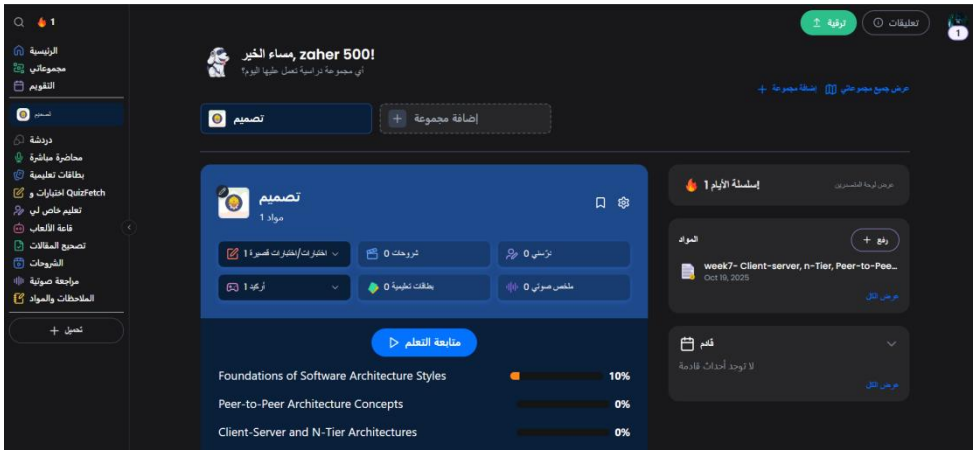


Figure 1 StudyFetch interface showcasing AI-powered study tools.

Humata AI

Humata AI is an advanced document assistant powered by artificial intelligence, designed to help users interact with and extract insights from complex documents—especially PDFs. It enables real-time question answering, summarization, and

Chapter 2 – Basic concepts and literature review

comparison across multiple files, making it a valuable tool for researchers, students, and professionals dealing with technical or lengthy content.

Core Features:

- **AI-powered PDF interaction**: Users can upload unlimited documents and ask questions directly to the system.
- **Summarization and comparison**: The platform can summarize findings, compare documents, and extract key insights.
- **Citation-based answers**: Responses include citations linked to the original source files for transparency.
- **Security and scalability**: Offers encrypted storage, role-based access, and single sign-on for enterprise use.

Relevance to the Project:

Humata AI showcases how artificial intelligence can streamline document analysis and enhance productivity. Its ability to summarize and answer questions from uploaded files aligns with the goals of this project, especially in automating content extraction. However, Humata primarily focuses on written documents, while this project emphasizes **audio-based summarization** and **Arabic language support**, which are not core strengths of Humata.

Research Gap:

Although Humata AI excels in processing technical documents and offers multilingual interface support, its core AI functionalities are optimized for English-language text and PDF formats. It does not natively support Arabic audio input or educational workflows tailored to Arabic-speaking students. This project addresses that gap by developing a system that supports **Arabic audio summarization**, **student-focused interaction**, and**localized educational content**, offering a more inclusive and accessible solution.

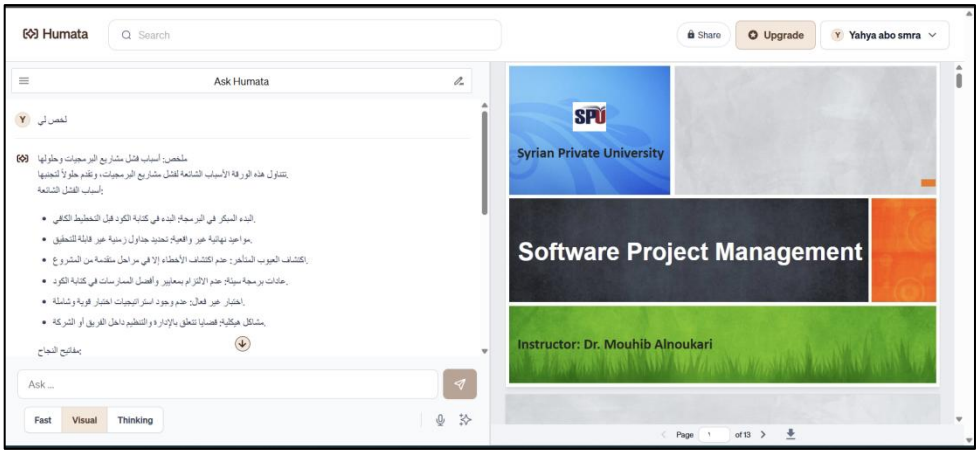


Figure2 Humata interface showcasing AI-powered study tools.

Genei

Genei is an AI research tool that helps students and researchers summarize articles, PDFs, and web pages. It generates concise summaries, extracts keywords, and organizes notes to speed up academic work.

Core Features:

- Summarizes long documents automatically
- Highlights key concepts
- Central workspace for notes and files
- Improves reading efficiency

Relevance to Moein:

Genei streamlines academic reading, but it focuses on English text and static documents. Moein, by contrast, supports Arabic lecture summarization and offers tools tailored to Arabic-speaking students.

Research Gap:

Genei lacks Arabic language support and audio interaction. Moein fills this gap with Arabic audio summarization, academic structuring, and a localized student assistant.

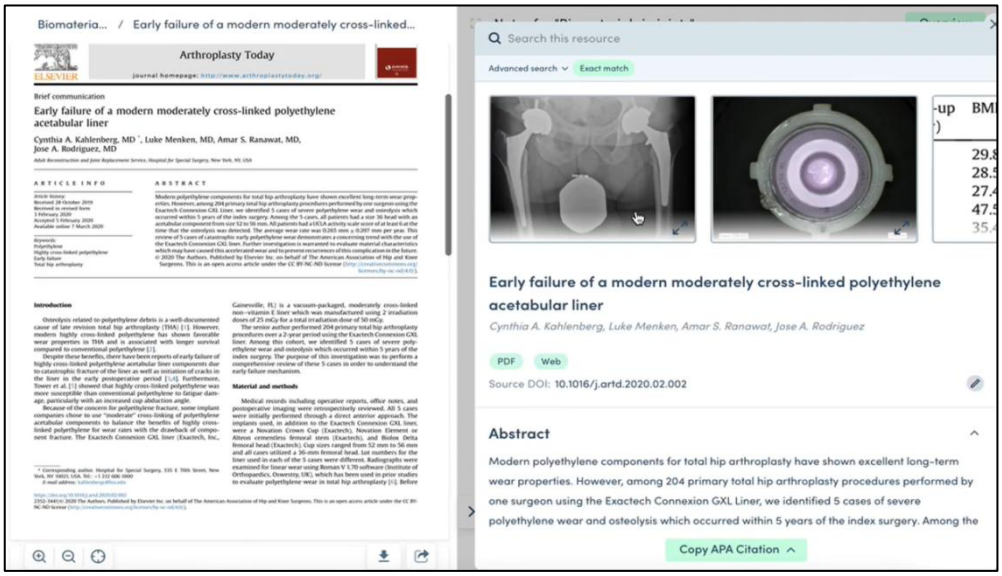


Figure3 Genei interface showcasing AI-powered study tools.

Comparative Analysis of AI-Based Educational Platforms

To highlight the unique value of **Moein**, the proposed system, the following table presents a comparative analysis between three existing AI-powered platforms — **StudyFetch**, **Humata AI**, and **AskYourPDF**— and **Moein**. The comparison focuses on key features relevant to Arabic-speaking students and educational workflows.

Table 2 Comparative analysis of AI-powered educational platforms

Feature	StudyFetch	Humata AI	Genei	Moein (Proposed System)
Text summarization	✔	✔	✔	✔
speech to text	✔	✘	✘	✔
Arabic interface support	✔	✘	✘	✔
AI support for Arabic content	✔	✔	✘	✔
Auto-generated quizzes	✔	✘	✘	✔
Simplified UI for local educational context	✘	✔	✔	✔
Responses include citations linked to the original	✘	✔	✔	✔
Student-focused smart assistant (Chat Bot)	✔	✔	✘	✔
Document and content library management	✔	✔	✔	✔

Chapter3 Project Management

1. Introduction:

In this chapter, we will dive into the management phase of the project, which is a critical aspect of ensuring the project's success. We will examine the project charter, project plan, Statement of Work (SOW) document, stakeholder analysis, and risk management strategies to effectively guide and control the project from initiation to completion.

2. Project charter:

A project charter is a formal document that serves as an official authorization for the start of a project. It acts as a reference point throughout the project, providing a clear understanding of the project's purpose and establishing a foundation for decision-making and project governance.

Project title	*AI Student Assistant*
Project start date	*30 – 10 – 2025*
Project finish date	*13 – 1 - 2026*
Project manager	*Dr. Riad Sonbol & Eng. Raghad Al Hossny*
Project objectives	*Develop a web-based educational system that assists students in managing their university courses and lecture materials in an efficient and organized manner that supports effective learning. By using artificial intelligence algorithms and natural language processing techniques to analyze uploaded lecture files (PDF, DOCX, PPTX), the system automatically generates concise text summaries and multiple-choice questions to help students understand and review course content.* *The system aims to reduce the time and effort required for manual revision by providing structured summaries and instant assessment through automatic question generation and grading. Through secure user authentication, course management features, and intelligent text processing, the system enhances the overall learning experience and provides a scalable foundation for future improvements and extensions.*
Approach	*1. Determining the scope of the project.* *2. Perform literature review for similar tools or software.*

3. perform analysis for the requirements of the project.

4. perform system decomposition and design.

5. Start the first increment and develop a subsystem that is responsible for account and project management functionalities. 6. Testing the functionalities of the first subsystem

7. Start the second increment and develop the AI components that are responsible for the reviews analysis process.

8. Testing the functionalities of the other subsystem.

9. Document every step of the work.

Roles and responsibilities:

Name	*Role*	*Responsibility*
Dr.Riad Sonbol	*Supervisor*	*Highly Project management and work monitoring*
Eng.Raghad Al Hossny	*Supervisor*	*Work monitoring*
Zaher Kraym	*Engineer*	*Backend development*
yahia Abou Samra	*Engineer*	*Backend development*
Awab Kheir	*Engineer*	*Frontent development*
Othman Darkal	*Engineer*	*Frontent development*

3. The SOW document

A Statement of Work (SOW) is a comprehensive document that defines the scope of work for a project. It outlines the specific tasks, deliverables, timeline, and responsibilities. The SOW document provides a clear understanding of what needs to be accomplished, the project's objectives, and the criteria for success.

Project Title:	*Moein – AI-Powered Student*
Project Description and Objectives	*The project aims to develop a web-based system that helps students manage courses and lecture materials while using AI to generate summaries and quizzes. The objective is to deliver a secure and scalable solution using a microservices architecture.*
Project Scope	*This project involves developing a full-stack software system that integrates backend microservices, a frontend web application, and AI-based services. The system enables students to create accounts, manage personal information, create and manage courses, upload lecture files, and automatically receive AI-generated summaries and quizzes. The architecture follows a microservices approach with an API Gateway to ensure scalability, maintainability, and secure communication between services.*
Project Goals	○ Develop a scalable backend system using Django Rest Framework and microservices architecture. ○ Implement secure user authentication and account management features. ○ Enable efficient management of courses and lecture materials. ○ Integrate AI-based services for automatic lecture summarization and quiz generation. ○ Apply software engineering best practices such as version control, configuration management, and documentation.

Project Deliverables	• Project plan. • Working software system – web application. • Final project report.
Technology and Tools	○ Programming Languages: Python, JavaScript, HTML, CSS. ○ Frameworks: Django Rest Framework for the backend, React for the frontend. ○ Database: MySQL, MongoDB. ○ AI Models / Services: Hugging Face models for text summarization and quiz generation. ○ Tools: Git, GitHub, Docker, RabbitMQ, Ngrok.
Assumptions	• Continuous availability of project team members throughout the development lifecycle. • Regular supervision and feedback from the project supervisor. • Stable development environment and internet connectivity. • Availability of third-party AI services during development and testing.
Project Resources	*Human Resources:* *Dr. Raid Sonbol - Project Manager* *Eng. Raghad al Hossny - supervisor* *Zaher Krayem - Back-end developer* *Yahia Abo Samra - AI contributor* *Othman Darkal - Front-end developer* *Awab Kheir - Front-end developer*
Schedules	○ *Project Start Date: October 18, 2025* ○ *First Project Seminar: November 15, 2025* ○ *Second Project Seminar: December 20, 2025* ○ *Technical Committee Review: January 10, 2026* ○ *Final Project Seminar: January 31, 2026* ○ *Project Finish Date: January 31, 2026*

4. Project plan – Gantt chart:

A document outlining tasks, deadlines, and resources needed to achieve project objectives, serving as a roadmap for successful project execution.

Task Name	Duration	Start	Finish
Sprint 0 – Scope Definition & Analysis	14 days	Sat 25-10-25	Thu 25-10-30
Sprint 0 – System Design	2 days	Thu 25-11-06	Fri 25-11-07
Sprint 1 - Frontend – Management services	11 days	Sat 25-11-08	Sat 25-11-15
Sprint 1 - Backend – Management services	11 days	Sat 25-11-08	Sat 25-11-15
Prisentation 01	1 day	Sat 25-11-15	Sat 25-11-15
Sprint 2 - Frontend – Course & Lecture service	14 days	Sat 25-11-15	Mon 25-11-24
Sprint 2 - Backend – Course & Lecture service	14 days	Sat 25-11-15	Mon 25-11-24
Sprint 3 - Frontend – Summarization service	24 days	Wed 25-11-26	Thu 25-12-11
Sprint 3 - Backend – Summarization service	24 days	Wed 25-11-26	Thu 25-12-11
Presentation 02	1 day	Sat 25-12-20	Sat 25-12-20
Sprint 4 - Frontend – Chatbot service	14 days	Sat 25-12-20	Mon 25-12-29
Sprint 4 - Backend – Chatbot service	14 days	Sat 25-12-20	Mon 25-12-29
Sprint 5 - Frontend – Quiz service	14 days	Sat 26-01-03	Mon 26-01-12
Sprint 5 - Backend – Quiz service	14 days	Sat 26-01-03	Mon 26-01-12
Documentation	121 days	Sat 25-10-25	Tue 26-01-13

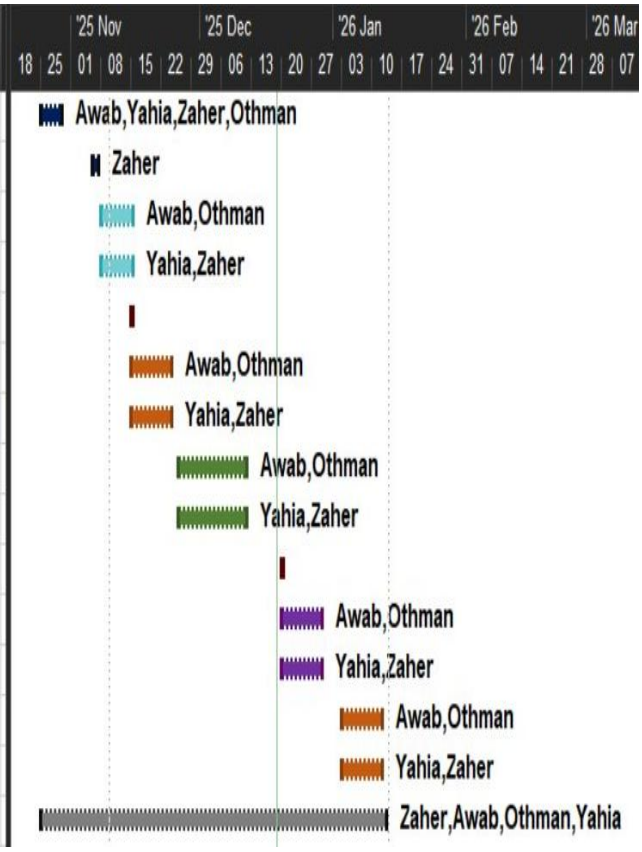


Figure4 Gantt chart

5. Risk Management:

the process of identifying, evaluating, and mitigating potential risks that could impact the success of a project or a team.

Table 3 Risk Management

Risk_ID	RK-01	RK-02	RK-03	RK-04
Risk title	team of only two students	fuzzy project scope	must learn and use new technologies	big Separation of work in the implementation stage
Risk Description	if one of the students stops for some reason, the whole project progress will be impacted	at the beginning of the work, a misunderstanding one member about the project scope will lead to a big mistake in his work	in our project, we need to use some AI components and our knowledge of developing it where low	one of us, where responsible for the backend development, and the other where responsible for the front-end development
Raised Date	22/10/2025	22/10/2025	17/12/2025	2/12/2025
Tracking Frequency	weekly	daily	weekly	daily
State	active	closed	Under Mitigation	active
Impact	high	medium	high	medium
Mitigation Plan	the one who end first will help the other one, in his work	meeting in the closing of each step		make one of us as the implementation leader

6. Version Control and Configuration Management

6.1 Project Description

The Moein project is a backend system developed using Django REST Framework and a microservices architecture. The main objective of the project is to provide a scalable and modular backend for an AI-based student assistant, supporting user management, course and lecture management, and request routing through an API Gateway. The system is designed to be extensible, allowing future integration of AI services and asynchronous communication.

6.2 Git Repository Structure

Version control for the project is managed using Git and hosted on GitHub. The backend source code is maintained in a dedicated repository that contains all backend microservices.

Backend Repository:

- **Repository Name:** Junior-project-moein-backend
- **Purpose:** Contains all backend microservices, including the Account User Service, Course & Lecture Service, and API Gateway.
- The repository is structured to keep each microservice isolated, enabling independent development and deployment.

This separation improves maintainability and supports incremental system growth.

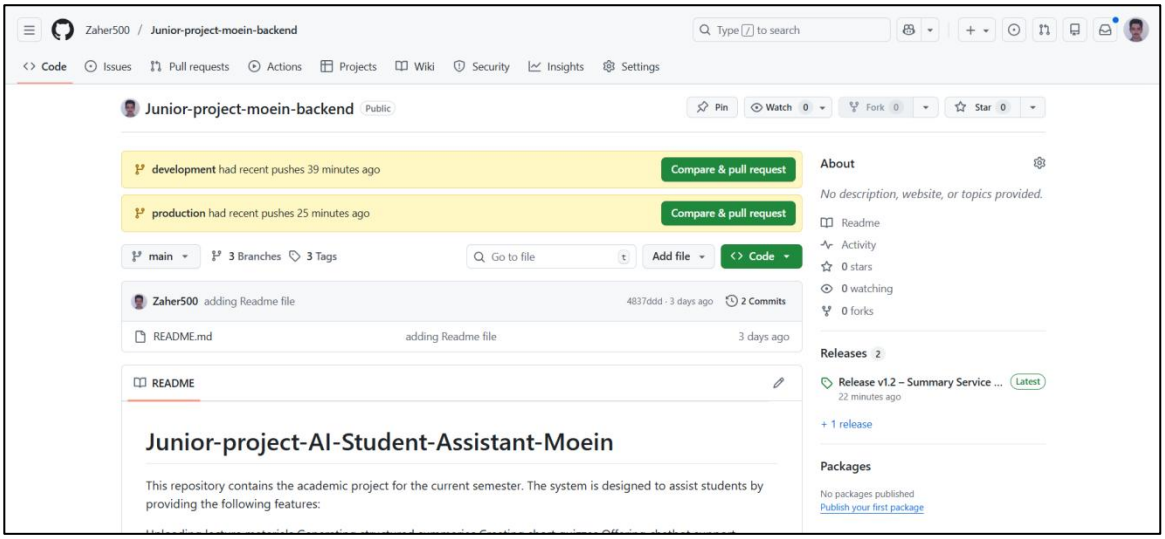


Figure 5 Backend GitHub repository for the Moein project.

Frontend Repository:

- **Repository Name:** Junior-project-moein-frontend
- **Purpose:** Contains the complete frontend source code of the Moein platform, including UI components, pages, services, styles, and configuration files.
- The repository is structured to separate components, pages, services, and assets, improving code organization and maintainability.
- This separation from the backend enhances development flexibility and supports incremental system growth.

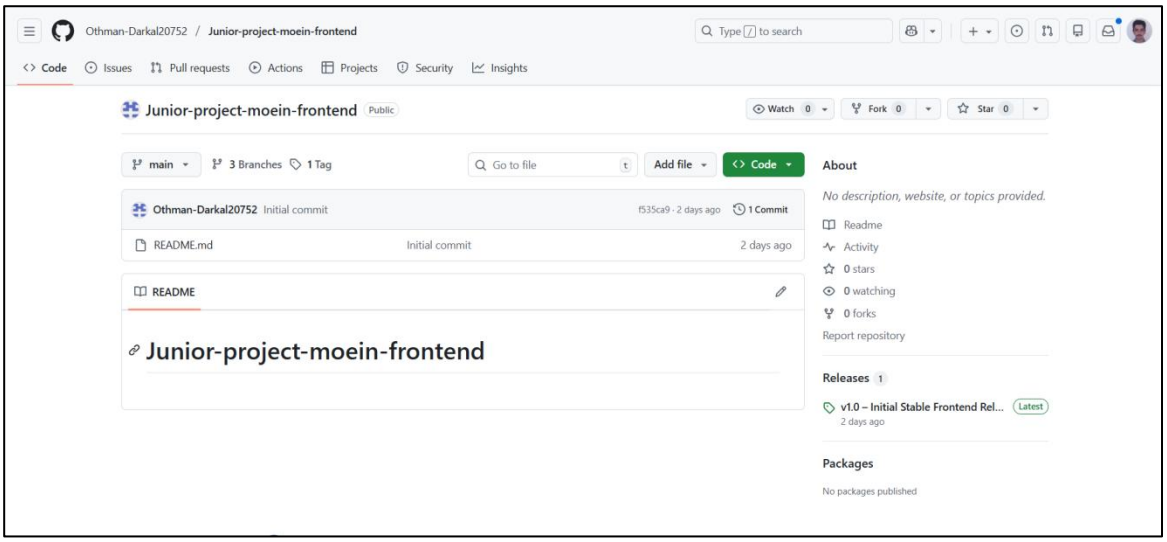


Figure 6 frontend GitHub repository for the Moein project.

Frontend–Backend Integration

Although the frontend and backend are maintained in separate repositories, both follow a consistent version control strategy, including branch management, pull requests, and release tagging. This approach enables coordinated integration while preserving the ability to develop and release each part of the system independently.

6.3 Branching and Merging Strategy

A simplified branching strategy was adopted across both the backend and frontend to support agile and incremental development while maintaining consistency throughout the project.

The following branches are used in **both repositories**:

- **production:** Contains the stable and production-ready versions of the system.
- **development:** Used for ongoing development and integration of new features.
- **feature/***: Temporary branches created for implementing specific features or services.

Branch Flow

- Feature branches are created from the `development` branch
- Completed features are merged back into `development` after testing
- Stable milestones are merged from `development` into `production`

For the **backend**, this strategy ensures that new microservices or backend enhancements do not affect the production-ready system until they are fully integrated and tested.

For the **frontend**, the same strategy ensures that user interface changes and new features (such as authentication interfaces or lecture workflows) are safely developed and validated before being released to production.

This unified branching strategy promotes consistency, controlled integration, and system stability across the entire project.

6.4 Team Members and Responsibilities

The project was developed by a team of four members. All team members collaboratively participated in the system analysis and architectural design phases, including requirements analysis, system modeling, and defining the overall system architecture.

The implementation responsibilities were divided as follows:

- **Backend Development:**
  Responsible for implementing the backend microservices using Django REST Framework, including user management, course and lecture services, API Gateway integration, database design, and backend version control management.
- **Frontend Development:**
  Responsible for implementing the frontend application, including user interface components, page layouts, client-side services, and integration with backend APIs.
- **AI Service Development:**
  Responsible for implementing the AI service, including model integration, text summarization logic, and communication with backend services.

All team members contributed to version control activities. Backend- and frontend-related branches, merge operations, releases, and repository management were handled by the respective developers. Pull requests were used to merge feature branches into the development branch, ensuring controlled integration and traceability of changes.

6.5 Development Workflow

The development workflow for both the backend and frontend follows an incremental and task-based approach to ensure consistency, stability, and controlled integration across the entire system.

The workflow is summarized as follows:

1. A development task or feature is identified
2. A feature branch is created from the `development` branch
3. The feature is implemented and tested locally
4. A pull request is created to merge the feature branch into `development`
5. After integration and validation, the `development` branch is merged into `production` for release

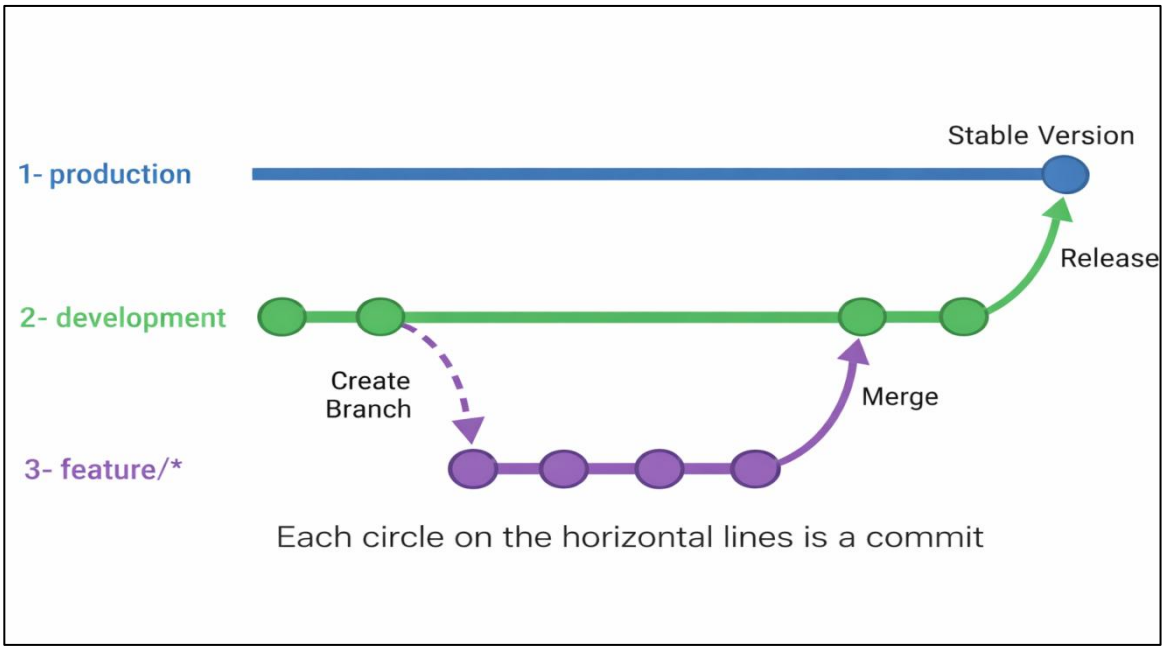


Figure 7 Branching and merging workflow from feature development to production release.

