Mutah University, Karak Jordan
Faculty of Information Technology

Quiz AI

A project submitted

in partial fulfillment of the requirements for the
B.Sc. Degree in Information Technology

By

Faisal Marwan Albaba (120222231068/Software Engineering)
Qotiph Khaled Albayaydah (120222231013/Software Engineering)
Bara'a Bader Eqnaibi (120222231123/Software Engineering)
Shahm Mohannad Abo-Alheja (120212212145/Cyber Security)
Nart Mohammed Mo'af (120222211003/Computer Science)

Supervised by

Dr. Khalid Awad Al-Tarawneh

& Njoud O. Al-maitah

September 2025

CERTIFICATE

It is hereby certified that the project titled ***Quiz AI***, submitted by undersigned, in partial fulfillment of the award of the degree of "Bachelor in Software Engineering" embodies original work done by them under my supervision.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Faisal Albaba (120222231068)

Qotiph Albayaydah (12022223013)

Nart Mo'af (120212212145)

Shahm Abo-Alheja (120212212145)

Bara'a Eqnaibi (120222231123)

ABSTRACT

Problem Statement: This software is an AI-powered educational tool designed to transform lecture audio into tailored exams, enhancing students' comprehension and retention. Users can record lectures directly or upload existing audio files, which are then processed using advanced speech recognition and natural language processing (NLP) techniques. The system identifies key concepts, learning objectives, and contextual cues from the lecture, generating quizzes or exams with customizable difficulty levels and question types. The platform aims to bridge the gap between passive listening and active learning by converting unstructured lecture content into interactive, measurable assessments. By providing immediate feedback and promoting self-testing, it fosters deeper engagement, better knowledge retention, and improved academic performance.

Proposed Solution: The project involves developing a platform that converts lecture audio into automatically generated exams. Users can record lectures directly or upload audio files (MP3, MP4). The system applies speech recognition and NLP to extract key concepts and produce various question types, including multiple-choice, fill-in-the-blanks, true/false, and short answer. Users can save generated exams, track their progress, and engage in efficient revision for improved understanding of lecture material.

Project Aim: To enhance students' comprehension and retention by transforming lecture audio into personalized exams, enabling active learning through immediate, targeted self-assessment and focused revision.

Objectives:

1. Support lecture audio recording and upload for automatic exam generation.
2. Extract key concepts from audio using speech recognition and NLP.
3. Generate diverse question types (multiple-choice, fill-in-the-blanks, true/false, short answer).
4. Allow users to save, organize, and manage generated exams in personal libraries.

5. Provide instant feedback and performance tracking to support adaptive learning.

Expected Benefits:

- Time-saving study sessions through automated quiz generation.
- Personalized learning by identifying weak areas and enabling repeated practice.
- Enhanced student engagement and knowledge retention through interactive assessments.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to our families and friends for their continuous support, encouragement, and understanding throughout the course of this project. Their patience and motivation have been invaluable.

We extend our heartfelt thanks to our instructors and doctors for their guidance and advice. In particular, we are deeply grateful to **Dr. Khalid Awad Al-Tarawneh**, who served as our supervisor and mentor during this journey. His expertise, encouragement, and dedication greatly contributed to the success of our work.

TABLE OF CONTENTS

ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
NLP	Natural Language Processing
PDF	Portable Document Format
MS	Microsoft
MP3	MPEG Audio Layer III
MP4	MPEG-4 Part 14
UI	User Interface
UX	User Experience
LMS	Learning Management System
SQL	Structured Query Language
IDE	Integrated Development Environment
VRAM	Video Random Access Memory
SSD	Solid State Drive
UAT	User Acceptance Testing
JSON	JavaScript Object Notation
CPU	Central Processing Unit

Abbreviation	Full Form
RAM	Random Access Memory
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
LoRA	Low-Rank Adaptation
ASR	Automatic Speech Recognition
OCR	Optical Character Recognition
GAN	Generative Adversarial Network
ML	Machine Learning
MCQ	Multiple Choice Question
DFD	Data Flow Diagram
UML	Unified Modeling Language
CRUD	Create, Read, Update, Delete
SSMS	SQL Server Management Studio
SQLi	SQL Injection
XSS	Cross-Site Scripting

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

This chapter provides an overview of the project, beginning with the challenges of transforming raw learning materials into meaningful assessments and summaries. It highlights the project's goals, scope, and boundaries, alongside the software and hardware requirements necessary for both development and deployment. In addition, it outlines the limitations, expected outputs, project schedule, and associated risks. Together, these sections establish the foundation for understanding the motivation, objectives, and constraints guiding the design and implementation of the AI-powered quiz-generation system.

1.1 Problem Statement: Successfully Translating Raw Learning Materials into Assessments and Summaries

The digital learning age has brought the creation of course material in a wide variety of formats, from audio and video files to PDFs, Word documents, and presentations. Translating this blended content into official quizzes and brief summaries is normally time-consuming, hard work, and prone to human error.

A majority of the existing tools are limited to one file type or must be pre-prepared beforehand by hand, which is inefficient and discourages frequent use. This creates a problem for students as well as teachers in accessing interactive, stimulating, and useful study material, thus slowing down the learning process within hectic academic as well as training environments.

1.2 Project Goal: From Formats to Knowledge — AI for Intelligent Learning Tools

The aim of this project is to develop an AI-driven system capable of extracting important details from various content forms and converting it into quizzes and abstracts. Through the automation of assessment and summarization, the system will help students, instructors, and self-learners save valuable time, promote active learning, and enable the easy creation of good-quality learning materials.

In the long term, the solution is designed to reduce teaching load, improve knowledge retention, and promote more individualized, adaptive learning experiences across different learning environments.

1.3 Project Scope: Establishing the Boundaries for AI-Powered Content Transformation

This project is designed to develop an AI system that can accept various types of content like MP3, MP4, PDF, MS Word files, and PowerPoint presentations, and perform real-time audio and video content transcription. The system will automatically generate quizzes and summaries from the inputs via an easy, intuitive web-based interface.

1.3.1 Key Features

1. Content parsing and intelligent file format handling
2. Natural Language Processing (NLP) to pull out major points and ideas
3. Robot quiz creation from various question types (e.g., multiple choice, true/false, short answer)
4. Fast revision-boosted summary creation

1.3.2 Within Scope

1. File uploading in accepted formats
2. Real-time transcriptions
3. English-language content
4. Arabic-language content
5. Exporting quizzes in common formats
6. Interactive in-platform quiz-taking

1.3.3 Out of Scope

1. Third-party LMS integration (e.g., Blackboard, Moodle)
2. Non-English and non-Arabic language support
3. Mobile app development in this release

1.4 Project Software and Hardware Requirements

1.4.1 Hardware Requirements

Hardware requirements can be split into two parts: requirements for development and requirements for end-users.

1.4.1.1 Requirements for Development

React

- CPU: Intel Core i3
- RAM: 8 GB
- Storage: 10 GB (Kinsta, n.d.)

C# Development using Visual Studio IDE

- Processor: 1.8 GHz or faster processor (dual-core or better recommended)
- RAM: 2 GB (4 GB recommended, 2.5 GB minimum on a virtual machine)
- Hard Disk Space: Up to 130 GB depending on features installed; typical installations require 20–50 GB. SSD recommended
- Video Card: Supports minimum display resolution of 720p (1280 × 720); WXGA (1366 × 768) or higher recommended (Microsoft, 2025)

1.4.1.2 Hardware Requirements for End-Users

1. CPU: 4 cores or more
2. System RAM: at least 16 GB
3. GPU: NVIDIA GPUs only, 12 GB VRAM or higher
4. Storage: 60 GB SSD (250 GB recommended for additional models)
5. Wired internet connectivity during setup; wired LAN afterwards (InsightReactions, 2025)

1.4.2 Software Requirements

1.4.2.1 Requirements for Development

1. React: Windows 10/11, Ubuntu 16, or macOS 10.10 (Kinsta, n.d.)

2. C# development using Visual Studio IDE: Windows 10/11 (Microsoft, 2025)
3. meta-llama-3.1-8b-instruct Q5_K_M: Windows 10/11, Ubuntu 16, or macOS 10.10 (, 2025)

1.4.2.2 Requirements for End-Users

- Windows 10/11, Ubuntu 16, or macOS 10.10

1.5 Project Limitations

1.5.1 Technical Limitations

1. Uses an open-source model agent as a base. Internet connection required (ITU, 2025).

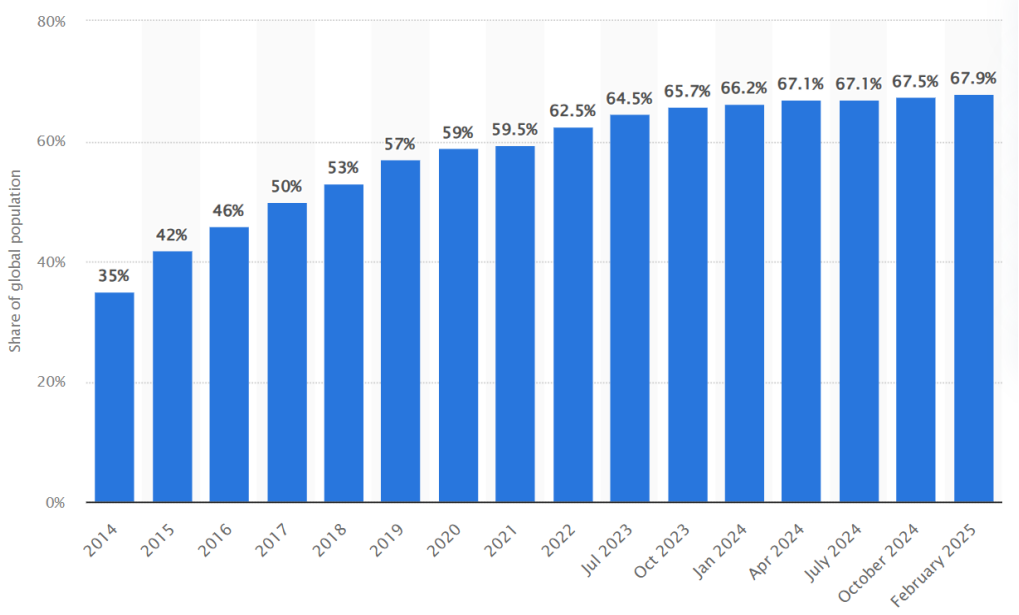


Figure 1.1: Global internet penetration in 2025, highlighting the 67.9% population with access (Source: ITU, 2025).

2. There is no application version of Quiz AI, which means it may not be as attractive to most of the population (GSMA, 2025).

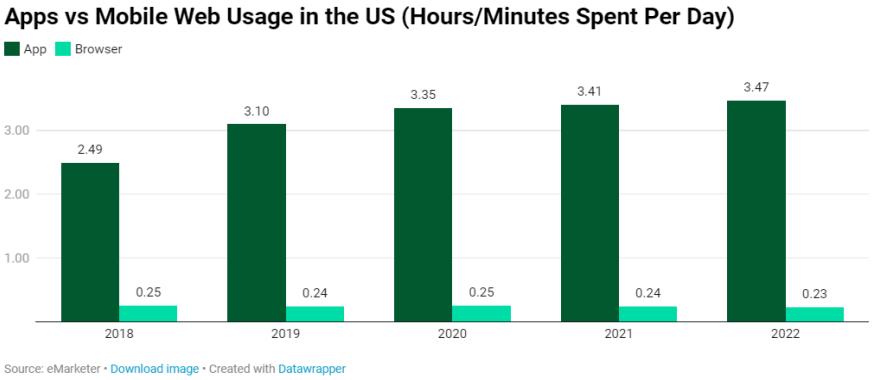


Figure 1.2: Lack of a Quiz AI mobile app may limit user adoption, illustrated by global mobile app engagement trends (Source: GSMA, 2025).

1.5.2 Functional Limitations

- Unsupported: Extracting audio from video not implemented.
- Workarounds: Use third-party APIs.

1.5.3 Security Limitations

- No login security measures or encryption for login database.

1.5.4 Usability & Accessibility

- Supported file formats: txt, pdf, pptx, docx, mp3, mp4.

1.6 Project Expected Output

1.6.1 Functional Outputs

1. Responsive web dashboard
2. Notifications for processing start and completion

1.6.2 Non-Functional Outputs

1. Fast output time
2. 4-chapter documentation in PDF
3. GitHub repository with source code

1.7 Project Schedule

The project schedule outlines the key milestones, activities, and deliverables required to complete the AI-powered quiz-generation website. It ensures each stage of the development process is completed on time, with clear dependencies between tasks.

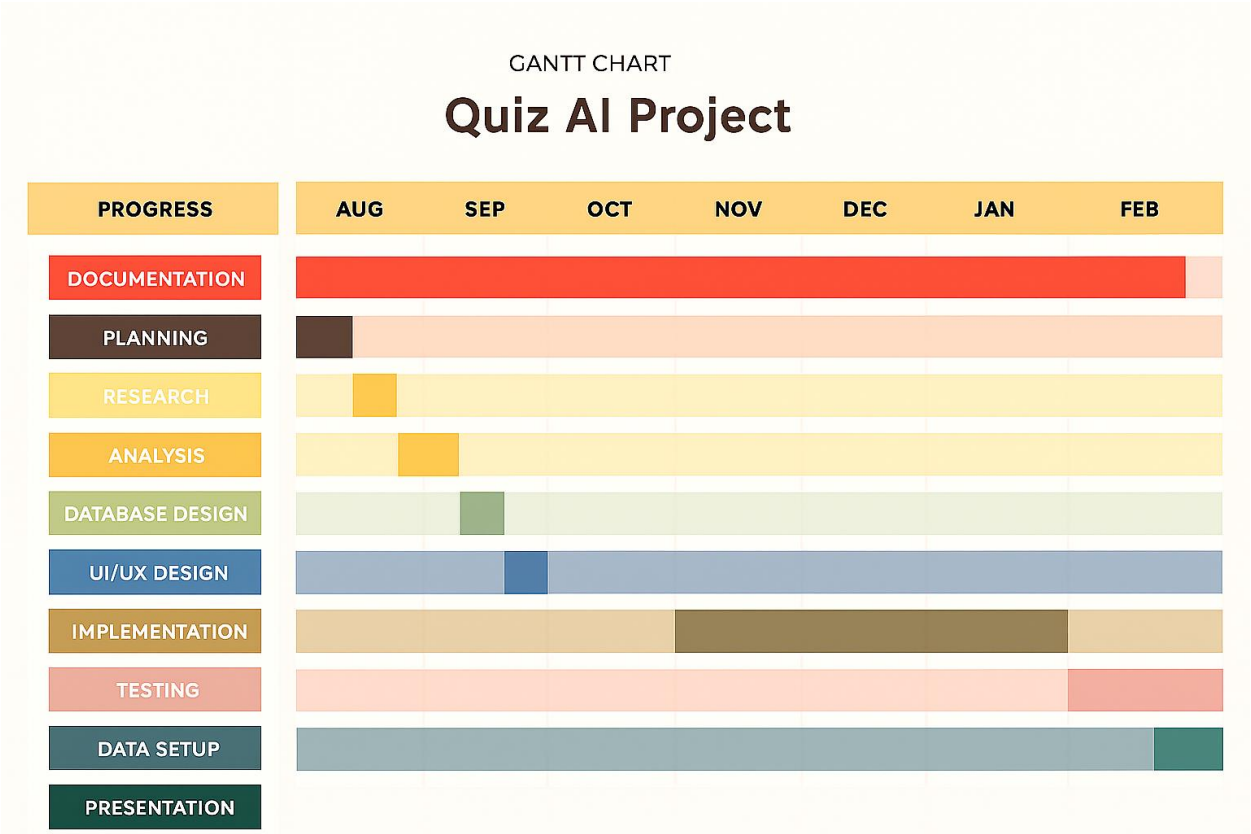


Figure 1.3: Project schedule Gantt chart illustrating key milestones, activities, and deliverables for the AI-powered quiz-generation website (Source: Project Team, 2025).

1.8 Methodology Framework Diagram

A methodology framework diagram is a visual representation that outlines the structured steps (the methodology) within a guiding structure (the framework) to carry out a process or research—often shown as boxes and arrows depicting stages and relationships (ResearchGate, 2008).

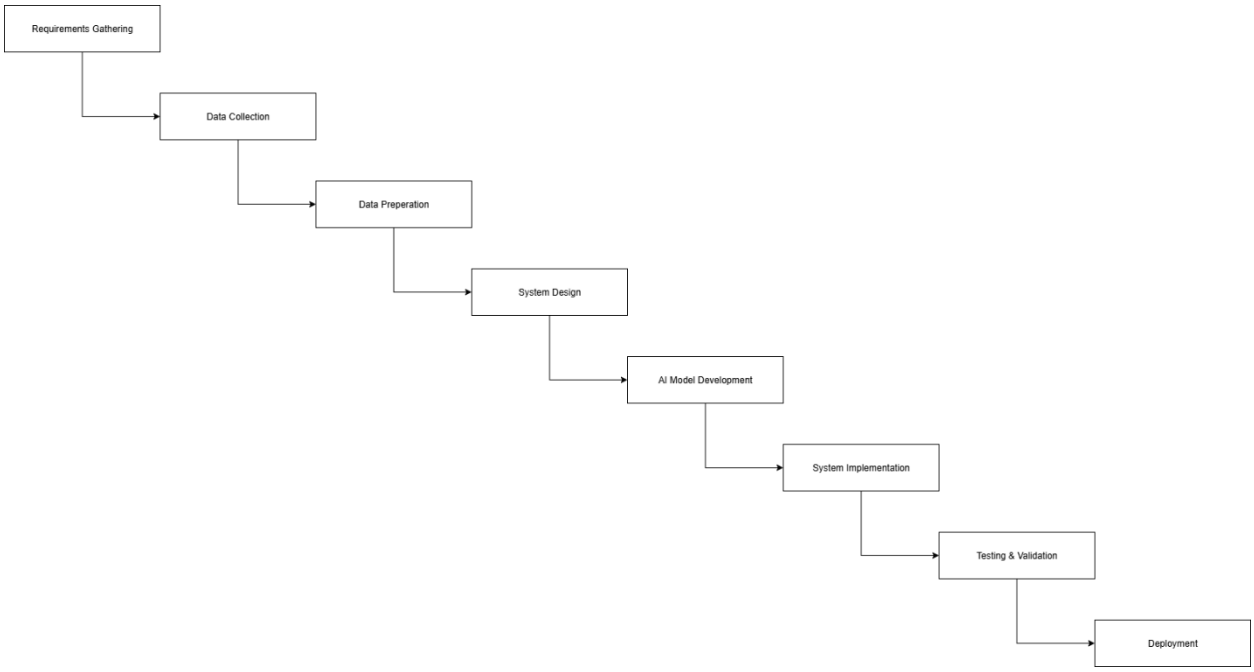


Figure 1.4: Methodology Framework Diagram

1.8.1 Workflow Steps

1. **Requirement Gathering:** Identify goals, define user roles, collect expectations.
2. **Data Collection:** Gather books, slides, notes, and other materials.
3. **Data Preparation:** Clean, organize, remove duplicates, structure topics.
4. **System Design:** Draw use case diagrams, DFDs, sequence diagrams.
5. **AI Model Development:** Build and train AI to generate quizzes, suggest topics, power chatbot.
6. **System Implementation:** Develop platform, user interfaces, integrate roles.