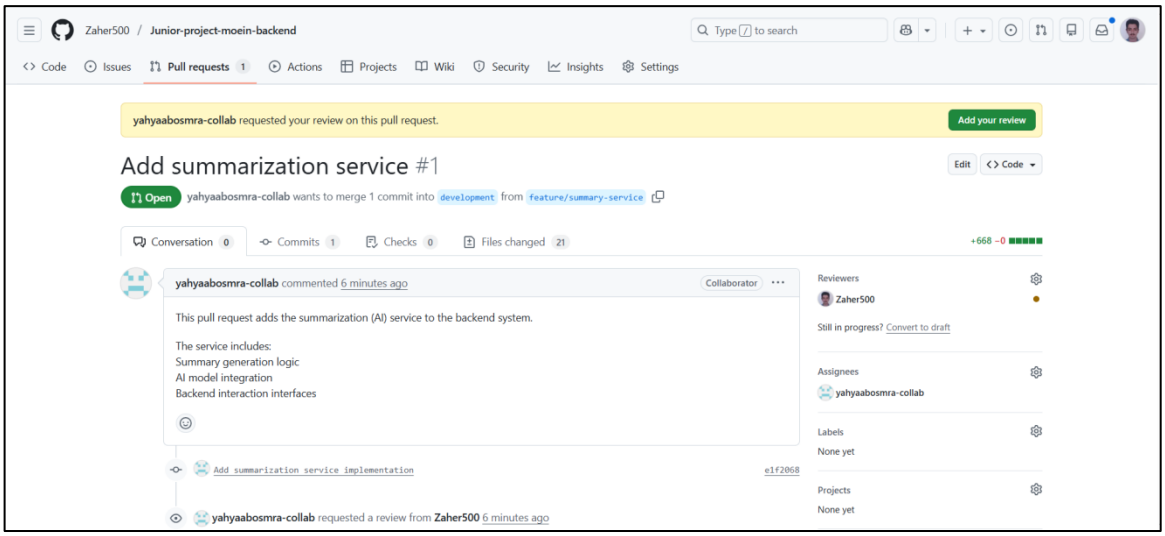


Figure 8 Pull request used to merge the summary service feature branch into the development branch.

This workflow is applied consistently to both backend and frontend development. For the backend, it ensures that new microservices or system enhancements are safely integrated before release. For the frontend, it guarantees that user interface changes and new client-side features are validated before being deployed.

Overall, this approach ensures code stability, clear change tracking, controlled integration, and organized progress throughout the project lifecycle.

6.6 Tags and Versioning

Git tags are used to mark important project milestones and stable releases across both the backend and frontend repositories. Tagging enables clear identification of release points, improves traceability, and supports controlled deployment.

v1.0 – Internal Development Version

Version **v1.0** was used as an internal development milestone during early stages of the project. This version was not considered stable due to missing core infrastructure components and incomplete system integration, and therefore was not treated as an official release.

v1.1 – Initial Stable Release

Version v1.1 represents the first fully functional and stable release of the system.
For the backend, this release includes the Account User Service, Course and Lecture Service, and the API Gateway.
For the frontend, it includes the stable user interface and full integration with backend APIs.

v1.2 – Summary Service Integration

Version v1.2 extends the system by integrating the summarization (AI) service.
This release includes the AI-based summary generation service developed by a team member and its integration with the existing backend microservices.
The release demonstrates collaborative development through feature branching, pull requests, and controlled merging into the production branch.

Only validated and production-ready versions were tagged as official releases.

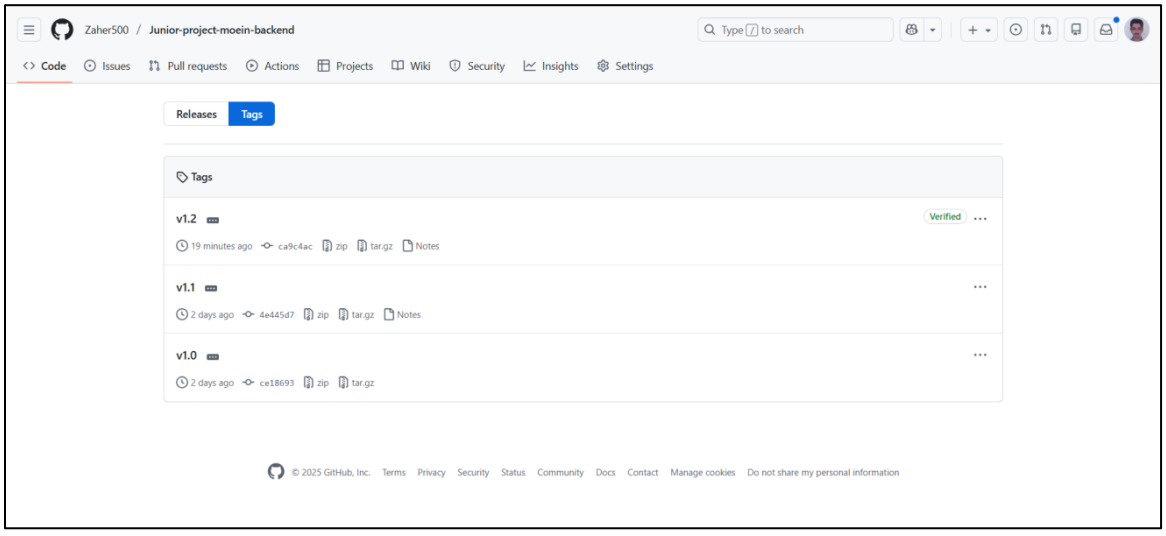


Figure 9 Git tags used to mark stable backend releases, highlighting versions v1.1 and v1.2.

6.7 Releases Management

The project follows a release-based development approach to deliver stable and incremental system functionality. Each release represents a validated milestone and is created only after successful integration, testing, and merging into the production branch.

GitHub Releases are used in combination with Git tags to document stable versions of the system, describe included features, and provide traceability throughout the project lifecycle.

Release v1.1 – Initial Stable Release

Release **v1.1** represents the first stable and fully functional version of the system. This release established the core backend and frontend infrastructure required for system operation.

Included components:

- Account User Service
- Course and Lecture Service
- API Gateway for request routing
- Stable frontend user interface with backend integration

This release served as the foundation for subsequent system extensions and enhancements.

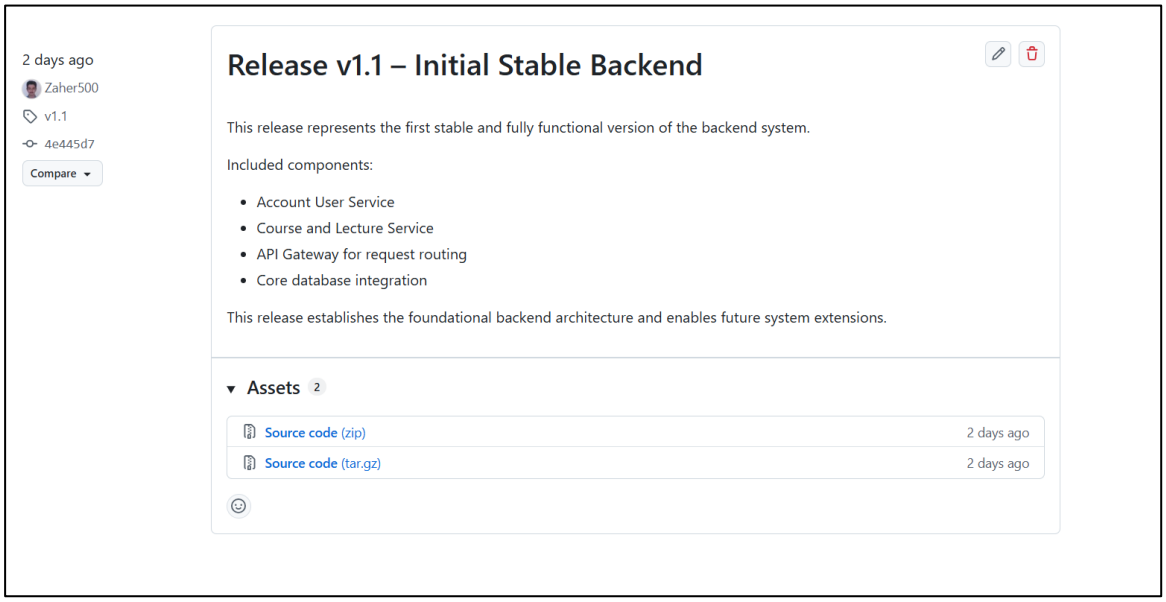


Figure 10 GitHub release illustrating stable system version v1.1

Release v1.2 – Summary Service Integration

Release **v1.2** extends the system by integrating the summarization (AI) service. This release demonstrates collaborative development and incremental feature delivery.

Key additions:

- AI-based summarization service implementation
- Integration of the summary service with existing backend microservices
- Controlled feature integration using pull requests and branch merging

This release reflects the evolution of the system beyond core functionality and highlights the use of version control practices to manage collaboration and feature growth.

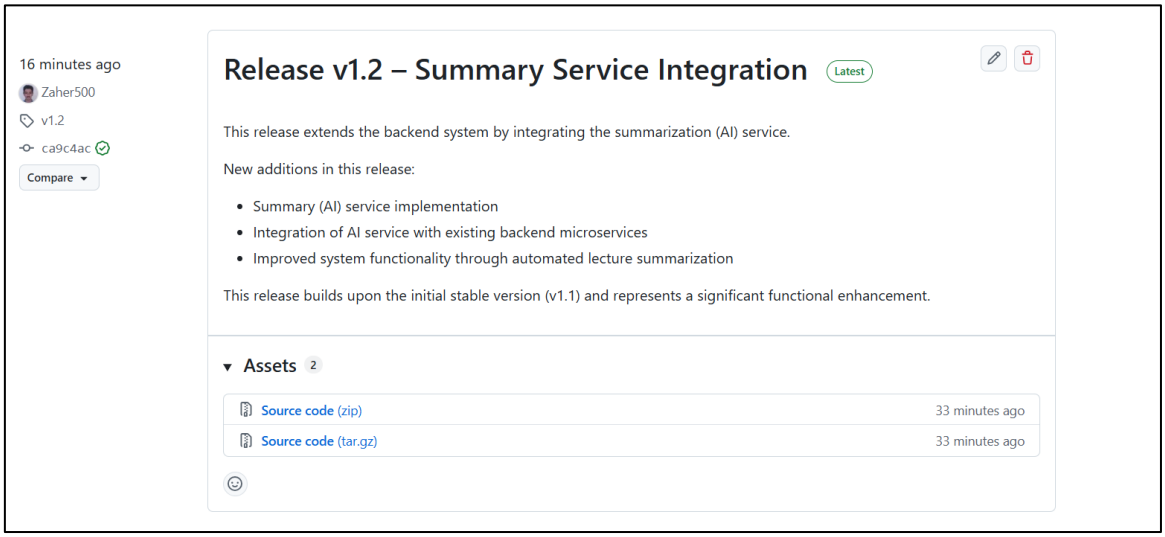


Figure 11 GitHub release illustrating stable system version v1.2

Release Strategy Summary

Only production-ready versions of the system are published as official releases. Earlier internal development versions were not released due to incomplete system components. This strategy ensures that each release represents a stable and deployable system state.

6. Summary:

In conclusion, good project management is vital for successful software system development. It provides structure, ensuring projects are completed within scope and schedule. Overall, strong project management enhances the delivery of high quality software systems that meet the goals of the project.

Chapter 4 System Analysis

1. Introduction

In this chapter we will introduce the analytical study of the system using the needed UML diagrams for system requirements modeling.

2. Requirements Elecitiation:

During the requirements elicitation process, we have taken several steps to ensure a comprehensive understanding of the project. We conducted a study and analysis of similar systems, also analyzed, and anticipated the needs of the application's users.

The resulted Requirement database:

Table 4 Requirement database

req-id	Requirement title	Actor	priority
RE-FR-AM-01	The system shall allow users to create new accounts.	User	high
RE-FR-AM-02	The system shall allow users to log in.	User	high
RE-FR-AM-03	The system shall allow users to manage their accounts	User	medium
RE-FR-AM-04	The system shall allow users to create a personal storage space for their uploaded files.	User	high
RE-FR-AM-05	The system shall allow users to upload files in PDF, DOCX, or PPTX formats.	User	High
RE-FR-AM-06	The system shall analyze the uploaded files using AI.	System	High

RE-FR-AM-07	The system shall generate summaries for the uploaded files using AI.	System	high
RE-FR-AM-08	The system shall allow users to export the generated summary as a PDF file.	System	medium
RE-FR-AM-09	The system shall store the generated summary along with the related PDF data.	System	medium
RE-FR-AM-10	The system shall generate exams based on the uploaded lecture files.	System	high
RE-FR-AM-11	The system shall generate various types of questions based on the uploaded files using AI.	System	high
RE-FR-AM-12	The system shall link each generated question to the file or lecture it was derived from.	System	medium
RE-FR-AM-13	The system shall automatically evaluate student answers based on the file content.	System	high
RE-FR-AM-14	The system shall provide explanations for answers based on the file content.	System	medium
RE-FR-AM-15	The system shall allow text-based search within all uploaded files.	User	medium
RE-FR-AM-16	The system shall provide an AI-powered chatbot that answers user questions based on uploaded files content.	User	high
RE-FR-AM-17	The system shall allow users to log out.	User	medium

3. SRS Document:

3.1 Purpose:

The purpose of this Software Requirements Specification (SRS) is to outline the functional and non-functional requirements for our system. This document will serve as a foundational reference to ensure alignment between stakeholders and the development team throughout the project lifecycle.

3.2 Project scope*:*

Helps students organize and manage their study process using artificial intelligence, Allows uploading of study materials like lectures or audio, Automatically summarizes the uploaded content for quick understanding.

Creates smart questions and short quizzes to support learning, Builds a personalized study plan that adapts to the student's progress.

Offers an AI tutor (chatbot) that answers questions and explains topics using the student's own materials

3.3 Document overview:

The software requirements specification SRS document will be structured as follows:

1. Introduction.

2. Overall description and product features.

3. Nonfunctional requirements.

4. Conclusion.

2. Overall description and product features:

1. Product features:

The product features derived from the systems use cases are:

- Sign Up (UC-01) — *Primary actor: Student.*

This feature allows a new student to create an account on the system by providing required details (email, password, name) and completing email verification.

- Log In (UC-02) — *Primary actor: Student.*

This feature enables registered users to authenticate (email/password) and receive a JWT for accessing protected APIs and services.

- Manage Account (UC-03) — *Primary actor: Student.*

This feature provides students the ability to view and update their profile information, change password, and manage personal settings.

- Upload Lecture (UC-04) — *Primary actor: Student.*

This feature allows a student to upload lecture files in supported formats (PDF, DOCX, PPTX) to a selected course.

- Parse Lecture File (UC-05) — *Primary actor: System (triggered by Upload).*

This feature extracts plain text and structure (titles, bullet points, slide text) from uploaded files to prepare content for AI processing.

- Analyze Uploaded File (UC-06) — *Primary actor: System*

This feature runs NLP preprocessing on the extracted text (cleaning, segmentation, embedding) as a preparation step for summarization and question generation.

- Generate Summary using AI (UC-07) — *Primary actor: System*

This feature automatically produces a concise, well-structured text summary of the lecture using Hugging Face models and stores the result for student access.

- Export Summary as PDF (UC-08) — *Primary actor: Student.*

This feature enables students to export any generated summary into a downloadable PDF document for offline review.

- Generate Quiz (UC-09) — *Primary actor: Student / System.*

This feature creates a quiz based on the lecture content; quizzes consist only of multiple-choice questions (MCQs) derived automatically from the lecture text.

Chapter 4 – System Analysis

- Generate MCQs (UC-10) — *Primary actor: System.*

This feature generates multiple-choice questions with one correct answer and plausible distractors from lecture text using transformer-based QG techniques.

- Auto-Evaluate Answers (UC-11) — *Primary actor: System.*

This feature automatically grades student quiz submissions by comparing answers to stored keys and records the score and result.

- Provide Answer Explanation (UC-12) — *Primary actor: System / Student.*

This feature supplies an explanation for each question's correct answer (generated or retrieved) after grading to support learning and feedback.

- AI Chatbot Q&A (UC-13) — *Primary actor: Student.*

This feature allows students to ask free-text questions about course content or summaries and receive AI-driven answers and clarifications in natural language.

- Lecture Management — View Lecture (UC-14) — *Primary actor: Student.*

This feature enables students to browse and open lecture files and their generated summaries for any course they own or are enrolled in.

- Lecture Management — Edit Lecture Info (UC-15) — *Primary actor: Student.*

This feature allows students to edit lecture metadata (title, description, tags, associated module/course) and trigger reprocessing if needed.

- Lecture Management — Search Lecture Content (UC-16) — *Primary actor: Student.*

This feature provides full-text search over lecture texts and summaries so students can quickly find concepts, keywords, or specific sections.

- Lecture Management — Delete Lecture (UC-17) — *Primary actor: Student.*

This feature allows students to remove a lecture file and its associated AI artifacts (extracted text, summary, generated MCQs) from a course.

- Course Management — Create Course (UC-18) — *Primary actor: Student.*

This feature enables creation of a new course container (title, code, description) to which lectures can be added.

- Course Management — Edit Course Info (UC-19) — *Primary actor: Student*

This feature permits updating course metadata, enrollment settings, and visibility options.

- Course Management — Delete Course (UC-20) — *Primary actor: Student*

This feature allows the removal of an entire course and optionally all contained lectures and artifacts.

Chapter 4 – System Analysis

- Manage Accounts (UC-21) — *Primary actor: Student* .

This feature enables the administrator to create, update, suspend, or delete user accounts and to manage role assignments.

2. User classes and characteristics:

Our system has two main types of users:

▪ **Admin (System Administrator):**
The Admin is a high-privilege user responsible foroverseeing the overall operation and maintenance of the application. Dutiesinclude managing user accounts (create, update, suspend, delete), creating ordeleting courses when needed, monitoring system usage and processingperformance, moderating content, and configuring system-wide settings such asupload limits or retention policies. Admins are expected to have above-averagetechnical proficiency (comfortable with dashboards, logs, and basictroubleshooting) and require reliable, secure access (strong authentication andaudit trails). They typically use the system less frequently than students butneed powerful tools for management, and emergency intervention.Security, data privacy, and the ability to view system analytics are keycharacteristics of this role.

▪ **Student (Primary user):**
The Student is the main consumer of the system's learning and content-management features. Students register and manage their accounts, create and organize courses, upload lecture files (PDF, DOCX, PPTX), request AI summaries and quizzes, view/export summaries, take auto-generated MCQ quizzes, and review their graded results and explanations. They are assumed to have basic computer literacy (familiarity with web apps, file uploads, and reading documents) and will interact with the system frequently—often on mobile devices or laptops—primarily for studying and course organization. Students expect quick, accurate summaries, straightforward UX for uploads and quizzes, and clear privacy controls over their uploaded materials and generated artifacts.

3. Assumptions:

Network Connection: It is assumed that users will have a stable and reliable network connection. The system's performance and accessibility depend on a consistent internet connection, as the application is web-based.

Browser Requirements: Users are expected to access the system through a modern web browser that supports HTML5, CSS3, and JavaScript. The system will be optimized for browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge.

Project in Production: It is assumed that the Product Owner has their project in the production stage. This is necessary for them to gather and analyze user reviews, which are critical for leveraging the system's insights and analytical capabilities.

3. Nonfunctional requirements:

▪ **RE-NF-01 (Usability – User-Friendly Interface):**
The system shall provide simple and intuitive user interfaces for both Students and Admins. A new user should be able to understand the main system functions (course creation, lecture upload, summary viewing, and quiz usage) within **15 minutes** of initial use without external assistance.

▪ **RE-NF-02 (Security – Authentication and Data Protection):**
The system shall ensure the security and privacy of all user data by implementing JWT-based authentication, role-based access control, and secure password policies (minimum 8 characters including at least one number). All data transmission must occur over secure HTTPS connections.

▪ **RE-NF-03 (Performance – Responsiveness and AI Processing):**
The system shall respond to standard user operations (navigation, search, viewing summaries) within **10 seconds**. AI-based operations such as lecture summarization and quiz generation shall be processed asynchronously and completed within an acceptable time frame, with progress status provided to the user.

▪ **RE-NF-04 (Scalability and Extensibility):**
The system shall be designed with a modular and scalable architecture that allows future expansion, including upgrading AI models, adding new features (search, export, multimedia support), and handling increased numbers of users and lecture files without major system redesign.

4. Conclusion:

This software Requirements Specification (SRS) document outlines the critical functionalities, user roles, and assumptions essential for developing a robust and

user-friendly system. By adhering to these requirements, the system will effectively support software product owners in enhancing their products to meet user needs and streamline project management processes.

4. Requirements modeling:

Use case diagram: use-case diagrams model the behavior of a system and help to capture the requirements of the system.

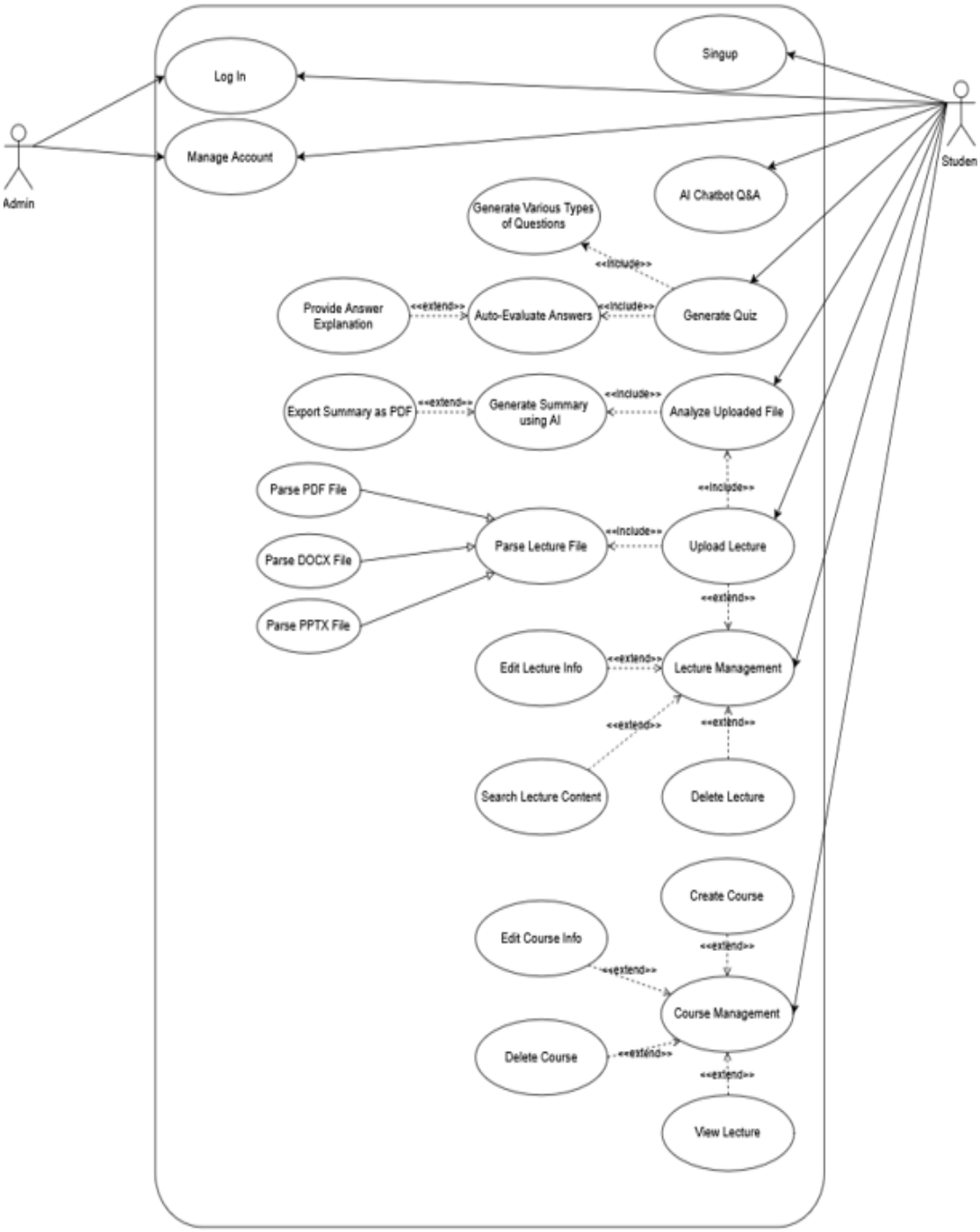


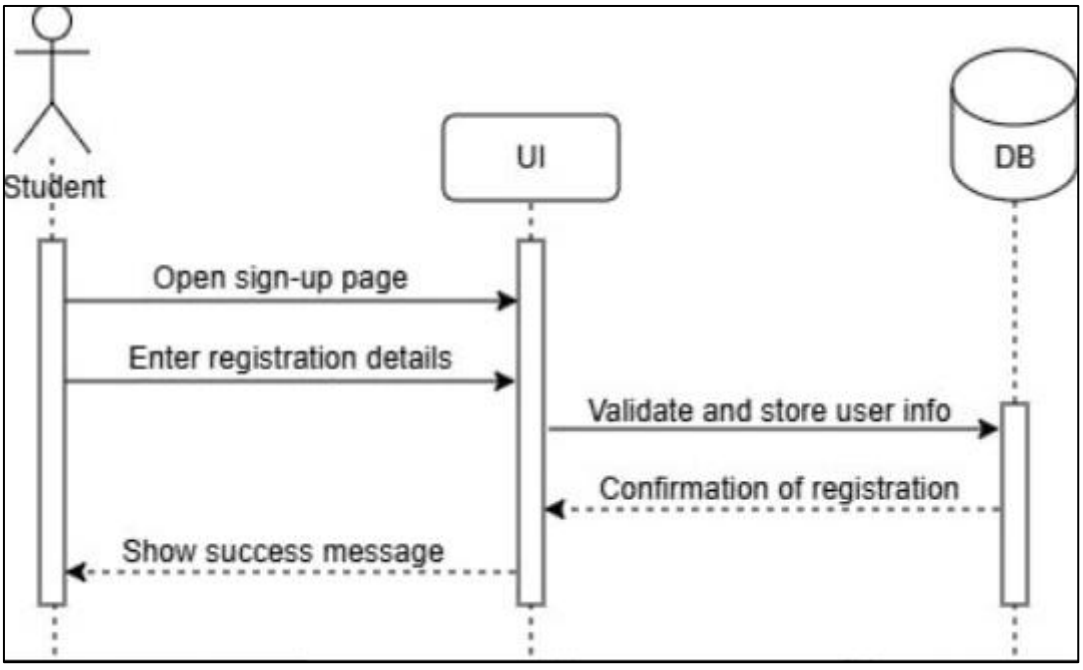
Figure12 Use case diagram

System features – use cases:

SignUp (UC-01):

Table 5 Use case specification for siginup

Use case name:	UC-01 Signup
Participating Actors:	Student
The flow of events:	1. Open sign-up page. 2. Enter name, email, phone, password. 3. System validates data. 4. System stores user. 5. System displays success message.
Alternative flows:	Alternative flow A1: starts at step 3 from the main flow: invalid or duplicate email is entered. 4. The system displays a validation error message and prevents account creation.
Entry condition:	• User is not registered
Exit conditions:	• Account created successfully



Sequence diagram for signup13 Figure

Login (UC-02):

Table 6 Use case specification for login

Use case name:	UC-02 Log In
Participating Actors:	Student, Administrator
The flow of events:	1. Open login page. 2. Enter email and password. 3. System verifies credentials. 4. System grants access.
Alternative flows:	Alternative flow A1: starts at step 3 from the main flow: incorrect email or password is entered. 4. The system displays an authentication failure message and denies access.
Entry condition:	• User has an existing account
Exit conditions:	• User authenticated successfully

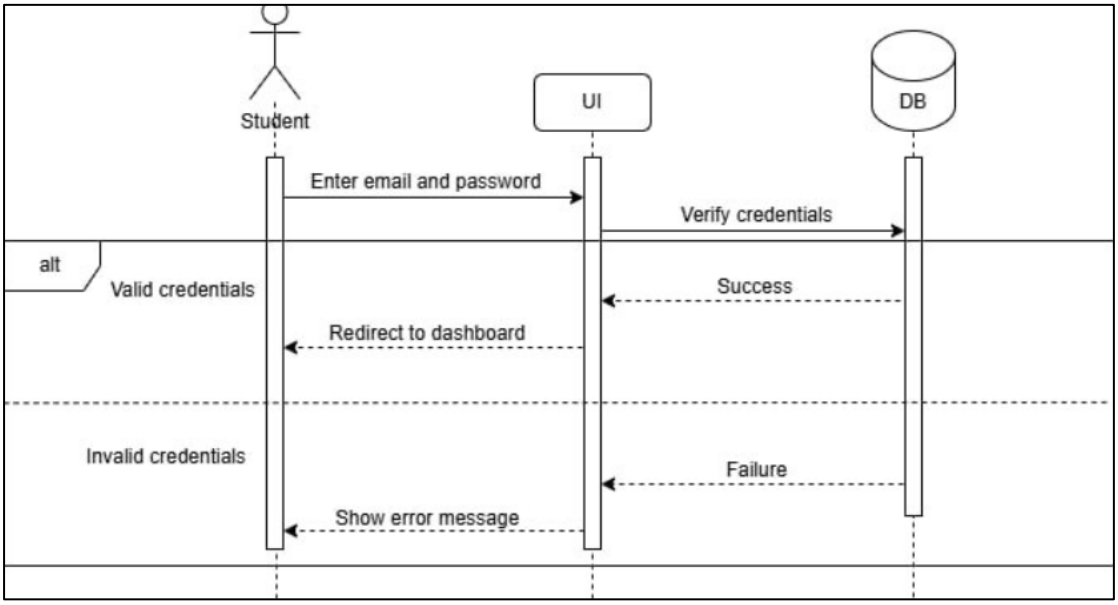
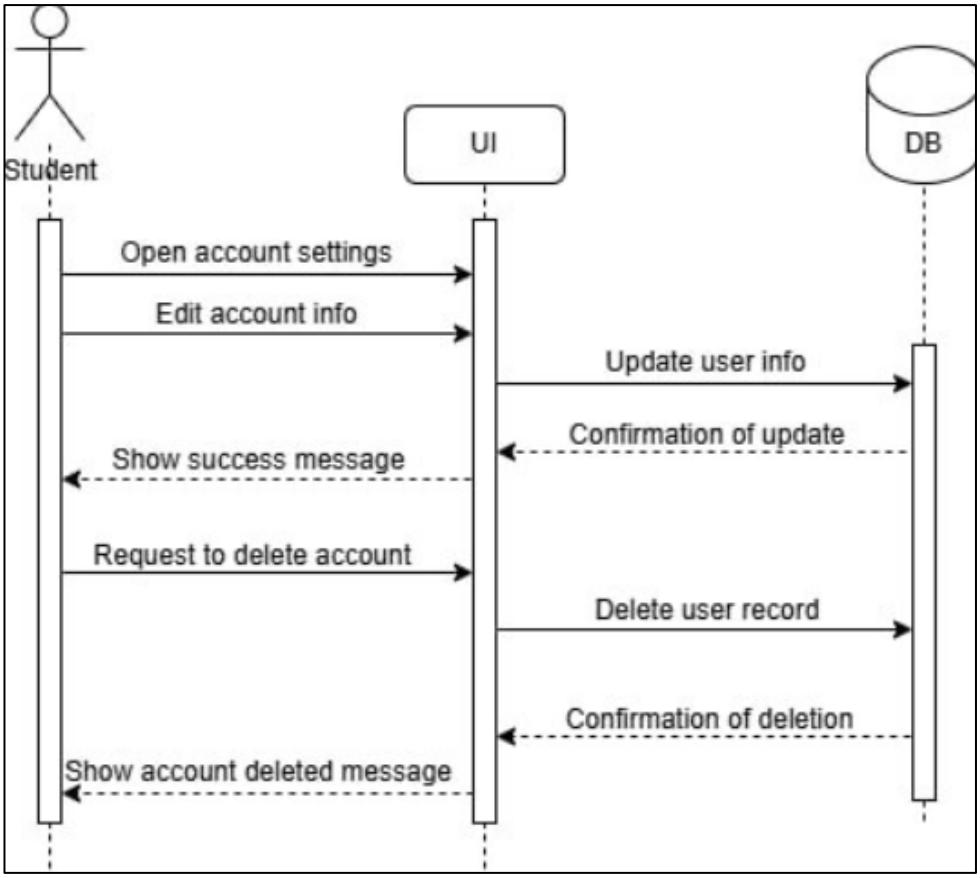


Figure14 Sequence diagram for login

Account Managment (UC-03):

Table 7 Use case specification for account management

Use case name:	UC-03 Manage Account
Participating Actors:	Student, Administrator
The flow of events:	1. Open account settings. 2. Edit personal information. 3. System validates data. 4. System updates data. 5. System displays confirmation
Alternative flows:	Alternative flow A1: starts at step 3 from the main flow: updated data is invalid. 4. The system rejects the update and displays a validation error message.
Entry condition:	• User is logged in
Exit conditions:	• Account updated successfully



Sequence diagram for accountmanagement15 Figure

Generate Quiz (UC-04):

Table 8 Use case specification for generate quiz

Use case name:	UC-04 Generate Quiz
Participating Actors:	Student
The flow of events:	1. The student selects a lecture file or summary from the available course materials. 2. The student selects the desired exam type (e.g., topic-based, difficulty level). 3. The system retrieves the selected content. 4. The system analyzes the content and generates a set of questions (e.g., multiple choice, true/false, short answer). 5. The system displays the generated questions to the student. 6. The student begins answering the questions one by one. 7. For each answer, the system evaluates the response and provides immediate feedback (correct/incorrect + explanation). 8. After completing all questions, the student clicks "Finish Exam". 9. The system calculates the final score and displays it to the student.
Alternative flows:	Alternative flow A1: starts at step 6 from the main flow: the student submits a blank answer. 7. The system prompts the student to submit an answer before proceeding.
Entry condition:	• The student must be logged in. • A valid lecture file or summary must be available and selected
Exit conditions:	• All answers are automatically evaluated and feedback is provided. • The final grade is calculated and displayed to the student

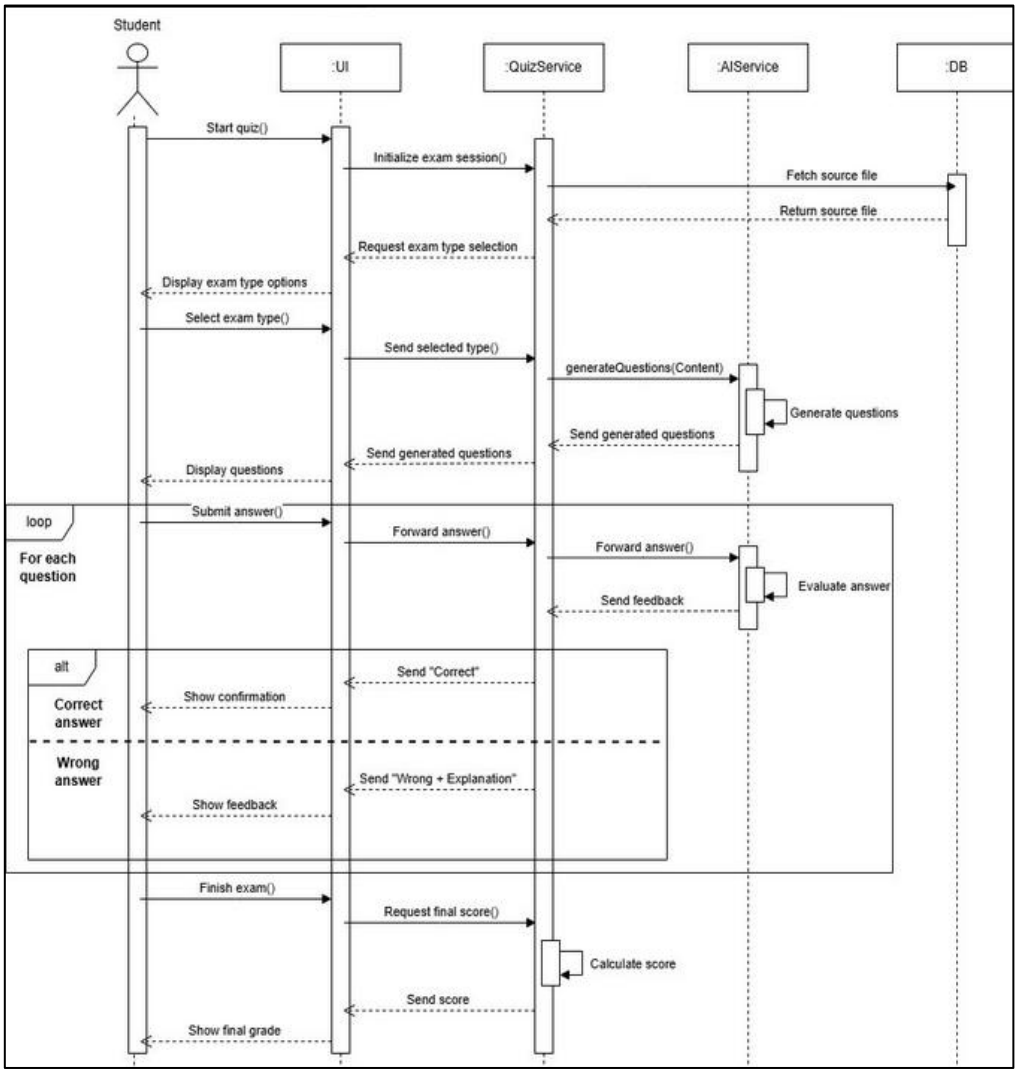


Figure16 Sequence diagram for generate quiz

Upload Lecture& Parse Lecture (UC-05):

Table 9 Use case specification for upload & parse lecture

Use case name:	UC-05 Upload & Parse Lecture
Participating Actors:	Student
The flow of events:	1. The student logs in and navigates to the selected course. 2. The student chooses Upload Lecture and selects a file in a supported format (PDF, DOCX, PPTX). 3. The student submits the upload request. 4. The system validates the file (file type, basic integrity / not empty). 5. The system stores the original file in persistent storage and returns an upload confirmation with a job-id for processing. 6. The system enqueues a parsing job and immediately returns control to the student (upload is accepted). 7. A background parser worker retrieves the stored file and extracts textual content and structure (titles, slide bullets, headings, paragraph text). 8. The parser normalizes and cleans extracted text (remove headers/footers, fix encoding, merge multi-line sentences). 9. The parser saves the extracted plain text and structured metadata (page/slide boundaries, detected headings) as an AI-ready artifact in the document store. 10. The system updates the job status to Completed and notifies the student (or the frontend reports completion on next status poll). 11. The student views the parsed text and can trigger downstream actions (Generate Summary, Generate Quiz, or Edit lecture metadata).

Alternative flows:	• Alternative A1 — Unsupported file type:
	  • Occurs at step 4. The system rejects the upload and returns an error message explaining supported formats. The student may choose a supported file and retry.
	• Alternative A2 — File corrupted / unreadable:
	  • Occurs during step 4 or 7. The system marks the job Failed, logs the error, and notifies the student with a suggested action (re-upload or contact admin).
	• Alternative A3 — Partial extraction (e.g., images-only slides):
	  • Occurs at step 7. The parser extracts available textual content and flags sections requiring manual review or OCR. The system stores the partial text, marks the job Partial, and prompts the student to review or re-

	upload a text-native file. • Alternative A4 — Asynchronous timeout / queue delay: • If processing is delayed due to heavy load, the system keeps the job in In Progress and provides estimated time or position in queue; the student may continue other tasks and check back later.
Entry condition:	• The student must be authenticated (logged in). • The student must have permission to add lectures to the selected course (owner/enrolled as allowed). • The selected file must be locally available to the student for upload.
Exit conditions:	• Success: Extracted plain text and structured metadata are stored and associated with the lecture record; the lecture status is Parsed/Ready; the student is notified; downstream AI tasks can begin. • Failure / Partial: The lecture record reflects the failure/partial state with an explanatory message; the original file remains stored (unless the student deletes it); student may retry upload or request support.

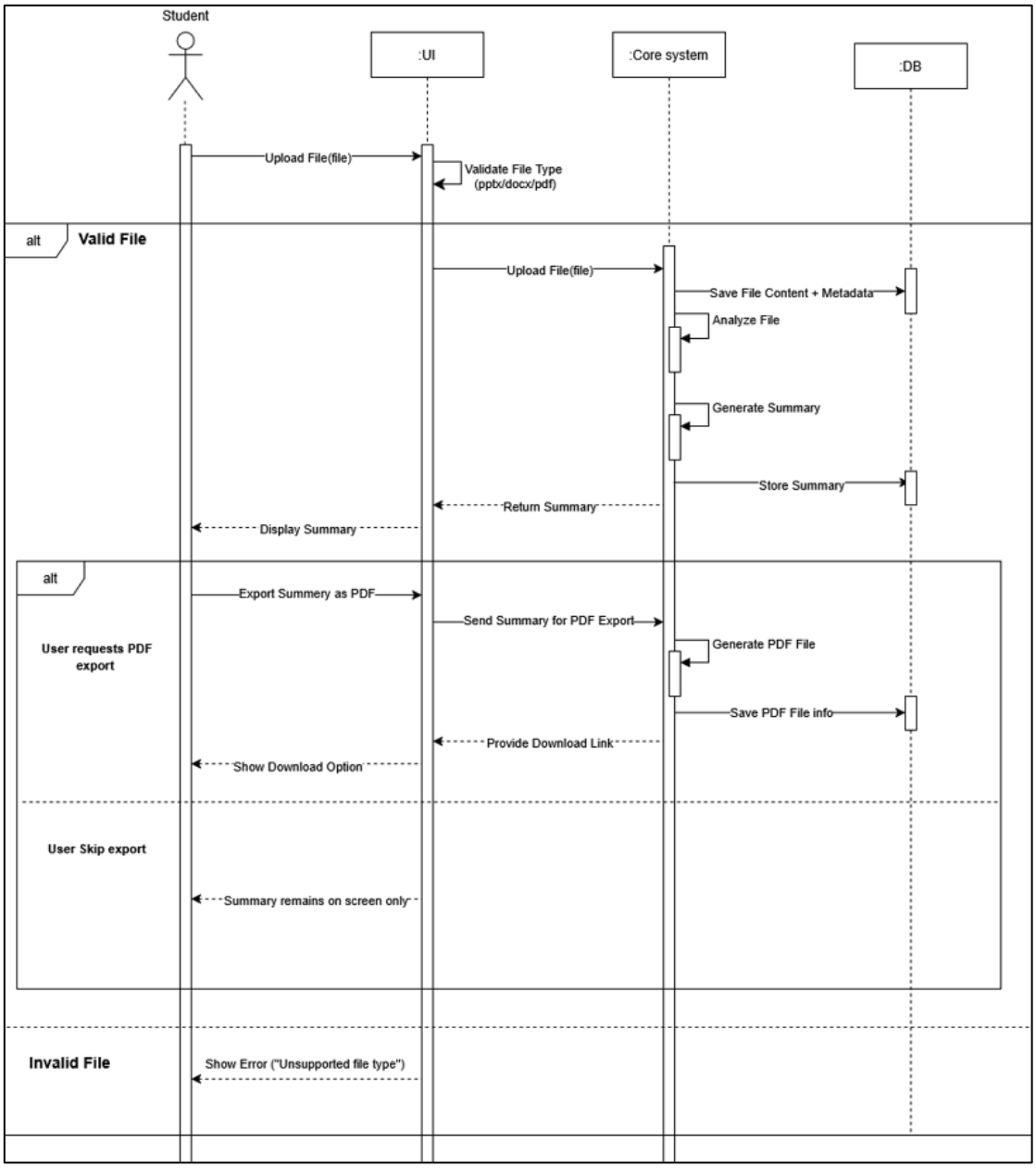


Figure17 Sequence diagram for upload lecture

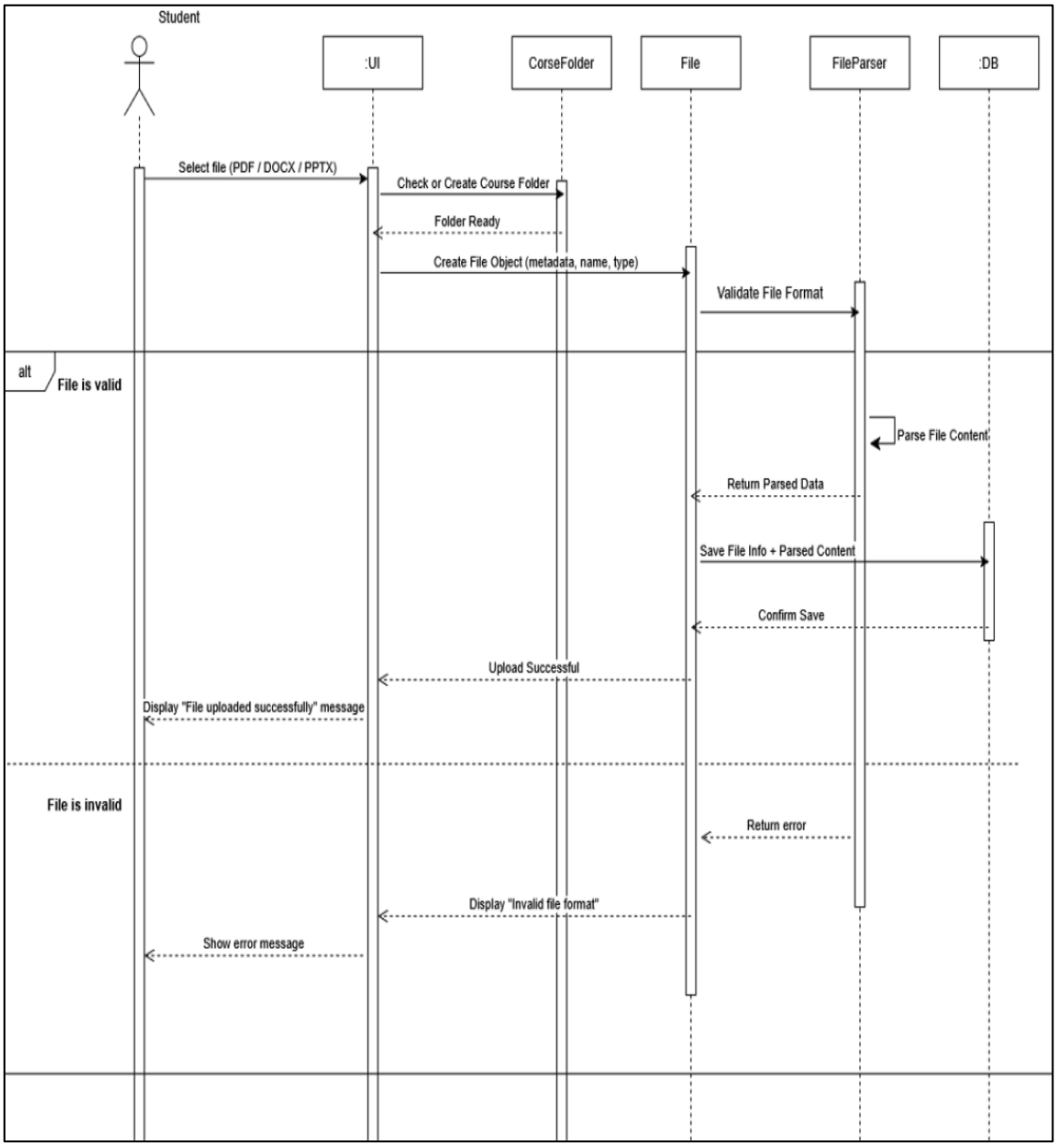


Figure18 Sequence diagram for parse lecture

Edit Lecture Info (UC-06):

Table 10 edit lecture info

Use case name:	UC-06 Edit Lecture Info
Participating Actors:	Student
The flow of events:	1. The student logs in and navigates to the selected course. 2. The student opens the lecture list and selects a lecture to edit. 3. The student clicks **Edit Lecture** and updates metadata fields (title, description, tags, module, visibility). 4. The student may optionally upload a replacement file or request re-processing. 5. The student submits the changes. 6. The system validates the input and, if a new file was provided, stores it and enqueues parsing as needed. 7. The system updates the lecture record with new metadata and timestamps, and returns a success confirmation. 8. The student views the updated lecture info.
Alternative flows:	• **A1 — Validation error:** occurs at step 6 if required fields are missing or invalid (e.g., title empty). The system rejects the change and displays validation messages; the student corrects input and resubmits. • **A2 — File reprocessing fails:** if a replacement file is uploaded and parsing fails, the system marks parsing **Failed/Partial**, retains the original file (if chosen), and notifies the student with remediation steps. • **A3 — Insufficient permission:** if the student is not authorized to edit this lecture, the system denies access and shows an authorization error.
Entry condition:	• Student is authenticated. • Student has permission to edit the selected lecture (owner or granted role). • Lecture exists in the selected course.
Exit conditions:	• **Success:** Lecture metadata is updated; if a new file was uploaded it is stored and parsing job is created. • **Failure:** No changes applied; system provides error details. All actions are logged.

Search Lecture Content (UC-07):

Table 11 search lecture content

Use case name:	UC-07 Search Lecture Content
Participating Actors:	Student
The flow of events:	1. The student logs in and navigates to a course or global search interface. 2. The student enters a search query (keywords, phrase, tag, or filter by course/module/date). 3. The system receives the query and queries the full-text index (lecture text, summaries, tags, metadata). 4. The system returns ranked search results showing matching lectures, snippets of matched text, and relevance scores. 5. The student selects a result to view the matching lecture and highlighted snippets within the summary or extracted text.
Alternative flows:	• **A1 — No results found:** system shows "no matches" with suggestions (broaden query, check spelling, remove filters). • **A2 — Partial index / stale index:** if content is recently uploaded and not yet indexed, the system indicates results may be incomplete and offers to retry after indexing completes. • • **A3 — Advanced filters produce zero matches:** system suggests removing or changing filters..
Entry condition:	• Student is authenticated. • Indexing service is available (or system will transparently handle delayed indexing).
Exit conditions:	• **Success:** Ranked list of matching lectures returned; user can navigate to selected content. • **Failure:** Search service error displayed and logged; student may retry.

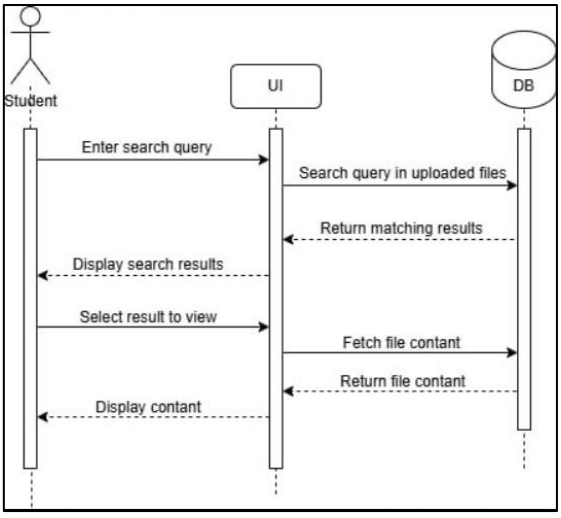


Figure19 Sequence diagram for search lecture content

Delete Lecture (UC-8):

Table 12 Use case specification for delete lecture

Use case name:	UC-8 Delete Lecture
Participating Actors:	System
The flow of events:	1. The student logs in and navigates to the course lecture list. 2. The student selects the lecture to delete and clicks **Delete**. 3. The system prompts for confirmation (shows consequences: removal of file, summaries, MCQs, scores). 4. The student confirms deletion. 5. The system marks the lecture as deleted (soft-delete) and removes or flags associated AI artifacts and generated quizzes. 6. The system returns success confirmation and updates the lecture list.
Alternative flows:	• **A1 — Cancel deletion:** student cancels at confirmation prompt; no change occurs. • **A2 — Admin forced delete / policy restriction:** if retention policy or admin settings prevent immediate removal, the system flags the lecture as **Pending Deletion** and schedules final purge after retention window; student is notified. • **A3 — Deletion failure (storage error):** system reports an error, retries or logs the failure, and notifies student to retry or contact admin.
Entry condition:	• Student is authenticated and authorized to delete the lecture (owner or permitted role). • Lecture exists.
Exit conditions:	• **Success (soft-delete):** Lecture marked deleted and hidden from normal views; associated artifacts flagged; deletion action logged with user/timestamp. • **Success (hard-delete):** If policy allows and retention expired, lecture and all related files/artifacts are permanently removed from storage. • **Failure:** Lecture remains; error returned and logged.

Create Course (UC-9):

Table 13 Use case specification for create course

Use case name:	UC-9: Create Course.
Participating Actors:	Student.
The flow of events:	1. The student selects the option to create a new course. 2. The system displays a form to enter course details. 3. The student submits the course information. 4. The system validates and saves the course data.
Alternative flows:	Exceptional flow E1: starts at step 3 from the main flow: The entered course information is incomplete or invalid. 4. The system shows an error message.
Entry condition:	• The student must be authenticated and logged into the system.
Exit conditions:	• A new course is successfully created and stored in the system.