7. **Testing & Validation:** Test AI accuracy, usability, and system performance.
8. **Deployment:** Publish system for real users online.
9. **User Training & Documentation:** Provide manuals, guides, training for instructors and students.

1.9 Project / Product Schedule Risks

1.9.1 Schedule Risk

- Delays may occur due to AI model integration, file handling, or unforeseen bugs.

1.9.2 Impact

- Reduced testing time could affect stability and quality.

1.9.3 Mitigation Strategies

1. Begin AI research concurrently with UI/UX design.
2. Use pre-trained NLP models to speed development.
3. Implement fallback mechanisms for simplified quiz generation.
4. Add buffer periods and conduct weekly progress reviews.

1.10 Report Organization

- **Chapter 2** – Theoretical Background & Literature Review: Existing AI learning tools.
- **Chapter 3** – Requirements Analysis: Functional and non-functional system requirements.
- **Chapter 4** – Software Design: System architecture, database schema, UI prototypes.

1.11 Operational Definitions

- **System** → The Quiz AI platform.
- **User** → Any individual interacting with the system.
- **Instructor** → Uploads materials, approves content, manages quizzes.

- **Student** → Studies materials, selects topics, takes quizzes.
- **Admin** → Manages accounts, access rights, and system settings.
- **Knowledge Base** → Collection of study resources.
- **Topic** → Subject area for quiz questions.
- **Quiz** → Automatically or instructor-generated questions.
- **Question Bank** → Repository of questions.
- **Competitive Exam Mode** → Multiple students compete for points.
- **Self-Assessment** → Quiz mode with instant results.
- **Gamification** → Levels, points, badges for engagement.
- **Content Approval** → Instructor validation of AI content.
- **Exam Log** → Records of quiz attempts.
- **AI Suggestion** → Recommendations on topics and questions.
- **Voice Reading** → System reads quiz questions aloud.
- **Chatbot Assistant** → Conversational tool for content questions.
- **Tool Support** → Additional resources like calculators or whiteboards.
- **Academic Integrity** → Ensures fair quizzes/exams.
- **Data Privacy** → Protects user data.

CHAPTER 2: THEORITICAL BACKGROUND & LITERATURE REVIEW

This chapter explores the theoretical foundations and prior research relevant to AI-driven quiz generation and multimedia learning tools. It begins with an overview of artificial intelligence and its role in processing multimodal content, followed by the process of AI-generated quiz creation. The chapter then reviews AI-powered multimedia generation models—ranging from text-to-text, text-to-image, text-to-audio, and text-to-video—and their applications in education. Finally, it examines existing AI quiz-generation tools, their functionalities, challenges, and emerging trends, providing a basis for understanding how this project builds upon and extends current technologies.

2.1 Artificial Intelligence (AI) Overview

The field of artificial intelligence (AI) is concerned with building machines that are able to carry out operations like perception, reasoning, and decision-making that normally call for human intelligence. Rule-based systems gave way to data-driven machine learning in modern AI, with large-scale deep learning models—particularly transformers—being the driving force behind recent advances. These days, large language models (LLMs) like GPT (OpenAI, 2025), Llama (Meta AI, 2024), Claude (Anthropic, 2024), and Gemini (Google DeepMind, 2024) serve as flexible reasoning engines that can process multimodal inputs, text, and code.

By processing text, audio, images, and video in a single pipeline, multimodal AI expands these capabilities (Yin et al., 2024) even further, opening the door for uses like lecture transcription and analysis. While parameter-efficient fine-tuning techniques (e.g., LoRA) enable cost-effective customization (Hu et al., 2022) for domain-specific tasks, retrieval-augmented generation (RAG) techniques increase accuracy (Lewis et al., 2020) by connecting models to external knowledge sources. Strong text-to-image, audio, and video models are another example of generative AI advancements that increase the creative and analytical potential.

AI is better able to handle big, complicated datasets, like complete lecture transcripts, when it has longer context windows and uses agent-like tools. In addition to these capabilities, safety, alignment, and governance frameworks are becoming more and

more important in order to ensure responsible deployment, particularly in delicate areas like education.

2.2 AI-Generated Quiz Creation Process

A multi-step process that combines knowledge comprehension, question formulation, and content extraction is used to create AI-generated quizzes. The system starts by ingesting source material, such as documents, audio recordings, videos, or lecture transcripts. Automatic speech recognition (ASR) transforms audio into text for non-text inputs, and optical character recognition (OCR) pulls text from slides or pictures.

Natural language processing (NLP) is applied to the extracted text in order to pinpoint important ideas, connections, and learning goals. Retrieval-augmented generation (RAG) guarantees that questions generated are based on the original content, while summarization models distill extensive content into targeted sections. These concepts are converted into multiple-choice, fill-in-the-blank, true/false, and short answer question formats by large language models (LLMs) like GPT or LLaMA, which are frequently refined using educational datasets.

Based on student performance data or frameworks such as Bloom's Taxonomy, dynamic difficulty adaptation is feasible. These capabilities are demonstrated by platforms that support multimodal inputs and customizable outputs, like Questgen AI (2025), Quizbot AI, and VidVersityQG (Shahid, Hussain, & Shoaib, 2021). This procedure is a useful tool in contemporary education since it not only automates the creation of assessments but also customizes tests for focused revision.

2.3 AI Multimedia Generating Tools

2.3.1 Overview

Artificial intelligence systems created to automate or support the production of different media types, such as text, images, audio, video, animation, and interactive content, are referred to as AI multimedia tools.

In order to produce richer and more captivating user experiences, multimedia generally integrates various types of content—such as text, images, audio, and

video—into a single presentation. Through features like games, quizzes, and clickable elements, interactive multimedia makes content more dynamic and personalized while encouraging active user participation.

The creation of content has changed dramatically as a result of AI developments. AI can create organized and cohesive textual content, including blog posts, reports, and articles, thanks to Natural Language Processing (NLP). While AI-powered tools support music composition, sound editing, voice synthesis, video production, and animation, Generative Adversarial Networks (GANs) and other generative models help create realistic images. When taken as a whole, these innovations facilitate customized multimedia experiences, improve creative efficiency, and lessen manual labor.

2.3.2 Text-to-Text Generation Models

The creation of content has changed dramatically as a result of AI developments. AI can create organized and cohesive textual content, including blog posts, reports, and articles, thanks to Natural Language Processing (NLP). While AI-powered tools support music composition, sound editing, voice synthesis, video production, and animation, Generative Adversarial Networks (GANs) and other generative models help create realistic images. When taken as a whole, these innovations facilitate customized multimedia experiences, improve creative efficiency, and lessen manual labor.

Institutions should create adaptable policies, train employees and students on moral AI use, and rethink tests to prevent abuse in order to meet these challenges. In order to identify AI-generated content, human judgment is still essential. In the AI era, preserving academic integrity requires involving students in policy-making and implementing a cooperative community approach.

2.3.3 Text-to-Image Models

AI programs that can produce images straight from descriptions in natural language are called text-to-image models. They allow users to create visuals that correspond with particular prompts by bridging the gap between linguistic and visual modalities.

Current methods consist of:

1. GAN-based Models: Using a generator–discriminator framework, Generative Adversarial Networks generate realistic images that correspond with text descriptions.
2. Diffusion Models: Stable Diffusion and DALL·E 2 gradually convert noise into coherent images through prompt-guided denoising steps.
3. Transformer-based Models: Use transformer architectures to model the relationships between text and visual elements for fine-grained image control.
4. Hybrid Approaches: Utilize a variety of methods to optimize semantic accuracy and visual quality.

   Applications: Digital art, marketing, product design, and instructional content production.

2.3.4 Text-to-Audio/Voice Models

Applications such as voice assistants, audiobooks, and music composition are made possible by these AI systems, which use text as input to produce speech, music, or sound effects. Examples include Uber's Jukebox and Google WaveNet.

2.3.5 Text-to-Video Models

New AI technologies automate video production processes by producing video content from minimal inputs or text descriptions. While still in the early stages of development, platforms such as Google's VEO 3, Meta's Make-A-Video, and Runway Gen-2 show promise.

2.3.6 Use Cases in Education

By creating rich, context-specific visual materials that are suited to different learning styles, text-to-image models can improve education by increasing engagement and comprehension (Chen & Wang, 2022).

- Less manual design is required because teachers can produce original diagrams, infographics, and illustrations aligned with lesson goals.
- Students can visualize historical occurrences, scientific procedures, and abstract ideas to improve memory and creativity.
- These resources can be incorporated into courses in creative fields like media studies, design, and art.
- Inclusive education benefits from accessible and culturally appropriate imagery for diverse linguistic contexts and students with disabilities.

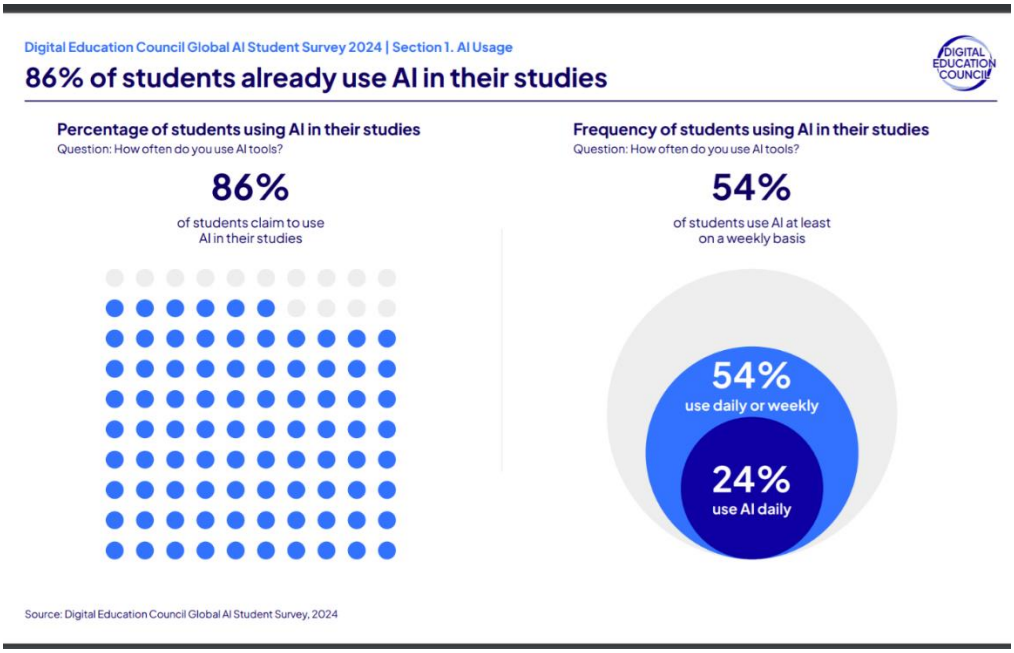


Figure 2.1: Student AI Usage Statistics. Adapted from Digital Education Council Global AI Student Survey (2024).

- 86% of students already use AI in their studies.
- 54% use AI daily or weekly, with 24% using it daily.

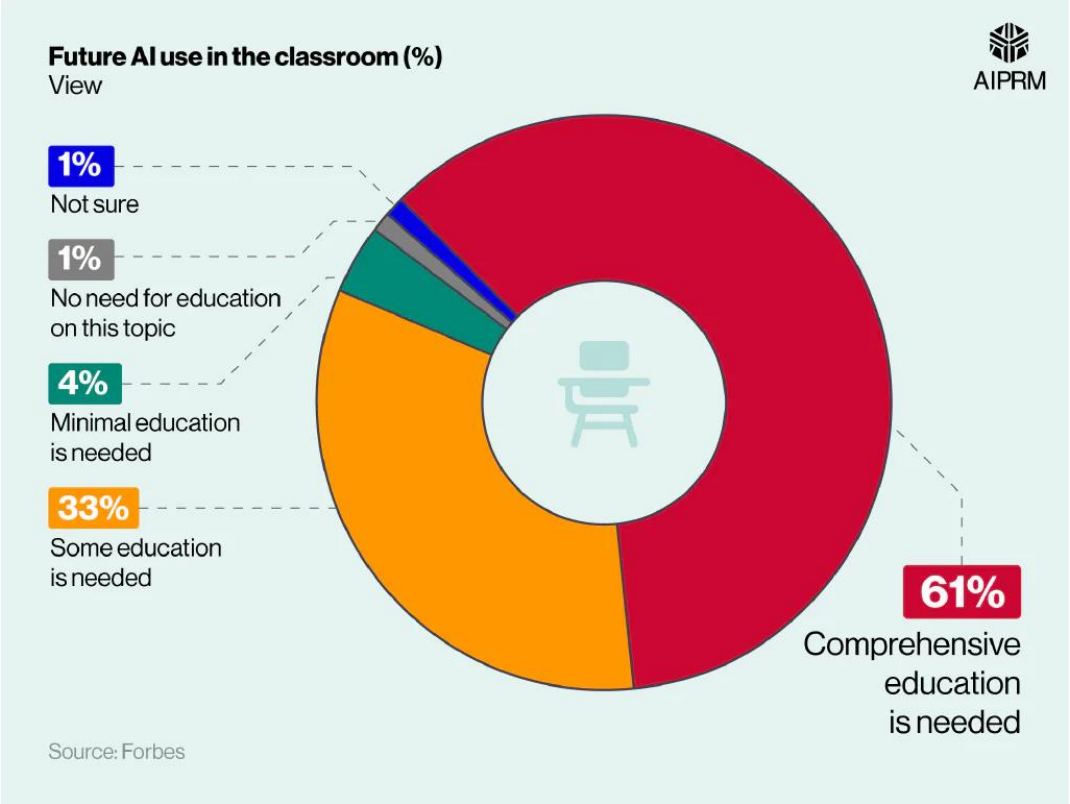


Figure 2.2: Student AI Usage Statistics. Adapted from *Digital Education Council Global AI Student Survey* (2024).

- Note: 86% of students use AI in their studies; 54% use AI daily or weekly, with 24% using it daily.

2.3.7 Challenges and Ethics

Even though AI multimedia tools have a lot of potential, their responsible use requires addressing key challenges:

- **Content Quality & Reliability:** AI may generate biased, erroneous, or misleading results.
- **Bias & Fairness:** Training data can reinforce stereotypes, affecting inclusivity.
- **Intellectual Property & Copyright:** AI may unintentionally replicate existing works.
- **Misinformation Risks:** Misuse for deepfakes or fake news undermines trust.
- **Privacy & Data Protection:** Sensitive data use raises ethical and legal concerns.

Solution: Collaboration among technologists, educators, policymakers, and ethicists is needed to ensure accountability, transparency, and fairness.

2.4 AI Tools to Generate Quizzes

2.4.1 Introduction

By making it possible to create quizzes automatically, the development of artificial intelligence (AI) has drastically changed educational assessment. Natural language processing (NLP) and machine learning (ML) algorithms are used by AI-driven quiz generation tools to produce a variety of assessments that can be customized to meet the needs of each learner. These resources have the potential to support adaptive learning environments, increase instructional efficiency, and offer instant feedback (Alsmadi & Almarashdeh, 2023). The basic processes underlying AI quiz creation, the range of question types generated, and the main advantages, difficulties, and restrictions related to their use in education are all covered in this section.

2.4.2 How AI Quiz Tools Work

AI quiz generation systems employ a combination of Natural Language Processing (NLP), Machine Learning (ML), and large language models (LLMs), such as GPT, to automatically generate questions from educational materials. NLP techniques allow systems to parse text, discover, and extract concepts, as well as to discern relationships between concepts and ideas. ML algorithms allow systems to identify existing patterns in question banks and learner performance data, and to formulate more contextualized and accurately structured questions. More advanced systems can include large language models that can actually create a new piece of content, paraphrase a complex idea, and be able to change the complexity of the language to an appropriate level for the audience. Most systems can accept a range of formats—like documents, lecture transcripts, or video—and produce diverse question formats, such as multiple-choice, true/false, open-ended, and scenario-based items. Finally, the system usually produces a validation step to ensure pre-generated questions meet learning objectives and standards for clarity and accuracy before being distributed for final delivery.

2.4.3 Types of Questions Generated

Artificial intelligence quiz tools generate many types of questions, and their benefits are specific to their academic purpose. Multiple-choice questions (MCQs) are the most common format - students demonstrate recognition and recall by choosing from several answering options and only one correct answer to complete the task. True/False questions are valuable when quickly assessing binary knowledge statements, as they allow the user to collect many responses to evaluate factual understanding quickly. Short-answer responses allow for a gravitational active recall process as students generate short responses. More advanced AI systems allow for open-ended or essay-style questions that can assess higher-level thinking skills such as critical thinking, synthesis, and application. Because AI quiz tools can generate many question types, they can provide a range of assessment strategies to assess different learning styles and objectives.

2.4.4 Personalization and Adaptivity

Although there have been some good examples of progress, the vast majority of tools for AI quiz generation today, provide little to no genuine personalization/adaptivity. Most often, these tools provide static quizzes which do not change the difficulty of questions or change topic depending on learner performance or preferences. Similarly, feedback systems are basic, and offer little if any, meaningful suggestions to assist the learner in their progression. These various forms of lack of adaptivity, means learners are having generic learning experiences that take advantage of the limited personalization/adaptiveness, thus also limiting the overall utility and engagement with educational opportunities.

2.4.5 Challenges and Limitations

AI quiz tools are capable of generating questions only too easily, but a consistent challenge is generating coherent and context-related choice options. These tools, including Quizlet, Quizziz, Kahoot, QuestionPro, and ProProfs, often produce distractors that are completely unrelated to the question stem, i.e., a question that pertains to a date, has distractors that include ocean names or phrases that are unrelated (Observations, 2025). This is not only a misuse of an assessment tool, but it also detracts from the overall validity of the assessment. Also, many tools limit the characters of the questions and answer options, which limits their complexity and clarity. The inability to scale question difficulty is also widespread, limiting tools' ability to be effective for learners of multiple abilities. Overall, these issues result in poor quality, reliability, and use in the teaching/learning process of AI-created quizzes (Observations, 2025).

2.4.6 Future Trends

The rising use of advanced large language models (LLMs) like GPT, Gemini, and related technologies is one of the most exciting developments in AI quiz generation. When compared to traditional quiz methods, LLMs can produce better quality quiz questions, and aspects of difficultly will be contextualized, which allows for more personalized learning opportunities. However, the LLM strength still produces quizzes in a relatively static/non-interactive way, as the AI-generated quiz still provides limited real-time adaptivity when creating quizzes that are intended to engage the learner and give them dynamic real-time feedback. Future research and development of LLMs are likely to focus on integration with interactive platforms for assessment, to allow for a more adaptive experience, producing courses that are more immersive and responsive to students needs.

CHAPTER 3: REQUIREMENT CLOLLECTION/ENGENERRING AND ANALYSIS

This chapter defines the functional and non-functional requirements of the proposed Quiz AI system, along with its constraints and stakeholder interactions. It begins by identifying primary, secondary, and tertiary stakeholders, then outlines detailed software requirements covering authentication, content management, quiz generation, and reporting. Functional requirements are expressed through use cases, while non-functional requirements highlight performance, security, usability, and scalability aspects. The chapter also presents system constraints across resources, technology, operations, and external factors. Finally, software diagrams—including use case, context, and data flow diagrams—illustrate the system's structure and data interactions, providing a comprehensive view of its expected functionality.