Delete Course (UC-10):

Table 14 Use case specification for delete course

Use case name:	UC-10: Delete Course.
Participating Actors:	Student.
The flow of events:	1. The student selects the course to delete. 2. The system requests deletion confirmation. 3. The student confirms the deletion. 4. The system deletes the course and its related data.
Entry condition:	• The student must be authenticated and logged into the system.
Exit conditions:	• The selected course is permanently deleted.

Edit Course info (UC-11):

Table 15 Use case specification for edit course

Use case name:	UC-11: Edit Course Info.
Participating Actors:	Student.
The flow of events:	1. The student selects a course to edit. 2. The system displays current course information. 3. The student updates the desired fields. 4. The system saves the updated information.
Alternative flows:	Exceptional flow E1: starts at step 3 from the main flow: The updated course information fails validation. 4. The system rejects the update and displays an error message.
Entry condition:	• The student must be authenticated and logged into the system.
Exit conditions:	• Course information is successfully updated.

View Lecture (UC-12):

Table 16 Use case specification for view lecture

Use case name:	UC-12: View Lecture.
Participating Actors:	Student.
The flow of events:	1. The student selects a course. 2. The system displays a list of associated lectures. 3. The student selects a lecture to view its content.
Entry condition:	• The student must be authenticated and logged into the system.
Exit conditions:	• Lecture content is displayed to the student.

Analyze Uploaded File (UC-13):

Table 17 Use case specification for analyze uploaded file

Use case name:	UC-13 Analyze Uploaded Files.
Participating Actors:	Student.
The flow of events:	1. The student uploads a lecture file. 2. The system validates the file format and content. 3. The system prepares the file data for AI processing.
Alternative flows:	Exceptional flow E1: starts at step 2 from the main flow: The uploaded file format is not supported. 3. The system displays an error message indicating an invalid file type.
Entry condition:	• The student is authenticated and lecture file has been successfully uploaded.
Exit conditions:	• The uploaded file is successfully analyzed and ready for summarization.

Generate Summary Using AI (UC-14):

Table 18 Use case specification for generate summary using AI

Use case name:	UC-14 Generate Summary Using AI.
Participating Actors:	Student.
The flow of events:	1. The system sends the analyzed content to the AI service. 2. The AI model processes the text data. 3. The system generates a structured summary. 4. The summary is stored in the DB.
Entry condition:	• The student is authenticated and lecture file has been successfully uploaded.
Exit conditions:	• The lecture summary is successfully generated and saved.

Export Summary as PDF(UC-15):

Table 19 Use case specification for export summary as PDF

Use case name:	UC-15 Export Summary as PDF.
Participating Actors:	Student.
The flow of events:	5. The student selects the option to export the summary as PDF. 6. The system formats the summary content. 7. The system generates a PDF file. 8. The PDF file is provided for download.
Alternative flows:	Exceptional flow E1: starts at step 1 from the main flow: The summary does not exist or is not ready. 2. The system displays an error message indicating that summary is unavailable. Exceptional flow E2: starts at step 3 from the main flow: The system fails to generate the PDF file. 4. The system displays an error message: *"Failed to export summary as PDF*.
Entry condition:	• The student is authenticated and summary has already been generated.
Exit conditions:	• The summary is successfully exported as a PDF file.

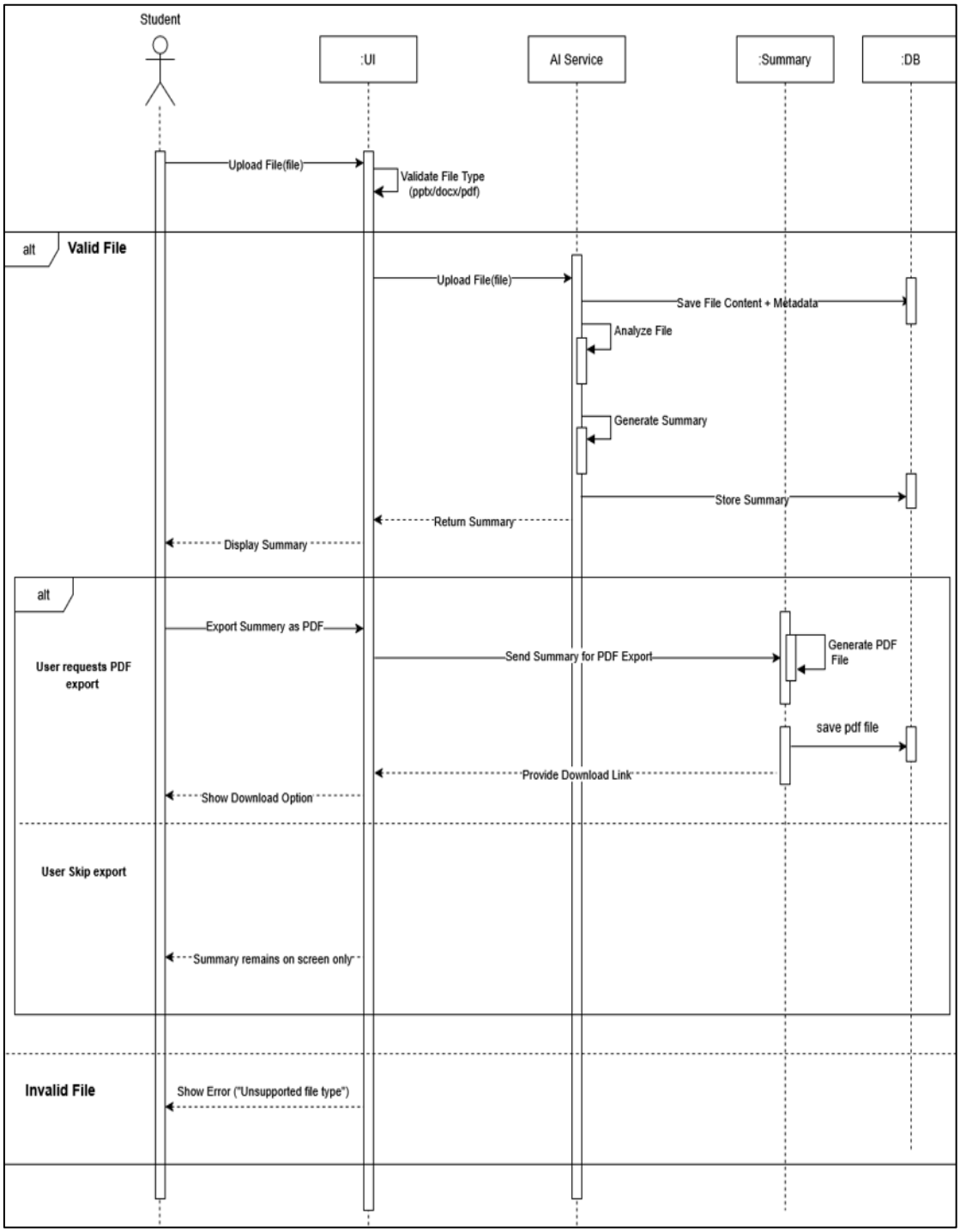


Figure20 Sequence diagram for analys uploaded file, generate summary using AI & export summary as PDF

Analysis class diagram:

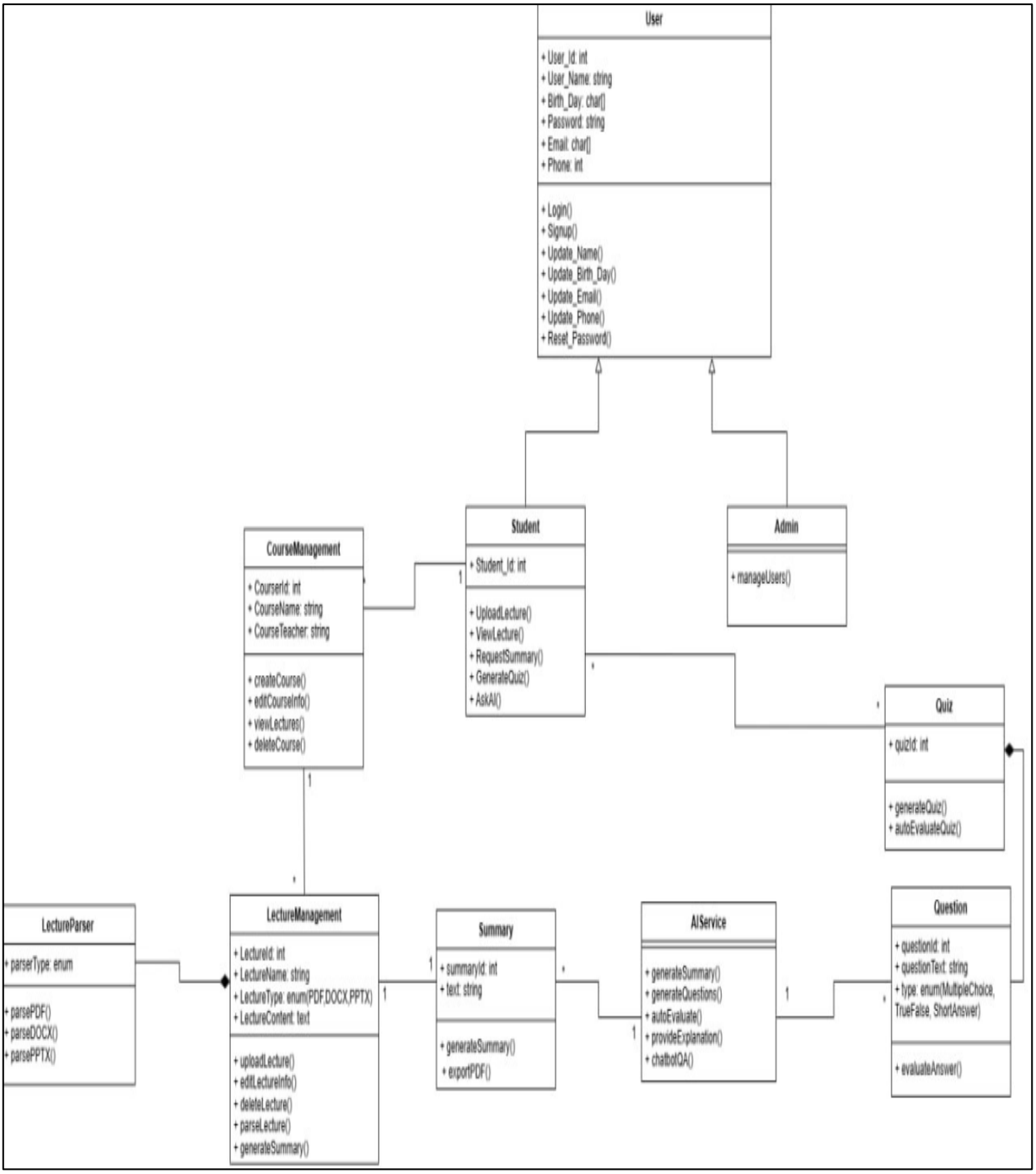


Figure21 class diagram

Entity Relationship Diagram ERD:

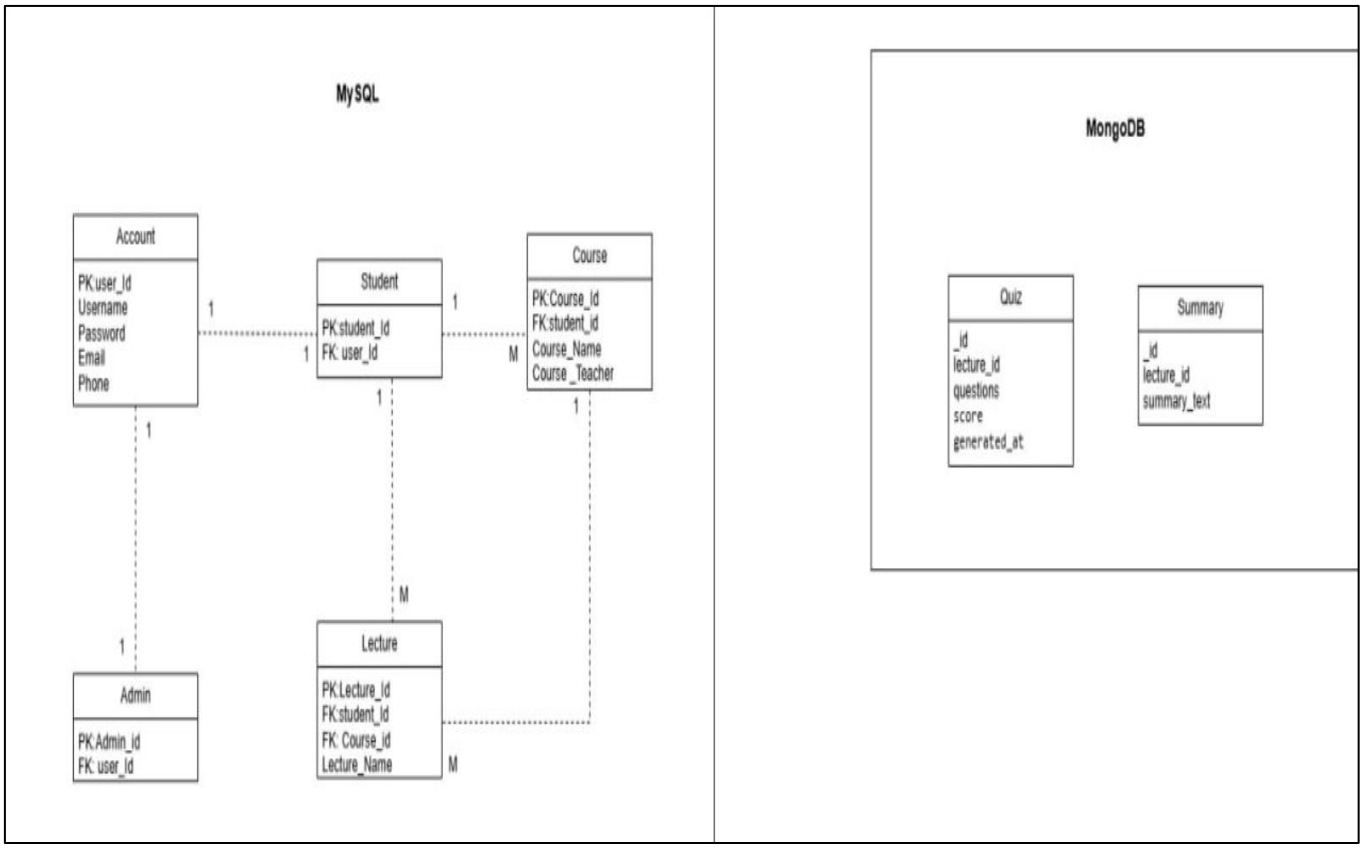


Figure22 Entity Relationship Diagram (ERD)

5. Initial Test Cases:

Test Case Scenario:		**Case 1: Signup**	
Test case id	Test case title	Test steps	Expected result
TC-01	Signup with valid data	1) Navigate to Signup page. 2) Enter valid username. 3) Enter valid email. 4) Enter strong password. 5) Click Sign Up.	Account is created successfully.
TC-02	Signup with email without @	1) Navigate to Signup page. 2) Enter username. 3) Enter email without @. 4) Enter valid password. 5) Click Sign Up.	Invalid email format error is displayed.

Test Case Scenario:		**Case 2: Login**	
Test case id	Test case title	Test steps	Expected result
TC-03	Login with valid credentials	1) Open system login page. 2) Enter valid email. 3) Enter valid password. 4) Click Login.	User is logged in successfully.
TC-04	Login with invalid email or password	1) Open system login page. 2) Enter invalid email or enter invalid password. 3) Click Login.	Error message is displayed.

Test Case Scenario:		**Case 3: Manage Account**	
Test case id	Test case title	Test steps	Expected result
TC-05	Update account information	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Click Save.	Account information updated successfully.
TC-06	Edit user data while leaving blank fields	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Leave fields blank) Click Save.5	Error message is displayed.
TC-07	Edit user data with an incorrect password entered	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Entering a password that does not match) Click Save.5	Error message is displayed.

Test Case Scenario:		**Case 4: Generate quiz**	
Test case id	Test case title	Test steps	Expected result
TC-08	Generate quiz successfully with valid input.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select ≥1 lecture. 4) Select question type. 5) Enter question count. 6) Click Generate.	Quiz is generated with requested number & type of questions.
TC-09	Generate quiz from multiple lectures.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select multiple lectures. 4) Select question type. 5) Enter question count. 6) Click Generate.	Quiz generated using all selected lectures.
TC-10	Generate quiz with mixed question types.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select multiple types. 5) Enter counts. 6) Click Generate.	Quiz contains selected mixed types.
TC-11	Generate quiz with no lecture selected.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Do not select any lecture. 4) Click Generate.	Error: Must select at least 1 lecture.
TC-12	No question type selected.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Do not select question type. 5) Click Generate.	Error: Must select question type.
TC-13	No question count entered.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Leave count empty. 6) Click Generate.	Error: Must enter number of questions.

TC-14	Zero question count	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Enter 0. 6) Click Generate.	Error: Question count must be greater than 0.
TC-15	Count > available questions	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Enter > available content. 6) Click Generate.	Error: Requested questions exceed available content.

Test Case Scenario:		**Case 5: Auto evaluate answers**	
Test case id	Test case title	Test steps	Expected result
TC-16	Submit a correct answer.	1) Login as student. 2) Navigate to Quiz. 3) Select an answer. 4) Click Submit.	Correct answer message shown.
TC-17	Submit an incorrect answer.	1) Login as student. 2) Navigate to Quiz. 3) Select an answer. 4) Click Submit.	Incorrect answer message shown + explanation .
TC-18	Submit an empty answer.	1) Login as student. 2) Navigate to Quiz. 3) Do not select answer. 4) Click Submit.	Must select an answer message shown.

Test Case Scenario:		**Case 6: Analyze Uploaded File**	
Test case id	Test case title	Test steps	Expected result
TC-19	Upload valid PDF.	1) Login as Student. 2) Open upload page. 3) Select a valid PDF file. 4) Click Upload.	The system accepts the PDF.
TC-20	Upload valid DOCX.	1) Login as student. 2) Open upload page. 3) Select a valid DOCX file. 4) Click Upload.	The system accepts the DOCX.
TC-21	Upload valid PPTX.	1) Login as student. 2) Open upload page. 3) Select a valid PPTX file . 4) Click Upload.	The system accepts the PPTX.
TC-22	Reject unsupported file type (TXT).	1) Login as student. 2) Open upload page . 3) Select a valid TXT file. 4) Click Upload.	Error: The file not supported.
TC-23	Empty file content.	1) Login as student. 2) Open upload page. 3) Select a PDF/DOCX/PPTX. 4) Click Upload.	Error: The file is Empty.

Test Case Scenario:		**Case 7: Generate Summary using AI**	
Test case id	Test case title	Test steps	Expected result
TC-24	Generate summary from a valid file.	1) Student logs in successfully. 2) Student opens the document upload page. 3) Student selects a valid file (PDF/DOCX/PPTX) that contains readable content. 4) Student clicks on “Generate Summary”.	The system generate summary and displayed to the student .

Test Case Scenario:		**Case 8: Export Summary as pdf**	
Test case id	Test case title	Test steps	Expected result
TC-25	Export summary to PDF.	1) Login as Student. 2) Uploads a valid file. 3) Generates summary successfully. 4) Clicks Export as PDF.	The system Export Summary to PDF and download link displayed to student.
TC-26	Attempt export without summary.	1) Login as student. 2) Uploads a valid file. 3) skip summary generation. 4) Click Export as PDF.	Error: "please generate a summary before exporting"; no PDF created.

Test Case Scenario:		**Case 9: Create Course**	
Test case id	Test case title	Test steps	Expected result
TC-27	Create Course with Valid Data.	1) The student selects Create Course. 2) The system displays the course creation form. 3) The student enters valid course information. 4) The student submits the form.	The course is created successfully and added to the course list.

TC-28	Create Course with Missing Required Fields.	1) The student selects “Create Course. 2) The student leaves one or more required fields empty. 3) The student submits the form.	The system displays an error message indicating missing or invalid data.

Test Case Scenario:		**Case 10: Edit Course Info**	
Test case id	Test case title	Test steps	Expected result
TC-29	Edit Course with Valid Information.	1) The student selects an existing course. 2) The student chooses the edit option. 3) The student updates the course information with valid data. 4) The student submits the changes.	The course information is successfully updated and saved.
TC-30	Edit Course with Invalid Data.	1) The student selects a course to edit. 2) The student enters invalid or incomplete course data. 3) The student submits the changes.	The system rejects the update and displays an error message.

Test Case Scenario:		**Case 11: Delete Course**	
Test case id	Test case title	Test steps	Expected result
TC-31	Delete Course with Confirmation.	1) The student selects the course to delete. 2) The system displays a deletion confirmation message. 3) The student confirms the deletion.	The course and all related data are permanently deleted.

Test Case Scenario:		**Case 12: View Lecture**	
Test case id	Test case title	Test steps	Expected result
TC-32	View Existing Lecture.	1) The student selects a course. 2) The system displays the list of available lectures. 3) The student selects a lecture.	The selected lecture content is displayed successfully.
TC-33	View Lecture When No Lectures Exist.	1) The student selects a course.	The system displays a message indicating that no lectures are available.

Test Case Scenario:		**Case 13: Edit lecture info**	
Test case id	Test case title	Test steps	Expected result
TC-34	Edit lecture name successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Change the lecture name. 6) Click Save.	Lecture name is updated successfully and displayed with the new name.
TC-35	Edit lecture name with empty value.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Clear the lecture name field. 6) Click Save.	System shows validation error and lecture name is not updated.
TC-36	Edit lecture name with special characters.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Enter a name with special characters (e.g. @#lecture!) 6) Click Save.	System shows validation error and lecture name is not updated.

Test Case Scenario:		**Case 14: Delete lecture**	
Test case id	Test case title	Test steps	Expected result
TC-37	Delete lecture successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Delete. 5) Confirm deletion.	Lecture is deleted successfully and removed from the lectures list.

Test Case Scenario:		**Case 15: Search lecture content**	
Test case id	Test case title	Test steps	Expected result
TC-38	Search lecture content successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Enter a valid keyword in the search field. 5) Click Search**.**	Matching content inside the lecture is highlighted or displayed.
TC-39	Search with keyword not found in lecture.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Enter a keyword not present in the lecture. 5) Click Search**.**	System displays “No results found in this lecture”.

TC-40	Search with empty input inside lecture.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Leave search field empty. 5) Click Search**.**	System shows validation message asking for a keyword.

Test Case Scenario:		**Case 16: Upload Lecture**	
Test case id	Test case title	Test steps	Expected result
TC-41	Upload Lecture with Supported File.	1) The student selects the upload lecture option. 2) The student chooses a supported lecture file. 3) The student submits the file.	The lecture file is uploaded successfully and sent for parsing.
TC-42	Upload Lecture with Unsupported File Type.	1) The student selects an unsupported file format. 2) The student submits the file.	The system rejects the file and displays an error message.

Test Case Scenario:		**Case 17: Parse Lecture File**	
Test case id	Test case title	Test steps	Expected result
TC-43	Parse Lecture File (PDF).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as PDF. 3) The system extracts text content from the PDF file.	The lecture content is successfully extracted as text.
TC-44	Parse Lecture File (DOCX).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as DOCX. 3) The system extracts text content from the DOCX file.	The lecture content is successfully extracted as text.
TC-45	Parse Lecture File (PPTX).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as PPTX. 3) The system extracts text content from presentation slides.	The lecture content is successfully extracted as text.
TC-46	Parse Lecture File with Unsupported Format.	1) The system detects the uploaded file. 2) The system identifies an unsupported file format.	The system rejects the file and displays an error message.

7. Initial RTM (Requirements Traceability Matrix):

REQ-ID	Use cases	analysis	System design	Detailed design	coding	Test cases
RE-FR-SU-01	UC-01	analysis				TC-01 TC-02
RE-FR-LI-02	UC-02	analysis				TC-03 TC-04
RE-FR-MA-03	UC-03	analysis				TC-05 TC-06 TC-07
RE-FR-GQ-04	UC-04	analysis				TC-08 TC-09 TC-10 TC-11 TC-12 TC-13 TC-14 TC-15
RE-FR-AEA-05	UC-04	analysis				TC-16 TC-17 TC-18
RE-FR-U&PL-06	UC-05	analysis				TC-43 TC-44 TC-45 TC-46
RE-FR-ELI-07	UC-06	analysis				TC-34 TC-35 TC-36
RE-FR-SLC-08	UC-07	analysis				TC-38 TC-39 TC-40
RE-FR-DL-09	UC-08	analysis				TC-37
RE-FR-CC-10	UC-09	analysis				TC-27 TC-28
RE-FR-DC-11	UC-10	analysis				TC-31
RE-FR-ECI-12	UC-11	analysis				TC-29 TC-30
RE-FR-VL-13	UC-12	analysis				TC-32 TC-33

RE-FR-AUF-14	UC-13	analysis				TC-19 TC-20 TC-21 TC-22 TC-23
RE-FR-GSUAI-15	UC-14	analysis				TC-24
RE-FR-ESPDF-16	UC-15	analysis				TC-25 TC-26

Chapter4 System Design

1. Introduction:

This chapter offers a comprehensive overview of the design phase steps, encompassing system design and architecture determination, as well as detailed design and modeling. It underscores the importance of these steps in creating a robust system for collecting, analyzing, and utilizing user feedback to continuously enhance the software system.

2. System Architecture:

In this section we will discuss system design where system design involves defining the overall structure and architecture of the software, ensuring that all components work seamlessly together. This phase establishes the blueprint for efficient data flow, interaction, and integration within the system.

The first constraints to take into consideration when building an architecture for a software system are the non-functional requirements (quality attributes), so first we will start with the non-functional requirements of our system

1. **(Usability – User-Friendly Interface):**
   The system shall provide simple and intuitive user interfaces for both Students and Admins. A new user should be able to understand the main system functions (course creation, lecture upload, summary viewing, and quiz usage) within **15 minutes** of initial use without external assistance.
2. **(Security – Authentication and Data Protection):**
   The system shall ensure the security and privacy of all user data by implementing JWT-based authentication, role-based access control, and secure password policies (minimum 8 characters including at least one number). All data transmission must occur over secure HTTPS connections.
3. **(Performance – Responsiveness and AI Processing):**
   The system shall respond to standard user operations (navigation, search, viewing summaries) within **10 seconds**. AI-based operations such as lecture summarization and quiz generation shall be processed asynchronously and completed within an acceptable time frame, with progress status provided to the user.
4. **(Scalability and Extensibility):**
   The system shall be designed with a modular and scalable architecture that allows future expansion, including upgrading AI models, adding new features (search, export, multimedia support), and handling increased numbers of users and lecture files without major system redesign.

System decomposition - Component diagram:

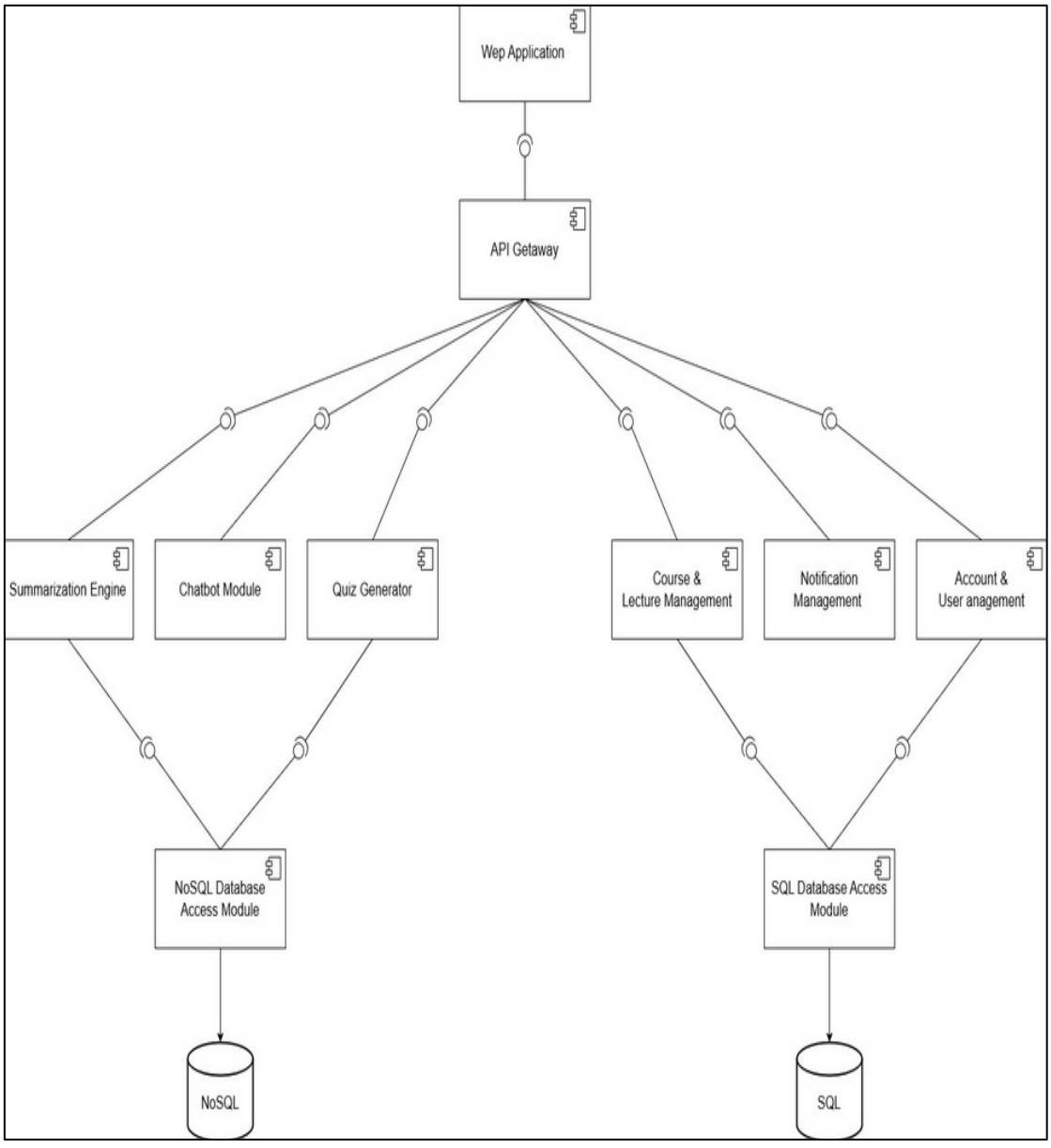


Figure23 Component diagram

Components functionalities:

1) Web Application (Frontend)

Responsibilities

- Single Page Application (React + Vite) that provides the UI for Students and Admin.
- Handles file selection/upload, course/lecture CRUD UIs, summary viewing, quiz taking, chatbot interaction, and notifications display.

Interfaces / Inputs

- Calls REST/HTTP(S) APIs exposed by API Gateway.
- Uses JWT for authenticated requests.

Outputs

- Presents API results to the user; uploads lecture files to Course & Lecture Management via the Gateway.
- Initiates requests for summaries, quizzes, chat, and exports.

Non-functional notes

- Keeps UI responsive by offloading long tasks to background jobs (polling / websocket notifications).
- Should include client-side validation for file types (pdf/docx/pptx) and size.

2) API Gateway

Responsibilities

- Single entry point for all frontend requests.
- Authentication & authorization (validate JWT), request routing to microservices.
- Enforce rate limits, SSL termination, request logging, basic input validation.
- Aggregate / proxy calls when needed (e.g., combine user profile + course metadata).

Interfaces

- REST endpoints for: `/auth/*`, `/courses/*`, `/lectures/*`, `/ai/summarize`, `/ai/quiz`, `/chat/*`, `/notifications/*`.

- Auth: `/auth/login`, `/auth/refresh`, `/auth/verify`.

Interactions

- Forwards requests to the appropriate microservice (Account, Course, Summarization, Quiz, Chatbot).
- Publishes uploaded-file events to a message queue/topic for async processing.

Non-functional notes

- Implement JWT verification here (no sensitive logic in frontend).
- Provide observability (request traces, metrics) and centralized logging.

3) Course & Lecture Management Service

Responsibilities

- CRUD for courses and lectures (metadata).
- Accepts lecture file uploads (stores original file metadata, triggers parsing job).
- Exposes endpoints for listing lectures, viewing parsed text/summaries, editing, deleting.
- Orchestrates storage of files (object storage / filesystem) and creates processing job entries.

Inputs / Outputs

- Input: upload requests (multipart), edit requests, delete requests.
- Output: lecture records, file URLs, parsed-text artifacts references.

Interactions

- Stores relational metadata in **MySQL** (course, lecture, user-course relations).
- Writes file metadata (and file location) to DB, pushes parse/upload event to message queue (RabbitMQ/Kafka).
- On deletion: triggers cleanup of AI artifacts from MongoDB and file storage (soft-delete recommended).

Non-functional notes

- Use presigned URLs or direct chunked uploads if files large.
- Validate file formats & size before persisting.

4) Parse / Summarization Engine (Summarization Engine)

Responsibilities

- Background worker/service that:
  - Retrieves uploaded files,
  - Extracts raw text (PDF/DOCX/PPTX parsers),
  - Cleans/normalizes text,
  - Runs NLP model(s) (via Hugging Face) to generate structured text summaries (headings, bullets, key concepts),
  - Stores summaries and intermediate artifacts (embeddings, extracted text) in the document store.

Inputs / Outputs

- Input: parse job message containing file location and lecture id.
- Output: extracted raw text, summary object (title, sections, bullets), embeddings, confidence/quality metrics.

Interactions

- Pulls jobs from the queue published by Course service.
- Loads model(s) using Python/Django workers (or separate Python microservice). Calls Hugging Face transformer models (local or hosted).
- Stores results in **MongoDB** (document: lecture_id, summary_text, sections, embeddings) and updates status back to Course service (via API or DB).
- Optionally triggers downstream flows (Generate Quiz, index for search).

Non-functional notes

- Run asynchronously; scale worker pool horizontally via Docker.
- Implement retry/backoff for transient model or network errors; persist job status and error messages.
- Provide a fallback: if abstractive model fails, use an extractive fallback (TextRank) to ensure some output exists.
- Cache model instances to reduce cold-starts; consider GPU nodes for heavy loads.

5) Chatbot Module

Responsibilities

- Provides conversational Q&A over lecture content and generated summaries.
- Accepts free-text questions and returns AI-generated answers referencing lecture material.

Inputs / Outputs

- Input: question text and context identifier (lecture_id, summary_id); JWT-authenticated user id.
- Output: answer text, optionally cited snippets/links to summaries or lecture passages.

Interactions

- Fetches lecture summary / extracted text and embeddings from MongoDB.
- Uses semantic search (embedding similarity) to select relevant context passages.
- Invokes Hugging Face model(s) (chain-of-thought / prompt with context) to generate an answer.
- Logs conversations and stores chat transcripts for reuse or analytics.

Non-functional notes

- Rate-limit users to avoid abuse and cost blow-up.
- Provide an option to show the source text snippets used to produce the answer for transparency.
- If models are unavailable, gracefully return a message and offer fallback (search results).

6) Quiz Generator

Responsibilities

- Automatically generates multiple-choice questions (MCQs) from lecture text or summaries.
- Generates distractors (plausible wrong answers) and marks the correct answer.
- Stores quizzes (question bank) and provides graded attempts.

Inputs / Outputs

- Input: generate-quiz job (lecture_id, options such as number of questions, difficulty).
- Output: a quiz object with MCQs, correct answers, distractors, explanations.

Interactions

- Triggered by student request or post-summary job; consumes lecture text and embeddings from MongoDB.
- Uses NLP QG models (Hugging Face) to propose questions and candidate answers.
- Uses semantic-similarity (embeddings) to select/score distractors and remove duplicates.
- Persists generated quiz and answer keys in MongoDB or MySQL (metadata in MySQL, question documents in MongoDB).
- On student submission, the Quiz service auto-evaluates answers, stores attempt and score, and returns feedback/explanations.

Non-functional notes

- Ensure MCQs meet quality thresholds: basic filters for grammar, length, and answer uniqueness.
- Provide admin/student flags to mark low-quality questions for review.

7) Notification Management

Responsibilities

- Delivers status updates and alerts: parsing/summarization completion, quiz ready, system messages.
- Supports multiple channels: in-app notifications, WebSocket push, email (SMTP), optional mobile push.

Inputs / Outputs

- Input: events from other services (job complete, job failed, admin announcements).
- Output: notifications persisted and delivered to users.

Interactions

- Subscribes to the message bus or receives webhook calls from other services.
- Writes notification records (MySQL) and pushes real-time updates to the frontend (via WebSocket or Server-Sent Events) through the API Gateway.

Non-functional notes

- Ensure reliable delivery (retry, exponential backoff).
- Keep notification history (user viewable) and support preference settings.

8) Account & User Management Service

Responsibilities

- User registration, login, profile management, password reset, role assignment (Student/Admin).
- Issues JWT tokens (or collaborates with Auth server) and handles email verification.

Inputs / Outputs

- Input: signup/login/profile update requests.
- Output: user records (MySQL), JWT tokens.

Interactions

- Stores user accounts and credentials (MySQL). Passwords hashed; email verification tokens persisted.
- Enforces password policy and rate-limits authentication endpoints at the Gateway.
- Provides endpoints the Gateway uses for auth checks (or the Gateway validates tokens locally).

Non-functional notes

- Implement audit logging for account changes.
- Provide admin endpoints for account management (suspend, list, delete).

9) NoSQL Database Access Module (MongoDB)

Responsibilities

- Encapsulates access patterns for document-oriented data: summaries, extracted text, embeddings, generated quizzes, chat transcripts, and any large JSON AI artifacts.

Usage

- Fast retrieval of AI artifacts and searches by lecture_id, embedding vectors for semantic search.
- Stores versioned artifacts (so changes/reprocessing can be tracked).

Non-functional

- Index embeddings and text fields for efficient retrieval.
- Periodic backup and TTL policies for transient artifacts.

10) SQL Database Access Module (MySQL)

Responsibilities

- Provides structured, relational storage for users, courses, enrollments, lecture metadata, audit logs, and notification metadata.

Usage

- Strong consistency for relational queries (who owns what course, permissions).
- Joins and transactional updates (e.g., course creation and default settings).

Non-functional

- Backups, migrations, and foreign-key constraints as needed.

Typical data flows (two common sequences)

A. Upload → Parse → Summarize → Notify

1. Web App POST `/lectures/upload` → API Gateway (auth check) → Course Service stores file metadata and file (object store) and enqueues `parse_job`.
2. Parser/Summarization workers pull `parse_job`, extract text, run summarization model (Hugging Face), store extracted text & summary in MongoDB and update lecture record in MySQL.
3. Parser publishes `summary_ready` event → Notification service notifies user (web UI or email) → Web App fetches summary and displays it.

B. Generate Quiz & Auto-Evaluate

1. Student requests quiz: Web App → Gateway → Quiz Service (generate request).
2. Quiz Service consumes lecture summary / text from MongoDB, generates MCQs, saves quiz object; notifies student when ready.
3. Student takes quiz → submits answers → Quiz Service auto-evaluates (compare to stored keys), writes attempt & score to MySQL, returns results and explanations.

Cross-cutting concerns & operational notes

- **Message queue**: use RabbitMQ/Kafka to decouple upload → parse → summarize → quiz flows and to scale workers.
- **Model hosting**: use local HF models in containers or call HF hosted endpoints. Cache models across worker processes to reduce cold-start.
- **Security**: API Gateway enforces TLS and JWT validation. Services authorize user actions (ownership checks) server-side.
- **Observability**: central logging (ELK/EFK), metrics (Prometheus), distributed tracing for performance debugging.
- **Error handling**: persist job status (`Pending`/`InProgress`/`Failed`/`Partial`/`Completed`) with human-readable error messages; implement retries and alerting for failed jobs.
- **Storage**: consider object storage (S3-compatible) for lecture files, MongoDB for AI artifacts, MySQL for relational data. Use daily backups and retention policies.
- **CI/CD & deployment**: Docker images per service, orchestrate with docker-compose for dev and Kubernetes for production. Use GitHub Actions for builds/tests and image publishing.

System Architecture:

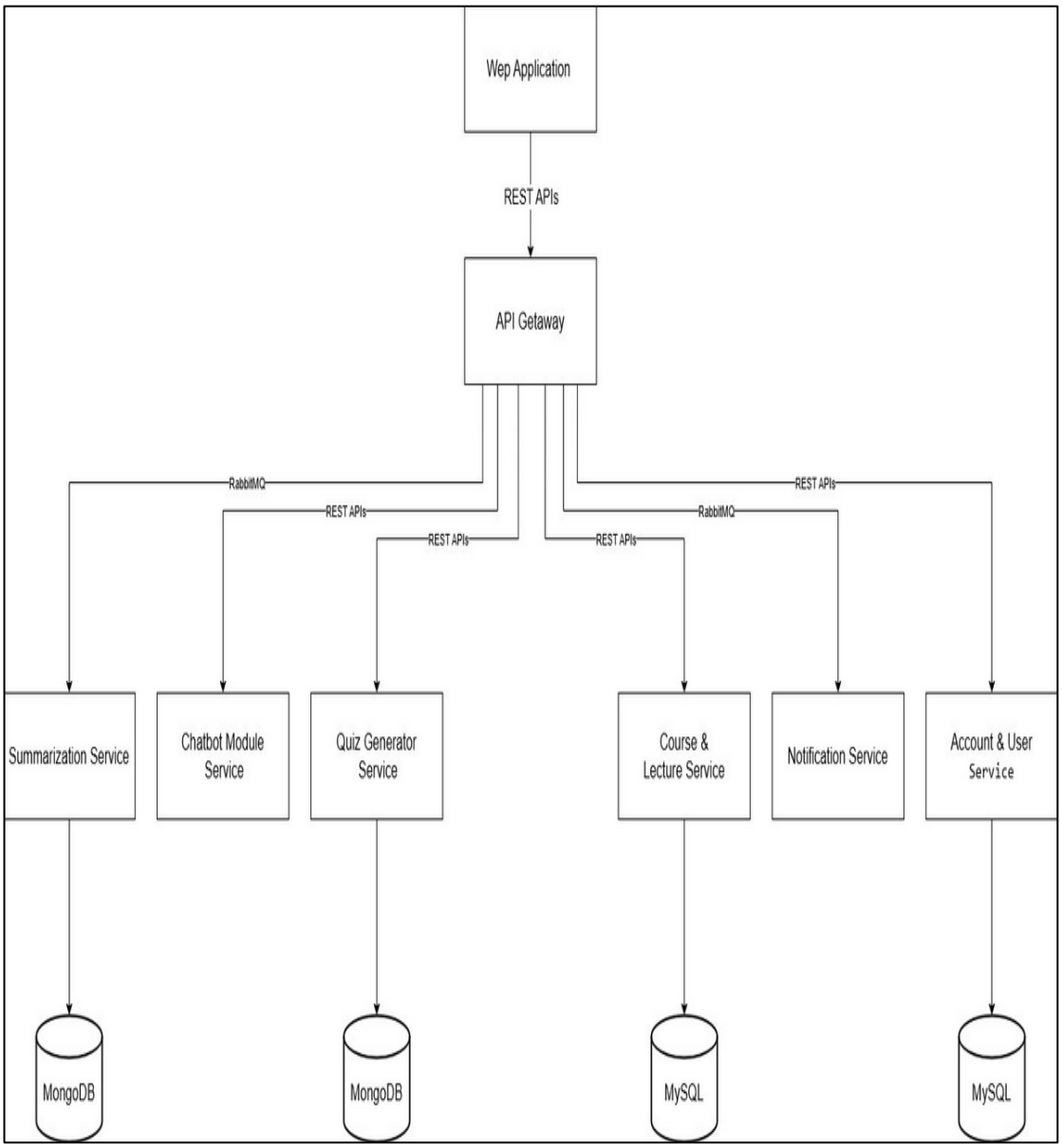


Figure24 System architecture

- We used Microservices Architecture combined with a client–server model to build the system in a scalable and modular way.

- The client side is a web application that communicates with the backend through RESTful APIs.

- The server side is composed of multiple independent backend services, each responsible for a specific business capability. These services are accessed through a centralized API Gateway, which acts as a single entry point for the system.

- The API Gateway is responsible for request routing, authentication, and communication between the client and the backend microservices using REST APIs.

- The backend is divided into the following microservices: Account & User Service: manages user registration, authentication, and account information. Course & Lecture Service: handles course creation, lecture management, and related academic data. Summarization Service: processes lecture content and generates AI-based summaries, storing results in a NoSQL database. Quiz Generator Service: generates quizzes automatically based on lecture content and summaries. Chatbot Module Service: provides an interactive AI-based assistant to help students with course-related questions. Notification Service: sends system notifications and updates asynchronously.

- The services communicate synchronously using REST APIs and asynchronously using RabbitMQ for background processing and event-driven communication, improving system performance and decoupling services.

- Each microservice has its own dedicated database, following the database-per-service pattern: MySQL is used for structured and relational data such as users, courses, and lectures. MongoDB is used for unstructured and AI-generated data such as summaries and quizzes.

- This architecture improves scalability, maintainability, fault isolation, and development flexibility, as each service can be developed, deployed, and scaled independently.

Used design patterns:

1. Facade Design Pattern

Pattern Overview

The **Facade Pattern** is a structural design pattern that provides a simplified and unified interface to a complex subsystem. It hides the internal complexity of the system and exposes only the necessary functionality to the client.

Application of the Facade Pattern in the System

In this system, the **API Gateway** implements the **Facade Pattern**.

Instead of allowing the Web Application to communicate directly with multiple backend microservices, the API Gateway acts as a single, unified interface. The client sends all requests to the API Gateway, which then internally forwards each request to the appropriate microservice such as Account & User Management, Course & Lecture Management, Summarization Engine, or Quiz Generator.

The client is therefore unaware of:

- The number of backend services.
- Their internal structure.
- Their communication mechanisms.

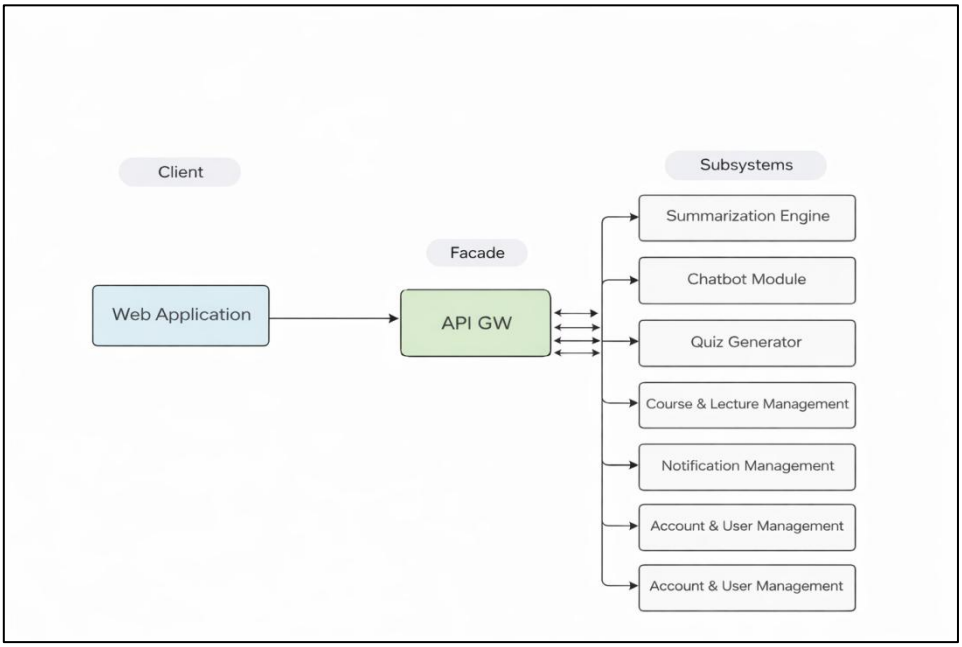


Figure 25 Facade pattern applied using the API Gateway.

2. Template Method Design Pattern

Pattern Overview

The **Template Method Pattern** is a behavioral design pattern that defines the skeleton of an algorithm in a base class while allowing subclasses to implement specific steps of the algorithm without changing its overall structure.

Application of the Template Method Pattern in the System

In this system, the **Template Method pattern** is applied in the **file parsing process** for lecture files.

Lecture files may be provided in different formats such as PDF, DOCX, or PDF. Although the overall parsing workflow remains the same, the **file parsing and text extraction step differs based on the file format**. A common parsing structure is defined, while format-specific parsing logic is implemented in specialized parsers.

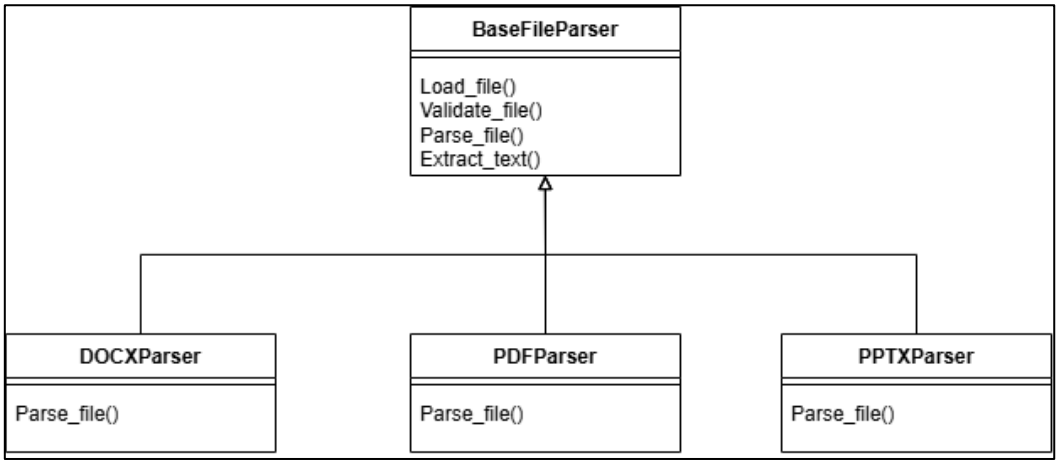


Figure 26 Template Method pattern for lecture file parsing.

Database Design:

NoSQL Database – MongoDB: For data related to user reviews and AI algorithm results, we employ MongoDB. This NoSQL database is well-suited for handling large volumes of unstructured data, providing the flexibility and scalability needed to manage diverse and dynamic datasets efficiently.

Review document structure:

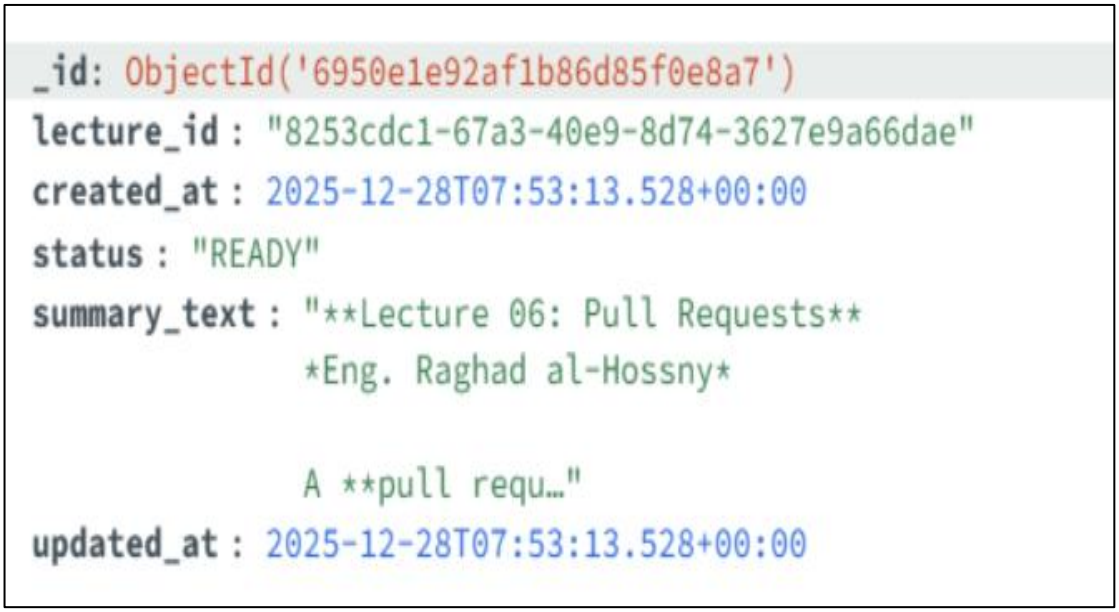


Figure27 MongoDB document structure

SQL Database – MySQL: In our system, MySQL is utilized for storing operational data related to accounts, projects, and requirements. This data is highly structured, making MySQL an ideal choice due to its efficiency in handling defined schemas and transactional integrity.