3.1 Stakeholders

3.1.1 Primary Stakeholders

- **Development Team**: Responsible for building and maintaining the system, thus its success or failure directly impacts them.
- **Students – Group A (Direct Users)**: Actively use the system as a tool for studying and reviewing their learning materials.
- **Teachers**: Benefit from AI-generated quizzes and exams, making assessment preparation faster and easier.
- **College & University Professors**: Similar to teachers, they leverage the AI system to efficiently generate quizzes and exams for higher-level courses.

3.1.2 Secondary Stakeholders

- **Students – Group B (Indirect Users)**: While they do not use the system directly, they may still take quizzes and exams created by it.
- **Educational Bodies**: Although not direct users, institutions may need to license or purchase the rights to use the system.

3.1.3 Tertiary Stakeholders

- **Textbook Publishers**: May experience reduced demand for premade question banks, negatively impacting sales.

- **Educational Researchers**: Interested in studying how AI usage influences assessment quality and student learning outcomes.
- **Educational Book Sellers & Libraries**: As schools rely more on AI-generated quizzes, demand for printed quiz books and related library resources may decrease.

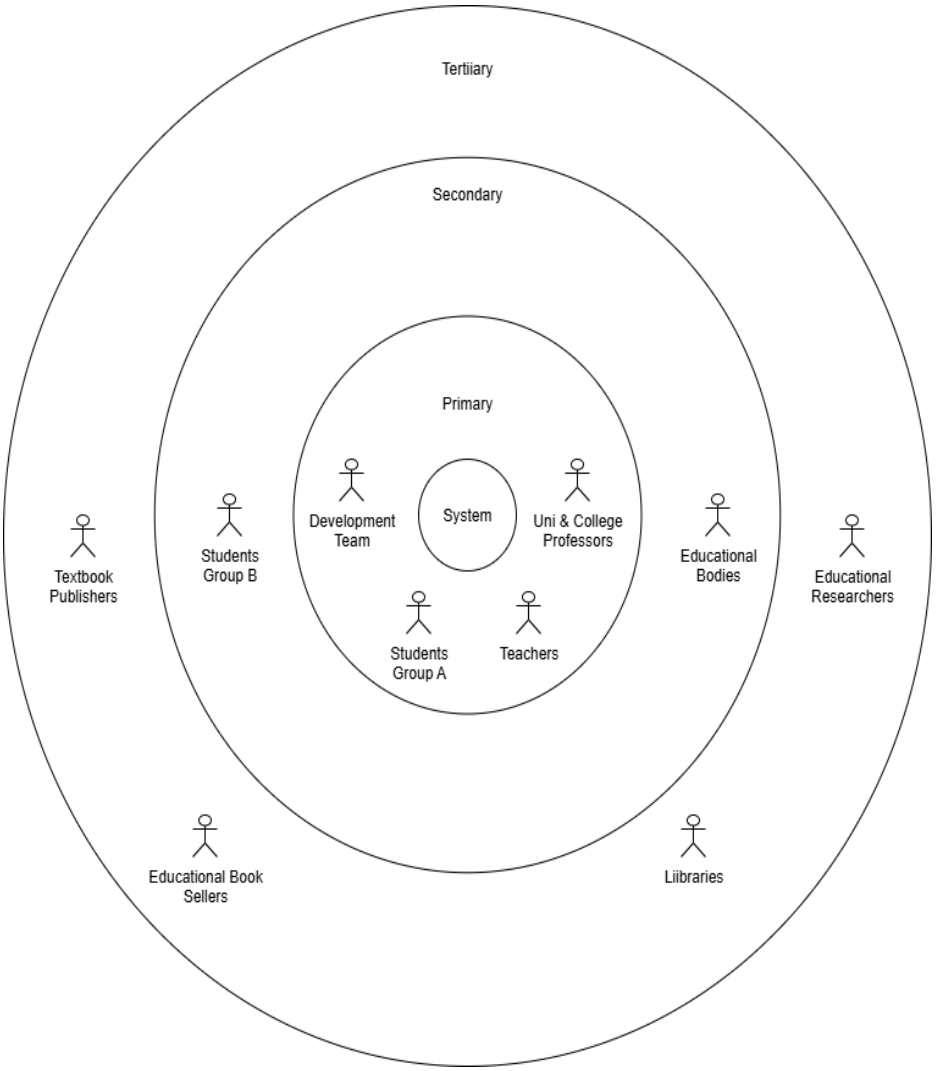


Figure 3.1: Stakeholders Diagram for the Quiz AI system, showing the primary, secondary, and tertiary stakeholders.

3.2 Software Requirements

1. **User Authentication and Security**
    - Users (students, instructors, admins) must be able to log in securely using their username and password.
    - Users must be automatically logged out if sessions remain idle for too long.
2. **Password Recovery**
    - Users must be able to reset their password through email verification.
    - Users must receive a password reset link or token if they forget their password.
3. **Content Management**
    - Students and instructors must be able to upload content (e.g., lecture slides, PDFs, videos) to generate quizzes.
    - Users must not be able to upload duplicate content files.
4. **Quiz Generation**
    - Users must be able to generate quizzes from uploaded content.
    - Users must be able to choose quiz difficulty level and number of questions.
    - Users must be able to access public quizzes, while private ones remain hidden.
    - Students must be able to view top attempts for public quizzes.
5. **Exam Creation and Publishing**
    - Instructors must be able to create and publish quizzes/exams.
    - Instructors must be able to set attempt limits for each quiz/exam.
6. **Quiz Participation**
    - Students must be able to attempt quizzes/exams within a time limit set by the instructor.
    - Students must have a secure and uninterrupted session during attempts.
7. **Results and Analytics**

- Students must be able to view their quiz results and weak areas in specific chapters.
- Instructors must be able to view analytics such as average marks, highest scores, and lowest scores.

8. **Error Handling and Logging**
    - Users must be notified if errors occur during login, uploads, or quiz attempts.
    - The system must ensure critical errors are escalated to administrators.
9. **Administrative Functions**
    - Admins must be able to manage users and their permissions.
    - Admins must be able to generate statistical reports about system usage and performance.

3.3 Functional User Requirements (Use Case Specifications)

1. Login to the System

Actor(s): Student, Instructor, Admin

Description: User enters username and password to access the system.

Precondition: User has an active account.

Normal Flow:

1. User navigates to the login page.
2. User enters username and password.
3. System verifies credentials.
4. System grants access to the dashboard.

Alternate Flow:

- If credentials are incorrect, the system displays an error message.
- If multiple failed attempts occur, the account is temporarily locked.

Postcondition: User is logged in and can access system features.

2. Handle Login Errors (Acceptance/Denial)

Actor(s): System

Description: System validates credentials and either accepts or denies access.

Precondition: Login request is made by the user.

Normal Flow:

1. System checks username and password against the database.
2. If valid, grants access.
3. If invalid, denies access.

Alternate Flow:

- System suggests *Forgot Password* after multiple failed attempts.

Postcondition: Access is either granted or denied.

3. Forgot Password Request

Actor(s): Student, Instructor, Admin

Description: User requests password reset through the system.

Precondition: User cannot log in to the system.

Normal Flow:

1. User clicks *Forgot Password*.
2. System prompts for registered email address.
3. User enters email address.
4. System sends password reset email.

Alternate Flow:

- If email is not registered, the system shows an error message.

Postcondition: Password reset email is sent to the user.

4. Send Password Reset Email

Actor(s): System
Description: Email Control System sends password reset email via Email Service.
Precondition: Password reset request has been made.

Normal Flow:

1. System generates password reset token.
2. System sends email with password reset link.

Alternate Flow:

- If email service is unavailable, the system logs an error and retries later.

Postcondition: User receives password reset email.

5. Upload Content (Lecture or Exam Material)

Actor(s): Student
Description: Student uploads lecture content for quiz generation.
Precondition: User is logged in and has permission to upload.

Normal Flow:

1. User navigates to upload section.
2. User selects content file.
3. System validates file format and size.
4. System stores the content.

Alternate Flow:

- If file is invalid, the system displays an error message.

Postcondition: Content is uploaded and ready for quiz generation.

6. Generate Quiz from Uploaded Content

Actor(s): Quiz AI Generation Engine
Description: System processes content and generates a quiz using AI.
Precondition: Content has been successfully uploaded.

Normal Flow:

1. System processes uploaded content.
2. AI analyzes content and generates questions.
3. System stores generated quiz.

Alternate Flow:

- If processing fails, the system logs an error and notifies the user.

Postcondition: Quiz is generated and stored in the system.

7. View Generated Quiz

Actor(s): Student
Description: Student reviews the AI-generated quiz.
Precondition: Quiz has been generated.

Normal Flow:

1. User navigates to quizzes section.
2. User selects generated quiz.
3. System displays the quiz.

Alternate Flow:

- If quiz is not available, the system shows an error message.

Postcondition: User views the quiz.

8. Create Public Exam

Actor(s): Instructor
Description: Instructor creates and publishes an exam for public access.
Precondition: Instructor is logged in.

Normal Flow:

1. Instructor navigates to exam creation page.
2. Instructor sets exam details and questions.

3. Instructor publishes the exam.

Alternate Flow:

- If publishing fails, the system shows an error message.

Postcondition: Exam is publicly available.

9. View Public Exam Results and Analytics

Actor(s): Instructor
Description: Instructor checks analytics and results of public exams.
Precondition: Exams have been conducted.

Normal Flow:

1. Instructor navigates to analytics section.
2. System displays exam performance data.

Alternate Flow:

- If data is unavailable, the system notifies the user.

Postcondition: Instructor views analytics.

10. Access Quiz Results and Reports

Actor(s): Student
Description: Student checks performance and results after taking a quiz.
Precondition: Quiz attempt is completed.

Normal Flow:

1. User navigates to results section.
2. System displays quiz results and reports.

Alternate Flow:

- If results are delayed, the system notifies the user.

Postcondition: User views quiz results.

11. Request Statistical Reports

Actor(s): Admin

Description: Admin requests system analytics and reports.

Precondition: Admin is logged in.

Normal Flow:

1. Admin navigates to reporting section.
2. System generates statistical reports.
3. System displays the reports.

Alternate Flow:

- If report generation fails, the system logs an error.

Postcondition: Admin views system analytics.

12. Handle System Errors (Error Control System)

Actor(s): System

Description: System logs and manages errors during processes.

Precondition: System encounters an error.

Normal Flow:

1. System detects an error.
2. System logs error details.
3. System notifies relevant stakeholders.

Alternate Flow:

- If error persists, the system escalates to technical support.

Postcondition: Errors are managed and logged.

3.4 Non-Functional Requirements

3.4.1 Execution Qualities

- **Safety**
  The system shall reject quizzes that are inappropriate, harmful, or offensive. Errors that could confuse students or undermine academic integrity shall be avoided.
- **Security**
  The system shall protect user data through safe authentication (e.g., secure password policies and optional two-factor authentication). Sensitive data, including user credentials, shall be encrypted during both storage and transmission. User activities shall be logged for security auditing.
- **Usability**
  The system shall provide a user-friendly and intuitive interface, making it accessible for users with limited technical skills. Accessibility features shall be supported, and user manuals/help guides shall be provided.
- **Performance Efficiency**
  The system shall provide a response time of less than five seconds for common operations. It shall support up to 250 concurrent users without performance degradation and ensure scalability to handle increased demand.
- **Reliability & Availability**
  The system shall ensure at least 99% uptime and allow automatic recovery from failures. Data shall be backed up daily to prevent loss. The system shall be accessible 24/7 except during scheduled maintenance, with downtime not exceeding 5 hours per month. Real-time notifications shall be provided during outages.

3.4.2 Evolution Qualities

- **Testability**
  The system shall support comprehensive testing, including unit, integration, and performance testing, to ensure correctness and reliability.
- **Maintainability**
  The system shall follow a modular architecture with proper documentation to enable easy bug fixes, updates, and feature additions. Updates shall be deployable with minimal downtime.

- **Portability**
  The system shall be deployable across Windows, Linux, and macOS platforms and support both desktop and mobile web browsers. It shall be easily transferable to cloud environments.
- **Scalability**
  The system shall allow horizontal and vertical scaling to handle workload increases. Future expansions shall support up to 10,000 users and integration with cloud services.
- **Interoperability**
  The system shall integrate with existing educational tools and platforms. It shall support standard data exchange formats (e.g., JSON, XML) and provide APIs for third-party integration.
- **Legal and Compliance**
  The system shall comply with relevant data protection regulations (e.g., GDPR). Copyrighted content shall not be used without proper licensing, and institutional policies on academic integrity shall be respected.

3.4 Constraints

a. Resource Constraints

- **Financial Resources:** The development and operation of the Quiz AI system must stay within the allocated budget, covering expenses such as development, AI model training, testing, deployment, and user training.
- **Human Resources:** Availability of team members skilled in software development, AI/ML, database management, and UX design may impact the project timeline.
- **Time Constraints:** The project must meet milestones for development, testing, and final submission within the semester or allocated timeline.

b. Technological Constraints

- **Platform Compatibility:** The system must work across modern web browsers on desktops, laptops, and mobile devices for students, instructors, and admins.
- **Database Performance:** The system must handle multiple concurrent quiz attempts, content uploads, and result queries efficiently.
- **AI and API Dependence:** Features such as quiz generation and content classification rely on AI models and stable API integrations.

- **Security Measures:** The system must use encryption, secure authentication, and input validation to protect user data and prevent unauthorized access.

c. External Constraints

- **User Availability:** Successful adoption depends on students, instructors, and admins actively using the system and engaging with training materials.
- **Data Quality:** AI-based quiz generation requires accurate, well-formatted content to produce meaningful quizzes.
- **Network Reliability:** Features like quiz attempts, content uploads, and notifications depend on stable internet connectivity.
- **Legal Compliance:** The system must respect data privacy regulations (like GDPR) and institutional rules regarding academic integrity.

d. Operational Constraints

- **Authentication Protocols:** Secure login and session management are required to prevent unauthorized access.
- **Data Synchronization:** Quiz attempts, results, and content uploads must sync across devices in real time.
- **Error Handling:** The system must provide clear error messages for invalid input, failed uploads, or unsuccessful quiz generation.
- **Audit Trail Maintenance:** Logs of quiz creation, user attempts, and system errors must be maintained for review and quality assurance.

3.6 Software Diagrams

A. Usecase Diagram

- The Use Case Diagram provides a high-level view of the system's functionalities and interactions with external actors.

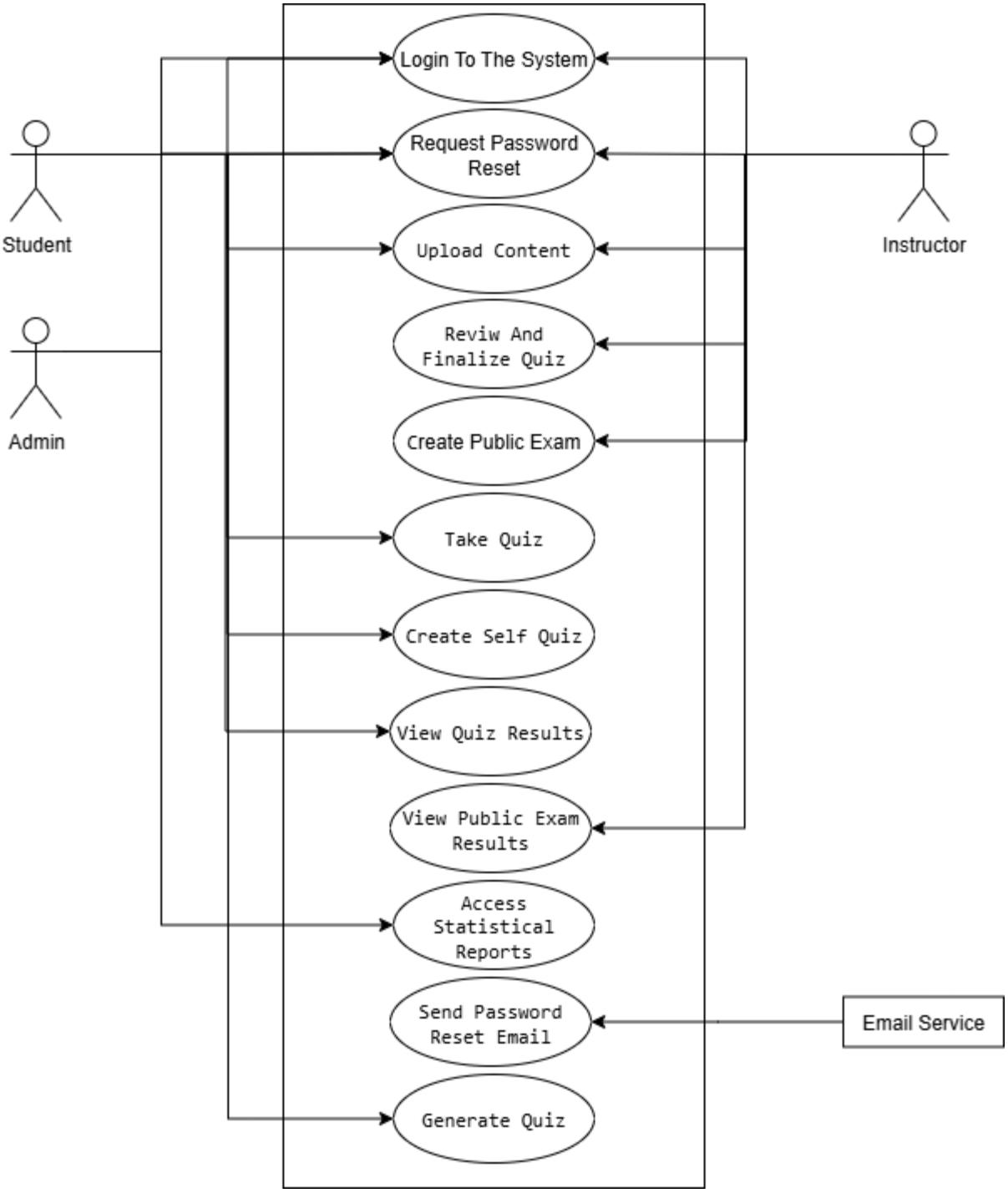


Figure 3.2: Use Case Diagram For Quiz AI System.

B. Context Diagram (Equivalent to DFD LV.0)

- The context diagram is an overview of an organizational system that shows the system boundaries, external entities that interact with the system, and the major information flows between the entities and the system.