MySQL- database modeling:

Updating the RTM:

REQ-ID	Use cases	analysis	System design	Detailed design	coding	Test cases
RE-FR-SU-01	UC-01	analysis	System design	Detailed design		TC-01 TC-02
RE-FR-LI-02	UC-02	analysis	System design	Detailed design		TC-03 TC-04
RE-FR-MA-03	UC-03	analysis	System design	Detailed design		TC-05 TC-06 TC-07
RE-FR-GQ-04	UC-04	analysis	System design	Detailed design		TC-08 TC-09 TC-10 TC-11 TC-12 TC-13 TC-14 TC-15
RE-FR-AEA-05	UC-04	analysis	System design	Detailed design		TC-16 TC-17 TC-18
RE-FR-U&PL-06	UC-05	analysis	System design	Detailed design		TC-43 TC-44 TC-45 TC-46
RE-FR-ELI-07	UC-06	analysis	System design	Detailed design		TC-34 TC-35 TC-36
RE-FR-SLC-08	UC-07	analysis	System design	Detailed design		TC-38 TC-39 TC-40
RE-FR-DL-09	UC-08	analysis	System design	Detailed design		TC-37
RE-FR-CC-10	UC-09	analysis	System design	Detailed design		TC-27 TC-28
RE-FR-DC-11	UC-10	analysis	System design	Detailed design		TC-31
RE-FR-ECI-12	UC-11	analysis	System design	Detailed design		TC-29 TC-30
RE-FR-VL-13	UC-12	analysis	System design	Detailed design		TC-32 TC-33
RE-FR-AUF-14	UC-13	analysis	System design	Detailed design		TC-19 TC-20 TC-21 TC-22 TC-23
RE-FR-GSUAI-15	UC-14	analysis	System design	Detailed design		TC-24

RE-FR-ESPDF-16	UC-15	analysis	System design	Detailed design		TC-25 TC-26

Chapter 5 Account Service

1. Overview and Purpose

The Account Management Service is a dedicated microservice responsible for all user-related functionality: user registration, authentication, authorization, profile management, and administrative account actions. It provides a secure, auditable, and scalable foundation for controlling access to the rest of the platform (courses, lecture uploads, AI features). The service issues and validates JWT tokens used by the API Gateway and enforces account-level policies (password rules, email verification, lockout).

Primary goals:

- Provide robust, standards-based authentication and authorization.
- Expose a clear, versioned REST API for frontend and other services.
- Securely store and manage user credentials and profile metadata.
- Offer administrative endpoints for account lifecycle management (suspend, delete).
- Produce testable behavior and comprehensive audit logs.

2. Responsibilities

- **User Registration**: accept sign-up requests, validate input, create user record, send email verification token.
- **Authentication**: verify credentials, issue short-lived access tokens (JWT) and longer-lived refresh tokens.
- **Account Management**: view/update profile, change password, manage email address, delete or deactivate account.
- **Authorization Support**: provide role information (Student, Admin) and simple RBAC checks for downstream services.
- **Security & Auditing**: enforce password policy, log suspicious activity, provide audit trail for account changes.

3. Functional Workflows (textual sequence)

3.1 User Signup (email/password)

1. Frontend collects user details (email, password, full name, optional metadata) and POSTs to `/auth/signup`.
2. Service validates fields and password strength policy.
3. If valid, service creates a new user record in MySQL with status `pending_verification`, hashes password (bcrypt/Argon2) and stores verification token (or issues email link).
4. Service sends email verification message containing a one-time token (or link) to user's email.
5. User clicks verification link; frontend calls `/auth/verify?token=...`. Service marks user as `active`.
6. Service returns 201 Created with user id (no password) or appropriate error.

3.2 Login (authenticate)

1. Frontend POSTs credentials to `/auth/login`.
2. Service verifies credentials, checks account state (active/suspended/locked).
3. On success, service issues:
    - short-lived **access JWT** (e.g., expires in 15 minutes),
    - long-lived **refresh token** (stored securely, e.g., httpOnly cookie or DB).
4. Service returns tokens and minimal profile (id, roles, display name).
5. API Gateway validates access JWT for protected routes; refresh endpoint `/auth/refresh` exchanges valid refresh token for a new access JWT.

3.3 Password Reset

1. User requests reset via `/auth/password-reset/request` with email.
2. If email exists, service creates short-lived reset token and sends email with reset link.
3. User follows link and submits new password to `/auth/password-reset/confirm` with token.
4. Service validates token, updates password (hash), invalidates existing refresh tokens and returns success.

3.4 Profile Management (Manage Account)

1. Authenticated user calls `/users/{id}` GET to view profile.
2. To update, user calls `/users/{id}` PUT/PATCH with allowed fields (display name, avatar URL, preferences).
3. For password change, user calls `/users/{id}/password` with old and new passwords; service validates old password and enforces strength on new password.

4. For account deletion, user calls `/users/{id}` DELETE; depending on policy, service either soft-deletes (flags `deleted_at`) or initiates retention/purge workflow. Admin may force-delete.

3.5 Administrative Actions

- Admin endpoints allow listing users, changing roles, suspending accounts, unlocking locked accounts, and viewing audit logs.

4. API Contract — Example Endpoints

Note: Return codes and request bodies are abbreviated for clarity. Final API should include OpenAPI/Swagger spec.

- `POST /auth/signup`
  Request: `{ "email": "...", "password": "...", "name": "..." }`
  Responses: `201 Created` (user created), `400 Bad Request` (validation), `409 Conflict` (email in use).
- `GET /auth/verify?token=...`
  Responses: `200 OK` (verified), `400/410` (invalid/expired token).
- `POST /auth/login`
  Request: `{ "email": "...", "password": "..." }`
  Responses: `200 OK { "accessToken": "...", "refreshToken": "...", "user": {...} }`, `401 Unauthorized`.
- `POST /auth/refresh`
  Request: `{ "refreshToken": "..." }`
  Response: `200 OK { "accessToken": "...", "refreshToken": "..." }`.
- `POST /auth/password-reset/request`
  Request: `{ "email": "..." }`
  Response: `200 OK` (always return generic message to avoid user enumeration).
- `POST /auth/password-reset/confirm`
  Request: `{ "token": "...", "newPassword": "..." }`
  Response: `200 OK` or `400 Bad Request`.
- `GET /users/{id}`
  Response: `200 OK` with profile (limited to allowed fields to requester).
- `PATCH /users/{id}`
  Request: partial fields (name, avatar, preferences). Response: `200 OK`.
- `DELETE /users/{id}`
  Admin-only or owner-only. Response: `200 OK` or `204 No Content`.

Errors must return structured JSON with code, message, and optional details:

}

" error} :"

" code": "INVALID_PASSWORD,"

" message": "Password must contain at least 8 characters and one number,".

" details[...] :"

{

{

5. Data Model (example MySQL schema)

Table: users

- `id` (UUID, PK)
- `email` (varchar, unique, indexed)
- `password_hash` (varchar)
- `name` (varchar)
- `role` (enum: 'student','admin')
- `status` (enum: 'pending_verification','active','suspended','deleted','locked')
- `created_at`, `updated_at`, `deleted_at` (timestamps)
- `last_login_at` (timestamp)
- `failed_login_attempts` (int)
- `password_changed_at` (timestamp)
- `metadata` (JSON) — optional profile preferences

Table: auth_tokens (refresh tokens / verification / reset tokens)

- `id`, `user_id`, `token_hash`, `type` (refresh/verify/reset), `expires_at`, `created_at`, `revoked` (boolean)

Indexes:

- index on `email` (unique)
- index on `status` for admin queries

Notes:

- Refresh tokens should be stored as hashed values; keep minimal PII.
- Use foreign keys for referential integrity where appropriate.

6. Security Considerations & Policies

- **Password hashing**: use Argon2id or bcrypt with strong parameters. Never store plaintext passwords.
- **Password policy**: minimum 8 characters, at least one digit; optionally require a mixture of upper/lowercase and symbols. Enforce on signup and password-change endpoints.
- **Email verification**: require account activation via email link before granting access to privileged features.
- **JWT usage**: sign tokens with a strong secret/key (or RSA key pair). Access tokens should be short-lived; store refresh tokens securely and allow revocation.
- **Transport**: TLS (HTTPS) for all client-server and inter-service communication; enforce HSTS at the gateway.
- **Rate limiting & brute-force protection**: throttle login attempts per IP and per account; increment `failed_login_attempts` and lock account after threshold (e.g., 5 attempts) with cooldown and admin override.
- **Session invalidation**: On password reset or forced logout, invalidate refresh tokens and possibly revoke active sessions.
- **Input validation & sanitization**: validate email format and escape stored strings to prevent injection. Use parameterized queries/ORM.
- **Least privilege**: admin endpoints protected and audited. Services call the Account service or verify JWT claims for authorization info.
- **Secrets management**: store keys, SMTP credentials, DB passwords in secure secret store (Kubernetes secrets, Vault) — not in source control.

7. Test Cases & Acceptance Criteria

Below is a professional way to present the test cases you already have, plus a recommended structure and mapping to acceptance criteria. Present the tests in tabular or enumerated form in the final document.

Recommended test-case structure (one-line example)

- **TC-01 — Signup valid**
  Preconditions: no existing user with given email.
  Steps: POST `/auth/signup` with valid payload.
  Expected: 201 Created; verification email queued; user record `pending_verification`.
- **TC-02 — Signup duplicate email**
  Steps: POST `/auth/signup` with email that already exists.
  Expected: 409 Conflict.
- **TC-03 — Login success**
  Preconditions: verified active user.
  Steps: POST `/auth/login` with correct credentials.
  Expected: 200 OK; accessToken + refreshToken returned.
- **TC-04 — Login invalid password**
  Steps: POST `/auth/login` with wrong password.
  Expected: 401 Unauthorized; failed_login_attempts incremented.
- **TC-05 — Password reset flow**
  Steps: request reset, confirm reset with token and new valid password.
  Expected: 200 OK; new password accepted; old refresh tokens invalidated.
- **TC-06 — Profile update**
  Steps: PATCH `/users/{id}` with new name and avatar.
  Expected: 200 OK; values updated in DB.
- **TC-07 — Access control**
  Steps: non-admin attempts to call admin-only endpoint.
  Expected: 403 Forbidden.
- **TC-08 — Email verification expired token**
  Steps: use expired token.
  Expected: 400/410 and no activation.

For each test case include: ID, title, preconditions, steps, expected result, postconditions, and links to UI screenshots if relevant.

Mapping to acceptance criteria

- Signup must create user in `pending_verification` state and send verification mail (pass: TC-01).
- After verification, login must return valid JWTs (pass: TC-03).
- Password reset must prevent reuse of invalidated refresh tokens (pass: TC-05).
- Account lockout should trigger after N failed attempts (configured value) and require admin/unlock or cooldown (testable by repeated TC-04).
- Admin endpoints must be inaccessible to non-admins (pass: TC-07

8. Logging, Monitoring & Auditing

- **Audit logs**: record actions such as `signup`, `email_verified`, `login_success`, `login_failed`, `password_reset_request`, `password_reset_confirm`, `profile_update`, `admin_suspend`. Include `user_id`, `timestamp`, `actor_ip`, `user_agent`, and `request_id`.
- **Security logging**: log suspicious patterns (multiple failed logins, many password resets) and trigger alerts.
- **Metrics**: number of signups/day, login success/failure rates, average time to verify email, number of locked accounts. Export metrics to Prometheus.
- **Tracing**: correlate requests via `X-Request-ID` propagated from API Gateway; instrument for distributed tracing (OpenTelemetry).
- **Retention**: keep audit logs for at least 90 days (configurable).

9. Deployment & Operational Notes

- Package the service as a container (Docker). Build pipeline should run unit tests and static analysis, then publish images to registry (GitHub Container Registry / Docker Hub).
- Environment variables (DB connection strings, JWT signing keys, SMTP config) must be injected at runtime and retrieved from a secrets manager.
- Database migrations managed by a migration tool (Django migrations or Flyway) and run during deployment.
- Scale horizontally: the service should be stateless (session state in tokens or DB), so replicas behind the API Gateway/load balancer are possible.
- Backups: schedule daily backups for MySQL; ensure restore procedures are tested.
- Health checks: implement liveness and readiness endpoints for orchestration (Kubernetes).

10. Risks & Mitigations

- **Risk**: Email delivery failures block signup verification.
  Mitigation: implement retry queue, alternative delivery logs, and allow resending verification. Provide helpful UI messaging.
- **Risk**: Token theft (refresh/access).
  Mitigation: short-lived access tokens, hashed refresh tokens, detect reuse, revoke on suspicious activity, encourage secure cookie usage for refresh tokens.
- **Risk**: Brute-force attacks.
  Mitigation: rate-limiting, account lockout, CAPTCHA after repeated failures.
- **Risk**: Data loss during migration.
  Mitigation: test migrations in staging, backup before migration.

11. Test cases

Test Case Scenario:		**Case 1: Signup**	
Test case id	Test case title	Test steps	Expected result
TC-01	Signup with valid data	1) Navigate to Signup page. 2) Enter valid username. 3) Enter valid email. 4) Enter strong password. 5) Click Sign Up.	Account is created successfully.
TC-02	Signup with email without @	1) Navigate to Signup page. 2) Enter username. 3) Enter email without @. 4) Enter valid password. 5) Click Sign Up.	Invalid email format error is displayed.

Test Case Scenario:		**Case 2: Login**	
Test case id	Test case title	Test steps	Expected result
TC-03	Login with valid credentials	1) Open system login page. 2) Enter valid email. 3) Enter valid password. 4) Click Login.	User is logged in successfully.
TC-04	Login with invalid email or password	1) Open system login page. 2) Enter invalid email or enter invalid password. 4) Click Login.	Error message is displayed.

Test Case Scenario:		**Case 3: Manage Account**	
Test case id	Test case title	Test steps	Expected result
TC-05	Update account information	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Click Save.	Account information updated successfully.
TC-06	Edit user data while leaving blank fields	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Leave fields blank) Click Save.5	Error message is displayed.
TC-07	Edit user data with an incorrect password entered	1) Login as student. 2) Navigate to Manage Account. 3) Edit profile information. 4) Entering a password that does not match) Click Save.5	Error message is displayed.

12. Service screenshots

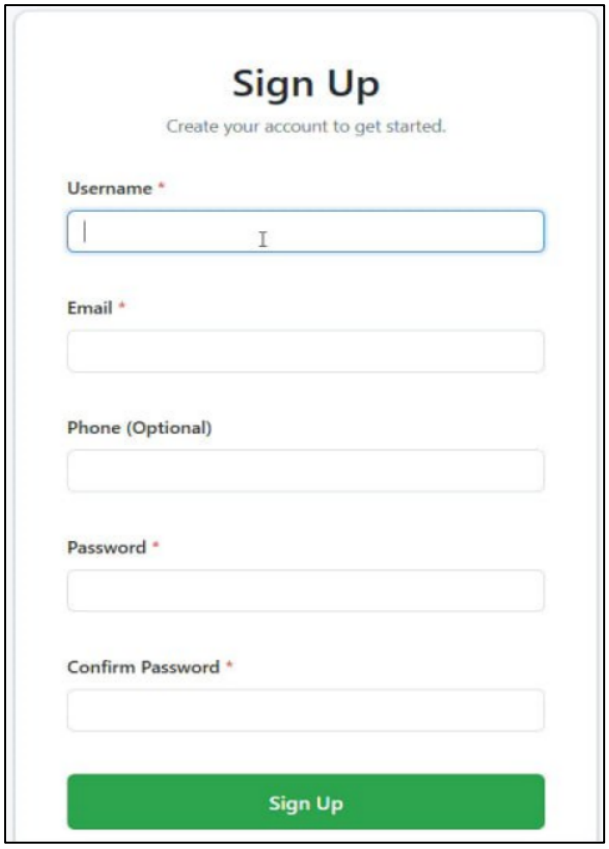


Figure28 Signup screenshot

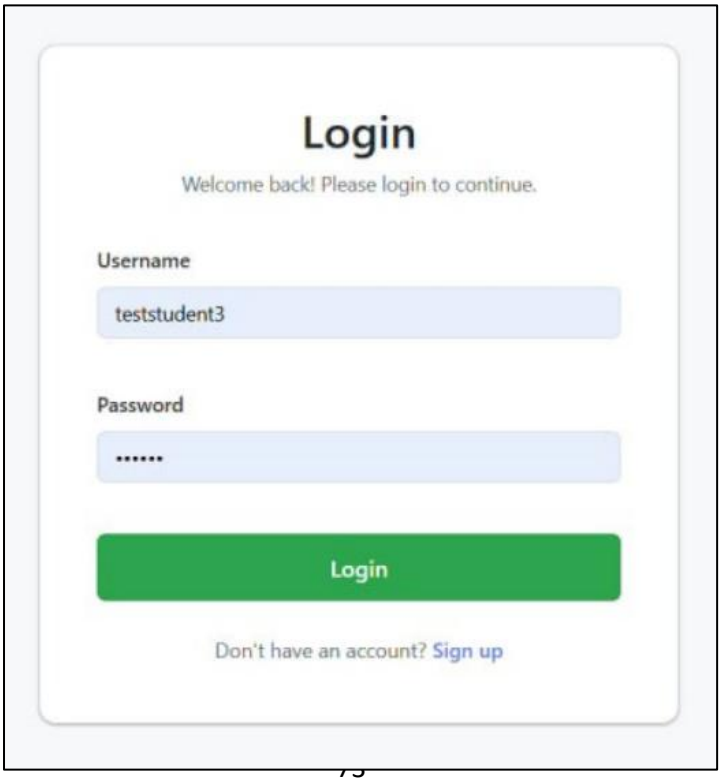


Figure29 Login screenshot

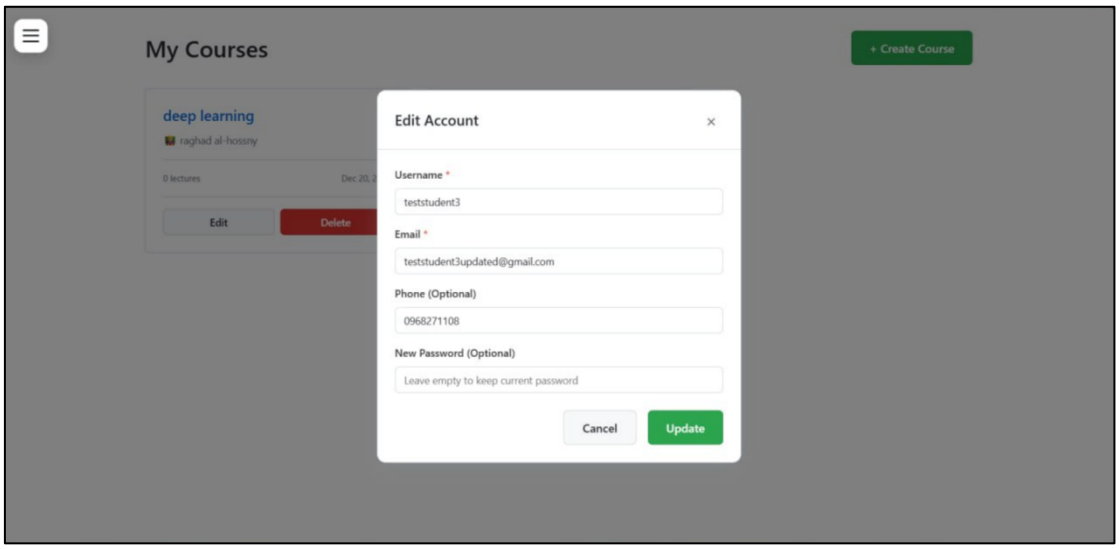


Figure30 Edit account screenshot

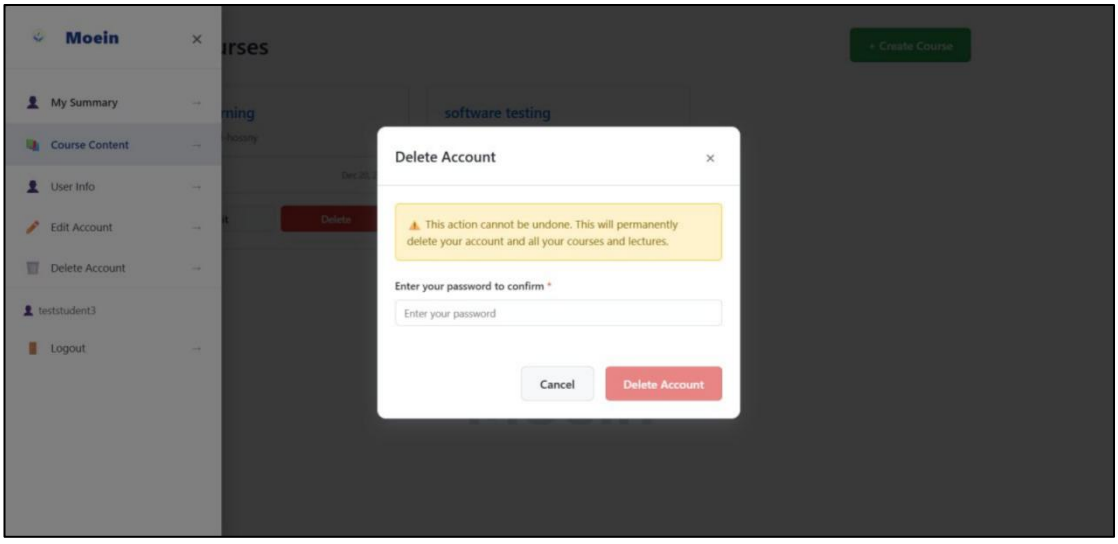


Figure31 Delete account screenshot

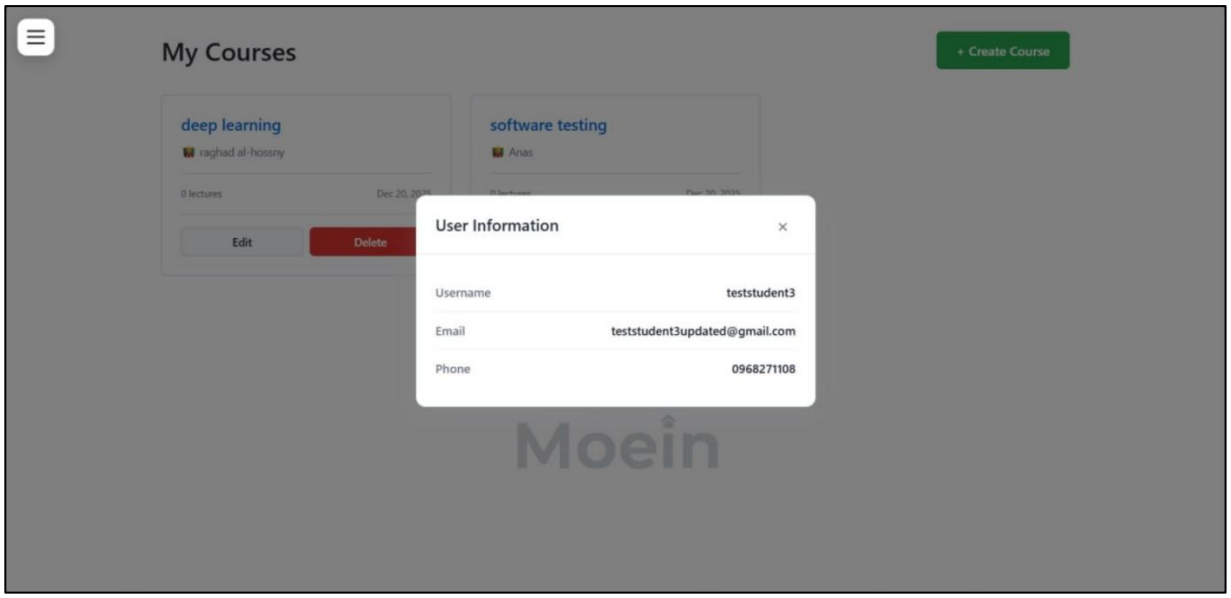


Figure32 user info screenshot

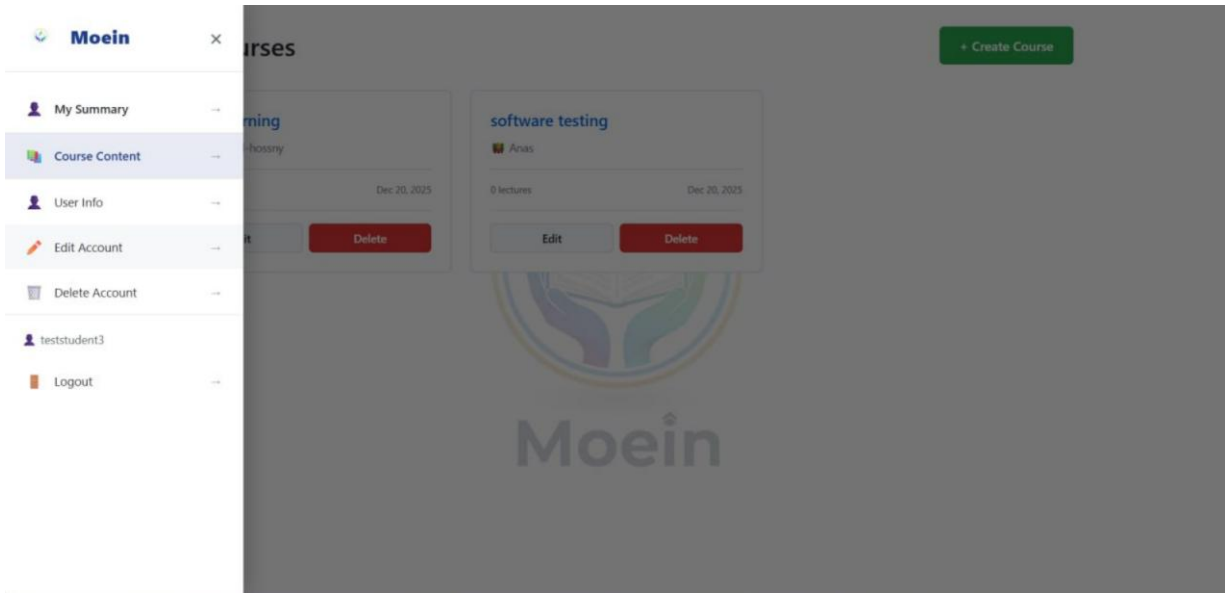


Figure33 side bar screenshot

Chapter 6 Course and Lecture Service

1. Overview and Purpose

The Course & Lecture Service is a domain-specific microservice responsible for managing course containers and lecture artifacts owned by students. Each student has a private workspace (their page) where they can create, update, and delete courses and the lectures inside them. The service handles metadata lifecycle, file ingestion orchestration (for PDF/DOCX/PPTX), storage references, job creation for downstream parsing and AI processing, and basic search/indexing hooks for lecture content.

Primary goals:

Provide robust CRUD operations for courses and lectures scoped to the authenticated student.

Orchestrate safe, efficient upload and storage of lecture files and their metadata.

Integrate with asynchronous processing pipelines (parsing & summarization) via a message bus or job queue.

Maintain consistency between relational metadata (owner, course membership) and document artifacts (extracted text, summaries).

Support operational concerns: soft-delete/restore, versioning of lecture metadata, auditability, and user-facing status reporting for background jobs.

2. Responsibilities

Course lifecycle management: create, read, update, delete course records (title, code, description, visibility, tags, created/updated timestamps).

Lecture lifecycle management: create, read, update, delete lecture records associated with a course; manage lecture metadata (title, description, tags, module), file references, and processing status.

File ingestion orchestration: accept uploads (multipart/form-data), validate type & size, store file to object storage, produce parsing job message.

Job status management: record and expose processing status (Pending → In Progress → Completed → Failed → Partial) and store job identifiers for client polling.