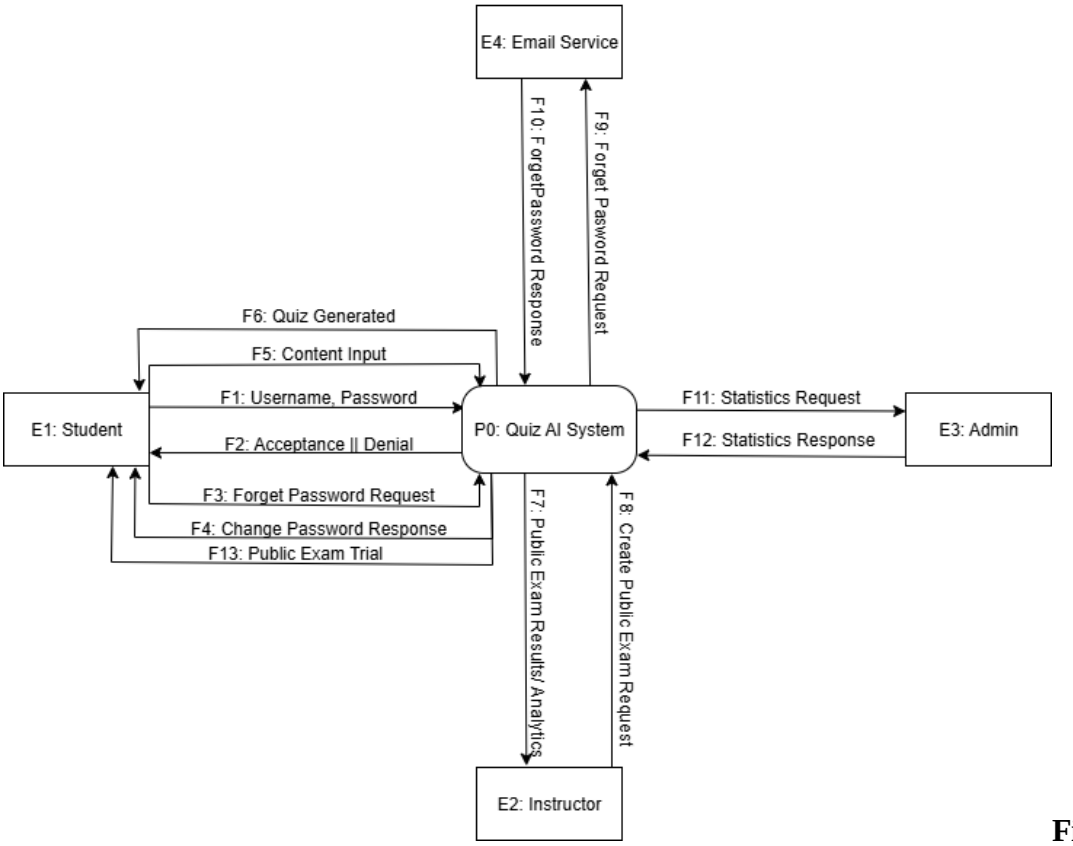


Figure 3.3: Context diagram of the proposed system.

Figure 3.3 presents the context diagram of the proposed Quiz AI System. This diagram illustrates the system's interactions with external entities, such as admin, instructors, and students. It provides a clear representation of data flow between the system and these actors, highlighting the system’s role in managing, and creating the quizzes by the system.

C. Data Flow Diagram Level 1

A Level 1 DFD shows the **detailed flow of data within the system**, breaking down major processes from Level 0 into more specific sub-processes. It illustrates **how data moves between actors, processes, and data stores**, helping readers understand the **internal workings of the system**. Level 1 DFDs provide clarity on **data handling, storage, and processing**, making it easier to follow each functional part of the system.

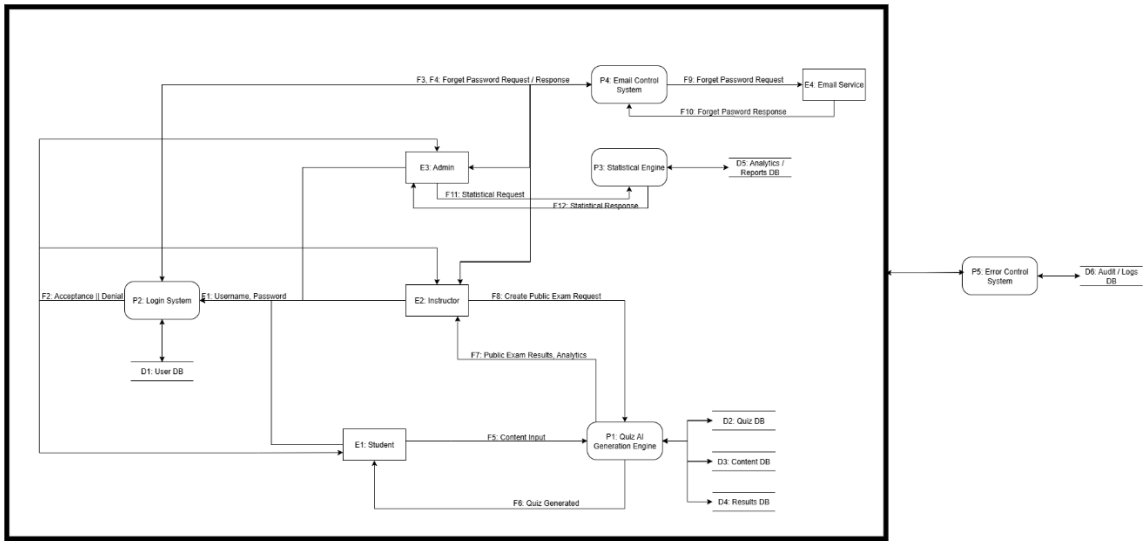


Figure 3.4: Level 1 DFD of the proposed system.

Figure 3.4 illustrates the Level 1 Data Flow Diagram (DFD) of the proposed Quiz AI System. This diagram depicts the main processes of the system, such as quiz creation, generating statistics, and error control. It shows how data flows between various processes, the database, and external entities, providing a clear understanding of the system's detailed operations and its role in achieving quality assurance objectives at the Quiz AI System.

D. Sequence Diagrams:

A sequence diagram is a type of UML diagram that shows **how objects and actors interact over time** to complete a process. It illustrates the **order of messages**, **activations**, and **conditions** between actors (like users) and system components (like controllers or databases).

It helps readers understand **step-by-step workflows**, such as login, content upload, quiz generation, or exam attempts, by showing **who does what and in what order**.

- Login Sequence Diagram:

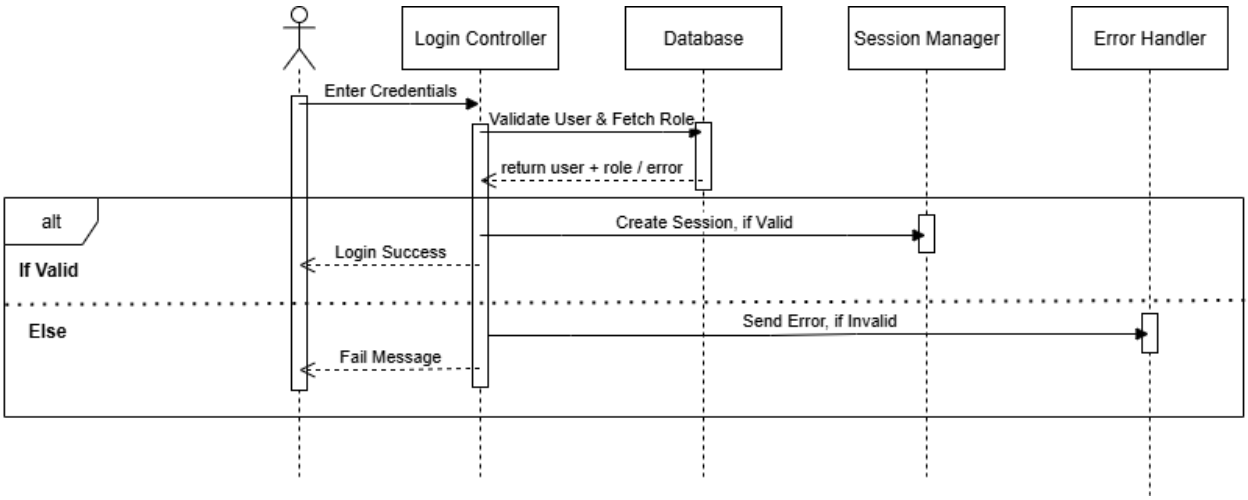


Figure 3.5: Login Sequence Diagram.

The login sequence diagram illustrates the step-by-step process a user follows to access the system. It shows how the **user interacts with the Login Controller**, how the system **validates credentials and fetches the user's role from the database**, and how it **creates a session or returns an error**. Conditional fragments indicate the alternative flows for **valid and invalid logins.**

- Student Attempt Exam Sequence Diagram:

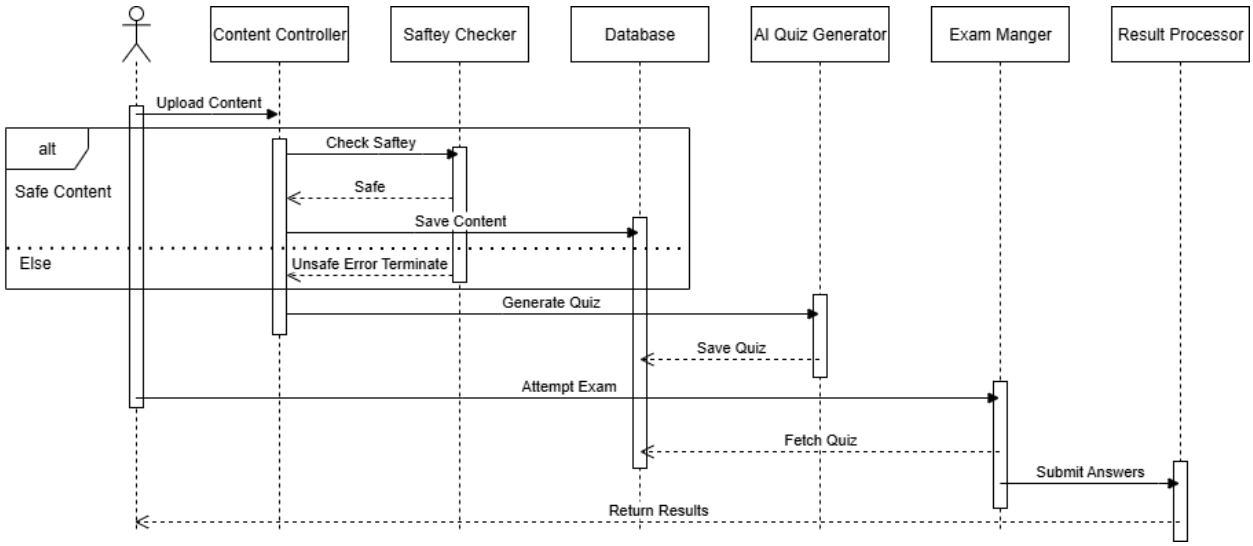


Figure 3.6: Student Attempt Exam Sequence Diagram.

This sequence diagram shows how a student interacts with the system to **upload content, generate a quiz, attempt an exam, and receive results**. It illustrates the flow between the **student, content controller, safety checker, database, AI quiz generator, exam manager, and result processor**. Conditional fragments represent **alternative flows**, such as unsafe content or late exam submissions. The diagram provides a clear view of how the system manages content, generates quizzes, handles exam attempts, and returns scores and feedback to the student.

- Instructor Quiz Management Sequence Diagram

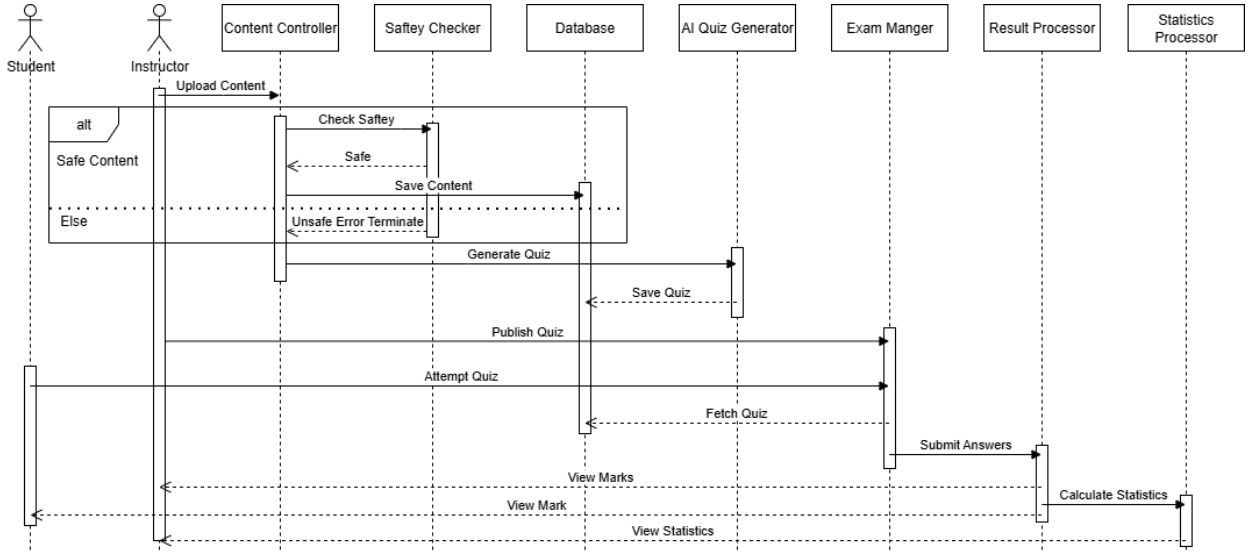


Figure 3.7: Instructor Quiz Management Sequence Diagram.

This sequence diagram illustrates how an instructor interacts with the system to **upload content, ensure its safety, generate quizzes via AI, publish them, and monitor student performance**. It shows the flow between the **instructor, content controller, safety checker, database, AI quiz generator, exam manager, result processor, and stats processor**. Conditional fragments capture alternative flows, such as **unsafe content or unapproved quizzes**. The diagram clearly depicts how the instructor manages quiz creation, publication, and accesses **marks and statistical reports** for evaluation.

3.7 Database Design

The database design for this project is structured to efficiently store, manage, and retrieve all necessary data related to users, courses, quizzes, and AI-generated content. An Entity-Relationship Diagram (ERD) has been created to visually represent the relationships between different entities, such as Students, Instructors, Admins, Questions, and Knowledge Bases. Each entity includes well-defined attributes, and the relationships ensure data integrity and consistency across the system. The design adheres to the **Third Normal Form (3NF)**, eliminating redundancy and ensuring that each piece of data is stored in only one place, which optimizes query performance and supports scalability and maintainability for future expansions.

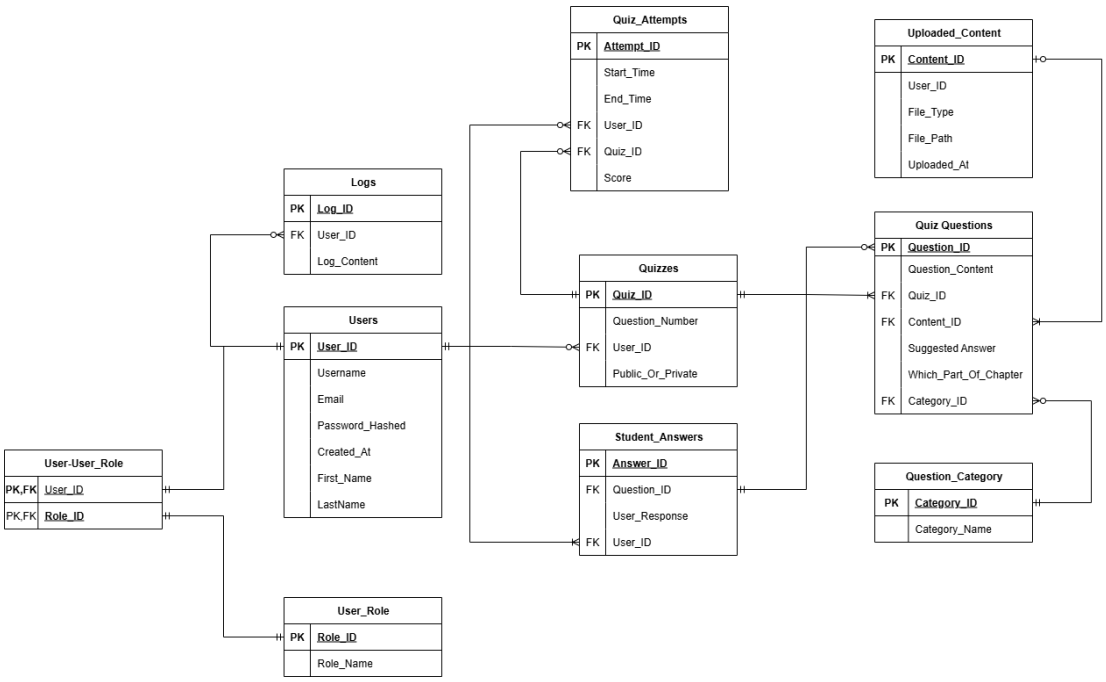


Figure 3.8: Quiz AI Entity Relationship Diagram.

CHAPTER 4: METHADOLOGY

4.1 Introduction

This chapter outlines the methodology followed in the development of the Quiz AI system. It explains the overall approach used to design, implement, and evaluate the system, highlighting the frameworks, models, and tools that guided the process. The chosen methodology emphasizes flexibility, adaptability, and continuous improvement, ensuring that the project can respond effectively to evolving requirements and feedback. Industry-recognized practices such as Agile software development, layered architecture, and structured testing were adopted to provide a systematic framework for building a reliable and scalable solution. Each section of this chapter discusses a key aspect of the methodology, from the project development approach and system architecture to tools, technologies, data collection, testing, and user interface design.

4.2 Project Development Approach

Our project follows the Agile Software Development Model. We chose this approach because it emphasizes flexibility, iterative development, and continuous improvement, which suit our project's dynamic nature. The system we are building includes user authentication, content upload, AI-based quiz generation, public exam creation, and analytics. It needs frequent adjustments based on feedback from stakeholders and changing requirements. Agile lets us make these changes without disrupting the overall development process.

Agile development works in short iterative cycles called sprints. Each sprint produces a functional part of the system. After each iteration, we gather feedback from stakeholders and make necessary refinements. This helps ensure that the final product is functional and meets user needs and expectations.

The advantages of Agile for this project include:

- Adaptability: We can modify or improve features and user interfaces during development without needing complete redesigns or causing significant delays.
- Early Delivery: The system is built step by step, allowing us to deliver functional parts like user login, content upload, or quiz generation early. This enables testing and user validation sooner.
- Customer Collaboration: Agile encourages close communication with stakeholders throughout development. This ensures the product evolves based on actual user requirements rather than assumptions.
- Risk Management: Continuous testing and incremental delivery help reduce the risk of major failures later on. Issues are identified and resolved early.

The project will progress through the following phases, repeated in cycles:

1. **Planning**: Define the overall scope of the system, prioritize features, and create a backlog of tasks for each sprint.
2. **Design**: Develop detailed user interface (UI) mockups, database architecture, and integration diagrams to clarify the technical structure.
3. **Implementation**: Code the core functionalities of the system, such as authentication, content upload, AI quiz generation, exam management, and analytics modules. Each feature is built incrementally during sprints.
4. **Testing**: Conduct rigorous testing at multiple levels. This includes unit testing for individual components, integration testing for module interactions, and user acceptance testing (UAT) to validate requirements.
5. **Deployment**: Release working versions of the application to a staging environment for stakeholder review, followed by full deployment to production.
6. **Maintenance & Continuous Improvement**: Address any issues reported post-deployment, release updates, add new features based on feedback, and enhance system performance and usability.

4.3 System Architecture

The layered architectural style was chosen for this software system because it organizes functions into separate layers. This structure promotes the separation of concerns by dividing the system into independent layers, each handling a specific group of related tasks. This setup improves maintainability, scalability, and testability by allowing changes within one layer without greatly affecting the others. It also makes the code easier to read and establishes a clear dependency direction, where upper layers depend on lower layers, which reduces coupling. These benefits make the layered architecture an excellent choice for complex systems that need clarity, modularity, and long-term flexibility.

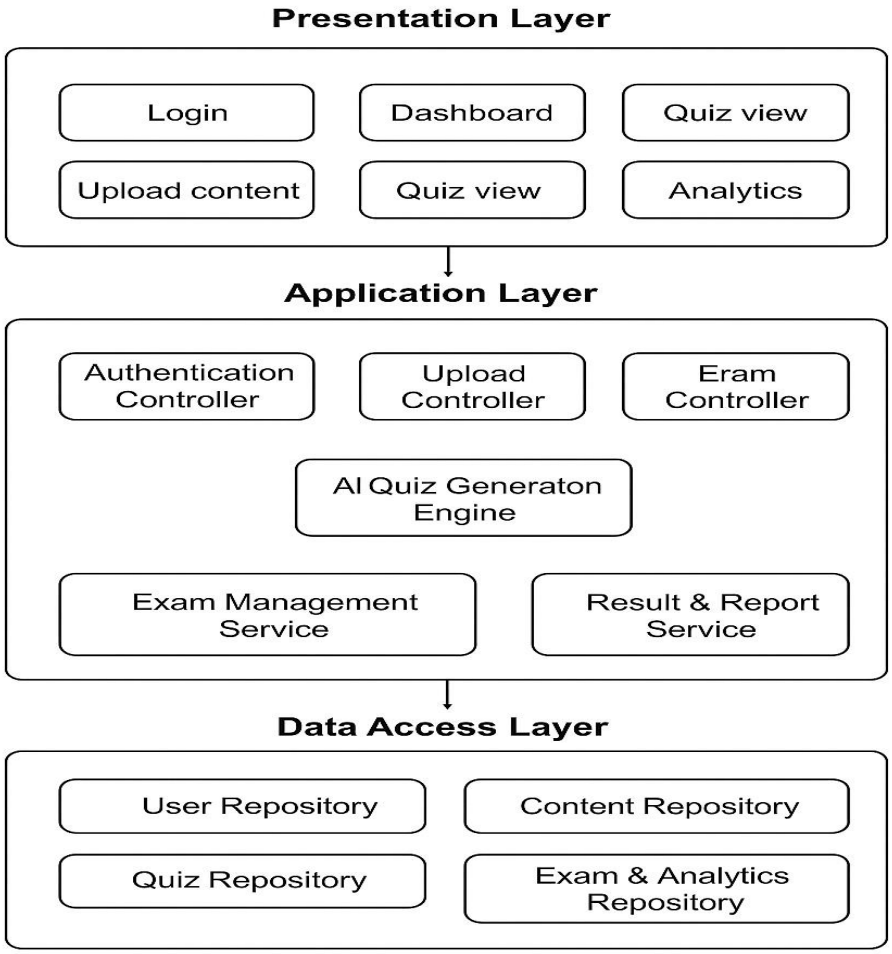


Diagram 4.1: Quiz AI Software Architecture.

This diagram illustrates the three-tier layered architecture of our Quiz System,

dividing the application into logical and distinct layers: the Presentation Layer, the Application Layer, and the Data Access Layer. This structured approach enhances modularity, maintainability, and scalability.

1. **Presentation Layer:** This is the topmost layer, responsible for the user interface and handling all user interactions. It presents information to the user and captures their input. Key components here include:
    - **Login:** Manages user authentication and session initiation.
    - **Dashboard:** Provides an overview and quick access to system functionalities.
    - **Quiz view:** Displays quizzes for users to take and review.
    - **Upload content:** Allows administrators or content creators to add new quiz materials.
    - **Analytics:** Presents insights and reports based on quiz data.
2. **Application Layer:** Often referred to as the Business Logic Layer, this central tier contains the core functionality and business rules of the Quiz System. It processes requests from the Presentation Layer and interacts with the Data Access Layer. Components include:
    - **Authentication Controller:** Manages user login, registration, and session validity.
    - **Upload Controller:** Handles the processing and validation of uploaded content.
    - **Exam Controller:** Orchestrates the creation, management, and delivery of exams/quizzes.
    - **AI Quiz Generation Engine:** A core component responsible for intelligently generating quizzes (e.g., from uploaded content).
    - **Exam Management Service:** Provides services related to scheduling, administering, and monitoring exams.
    - **Result & Report Service:** Processes quiz submissions, calculates scores, and generates performance reports.

3. **Data Access Layer:** This bottom layer is responsible for interacting with the database and managing all data persistence operations. It abstracts the underlying database technology from the Application Layer. Components within this layer include:
    - **User Repository:** Handles CRUD (Create, Read, Update, Delete) operations for user data.
    - **Content Repository:** Manages the storage and retrieval of quiz questions, multimedia, and other content.
    - **Quiz Repository:** Deals with saving, loading, and managing quiz definitions and structures.
    - **Exam & Analytics Repository:** Stores data related to conducted exams, user attempts, scores, and information used for analytics.

4.4 Tools and Technologies

Programming Languages

C#

It is reliable, efficient, and well-suited for building robust backend systems or desktop applications. It integrates seamlessly with .NET libraries, offering strong type safety and excellent performance.

It reduces runtime errors, provides scalability for large systems, and allows rapid development of stable APIs or services that power the application.

Python

Python is widely used in machine learning and AI development because of its clean syntax, extensive libraries, and community support.

React

React is rather ideal for creating web interfaces with a component-based architecture that promotes reusability.

It also allows a smoother user experience than some alternatives, reduces development effort for the frontend, and ensures scalability as the application grows.

Frameworks

QLoRA

QLoRA (Quantized Low-Rank Adaptation) fine-tunes large language models efficiently by lowering memory usage without compromising accuracy.
It also makes it affordable and practical to adapt advanced AI models to our project requirements.

LM Studio

LM Studio provides an extensive and scalable easy to deploy LLM server with support to all GGUF format LLMs

---

Development Tool

Microsoft Visual Studio Code

Why chosen: VS Code is a lightweight yet powerful code editor with broad language support, integrated debugging, Git control, and an extensive extension marketplace.
How it helps the project: It increases developer productivity, streamlines workflows across different programming languages, and enables rapid switching between backend, AI, and frontend development.

SQL Server Management Studio

SSMS provides a comprehensive and user-friendly interface for managing SQL Server databases. It's a key tool for database administrators and developers, offering a visual way to design, query, and maintain databases. Its powerful features streamline tasks like performance monitoring, security management, and data backup, ensuring the database is stable, secure, and performant.
As our backend is powered by C#, we'll be using a SQL Server database for data storage. SSMS is the standard tool for working with this database. It allows us to easily design the database schema, write and debug complex queries, and manage the database's health and security. This is crucial for building a scalable and reliable backend that can handle our application's data needs efficiently.

Overall Impact

- Backend stability and scalability (C# + VS Code)
- Easy AI integration (Python + LM Studio + meta-llama-3.1-8b-instruct Q5_K_M)
- Modern, user-friendly frontend (React)

4.5 Data Collection and Analysis

The data collection process for the proposed Quiz AI system focuses on preparing educational resources that can be used to test the performance of multiple AI models using LM Studio to figure the best option for fast and accurate replies.

4.6 Testing Strategy

Complete, multi-layered testing will be employed to certify that the Quiz AI is accurate, reliable, and meets all functional and non-functional requirements. The strategy will use automated and manual testing techniques throughout the development process.

4.6.1 Testing Methodology

The testing approach will be centered on Unit Testing and Burp Suite.

Unit Testing:

- **Objective**: To confirm every single method, function, and class operates correctly independently of the other system parts. It is highly recommended to detect logic bugs as early as possible during the development cycle.
- **Method**: Automated tests will be written by developers for the lowest level of units of code. These are testing functions for text processing, quiz question generation, input validation, and data formatting. External dependencies such as calls to AI APIs or file system calls will be mocked out with mock objects so that tests are fast, consistent, and isolated.

- **Tool**: The project will utilize the unit test framework, Python's default unit test tool. The framework provides a solid base upon which to build and run a solid set of tests.

Penetration Testing & Security Scanning:

- **Purpose**: To actively scan and fix security vulnerabilities in the application before it is deployed.
- **Procedure**: The deployed application will be scanned for typical web vulnerabilities systematically. This includes testing injection flaws (SQLi, XSS), broken authentication, insecure direct object reference, and other on the OWASP Top 10 list.
- **Tool**: The primary tool for manual and automated security testing will be Burp Suite. It will be used to intercept, observe, and manipulate HTTP/S requests between the server and client in an effort to find security vulnerabilities.

4.6.2 Error Identification and Correction

A transparent process will be employed to deal with problems that are found:

- **Identification**: Problems will be identified by automated unit test failures and security scans.
- **Triage & Logging**: Problems from functional bugs to severe security bugs will be logged, prioritized for repair, and sorted based on severity.
- **Resolution**: Bugs will be repaired by developers, while security bugs will be repaired using a patch. Security patches will be tested to ensure that they do not introduce new vulnerabilities.
- **Validation**: The unit test suite and The Burp Suite will be re-run to verify that the fix is working without causing any regressions.

4.6.3 Test Cases and Validation

The following table provides examples of test cases for both unit and security testing.

TEST CASE ID	TYPE	DESCRIPTION	INPUT EXPECTED	OUTCOME
TC-U-101	Unit Test	Test input validation for file upload.	A file with type .exe	Function raises a ValueError.
TC-U-102	Unit Test	Test JSON formatting of a quiz question.	Question data object.	A valid JSON string with correct fields.
TC-SEC-201	Security (Burp Suite)	Test for SQL Injection in login form.	username: admin' --	Returns a generic error message, not a database error.
TC-SEC-202	Security (Burp Suite)	Test for Cross-Site Scripting (XSS) in quiz output.	<script>alert('test')</script>	Input is sanitized; script tags are not executed.

4.7 User Interface Design (Prototype)

The User Interface (UI) design prototype for this project provides a clear and intuitive layout that guides users through the system's features, including course selection, quiz taking, and AI-assisted content interaction. The prototype emphasizes usability and accessibility, with consistent visual elements, logical navigation paths, and responsive design principles to ensure a smooth experience across devices. It serves as a preliminary model to gather feedback from stakeholders, allowing for iterative improvements before the final implementation.

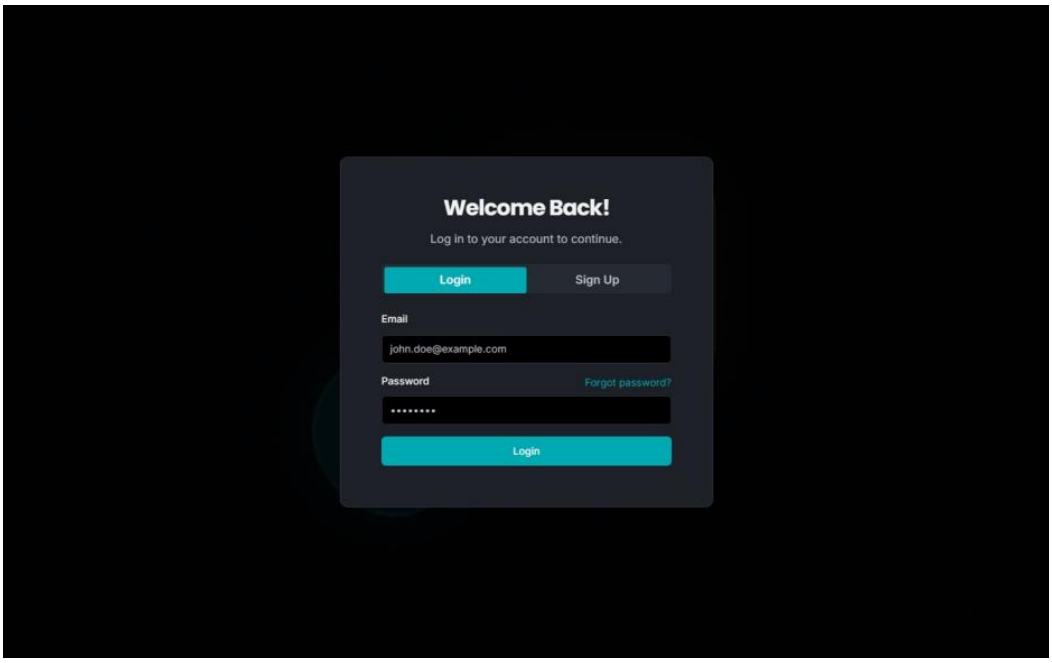


Figure 4.3: Login Screen.

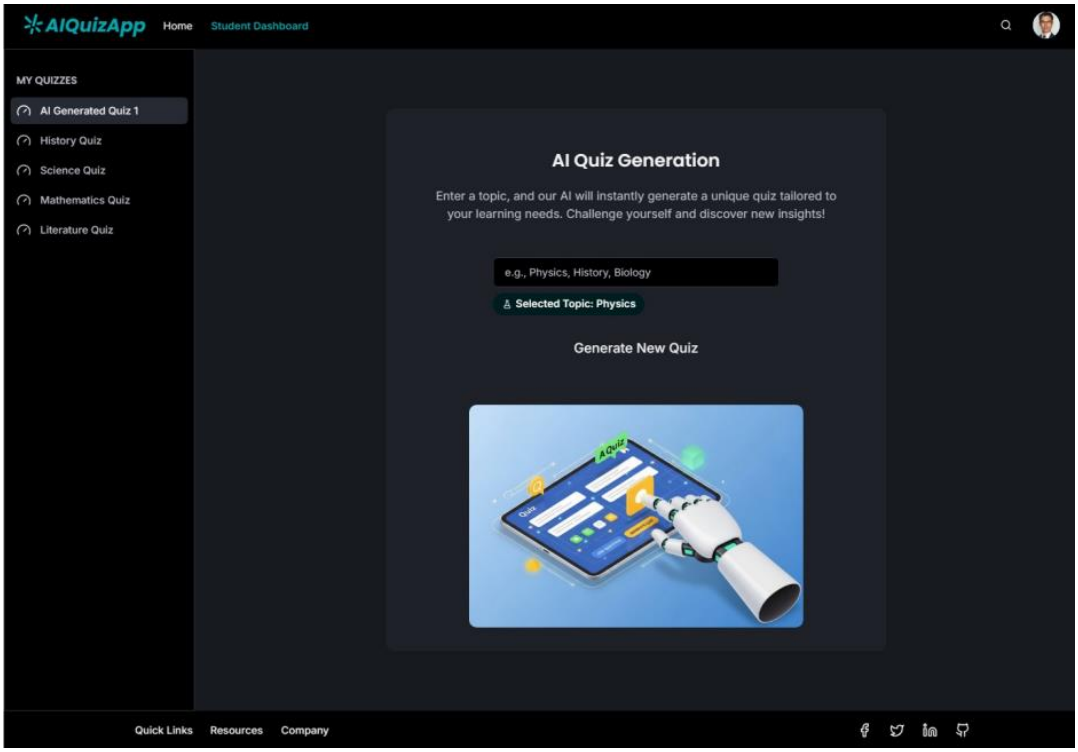


Figure 4.4: Quiz Generation Page.

5.1 Description of Implementation

5.1.1 System Deployment and Transition

The **QUIZ AI** platform will be deployed as a centralized web application. Since this is a greenfield project, no legacy data migration is required. The transition into an operational system will follow a **"Big Bang" deployment model**, where the system becomes fully available to students immediately upon the successful configuration of the host server.

Deployment Workflow:

1. **Environment Setup:** Configuration of the host PC with the necessary runtime environments (.NET for the backend and a web server for the React frontend).
2. **AI Model Integration:** Loading the fine-tuned AI model into the backend infrastructure.
3. **Site Launch:** Pointing the web domain or IP address to the host machine, making the "QUIZ AI" portal accessible to users.

5.1.2 Major Tasks and Components

The implementation effort is divided into the following key tasks:

- **Backend Deployment:** Hosting the **ASP.NET (C#)** API and ensuring the fine-tuned AI logic is responding to requests.
- **Frontend Integration:** Deploying the **React.js** build to the web server to provide the user interface.
- **AI Parameter Configuration:** Final calibration of the AI settings to ensure quiz generation matches user-selected settings and material accuracy.
- **Security Hardening:** Implementing authentication protocols and firewall rules on the host PC to prevent unauthorized access to the backend.

5.1.3 Resource Requirements

To support the implementation and ongoing operation of QUIZ AI, the following resources are required:

Resource Category	Description
Hardware	A dedicated "Main Server" PC provided by the development team to host the database, backend, and frontend.
Software	.NET Runtime, Node.js (for build), a Database Management System, and the fine-tuned AI model files.
Facilities	A stable power supply and high-speed internet connection for the host PC to ensure website uptime.
Personnel	The original student development team will act as the primary **Support Team** for maintenance and troubleshooting.

5.2 Programming Language and Technology

This section identifies the technical stack and tools used to implement the QUIZ AI system. The solution utilizes a hybrid hosting model where the frontend is deployed to a public web server, while the backend and database remain on a local secure host.

5.2 .1Core Programming Languages & Frameworks

Name	Code / Acronym	Type	Description
C#	C#	COTS (Open Source)	Primary language for backend logic and AI orchestration.
ASP.NET Core	ASP.NET	COTS (Open Source)	Framework used to build the RESTful API and manage security middleware.
React.js	React	COTS (Open Source)	Frontend library used to build the interactive student interface.
Transact-SQL	T-SQL	COTS	Scripting language used for database queries and management.

5.2 .2 Solution-Specific Software & AI Models

These components represent the custom-built "State-specific" parts of the project.

Name	Acronym	Type	Role
Fine-tuned TinyLlama	TinyLlama-1B	**State-specific**	Small Language Model (SLM) fine-tuned on 1B tokens for specialized quiz generation.
QUIZ AI DB	DB	**State-specific**	SQL Server database containing custom schemas for user materials and quiz data.
SQL Server	MSSQL	COTS	Relational Database Management System (RDBMS) hosted on the main server PC.

5.2 .3 Implementation & Security Technologies

Technology	Category	Usage
JSON Web Tokens	**JWT**	Used for secure, stateless authentication between the React frontend and C# backend.
Swagger / OpenAPI	**Testing Tool**	Integrated into the ASP.NET backend to facilitate API testing and documentation.
Base Model API	**Interface**	The bridge used by the C# backend to communicate with the locally running TinyLlama model.

5.2.4 Support & Development Software

Software Name	Purpose
Visual Studio (VS)	Primary IDE for C# backend development and SQL management.
VS Code	Used for React.js frontend development and styling.
Localhost Environment	Facilitates the execution of the backend and AI model without public cloud exposure.
Production "dist" Folder	The optimized React build folder deployed to the separate hosting server for the UI.

Technical Note on Implementation

The system follows a **Decoupled Architecture**. The user interface is served via a public URL (hosting the dist folder), which securely calls the backend API hosted on the "Main Server" PC. Traffic is protected using **JWT**, ensuring that only authorized users can trigger the AI quiz generation.