Integration: publish events/messages for other services (Summarization Engine, Quiz Generator, Notification Service) and consume callbacks/status updates.

Access control: enforce that only the owning student (or Admin when appropriate) can modify or delete their courses/lectures.

Soft-delete & retention: implement non-destructive deletion with optional purge policies.

Search hooks: trigger indexing after parsing/summary creation to enable search capabilities.

3. Functional Workflows

3.1 Create Course (main flow)

Authenticated student navigates to their page and requests course creation with metadata (title, description, tags).

Service validates input and enforces per-student constraints (e.g., max courses).

Service creates a new course record in MySQL and returns 201 Created with course id and metadata.

Frontend displays the new course and links for adding lectures.

Alternative flows / notes

Validation failures return 400 Bad Request with helpful messages.

If the student reaches a configured limit, return 403 Forbidden with reason.

3.2 Update Course

Student opens the course edit form and submits changes.

Service checks ownership, validates changes, updates the database, and returns 200 OK.

Changes are versioned/audited (who/when).

Alternative

If non-owner attempts update → 403 Forbidden.

3.3 Delete Course

Student selects delete and confirms.

Service performs soft-delete: sets deleted_at and status = deleted and flags contained lectures as orphaned/deleted.

Service enqueues cleanup tasks (optional) for artifact purge according to retention policy.

UI reflects removal; Admin can view and permanently purge or restore during retention window.

Alternative

If a course contains protected data or is subject to retention policy, mark as PendingDeletion and schedule purge.

3.4 Add Lecture (upload + create lecture metadata)

Student from course page clicks Add Lecture and provides metadata and selects a file (PDF, DOCX, PPTX).
Service validates file type and size (reject unsupported or too-large files).

Service stores file in object storage (S3-compatible) and creates a lecture record referencing the file location with status PendingParsing.

Service publishes a parse_job message to the message bus including lecture_id and file location.

Service returns 202 Accepted with job id and initial status to the frontend.

Notes:

The lecture record contains processing_status, job_id, uploaded_by, and timestamps.

The frontend polls job status or receives notification when parsing completes.

3.5 Update Lecture Info

Student edits lecture metadata (title, description, tags) and submits.

Service validates ownership and updates the lecture record; if user requests file replacement, treat as a new upload (store new file, enqueue parse job, optionally keep prior versions).

If a new file is uploaded, previous AI artifacts (summaries, MCQs) are flagged as stale and either invalidated or versioned.

3.6 Delete Lecture

Student requests delete; service prompts confirmation.

On confirm, perform soft-delete: flag lecture record deleted_at, mark artifacts stale, remove from course listing.

Publish artifact-cleanup job to purge from object storage and MongoDB per retention policy or leave for admin purge.

Alternative

Immediate hard-delete permitted only to Admin or after retention window.

4. Interaction with AI/Processing Pipeline

After file ingestion, the Course Service publishes a parse job to the message bus (RabbitMQ/Kafka). Job payload includes: lecture_id, file_url, uploader_id, course_id, and optional processing options (OCR flag).

Summarization Engine consumes the job, performs extraction & summarization, stores results in MongoDB, and sends a summary_ready callback/event with summary id and status.

Course Service listens for these events and updates the lecture record (attach summary_id, update status to Completed and set summary_ready_at).

When summary_ready, Course Service triggers indexing for search and notifies the user via Notification Service.

Design note: keep the Course Service stateless with respect to heavy processing; it should only maintain job metadata and state transitions.

5. API Examples (concise)

These examples are illustrative; do not constitute a full OpenAPI spec.

POST /courses
Body: { "title": "Algorithms 101", "description": "...", "tags": ["algo","cs"] }

Response: 201 Created { "id": "c-123", "owner": "u-456", ... }

PATCH /courses/{courseId}
Body: { "title": "Updated Title" }
Response: 200 OK

DELETE /courses/{courseId}
Response: 204 No Content (soft-delete initiated)

POST /courses/{courseId}/lectures (multipart file + metadata)
Form-data fields: title, description, file (pdf/docx/pptx)
Response: 202 Accepted { "lectureId": "l-789", "jobId": "job-001", "status": "PendingParsing" }

GET /courses/{courseId}/lectures/{lectureId}
Response includes lecture metadata, file URL, processing status, and links for summary/quiz if available.

PATCH /courses/{courseId}/lectures/{lectureId} (edit metadata / replace file)
Response: 200 OK

DELETE /courses/{courseId}/lectures/{lectureId}
Response: 204 No Content (soft-delete)

All endpoints require a valid access JWT and must enforce owner-only mutations.

6. Data Model Sketch (compact)

Course (relation / MySQL)

id (UUID)

owner_id (FK → users.id)

title, description, tags (JSON/array)

created_at, updated_at, deleted_at, status

Lecture (relation / MySQL)

id (UUID)

course_id (FK → courses.id)

uploader_id (FK → users.id)

title, description, module, tags

file_url (object storage pointer), file_size, file_type

processing_status (Pending, InProgress, Completed, Failed, Partial)

parse_job_id, summary_id (if available), created_at, updated_at, deleted_at

AI Artifacts (MongoDB document)

lecture_id, extracted_text, summary, embeddings, generated_quiz_ids, version, created_at

Notes:

Keep relational data (ownership, course structure) in MySQL and large/unstructured AI artifacts in MongoDB.

Consider a version field on lecture or artifact to manage reprocessing.

7. Security & Access Control

Authentication: requests must include a valid JWT signed by the Account Service. Gateway validates signature and passes claims.

Authorization: for all mutating endpoints, verify request.user_id ==

course.owner_id (or Admin override).

Upload validation: reject any file that is not application/pdf,

application/vnd.openxmlformats-officedocument.wordprocessingml.document, or PowerPoint MIME types; enforce maximum size (configurable, e.g., 50 MB.(

Input sanitization: sanitize titles, descriptions and tags to prevent XSS when rendering in the frontend.

Least privilege: the service only returns file URLs that are time-limited (presigned) or behind authenticated proxy endpoints to prevent direct public object store access.

8. Test Cases & Acceptance Criteria Mapping

Present your existing test cases in a structured manner and tie them to acceptance criteria. Below is a recommended mapping and example test cases (ID format to match your QA artifacts). Each TC should include steps, expected results and postconditions.

Example test cases

TC-C-01 — Create Course (happy path(

Preconditions: authenticated student, under course quota.

Steps: POST /courses with valid metadata.

Expected: 201 Created; course appears in the student's page; DB record exists with owner_id matching user.

Acceptance: UI shows new course within 2s of response.

TC-C-02 — Create Course (validation error(

Steps: POST /courses with empty title.

Expected: 400 Bad Request with error message. Acceptance: error shown on UI.

TC-L-01 — Add Lecture (upload valid file(

Preconditions: student owns course.

Steps: POST /courses/{id}/lectures with supported file and metadata.

Expected: 202 Accepted; lecture record created with processing_status = PendingParsing; jobId returned; file exists in object storage.

Acceptance: frontend receives job id and shows pending status.

TC-L-02 — Parse job completes (integration(

Steps: Wait for background job to complete or simulate summary_ready event.

Expected: lecture processing_status updated to Completed, summary_id attached; user is notified.

Acceptance: summary becomes available from lecture GET.

TC-L-03 — Update Lecture metadata

Steps: PATCH lecture title/description.

Expected: 200 OK; values updated in DB; audit log entry created.

Acceptance: UI reflects change.

TC-L-04 — Replace Lecture file

Steps: PATCH with new file upload.

Expected: new file stored, new parse_job created, old AI artifacts flagged stale.

Acceptance: previous summary is marked stale and new summary appears after processing.

TC-L-05 — Delete Lecture (soft-delete(

Steps: DELETE lecture.

Expected: 204 No Content; deleted_at set in DB; lecture hidden from default lists; artifacts flagged.

Acceptance: student cannot see lecture in normal listing; Admin can restore/purge.

TC-C-03 — Delete Course with lectures

Steps: DELETE course that has lectures.

Expected: course soft-deleted; associated lectures soft-deleted; retention/purge job scheduled.

Acceptance: all course content hidden from student view, retention policy applied.

Each test case should be automated where possible (unit + integration tests for endpoints; end-to-end tests for upload → parse → notify flow). Use fixture data in a staging environment to simulate background workers or use test doubles for the summarization engine.

9. Logging, Monitoring & Auditing

Action logs: log course_created, course_updated, course_deleted, lecture_uploaded, lecture_updated, lecture_deleted, parse_job_created, parse_job_completed with user_id, course_id, lecture_id, ip, request_id.

Job telemetry: track queue times, processing durations, failure rates; expose metrics to Prometheus (e.g., lecture_uploads_total, parse_job_duration_seconds.(

Storage metrics: monitor object storage usage, file size distribution, and total storage per user.

Alerts: alert on high parse failure rate, queue backlog growth, or storage quota exhaustion.

10. Deployment & Operational Notes

Package as a stateless container (Docker). Use environment variables for DB connections, object store credentials, and queue configuration.

Use presigned URLs or signed proxy endpoints for secure file upload/download.

Design the service to scale horizontally; keep file uploads in object-store and metadata in MySQL for straightforward scaling.

Implement migration scripts for schema changes and a migration/rollout strategy for production.

Provide health endpoints (/health, /ready) and graceful shutdown handlers (drain in-flight jobs).

11. Risks & Mitigations

Risk: Large file uploads can overwhelm storage or cause timeouts.

Mitigation: enforce file size limits, use resumable/chunked uploads, validate client-side before upload.

Risk: Background processing backlog delays availability of summaries.

Mitigation: autoscale worker pool, provide queue metrics & user-facing ETA, implement priority for small/interactive jobs.

Risk: Inconsistent state between course metadata and AI artifacts after reprocessing or failure.

Mitigation: use transactional updates for status changes where possible; store artifact version and mark artifacts stale on new uploads; implement compensating transactions.

Risk: Unauthorized modifications (broken authorization).

Mitigation: centralize ownership checks, enforce claims in Gateway + service, add integration tests for access-control rules.

12. Test cases

Test Case Scenario:		**Case 9: Create Course**	
Test case id	Test case title	Test steps	Expected result
TC-27	Create Course with Valid Data.	1) The student selects Create Course. 2) The system displays the course creation form. 3) The student enters valid course information. 4) The student submits the form.	The course is created successfully and added to the course list.
TC-28	Create Course with Missing Required Fields.	1) The student selects “Create Course. 2) The student leaves one or more required fields empty. 3) The student submits the form.	The system displays an error message indicating missing or invalid data.

Test Case Scenario:		**Case 10: Edit Course Info**	
Test case id	Test case title	Test steps	Expected result
TC-29	Edit Course with Valid Information.	1) The student selects an existing course. 2) The student chooses the edit option. 3) The student updates the course information with valid data. 4) The student submits the changes.	The course information is successfully updated and saved.
TC-30	Edit Course with Invalid Data.	1) The student selects a course to edit. 2) The student enters invalid or incomplete course data. 3) The student submits the changes.	The system rejects the update and displays an error message.

Test Case Scenario:		**Case 11: Delete Course**	
Test case id	Test case title	Test steps	Expected result
TC-31	Delete Course with Confirmation.	1) The student selects the course to delete. 2) The system displays a deletion confirmation message. 3) The student confirms the deletion.	The course and all related data are permanently deleted.

Test Case Scenario:		**Case 12: View Lecture**	
Test case id	Test case title	Test steps	Expected result
TC-32	View Existing Lecture.	1) The student selects a course. 2) The system displays the list of available lectures. 3) The student selects a lecture.	The selected lecture content is displayed successfully.
TC-33	View Lecture When No Lectures Exist.	1) The student selects a course.	The system displays a message indicating that no lectures are available.

Test Case Scenario:		**Case 13: Edit lecture info**	
Test case id	Test case title	Test steps	Expected result
TC-34	Edit lecture name successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Change the lecture name. 6) Click Save.	Lecture name is updated successfully and displayed with the new name.
TC-35	Edit lecture name with empty value.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Clear the lecture name field. 6) Click Save.	System shows validation error and lecture name is not updated.

TC-36	Edit lecture name with special characters.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Edit. 5) Enter a name with special characters (e.g. @#lecture!) 6) Click Save.	System shows validation error and lecture name is not updated.

Test Case Scenario:		**Case 14: Delete lecture**	
Test case id	Test case title	Test steps	Expected result
TC-37	Delete lecture successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Select a lecture. 4) Click Delete. 5) Confirm deletion.	Lecture is deleted successfully and removed from the lectures list.

Test Case Scenario:		**Case 15: Search lecture content**	
Test case id	Test case title	Test steps	Expected result
TC-38	Search lecture content successfully.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Enter a valid keyword in the search field. 5) Click Search.	Matching content inside the lecture is highlighted or displayed.

TC-39	Search with keyword not found in lecture.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Enter a keyword not present in the lecture. 5) Click Search**.**	System displays "No results found in this lecture".
TC-40	Search with empty input inside lecture.	1) Login as student. 2) Navigate to the lectures list. 3) Open a lecture. 4) Leave search field empty. 5) Click Search**.**	System shows validation message asking for a keyword.

Test Case Scenario:		**Case 16: Upload Lecture**	
Test case id	Test case title	Test steps	Expected result
TC-41	Upload Lecture with Supported File.	1) The student selects the upload lecture option. 2) The student chooses a supported lecture file. 3) The student submits the file.	The lecture file is uploaded successfully and sent for parsing.
TC-42	Upload Lecture with Unsupported File Type.	1) The student selects an unsupported file format. 2) The student submits the file.	The system rejects the file and displays an error message.

Test Case Scenario:		**Case 17: Parse Lecture File**	
Test case id	Test case title	Test steps	Expected result
TC-43	Parse Lecture File (PDF).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as PDF. 3) The system extracts text content from the PDF file.	The lecture content is successfully extracted as text.
TC-44	Parse Lecture File (DOCX).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as DOCX. 3) The system extracts text content from the DOCX file.	The lecture content is successfully extracted as text.
TC-45	Parse Lecture File (PPTX).	1) The system detects the uploaded lecture file. 2) The system identifies the file type as PPTX. 3) The system extracts text content from presentation slides.	The lecture content is successfully extracted as text.
TC-46	Parse Lecture File with Unsupported Format.	1) The system detects the uploaded file. 2) The system identifies an unsupported file format.	The system rejects the file and displays an error message.

13. Service screenshots

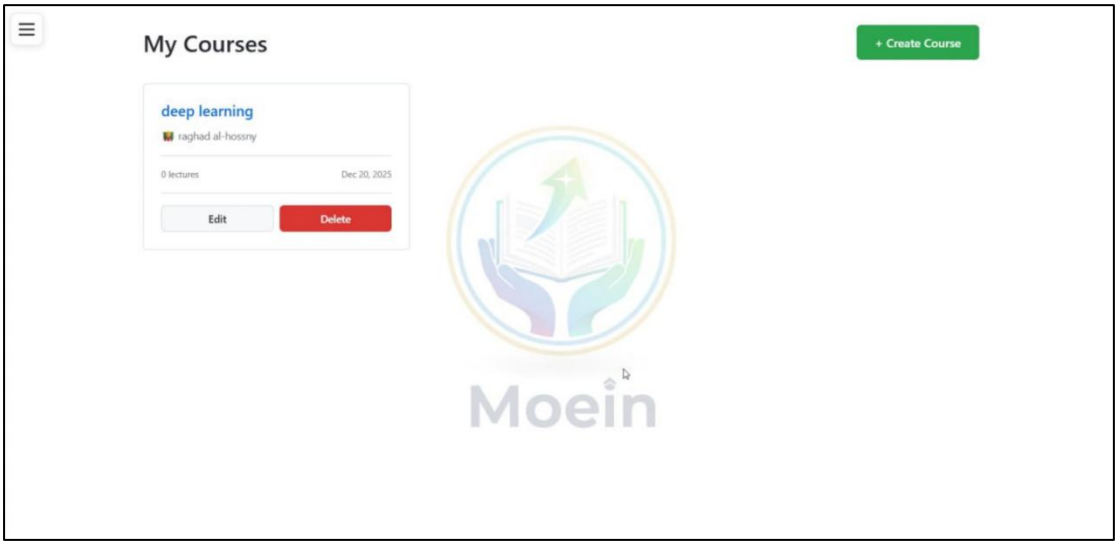


Figure34 Courses list screenshot

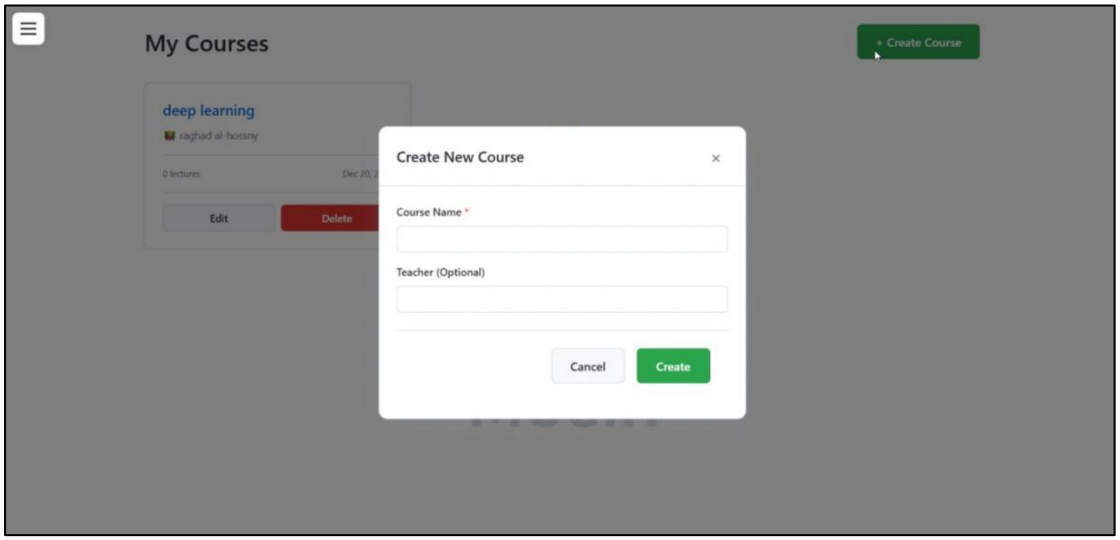


Figure35 Create course screenshot

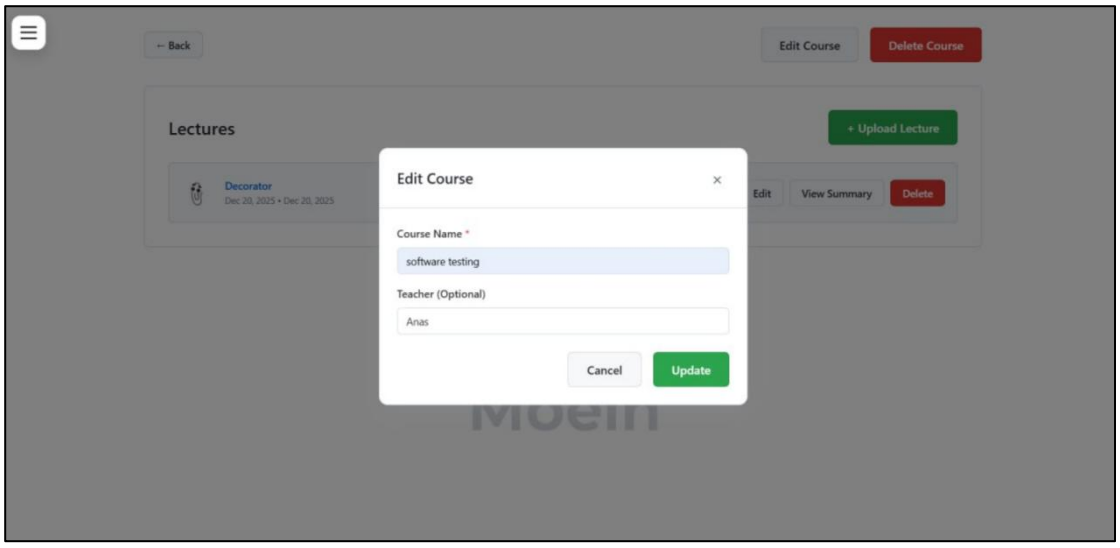


Figure36 Edit course info screenshot

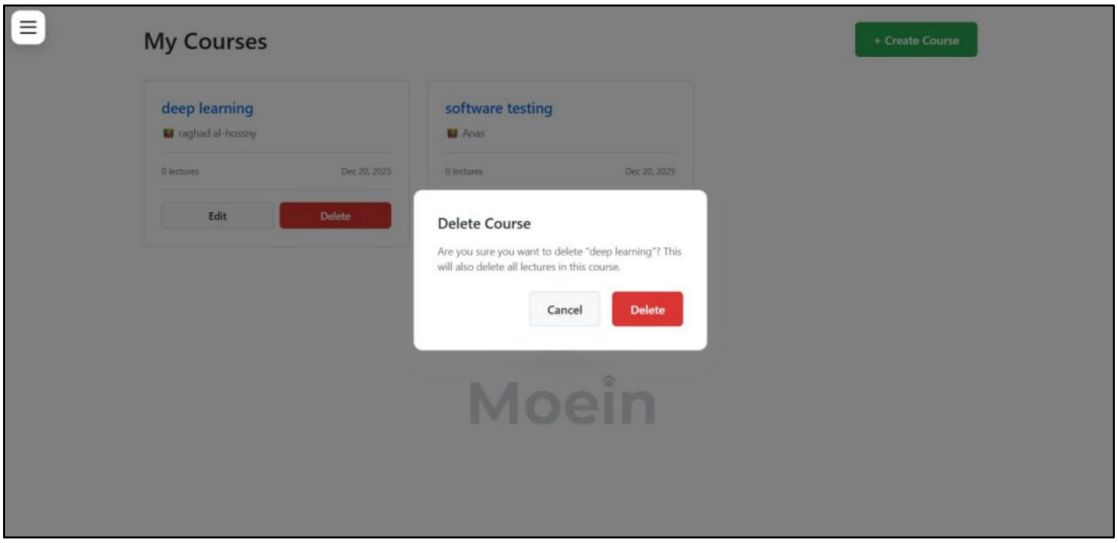


Figure37 Delete course screenshot

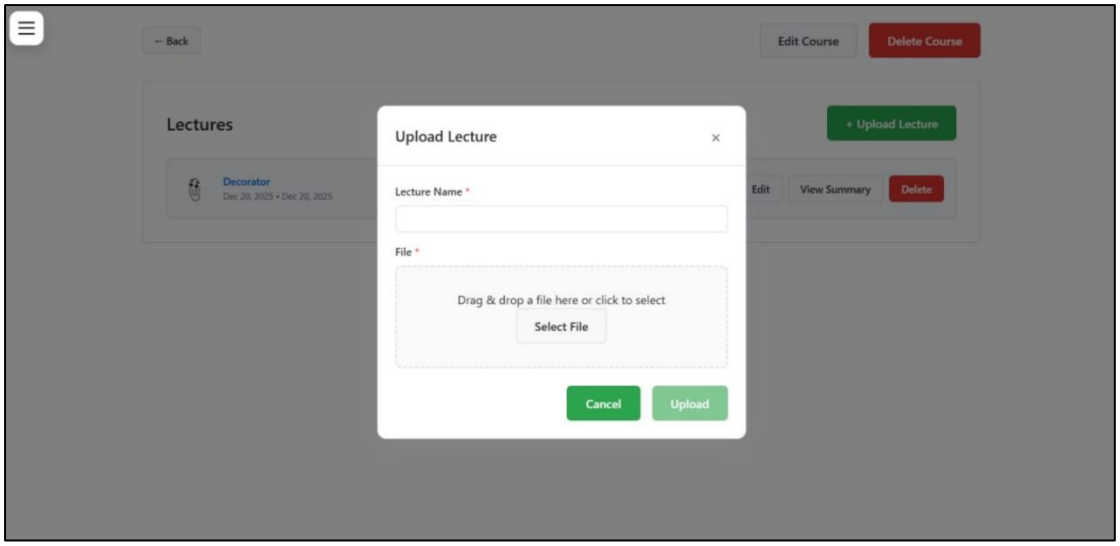


Figure38 upload lecture screenshot

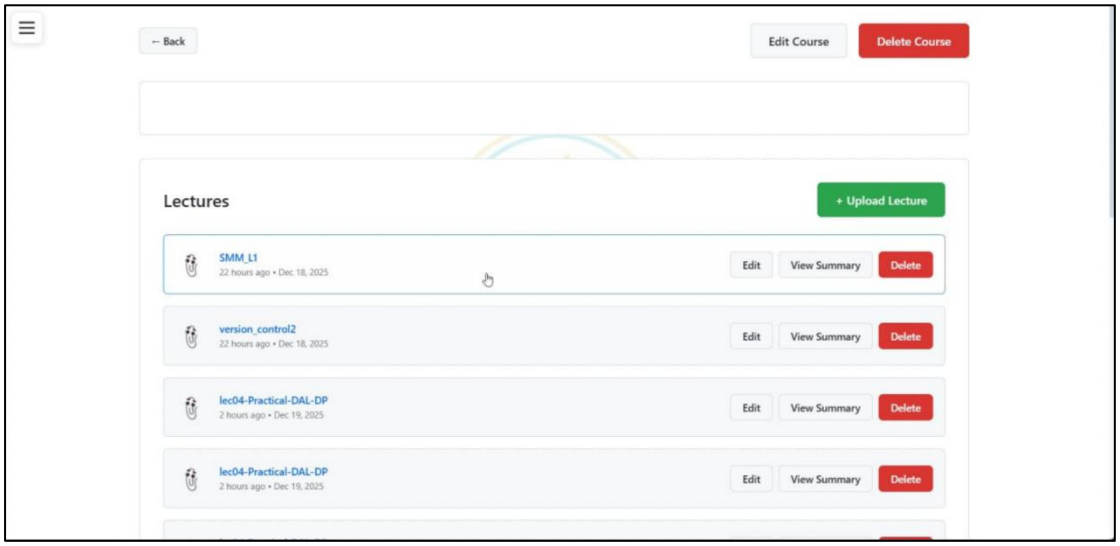


Figure39 Lectures list screenshot

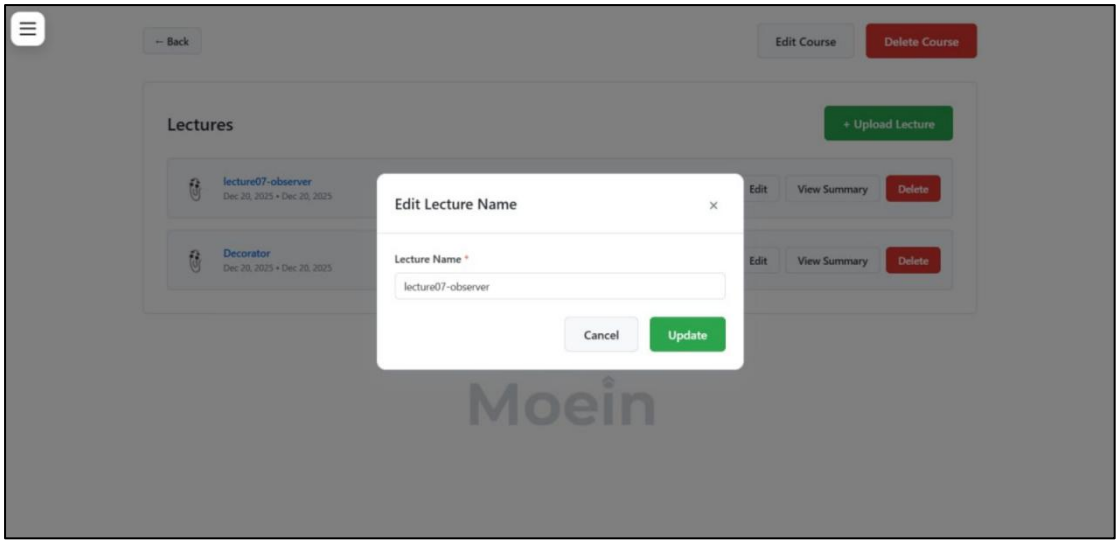


Figure40 Edit lecture name screenshot

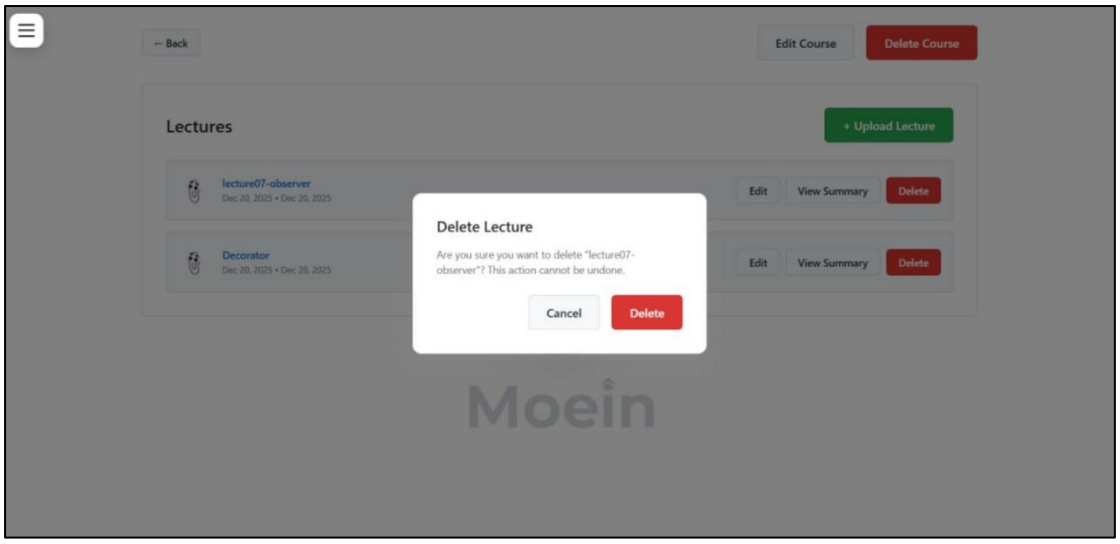


Figure41 Delete lecture screenshot

Chapter 7 AI Service

1. Introduction

Artificial Intelligence (AI) is increasingly transforming the education sector by automating content processing, enhancing learning efficiency, and supporting students in managing large volumes of academic material. University lectures often contain dense and lengthy information, making it difficult for students to manually extract key concepts and essential points in a limited time.