5.3 part of implementation

5.3.1 Data-Back Layer

The Data-Back Layer is responsible for all interactions with the database. This layer communicates with the database exclusively through **stored procedures**, which are executed at the database level.

Using stored procedures helps ensure **database integrity and security**, as queries are processed internally within the database rather than being sent as dynamic string-based queries from the C# application. This approach reduces the risk of SQL injection attacks and enforces a controlled access pattern to the database.

In the following section, we introduce and explain some of the stored procedures that were designed and used within this layer.

One of the most frequently used stored procedures in the system is **SaveGeneratedQuiz**. This procedure is responsible for creating a new quiz and storing it in the database. It receives several parameters, including the **User ID**, **Quiz Title**, **File Type**, **Questions JSON**, and **Quiz ID**, which together define the quiz content and its metadata.

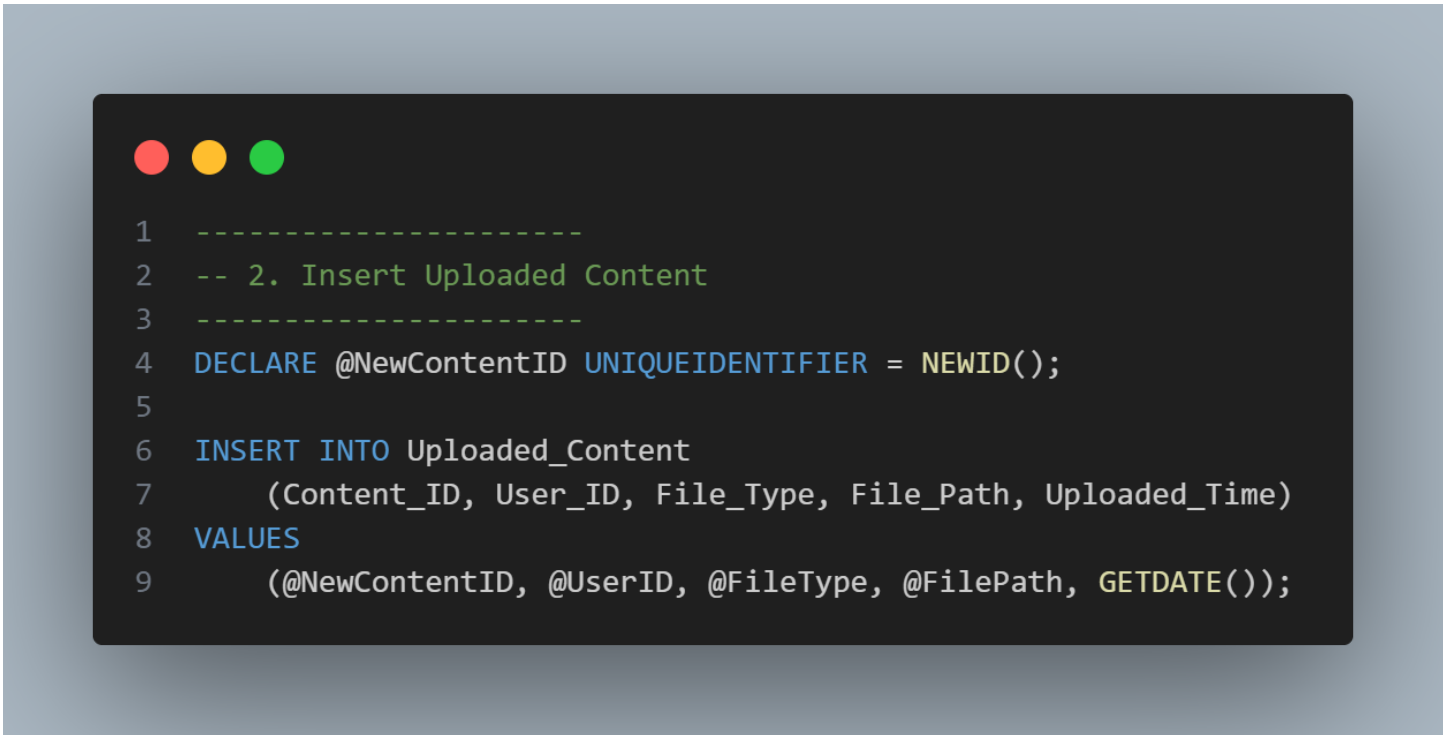
This section of the procedure is the **header**, where all required input parameters are declared. These parameters provide the necessary information for the procedure to correctly generate and persist the quiz data in the database.

```sql
ALTER PROCEDURE [dbo].[SP_SaveGeneratedQuiz]
    @UserID UNIQUEIDENTIFIER,
    @QuizTitle NVARCHAR(255),
    @FileType INT,
    @FilePath NVARCHAR(500),
    @QuestionsJson NVARCHAR(MAX),
    @QuizID UNIQUEIDENTIFIER = NULL -- 1. Made optional by defaulting to NULL
AS
BEGIN
    SET NOCOUNT ON;
```

The first part of the procedure is responsible for determining the **Quiz ID**. If the quiz is being created for the first time, a new **Quiz ID** is generated. Otherwise, the procedure uses the **Quiz ID** provided as an input parameter, allowing an existing quiz to be updated or reused.

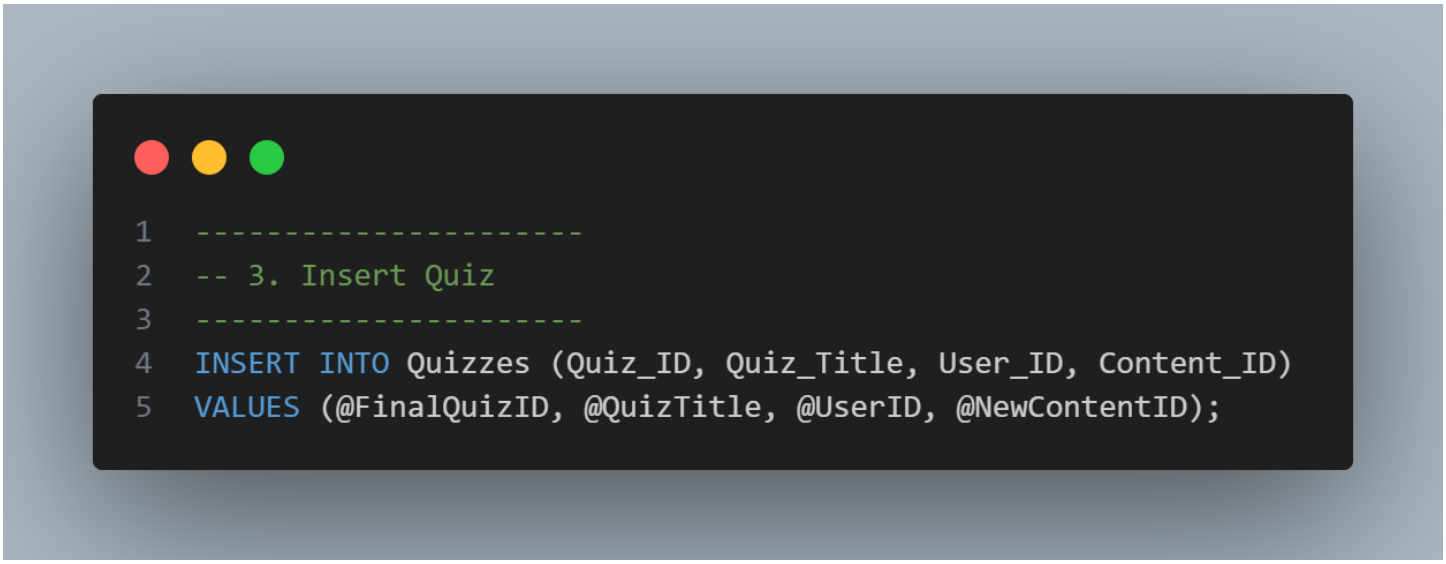
```sql
----------------------
-- 1. Determine the Quiz ID to use
----------------------
-- If the user passed a QuizID, use it. Otherwise, generate a new one.
DECLARE @FinalQuizID UNIQUEIDENTIFIER = ISNULL(@QuizID, NEWID());

```

The second part of the procedure is responsible for inserting the information related to the uploaded content, which is stored on the disk. This step records the file's metadata in the database, enabling the system to track and associate the uploaded content with the corresponding quiz.



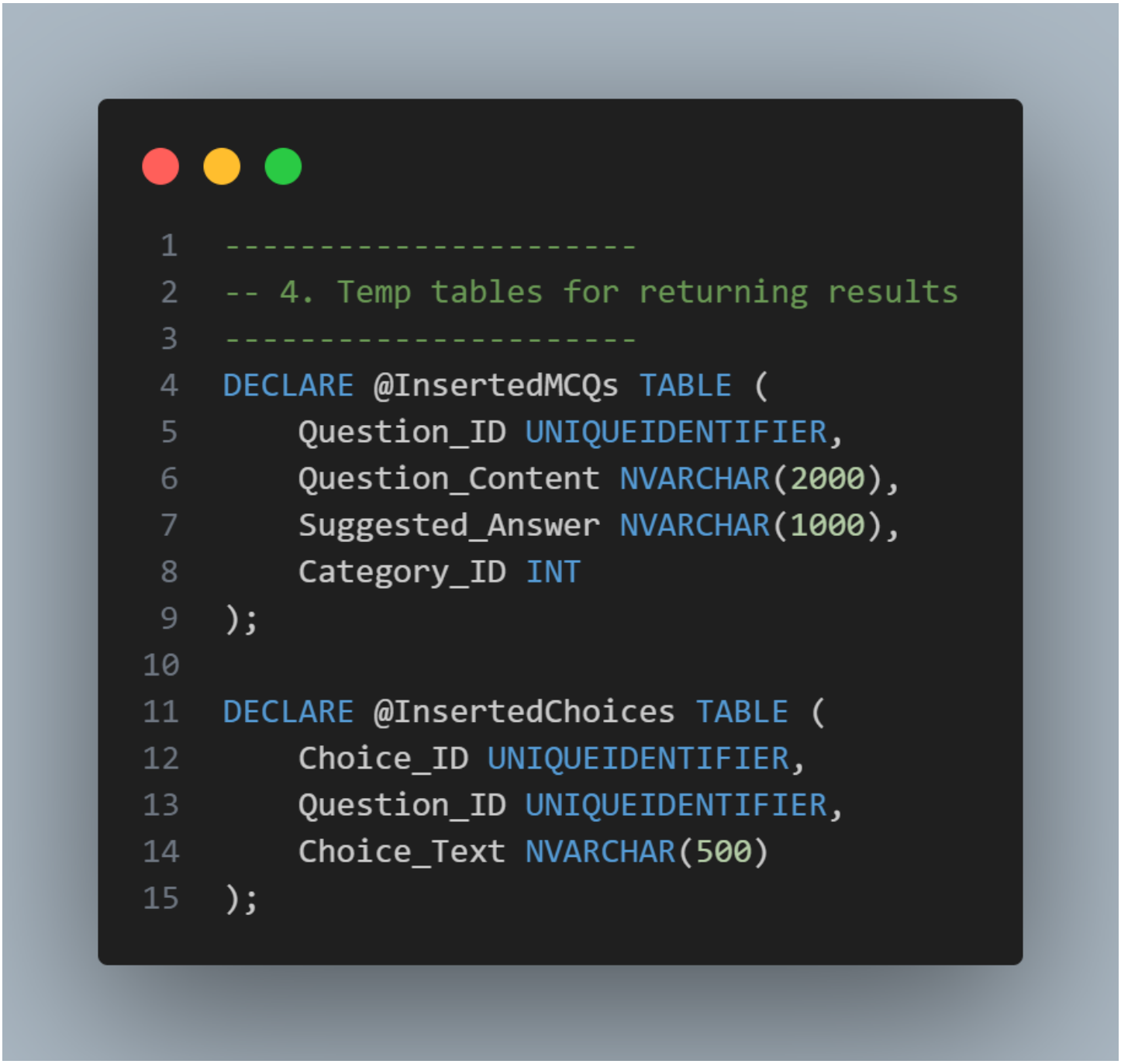
```
----------------------
-- 2. Insert Uploaded Content
----------------------
DECLARE @NewContentID UNIQUEIDENTIFIER = NEWID();

INSERT INTO Uploaded_Content
    (Content_ID, User_ID, File_Type, File_Path, Uploaded_Time)
VALUES
    (@NewContentID, @UserID, @FileType, @FilePath, GETDATE());
```

The third part of the procedure focuses on creating a new quiz by inserting the quiz information into the **Quizzes** table. This step stores the quiz's core details, such as its title, type, and associated identifiers, ensuring that the quiz is properly registered in the database.



```
----------------------
-- 3. Insert Quiz
----------------------
INSERT INTO Quizzes (Quiz_ID, Quiz_Title, User_ID, Content_ID)
VALUES (@FinalQuizID, @QuizTitle, @UserID, @NewContentID);
```

The fourth section creates a **temporary table** to store the results that will be returned to the backend (C#) and subsequently to the frontend. This temporary table holds the required **unique identifiers**, allowing the system to retrieve all necessary data without reopening an additional database connection. As a result, the entire operation is completed within a **single database connection**, improving performance and reducing overhead.



```
----------------------
-- 4. Temp tables for returning results
----------------------
DECLARE @InsertedMCQs TABLE (
    Question_ID UNIQUEIDENTIFIER,
    Question_Content NVARCHAR(2000),
    Suggested_Answer NVARCHAR(1000),
    Category_ID INT
);

DECLARE @InsertedChoices TABLE (
    Choice_ID UNIQUEIDENTIFIER,
    Question_ID UNIQUEIDENTIFIER,
    Choice_Text NVARCHAR(500)
);
```

The fifth part of the procedure is responsible for handling the **JSON data** received from the backend, which originates from the AI model. In this section, the JSON content is parsed and processed to extract the quiz questions.

Based on the extracted data, the quiz questions are inserted into the relevant database tables, along with their associated answer choices. This section is specifically designed to handle **multiple-choice questions (MCQs)** only.

```sql
----------------------
-- 5. Insert MCQs
----------------------
DECLARE
    @Q NVARCHAR(2000),
    @A NVARCHAR(1000),
    @O NVARCHAR(MAX),
    @QuestionID UNIQUEIDENTIFIER;

DECLARE MCQ_CURSOR CURSOR FAST_FORWARD FOR
SELECT
    JSON_VALUE(q.value, '$.question'),
    JSON_VALUE(q.value, '$.answer'),
    q.value
FROM OPENJSON(@QuestionsJson, '$.mcq_questions.questions') q;

OPEN MCQ_CURSOR;
FETCH NEXT FROM MCQ_CURSOR INTO @Q, @A, @O;

WHILE @@FETCH_STATUS = 0
BEGIN
    SET @QuestionID = NEWID();

    INSERT INTO Quiz_Questions
        (Question_ID, Question_Content, Quiz_ID, Content_ID, Suggested_Answer, Category_ID)
    VALUES
        (@QuestionID, @Q, @FinalQuizID, @NewContentID, @A, 1);

    INSERT INTO @InsertedMCQs
        (Question_ID, Question_Content, Suggested_Answer, Category_ID)
    VALUES
        (@QuestionID, @Q, @A, 1);

    INSERT INTO MCQChoices
        (Choice_ID, Question_ID, Choice_Text)
    OUTPUT
        inserted.Choice_ID,
        inserted.Question_ID,
        inserted.Choice_Text
    INTO @InsertedChoices
    SELECT
        NEWID(),
        @QuestionID,
        value
    FROM OPENJSON(@O, '$.options');

    FETCH NEXT FROM MCQ_CURSOR INTO @Q, @A, @O;
END

CLOSE MCQ_CURSOR;
DEALLOCATE MCQ_CURSOR;
```

The sixth section is responsible for handling **true/false (T/F) questions**. In this part, the processed data is inserted into the corresponding database tables, ensuring that true/false questions are stored and linked correctly to the associated quiz.

```
----------------------
-- 6. Insert True / False
----------------------
INSERT INTO Quiz_Questions
    (Question_ID, Question_Content, Quiz_ID, Content_ID, Suggested_Answer, Category_ID)
OUTPUT
    inserted.Question_ID,
    inserted.Question_Content,
    inserted.Suggested_Answer,
    inserted.Category_ID
INTO @InsertedMCQs
SELECT
    NEWID(),
    JSON_VALUE(q.value, '$.question'),
    @FinalQuizID,
    @NewContentID,
    JSON_VALUE(q.value, '$.answer'),
    2
FROM OPENJSON(@QuestionsJson, '$.true_false_questions.questions') q;
```

The seventh section is to return the results of the saved quiz, the questions in it, and the choices, with their id's, in separate query parts, I know this isn't the best practice, but when we do it in this part, we prevent opening multiple database connections, which decreases the speed.

```sql
    ----------------------
    -- 7. Return result sets
    ----------------------

    -- Quiz info
    SELECT
        @FinalQuizID   AS Quiz_ID,
        @QuizTitle     AS Quiz_Title,
        @NewContentID  AS Content_ID,
        @FileType      AS File_Type,
        @FilePath      AS File_Path;

    -- All Questions
    SELECT * FROM @InsertedMCQs;

    -- MCQ Choices
    SELECT * FROM @InsertedChoices;

    -- True / False Choices
    SELECT
        Question_ID,
        CAST('11111111-1111-1111-1111-111111111111' AS UNIQUEIDENTIFIER) AS Choice_ID,
        CAST('True' AS NVARCHAR(10)) AS Choice_Text
    FROM @InsertedMCQs
    WHERE Category_ID = 2
    UNION ALL
    SELECT
        Question_ID,
        CAST('00000000-0000-0000-0000-000000000000' AS UNIQUEIDENTIFIER),
        CAST('False' AS NVARCHAR(10))
    FROM @InsertedMCQs
    WHERE Category_ID = 2;

END;
```

In this section, we demonstrate how the stored procedure is handled within the backend, including the processing of its sub-queries.

The first part presents the **function header** and the parameters required by the method. This section also includes basic **data validation** and **error handling** to ensure that invalid input is detected before interacting with the database.

The function is implemented as an **asynchronous method**, allowing the system to continue executing other operations while waiting for the database to process and return the requested data. This approach improves system responsiveness and overall performance.

```
1  public static async Task<GenerateQuizResponseDTO> SaveQuizInfoToDataBaseAsync(Guid userID,
2   QuestionResponse QuizInfo, string FilePath, byte GenerateFlag, Guid? QuizID = null)
3  {
4      if (QuizInfo == null) throw new ArgumentNullException(nameof(QuizInfo));
5      GenerateQuizResponseDTO result = new GenerateQuizResponseDTO();
```

In this section, the database connection is established using a valid connection string retrieved from the **appconfig.json** configuration file. A SQL command object is also initialized to execute the database operation. Both the connection and the command are wrapped within **using statements** to ensure proper resource management and to prevent common human errors, such as forgetting to close database connections.

Finally, the command type is explicitly set to **Stored Procedure**, ensuring that the database operation is executed at the procedure level rather than as a raw SQL query.

```
1  using (SqlConnection con = new SqlConnection(Database._connectionString))
2  using (SqlCommand cmd = new SqlCommand("SP_SaveGeneratedQuiz", con))
3  {
4      cmd.CommandType = CommandType.StoredProcedure;
5
```

This section is dedicated to supplying the parameters required by the stored procedure. It defines and assigns all necessary input values, ensuring that the procedure receives the complete and correct data needed for

execution.

```
// --- Parameters ---
cmd.Parameters.AddWithValue("@UserID", userID);
cmd.Parameters.AddWithValue("@QuizTitle", QuizInfo.filename ?? "Generated Quiz");

string extension = System.IO.Path.GetExtension(QuizInfo.filename) ?? ".unknown";
cmd.Parameters.AddWithValue("@FileType", Database.SelectFileType(extension));
cmd.Parameters.AddWithValue("@FilePath", FilePath);

```

In this section, the class structures are serialized into **JSON format** before being sent to the database. This approach is necessary because **T-SQL does not support complex class or structure types**. By serializing the data into JSON, the stored procedure can parse and process the structured information within the database
layer.

```
// Serialize based on the new structure
string jsonBody = System.Text.Json.JsonSerializer.Serialize(QuizInfo);
cmd.Parameters.AddWithValue("@QuestionsJson", jsonBody);
```

In this section, the **ternary operator** is used to determine whether to send a value or NULL for the **Quiz ID**.

- If the **Generate Flag** is 1, this indicates a **new quiz**, and NULL is sent so that the database generates a new Quiz ID.
- If the **Generate Flag** is 0, this indicates a **regenerated quiz**, and the existing Quiz ID is used to ensure that the same quiz is updated in the database.

This approach allows the procedure to handle both new and regenerated quizzes efficiently while maintaining database consistency.

```
// Logic fix: Pass existing QuizID only if regenerating (Flag 0)
cmd.Parameters.AddWithValue("@QuizID", (GenerateFlag == 0 && QuizID.HasValue) ? (object)QuizID.Value : DBNull.Value);

```

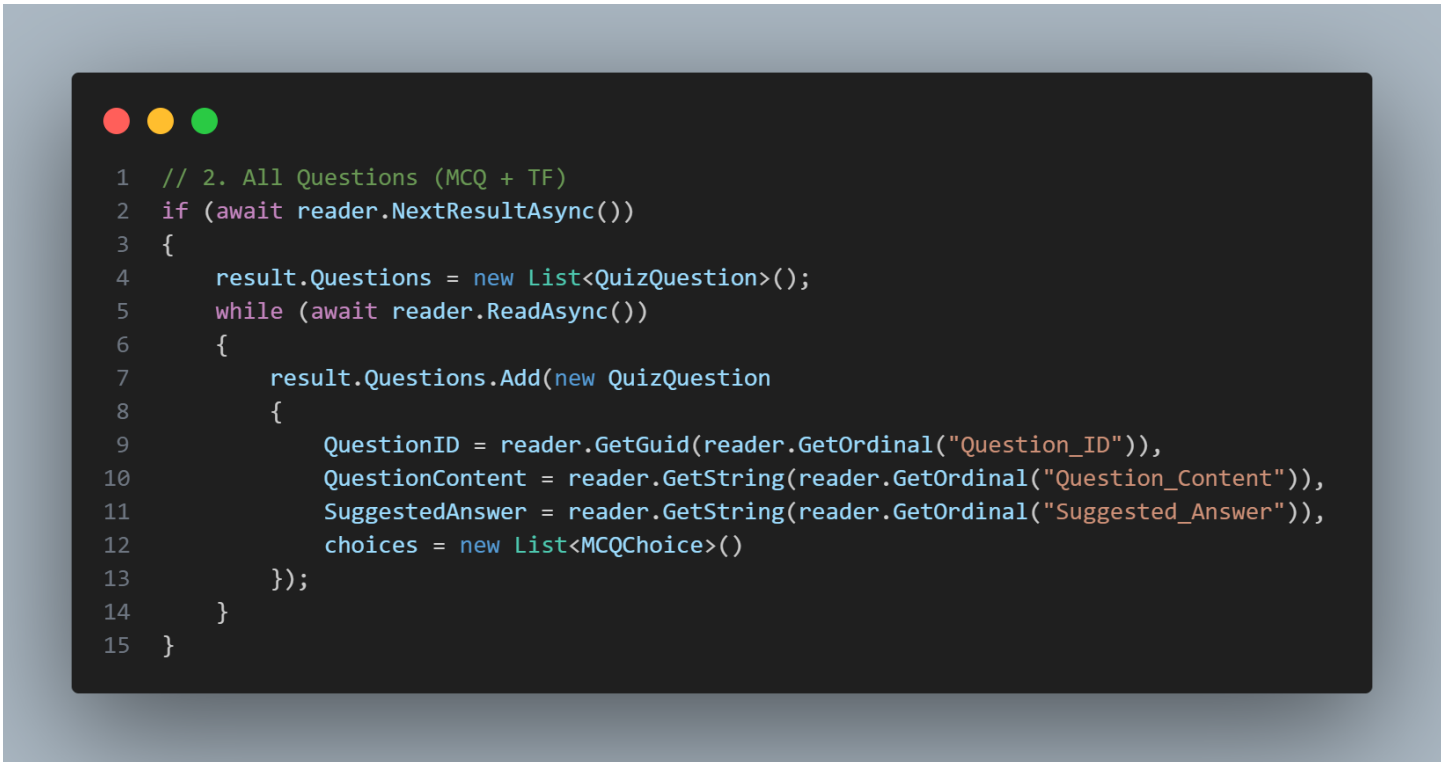
In this section, the method leverages **asynchronous programming** to prevent blocking the system while waiting for the database response. A **SqlDataReader** is used to efficiently handle queries that return **multiple rows**, allowing the system to read data row by row as it is retrieved from the database. This approach improves performance and resource utilization, especially when processing large result sets.

```
await con.OpenAsync();

using (SqlDataReader reader = await cmd.ExecuteReaderAsync())
{
```

In this section, only the **basic quiz information** is retrieved from the first result set. This includes essential details such as the **Quiz ID** and **Quiz Title**, which are then stored in the response object for further processing.

```
// 1. Quiz Basic Info
if (await reader.ReadAsync())
{
    result.QuizID = reader.GetGuid(reader.GetOrdinal("Quiz_ID"));
    result.QuizTitle = reader.GetString(reader.GetOrdinal("Quiz_Title"));
}
```

In this section, all **quiz questions** are retrieved from the second result set using a **while loop** that iterates over the rows returned by the SqlDataReader. Both **multiple-choice (MCQ)** and **true/false (T/F)** questions are processed and added to the response object, preparing them for the next step of associating answer choices.



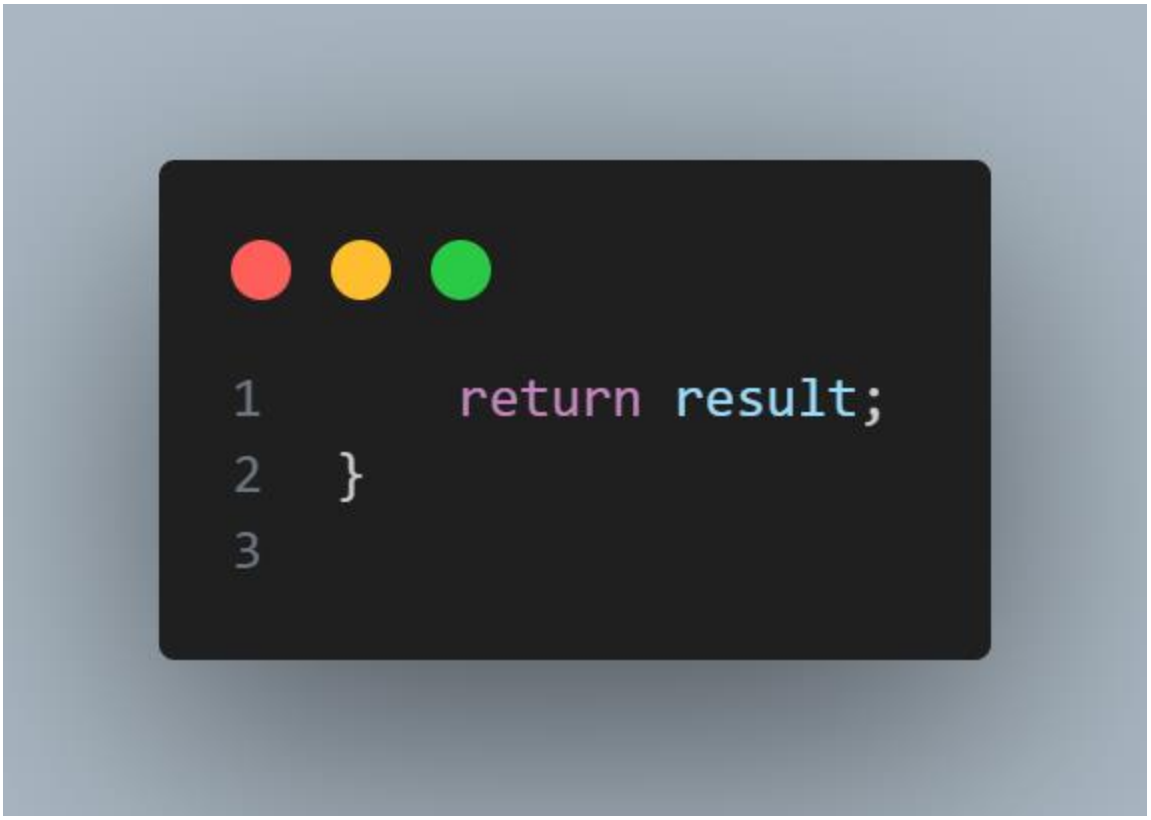
```
// 2. All Questions (MCQ + TF)
if (await reader.NextResultAsync())
{
    result.Questions = new List<QuizQuestion>();
    while (await reader.ReadAsync())
    {
        result.Questions.Add(new QuizQuestion
        {
            QuestionID = reader.GetGuid(reader.GetOrdinal("Question_ID")),
            QuestionContent = reader.GetString(reader.GetOrdinal("Question_Content")),
            SuggestedAnswer = reader.GetString(reader.GetOrdinal("Suggested_Answer")),
            choices = new List<MCQChoice>()
        });
    }
}
```

In this section, the **answer choices** for all questions are retrieved from the database. The process uses the same method as before, iterating over the result sets with **reader.Read()**. Each choice is matched to its corresponding question and added to the question's choices collection in the response

object.

```
        // 3. MCQ Choices AND 4. True/False Choices (Loop both result sets)
        while (await reader.NextResultAsync())
        {
            while (await reader.ReadAsync())
            {
                Guid qId = reader.GetGuid(reader.GetOrdinal("Question_ID"));
                var question = result.Questions.FirstOrDefault(q => q.QuestionID == qId);

                if (question != null)
                {
                    question.choices.Add(new MCQChoice
                    {
                        ChoiceID = reader.GetGuid(reader.GetOrdinal("Choice_ID")),
                        Question_ID = qId,
                        Choice = reader.GetString(reader.GetOrdinal("Choice_Text"))
                    });
                }
            }
        }
    }
}
```

Finally, we return the questions, with the choices of each question.



All **Data-Back functions** are designed to handle **direct communication with the database**. They contain **minimal logic**, focusing only on processing inputs and formatting outputs, while leaving business rules and complex operations to higher layers of the application. This ensures a clear separation of concerns and simplifies maintenance.

Business Logic Layer:

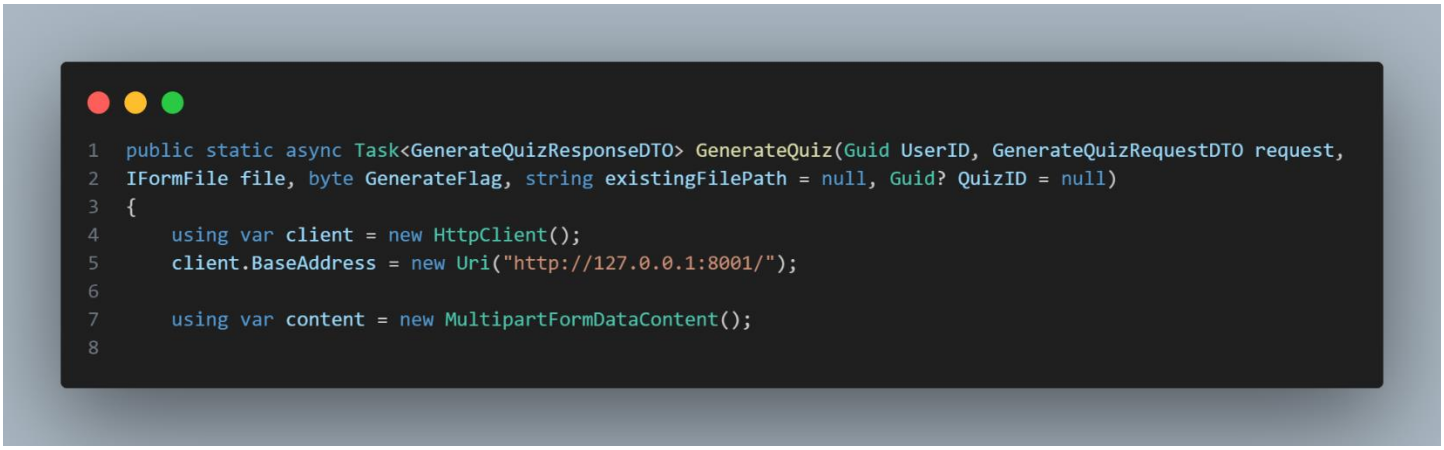
In the **Business Logic Layer**, we handle the program's core logic, following the workflow from the stored procedure, through the Data-Back layer, and into the business logic itself. This layer manages a variety of responsibilities beyond simple data operations.

It includes functionality for **JSON Web Tokens (JWTs)** to authenticate users, such as generating tokens for account verification, password recovery, and sharing quizzes. It also handles **file management**, including saving and deleting files, and monitors **server health**, including database connectivity, AI model readiness, disk space, and email service availability.

In the sections that follow, we will focus specifically on the **Generate Quiz process**, detailing how this layer orchestrates the flow between the AI model, file storage, and database operations.

In this section, we observe that the function is implemented as an **asynchronous method**, as explained earlier. Using asynchronous execution allows the system to remain responsive while waiting for external operations, such as file handling, AI model processing, and database interactions.

A key component that determines the workflow of this function is the **Generate Flag**. This flag controls whether the user is generating a **new quiz** or **regenerating an existing quiz**, enabling the function to follow the appropriate execution path for each scenario.

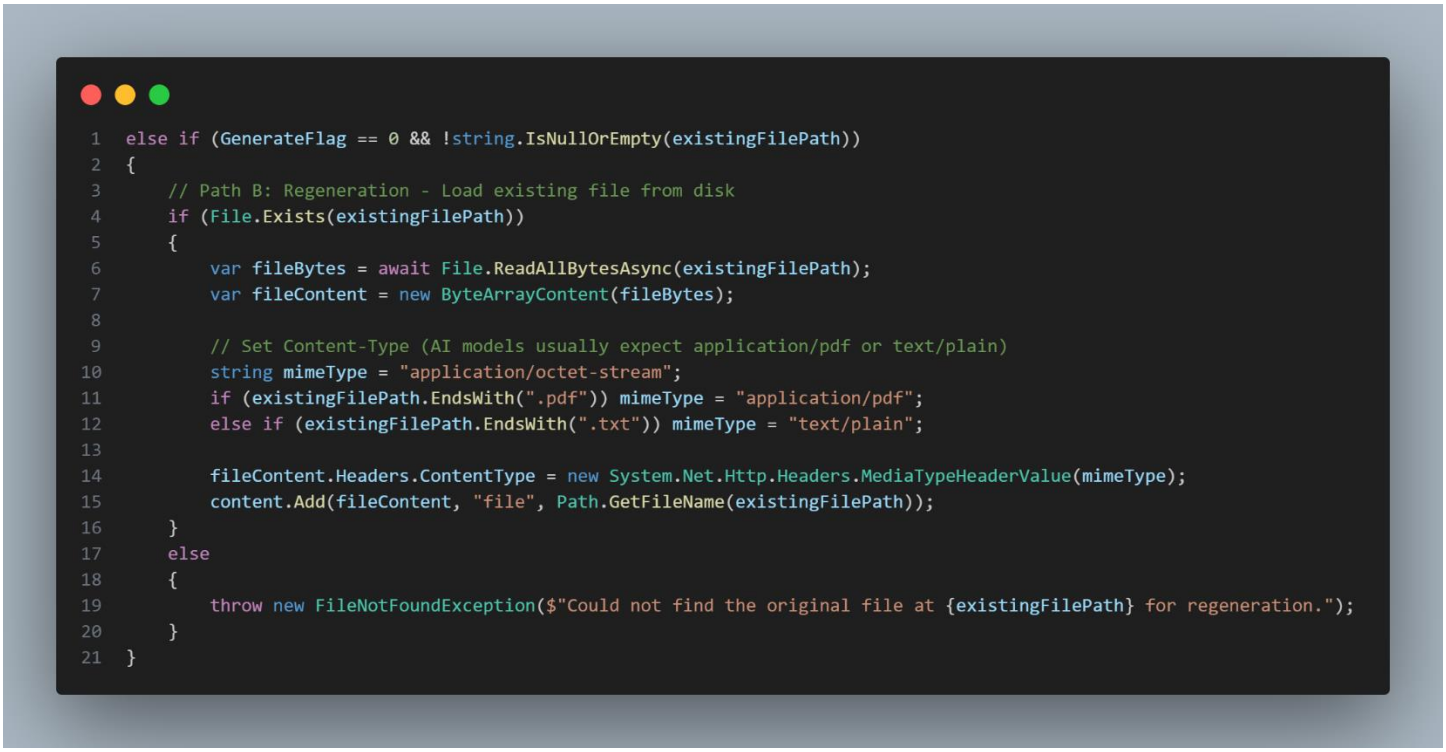


```
public static async Task<GenerateQuizResponseDTO> GenerateQuiz(Guid UserID, GenerateQuizRequestDTO request,
IFormFile file, byte GenerateFlag, string existingFilePath = null, Guid? QuizID = null)
{
    using var client = new HttpClient();
    client.BaseAddress = new Uri("http://127.0.0.1:8001/");

    using var content = new MultipartFormDataContent();

```

In this section, the **Generate Flag** is evaluated. In this scenario, the flag value is assumed to be 1, which indicates that a **new quiz** is being generated. Under this condition, the workflow follows these steps, starting with saving the uploaded file to the local disk.

```
// --- STEP 1: Attach the File ---
if (GenerateFlag == 1 && file != null && file.Length > 0)
{
    // Path A: New upload from the user
    var stream = file.OpenReadStream();
    var fileContent = new StreamContent(stream);
    var contentType = string.IsNullOrEmpty(file.ContentType) ? "application/octet-stream" : file.ContentType;
    fileContent.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue(contentType);
    content.Add(fileContent, "file", file.FileName);
}
```