The AI Processing Service in the proposed Student Course Management and AI Summarization System addresses this challenge by automatically generating structured summaries of uploaded lecture materials. The service leverages AI-based content analysis and summarization techniques to extract main topics, key points, and important concepts from lecture content.

Implemented as an independent microservice, the AI Processing Service operates asynchronously and is triggered after a lecture is uploaded. This design ensures scalability and prevents blocking other system components. By automating the summarization process, the service significantly reduces students' study time and improves content comprehension, thereby enhancing the overall learning experience.

- Key Functionalities of AI Service:

The AI service processes uploaded lecture content through a set of core components designed to automatically generate meaningful academic summaries. These components work together to transform raw lecture material into structured and easy-to-understand outputs.

- **Content Processing and Text Extraction:**
  The service analyzes uploaded lecture files or recordings and extracts textual content in a machine-readable format suitable for further analysis.

- **Summarization:**
  The AI service generates concise and well-structured summaries by identifying the main topics and important sections within the lecture content. This enables students to quickly review essential information without revisiting the entire lecture.

- **Key Concepts Extraction:**
  The service identifies essential concepts, definitions, and important points

discussed during the lecture, helping students focus on critical academic material and improve knowledge retention.

In this chapter, the key tools and technologies utilized in the AI Processing Service are presented. In addition, relevant literature related to AI-based content summarization is reviewed, followed by a detailed discussion of the design and testing phases of the AI Processing Service within the proposed system.

2. Basic Concepts and literature review:

2.1 Introduction:

This section provides an overview of the key concepts, technologies, and tools that are essential for the AI Processing Service. It also includes a review of relevant literature on similar models. By examining these foundational elements, the chapter highlights the significance of the proposed service and demonstrates how it enhances existing solutions in lecture summarization and content processing.

2.2 Basic concepts:

- AI pre-trained models:

AI pre-trained models have revolutionized the way artificial intelligence is developed and deployed across various industries. These models are trained on massive datasets, enabling them to understand patterns, recognize structures, and generate intelligent responses without the need for training from scratch. Instead of spending significant time, computational resources, and large amounts of labeled data to develop models from the ground up, pre-trained models offer a ready-to-use foundation that can be fine-tuned for specific tasks. Leveraging pre-trained models improves efficiency, accuracy, and scalability while significantly reducing the cost of AI development. This approach allows developers to focus on customization and optimization rather than starting with raw data.

- AI summarization:

AI-Based Summarization in this system leverages the Qwen/Qwen3-30B-A3B-Instruct-2507 model to automatically generate concise and meaningful summaries of lecture content. This advanced large language model is specifically designed for instruction-following tasks, making it well-suited for educational applications where structured, coherent summaries are required.

2.3 Literature review:

This section provides a comprehensive analysis of pre-trained AI models and existing text-to-text solutions, focusing on their applicability to the core functionalities of the

AI Processing Service. Various summarization models are examined, with an evaluation of their strengths, limitations, and alignment with the specific requirements of the lecture summarization system. The analysis highlights the rationale behind selecting Qwen/Qwen3-30B-A3B-Instruct-2507 for generating structured and coherent text summaries, ensuring high-quality output tailored for educational use.

- text to text models:

For text-to-text componet, several models from different families will be compared.

openai/gpt-oss-120b:groq:

This is a large-scale language model with approximately 120 billion parameters. It is designed for general-purpose tasks, including text generation, reasoning, and complex instruction following. The “groq” variant is optimized for high-performance hardware to speed up inference. It is highly flexible and powerful but requires substantial computational resources, making it more suitable for large-scale industrial applications rather than light academic projects.

google/gemma-2-2b-it:nebius:

This is a lightweight language model with around 2 billion parameters. It is part of the Gemma/Gemini series and is primarily designed for instruction-following tasks. Its small size makes it fast and efficient on limited hardware, but it may struggle with understanding long or complex academic texts. It is more suitable for simple text generation or short-form tasks rather than processing lengthy lecture notes.

deepseek-ai/DeepSeek-R1:novita:

DeepSeek-R1 is a language model designed with a focus on reasoning and analytical tasks. It excels at complex problem-solving, logical inference, and structured analysis. While it performs well on tasks requiring deep understanding and reasoning, it may be slower and less efficient when handling large volumes of text for summarization purposes. It is best suited for projects that require critical thinking or technical analysis rather than casual text summarization.

Qwen/Qwen3-Coder-480B-A35B-Instruct:novita

This is an extremely large language model with around 480 billion parameters, specifically optimized for coding and agentic tasks. It excels in code generation, debugging, and reasoning in technical workflows. While powerful, it is not primarily

designed for summarizing academic texts and may be excessive (“overkill”) for projects focused on lecture summarization. Its computational requirements are very high, making it suitable mainly for specialized industrial or research use.

zai-org/GLM-4.5:novita

GLM-4.5 is a very large Mixture-of-Experts (MoE) language model, with approximately 355 billion parameters in total, and around 32 billion active parameters used per inference. It is designed for general-purpose reasoning, text generation, and complex tasks requiring deep understanding. It supports very long contexts, making it suitable for processing extensive documents. However, it requires substantial computational resources, so it may not be practical for small academic projects.

Qwen/Qwen3-4B-Thinking-2507:nscale

Qwen3-4B-Thinking is a lightweight model with approximately 4 billion parameters, designed with enhanced reasoning capabilities. It is suitable for tasks that require logical thinking or basic analysis. Its small size makes it fast and efficient, but it may lack the depth needed for summarizing long and complex academic lectures. It is better suited for short texts or preliminary summarization.

Qwen/Qwen2.5-7B-Instruct-1M:featherless-a

This is a medium-sized general-purpose instruct model with approximately 7 billion parameters. It is designed to follow instructions for text generation and simple analytical tasks. While it is more capable than smaller models, it may still struggle with long academic texts and complex lecture materials. It is suitable for moderate summarization tasks but may not capture all intricate details in lengthy documents.

Qwen/Qwen2.5-Coder-32B-Instruct:nscale

Qwen2.5-Coder-32B is a large model with approximately 32 billion parameters, specifically optimized for coding tasks. It excels at code generation, debugging, and technical reasoning. While it can process text, it is not primarily designed for summarizing academic lectures, and using it for this purpose may be inefficient. It is best suited for programming-related projects rather than general text summarization.

Qwen/Qwen3-30B-A3B-Instruct-2507:nebius

Qwen3-30B-A3B-Instruct-2507 is a large Mixture-of-Experts (MoE) model with a total of approximately 30 billion parameters and around 3.3 billion active parameters per inference. It is specifically optimized for general-purpose instruction-following tasks, text summarization, and reasoning. It supports long contexts, making it highly suitable for processing and summarizing lengthy academic lectures. It offers a balanced combination of performance and computational efficiency, making it the most practical choice for lecture summarization projects.

Suitability for Lecture Summarization Project	Weaknesses	Strength	Model Type	Size	Model Name
Not suitable	Weak understanding of long texts	Fast, low resource usage	Instruct (Lightweight)	2B	google/gemma-2-2b-it
Limited	Limited summarization quality	Good logical reasoning, lightweight	Reasoning / Thinking	4B	Qwen/Qwen3-4B-Thinking-2507
Acceptable	Not sufficient for long academic lectures	Stable, moderate performance	General Instruct	7B	Qwen/Qwen2.5-7B-Instruct-1M
Not suitable	Not designed for academic text summarization	Strong in code and technical reasoning	Coder (Programming)	32B	Qwen/Qwen2.5-Coder-32B-Instruct
Acceptable with limitations	Slower, not optimized for summarization	Deep analytical and reasoning abilities	Reasoning-focused	Large	deepseek-ai/DeepSeek-R1

Overkill	High cost, impractical for this task	Very strong overall understanding	General / Agentic	120B	openai/gpt-oss-120b
Overkill	Extremely resource-intensive	Excellent accuracy, deep comprehension	General / Reasoning	355B	zai-org/GLM-4.5
Not suitable	Not suitable for text summarization	Outstanding for coding and agent workflows	Coder	480B	Qwen/Qwen3-Coder-480B-A35B
Best choice	Requires moderate resources	Excellent balance, strong summarization, long context	General Instruct	30B	Qwen/Qwen3-30B-A3B-Instruct-2507

3. Service Design and implementation:

3.1 Introduction:

In this section, we present the design of the AI Processing Service and the architecture it follows to efficiently handle text-based lecture content. The service is built to transform raw lecture texts into structured outputs, including concise summaries and key concepts, through a scalable and modular pipeline. It leverages cutting-edge tools and models, such as **Qwen/Qwen3-30B-A3B-Instruct-2507**, to ensure coherent, context-aware, and high-quality summaries suitable for educational use.

3.2 AI Service Design:

- Service Functional requirements:

- The service must support receiving text-based lecture content for processing.
- The service must generate concise and coherent summaries directly from the provided text content.

- The service must extract key concepts, definitions, and important points from the summarized text, ensuring clarity and relevance.
- The service must organize extracted key points into a structured format that facilitates easy review and study.
- The service must provide an API interface for external services or users to utilize its text processing pipeline.
- The service must store and track the status of text processing at each stage (Summarization, Key Concepts Extraction).
- The service must provide an API endpoint to fetch the final processing report, including the generated summary and extracted key concepts.

- Non-functional requirements:

- **Usability:** The service must support processing text content in both Arabic and English, ensuring multilingual capability.
- **Security:** All API endpoints must be protected by an API secret key, ensuring that only authorized users or services can access the AI processing pipeline.
- **Performance:** The service must process text inputs within a short response time, ensuring minimal latency and an optimal user experience
- **Scalability:** The service must efficiently handle multiple text processing requests simultaneously using queue-based task management, ensuring seamless workload distribution and system scalability.

- Processing Flow:

We designed our AI service to seamlessly process text-based lecture content, transforming it into structured and meaningful outputs, including concise summaries and extracted key concepts. This design ensures that students receive high-quality, easily digestible material to support efficient learning.

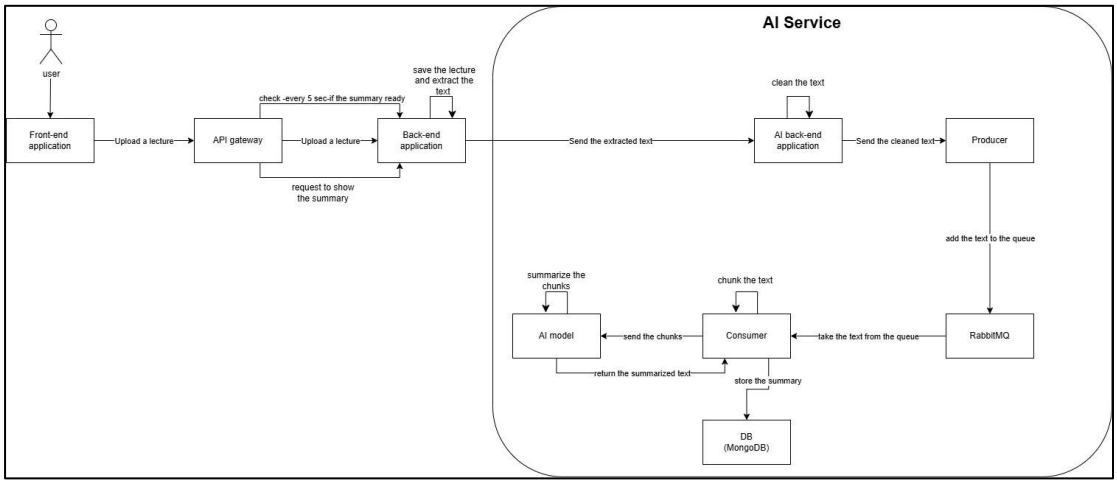


Figure 42 AI service processing flow

The AI service processes lecture text through a structured and asynchronous workflow to ensure efficiency, scalability, and reliable performance. When a student uploads a lecture through the front-end application, the request is routed via the API Gateway to the back-end service, which stores the lecture and extracts the textual content.

Once the text is extracted, it is sent to the AI service for processing. The AI back-end application first cleans and normalizes the received text to remove unnecessary characters and ensure consistency. The cleaned text is then forwarded to a producer component, which publishes the task to a message queue managed by RabbitMQ. This queue-based approach enables asynchronous processing and efficient task distribution.

The consumer component retrieves the text from the queue and splits it into manageable chunks suitable for AI processing. These chunks are then sent to the AI model, which generates summarized content for each chunk. The summarized outputs are collected and returned to the consumer, where they are combined into a final coherent summary.

Finally, the generated summary is stored in the database (MongoDB) for later retrieval. Throughout the process, the system updates the processing status, allowing the front-end application to periodically check whether the summary is ready and request its display once the processing is complete.

The importance of queue:

Implementing a queue system is essential for efficiently handling multiple text processing requests. It prevents system overload, ensures fair task distribution, and enables asynchronous processing, allowing multiple lecture texts to be processed concurrently without blocking new requests. This approach improves scalability, optimizes resource utilization, and ensures smooth handling of large workloads, providing a reliable and responsive AI service for students.

3.3 Used tools and models:

Based on the findings from our literature review, we carefully selected high-performance, resource-efficient models, well-integrated software solutions, and the most suitable frameworks to develop this service. This ensures a balance between optimal performance and cost-effectiveness while maintaining reliability and scalability.

- Django – python:

is a high-level Python web framework that encourages rapid development and clean, pragmatic design. especially we use DRF Django Rest Framework, which is a widely used, full-featured API framework designed for building RESTful APIs with Django

- MongoDB:

MongoDB is a NoSQL database that stores data in flexible, JSON like documents, making it ideal for handling unstructured data and large volumes of information. It offers high scalability and performance.

- RabbitMQ:

We use RabbitMQ as the messaging broker in our AI service to ensure efficient communication and task management across the service. It plays a crucial role in handling the queuing and distribution of tasks - audio processing requests.

3.4 APIs documentation:

API documentation is crucial for enabling seamless integration, clear communication, and efficient utilization of the service by developers.

- AI Service communication with the rest of the system:

The AI Service provides secure API endpoints for interaction with other system components. The base management service calls these endpoints when a lecture text file or document has been submitted and is ready for processing. This design ensures secure, reliable, and efficient communication between services, allowing the AI Processing Service to handle requests asynchronously while maintaining high-quality output.

API Endpoints Documentation:

Endpoint:	/lecture-text/
HTTP Method:	Post
Description:	Submits a lecture text for processing. The text is cleaned and sent to the processing queue.

Parameters:	• **`lecture_id`** (UUID, required): The unique identifier of the lecture. • **`text`** (string, required): The text content of the lecture.
Response:	• **`message`**: Success message confirming that the text has been received and sent to the queue. • **Status code:** 202 Accepted on success, 400 Bad Request if validation fails.

Endpoint:	`summary/status/<str:lecture_id>/`
HTTP Method:	Get
Description:	Retrieves the current processing status of a lecture summary, indicating whether the summary is ready.
Parameters:	• **`lecture_id`** (UUID, required, path parameter): The unique identifier of the lecture.
Response:	• **`lecture_id`:** The ID of the lecture. • **`ready`**: Boolean value indicating if the summary is ready (`true` or `false`). • **Status code:** 200 OK

Endpoint:	`summary/<str:lecture_id>/`
HTTP Method:	Get
Description:	Retrieves the final processed summary for a given lecture. Returns an error if the summary is not yet ready.
Parameters:	• **`lecture_id`** (UUID, required, path parameter): The unique identifier of the lecture.

Response:	• **Success (200 OK):**
	  ○ **`lecture_id`**: The ID of the lecture.
	  ○ `summary`: The generated summary text for the lecture.
	• **Error (404 Not Found):**
	  ○ **`lecture_id`**: The ID of the lecture.
	  ○ `error`: Message indicating that the summary is not ready.

4. Test cases

Test Case Scenario:		**Case 4: Generate quiz**	
Test case id	Test case title	Test steps	Expected result
TC-08	Generate quiz successfully with valid input.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select ≥1 lecture. 4) Select question type. 5) Enter question count. 6) Click Generate.	Quiz is generated with requested number & type of questions.
TC-09	Generate quiz from multiple lectures.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select multiple lectures. 4) Select question type. 5) Enter question count. 6) Click Generate.	Quiz generated using all selected lectures.
TC-10	Generate quiz with mixed question types.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select multiple types. 5) Enter counts. 6) Click Generate.	Quiz contains selected mixed types.
TC-11	Generate quiz with no lecture selected.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Do not select any lecture. 4) Click Generate.	Error: Must select at least 1 lecture.
TC-12	No question type selected.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Do not select question type. 5) Click Generate.	Error: Must select question type.

TC-13	No question count entered.	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Leave count empty. 6) Click Generate.	Error: Must enter number of questions.
TC-14	Zero question count	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Enter 0. 6) Click Generate.	Error: Question count must be greater than 0.
TC-15	Count > available questions	1) Login as student. 2) Navigate to “Generate Quiz”. 3) Select lectures. 4) Select type. 5) Enter > available content. 6) Click Generate.	Error: Requested questions exceed available content.

Test Case Scenario:		**Case 5: Auto evaluate answers**	
Test case id	Test case title	Test steps	Expected result
TC-16	Submit a correct answer.	1) Login as student. 2) Navigate to Quiz. 3) Select an answer. 4) Click Submit.	Correct answer message shown.
TC-17	Submit an incorrect answer.	1) Login as student. 2) Navigate to Quiz. 3) Select an answer. 4) Click Submit.	Incorrect answer message shown + explanation .
TC-18	Submit an empty answer.	1) Login as student. 2) Navigate to Quiz. 3) Do not select answer. 4) Click Submit.	Must select an answer message shown.

Test Case Scenario:		**Case 6: Analyze Uploaded File**	
Test case id	Test case title	Test steps	Expected result
TC-19	Upload valid PDF.	1) Login as Student. 2) Open upload page. 3) Select a valid PDF file. 4) Click Upload.	The system accepts the PDF.
TC-20	Upload valid DOCX.	1) Login as student. 2) Open upload page. 3) Select a valid DOCX file. 4) Click Upload.	The system accepts the DOCX.
TC-21	Upload valid PPTX.	1) Login as student. 2) Open upload page. 3) Select a valid PPTX file . 4) Click Upload.	The system accepts the PPTX.
TC-22	Reject unsupported file type (TXT).	1) Login as student. 2) Open upload page . 3) Select a valid TXT file. 4) Click Upload.	Error: The file not supported.
TC-23	Empty file content.	1) Login as student. 2) Open upload page. 3) Select a PDF/DOCX/PPTX. 4) Click Upload.	Error: The file is Empty.

Test Case Scenario:		**Case 7: Generate Summary using AI**	
Test case id	Test case title	Test steps	Expected result
TC-24	Generate summary from a valid file.	1) Student logs in successfully. 2) Student opens the document upload page. 3) Student selects a valid file (PDF/DOCX/PPTX) that contains readable content. 4) Student clicks on “Generate Summary”.	The system generate summary and displayed to the student .

Test Case Scenario:		**Case 8: Export Summary as pdf**	
Test case id	Test case title	Test steps	Expected result
TC-25	Export summary to PDF.	1) Login as Student. 2) Uploads a valid file. 3) Generates summary successfully. 4) Clicks Export as PDF.	The system Export Summary to PDF and download link displayed to student.
TC-26	Attempt export without summary.	1) Login as student. 2) Uploads a valid file. 3) skip summary generation. 4) Click Export as PDF.	Error: "please generate a summary before exporting"; no PDF created.

5. Service screenshots

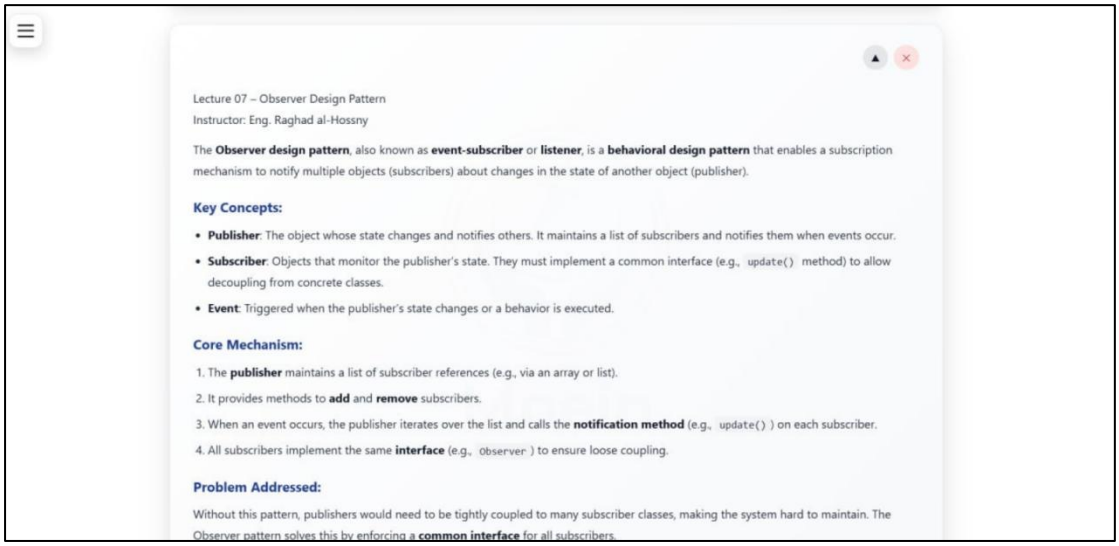


Figure43 Generated summary screenshot

Chapter 8: Conclusion.

1. Introduction:

As a result of our project, we have developed a software system designed to support software companies that follow the Agile methodology. This system streamlines the management of frequent and recurring meetings, ensuring that critical meeting artifacts are efficiently captured, organized, and utilized to enhance planning and decision-making for subsequent project phases. In this chapter we will conclude it all and put it all together, showing the final architecture, lessons learnt and the future possible enhancements.

2. Lessons Learnt

Throughout the development and deployment of AgileMeets, several valuable lessons emerged, shaping our understanding of microservices and cloud-based systems: • Importance of Containerization: Docker proved indispensable for maintaining consistency across environments, but managing Dockerfile configurations and Docker Compose orchestration required careful planning to avoid compatibility issues or resource conflicts. • Autoscaling Trade-offs: Fly.io's auto-scaling and auto-shutdown features significantly reduced costs demanded precise load metrics and monitoring to balance performance and expense effectively. • Inter-Service Communication: Implementing REST, WebSocket, and RabbitMQ required robust error handling and retry mechanisms to ensure resilience, particularly under high loads or network disruptions.

3. Future Recommendations

• Integrating or developing an Issue Management Service to create a comprehensive, all-in-one solution for projects and organizations.

• Implementing smart requirements extraction from meeting transcripts and key points to streamline project planning and documentation.

• Introducing a rich search experience, enabling semantic searches and requirement-based queries linked to meeting transcripts for improved accessibility.

• Enhancing the Meeting and AI Services to capture speaker identities within voice transcripts, distinguishing talkers for greater accuracy and context.

Appendices

- GitHub repository

References

- https://docs.djangoproject.com/en/5.0/

- https://www.google.com/search?q=react+documentation&oq=react+documentation&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIGCAEQRRg7MgYIAhBFGDsyBggDEEUYO9IBCDQzMTdqMGo3qAIAsAIA&sourceid=chrome&ie=UTF-8

- https://docs.python.org/3/

- https://www.mongodb.com/docs/

- https://www.rabbitmq.com/docs