In this section, the **quiz regeneration workflow** is handled. When the **Generate Flag** is set to 0, the system loads the previously uploaded file from the local disk using the stored file path. The file is read asynchronously and attached to the request sent to the AI model, ensuring that the quiz is regenerated using the same original content. The appropriate content type is set based on the file extension to allow correct processing by the AI model. If the original file cannot be found, an exception is raised to prevent invalid regeneration.



```
else if (GenerateFlag == 0 && !string.IsNullOrEmpty(existingFilePath))
{
    // Path B: Regeneration - Load existing file from disk
    if (File.Exists(existingFilePath))
    {
        var fileBytes = await File.ReadAllBytesAsync(existingFilePath);
        var fileContent = new ByteArrayContent(fileBytes);

        // Set Content-Type (AI models usually expect application/pdf or text/plain)
        string mimeType = "application/octet-stream";
        if (existingFilePath.EndsWith(".pdf")) mimeType = "application/pdf";
        else if (existingFilePath.EndsWith(".txt")) mimeType = "text/plain";

        fileContent.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue(mimeType);
        content.Add(fileContent, "file", Path.GetFileName(existingFilePath));
    }
    else
    {
        throw new FileNotFoundException($"Could not find the original file at {existingFilePath} for regeneration.");
    }
}
```

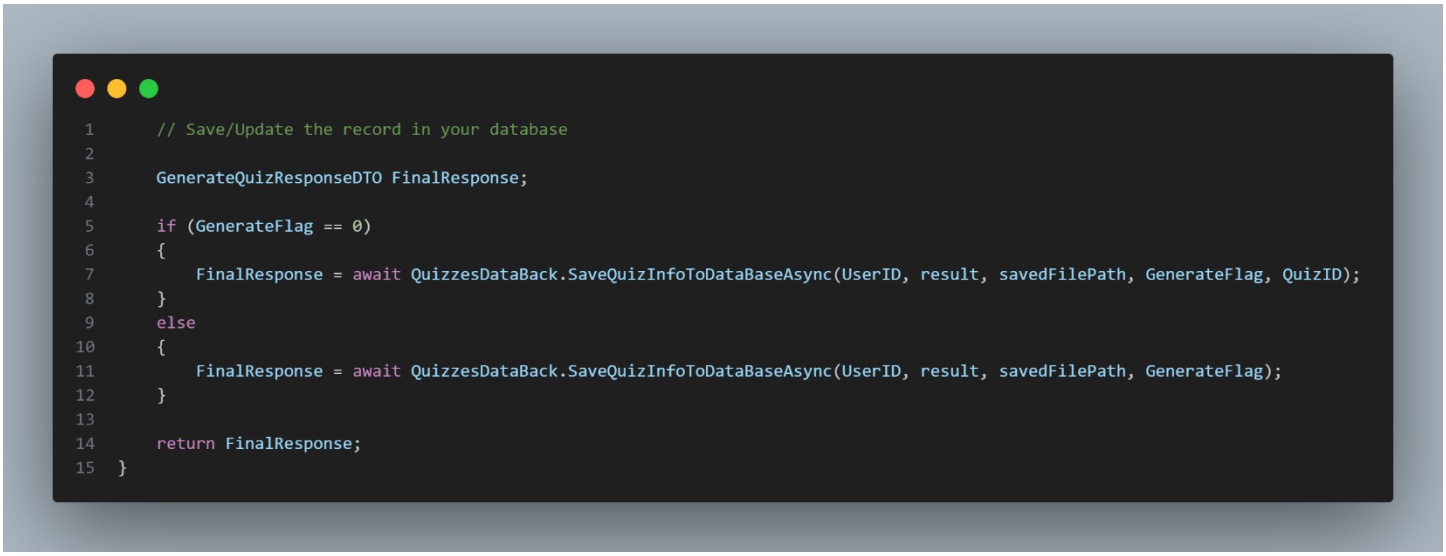
In this section, the request is sent to the **AI model**, which is exposed through a **Python FastAPI endpoint**. The model processes the provided content and generates the quiz, returning the result in **JSON format**. This response is then handled in the C# backend, where it is **deserialized into Data Transfer Objects (DTOs)** for further processing within the system.

```
// --- STEP 2: Call AI Model ---
string url = $"ask_ai_model?mcq_count={request.MCQCount}&tf_count={request.TFCount}";
HttpResponseMessage response = await client.PostAsync(url, content);

if (!response.IsSuccessStatusCode)
{
    // Capture the error message from the AI API for easier debugging
    string errorDetails = await response.Content.ReadAsStringAsync();
    throw new Exception($"AI Model Error ({response.StatusCode}): {errorDetails}");
}

var options = new JsonSerializerOptions
{
    PropertyNameCaseInsensitive = true
};

string jsonString = await response.Content.ReadAsStringAsync();
QuestionResponse? result = JsonSerializer.Deserialize<QuestionResponse>(jsonString, options);


if (result == null)
    throw new Exception("Failed to deserialize the response from AI model.");

```

In this section, the system handles **file persistence** for the quiz generation process. If the operation represents a **new quiz generation**, the uploaded file is saved to the local disk and its path is stored for later use. In the case of quiz regeneration, the existing file path is reused, avoiding unnecessary file duplication and ensuring consistency across quiz versions.

```
// --- STEP 3: Handle Database & Disk Persistence ---
string savedFilePath = existingFilePath; // Default to existing path if regenerating

if (GenerateFlag == 1 && file != null && file.Length > 0)
{
    // Only save to disk if this is a brand new generation
    contentResponseDTO uploadedContent = await ContentBusinessLayer.SaveUploadedFile(file, UserID);
    savedFilePath = uploadedContent.path;
}
```

In this final section, the generated quiz data is saved or updated in the database by invoking the Data-Back layer. If the operation represents a **quiz regeneration**, the existing Quiz ID is passed to ensure that the same quiz record is updated. Otherwise, a new quiz record is created. Once the database operation is completed, the fully populated response object is returned to the caller.



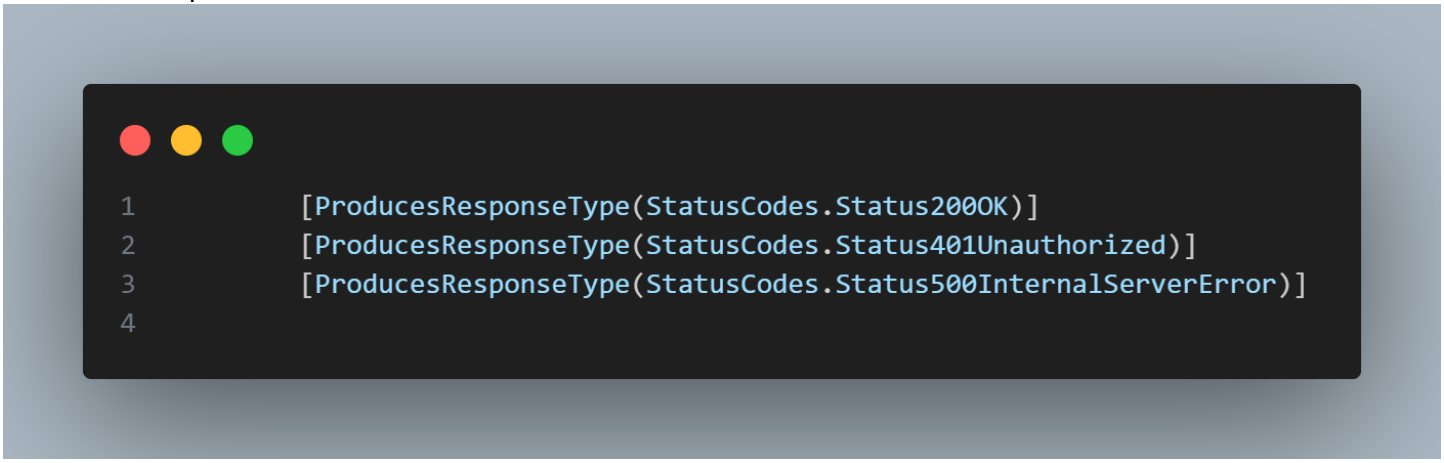
```
    // Save/Update the record in your database

    GenerateQuizResponseDTO FinalResponse;

    if (GenerateFlag == 0)
    {
        FinalResponse = await QuizzesDataBack.SaveQuizInfoToDataBaseAsync(UserID, result, savedFilePath, GenerateFlag, QuizID);
    }
    else
    {
        FinalResponse = await QuizzesDataBack.SaveQuizInfoToDataBaseAsync(UserID, result, savedFilePath, GenerateFlag);
    }

    return FinalResponse;
}
```

This section ensures that the user is authorized to use the function (i.e., the user is logged in and considered valid by possessing a valid JWT). Additionally, this section defines the endpoint's name, which will be called from the front

end.

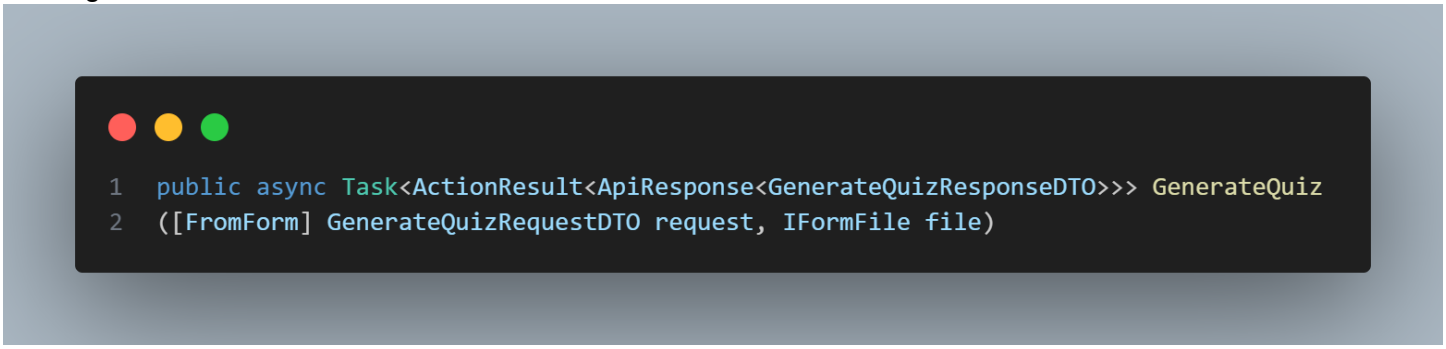
```
1  [Authorize]
2  [HttpPost("Quiz/Generate")]
```

In this section, we document the possible status codes that may be returned by the function, including successful responses, authorization failures, and internal server errors.



```
1          [ProducesResponseType(StatusCodes.Status200OK)]
2          [ProducesResponseType(StatusCodes.Status401Unauthorized)]
3          [ProducesResponseType(StatusCodes.Status500InternalServerError)]
4
```

In this section, we define the function signature, which contains the required parameters needed for the function to operate correctly. The function is also asynchronous, ensuring that its execution does not block other parts of the application while it is
running.

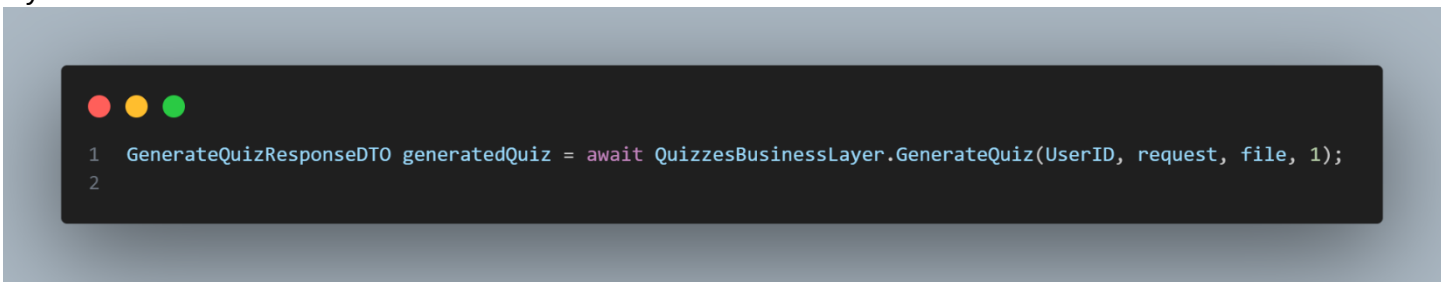


```
1  public async Task<ActionResult<ApiResponse<GenerateQuizResponseDTO>>> GenerateQuiz
2  ([FromForm] GenerateQuizRequestDTO request, IFormFile file)
```

In this section, we validate the user's identity using the JSON Web Token (JWT). If the token is invalid or missing, the user is not allowed to interact with the system. Otherwise, the request is authorized, and the user ID is extracted from the JWT to be used in subsequent operations.

```
try
{
    var userIdClaim = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier);
    if (userIdClaim == null)
    {
        return Unauthorized();
    }
    Guid UserID = Guid.Parse(userIdClaim!.Value);

```

In this section, we invoke the Business Logic Layer to handle the core workflow, including saving the required information, sending the request to generate the quiz, and coordinating all related operations. This abstraction keeps the controller lightweight and delegates the application logic to the appropriate layer.



```
GenerateQuizResponseDTO generatedQuiz = await QuizzesBusinessLayer.GenerateQuiz(UserID, request, file, 1);

```

In this section, if the process completes successfully without any errors, we return a **200 OK** status code along with the generated quiz data, indicating that the request was handled

correctly.

```
        return Ok(new ApiResponse<GenerateQuizResponseDTO>
        {
            Success = true,
            Status = 200,
            Message = "Request completed successfully.",
            Timestamp = DateTime.UtcNow,
            Data = generatedQuiz,
            Error = null
        });
    }
```

At the final level, if an unexpected error occurs during the process, we return a **500 Internal Server Error** status code to indicate a server-side failure.

```
        catch (Exception ex)
        {
            return StatusCode(500, new ApiResponse<object>
            {
                Success = false,
                Status = 500,
                Message = "Internal server error",
                Data = null,
                Error = new ApiError
                {
                    Code = "SERVER_ERROR",
                    Details = ex.Message
                }
            });
        }


}
```

This outlines the process of generating a new quiz, and the same workflow is applied across all other functions and endpoints. This approach reflects how the backend is structured, ensuring consistency, maintainability, and a clear separation of responsibilities.

This section covers how the frontend calls the endpoint and consumes its response. It explains how requests are sent, how data is passed to the backend, and how the returned quiz information is processed and displayed in the user interface.

This snippet shows how the frontend calls the backend to generate a quiz from an uploaded file: Builds a FormData payload containing the file (and optional settings like question counts). Sends an authenticated POST request to the generate endpoint (adds Bearer token). Handles non-OK responses and returns either the generated quiz data or a clear error object. This represents the project's core feature: turning user content into an auto-generated quiz through an API call.

```
export async function generateQuizFromFile(file, token, settings) {
  try {
    const formData = new FormData();
    formData.append("file", file);

    // Append settings as individual fields for easier backend parsing
    if (settings) {
      if (settings.mcqCount) formData.append("mcqCount", settings.mcqCount);
      if (settings.tfCount) formData.append("tfCount", settings.tfCount);
    }

    const res = await fetch(`/api/v1/quiz-ai/Quiz/Generate`, {
      method: "POST",
      headers: {
        Authorization: `Bearer ${token}`,
      },
      body: formData,
    });

    // Handle 401 Unauthorized or other non-OK status codes immediately
    if (!res.ok) {
      const errorData = await res.json().catch(() => ({}));
      throw new Error(errorData.message || `Server error: ${res.status}`);
    }

    const data = await res.json();
    return data;
  } catch (error) {
    console.error("Quiz Generation Error:", error);
    return {
      success: false,
      error: error.message || "Network error while generating quiz.",
    };
  }
}
```

And now, with some random codes from the frontend:

This snippet shows how the app is wrapped with context providers so authentication state (current user, token, login/logout) and exams state (exam list, selected exam, actions) become global and accessible to any component without prop-
drilling.

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import { AuthProvider } from
"./context/AuthContext.jsx";
import { ExamsProvider } from
"./context/ExamsProvider.jsx";

createRoot(document.getElementById("root")).
render(
  <StrictMode>
    <AuthProvider>
      <ExamsProvider>
        <App />
      </ExamsProvider>
    </AuthProvider>
  </StrictMode>
);
```

This snippet shows how the AuthContext.Provider exposes auth-related state and functions to the rest of the application through its value prop (for example: user, token, isAuthenticated, login, logout, loading, etc.). In other words, it’s the “public API” of your auth layer that other components consume via useContext(AuthContext).

```
return (
    <AuthContext.Provider
      value={{
        user,
        setUser,
        token,
        isLoggedIn: !!token,
        loading,
        login,
        logout,
        signup,
        error,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
}
```

This snippet shows how the exams provider exports exams-related state and operations (for example: exams, fetchExams, currentExam, submitAnswers, loading, etc.) to any screen/component. It represents the “shared exams/quiz state” that multiple pages can use consistently (library page, quiz page, shared exam route, etc.).

```
return (
    <ExamsContext.Provider
      value={{
        exam,
        setExam,
        getExamData,
        exams,
        loading,
        regeneratingQuiz,
        examResetNonce,
        feedback,
        clearFeedback,
        loadExams,
        loadSharedExam,
        updateExam,
        deleteExam,
        addExam,
        renameExam,
        regenerateWholeExam,
        regenerateExamQuestion,
        deleteExamQuestion,
        submitExamAnswers:
submitExamAnswersToBackend,
        error,
      }}
    >
      {children}
    </ExamsContext.Provider>
  );
```

CHAPTER 6: TESTING PLAN

In the AI testing phase, we evaluate the reasoning speed, API performance, and formatting reliability of various local AI models under high context loads. The approach focuses on comparing model efficiency for specific educational tasks, such as True or False (T/F) and Multiple-Choice Question (MCQ) generation.

The testing environment used for these tests is as follows:

- **Software Tools:** LM Studio (built-in chat and Local Server) and Silly Tavern.
- **Context Length:** 80,000 tokens.
- **Hardware (Test Bench):**
  - **CPU:** Intel Core I7-12700H.
  - **GPU:** Nvidia GeForce RTX 3070TI Laptop edition.
  - **RAM:** 32GB 4800 MT/s DDR5 running in dual channel.

6.1 Black-box

We utilized black-box testing to observe model behavior and output quality based on external prompts without modifying internal weights or code.

Test Cases and Outcomes:

- **Test Case 1 100 (15 T/F Prompts):** Evaluate the average reasoning speed for True or False questions.

Results*:*

nemotron-3-nano 30B Q_4_K_M (31.89s).

gpt-oss-20b Q_4_K_S (34.55s).

meta-llama-3.1-8b-instruct Q5_K_M (0.679s).

deepseek-r1-0528-qwen3-8b Q_4_K_M (23.76s).

ministral-3-14b-reasoning Q_4_K_M (2m 31s).

- **Test Case 2 100 (15 MCQ Prompts):** Evaluate the average reasoning speed for Multiple Choice Questions.

Results:

nemotron-3-nano 30B Q_4_K_M (59.35s).

gpt-oss-20b Q_4_K_S (26.31s).

meta-llama-3.1-8b-instruct Q5_K_M (4.365s).

deepseek-r1-0528-qwen3-8b Q_4_K_M (1m 48s).

ministral-3-14b-reasoning Q_4_K_M (exceeded 5 minutes).

- **Test Case 3 (Format Correctness):** Verify if the model maintains the required output structure.

Results*:*

nemotron-3-nano 30B Q_4_K_M (= 90%).

gpt-oss-20b Q_4_K_S (= 100%).

meta-llama-3.1-8b-instruct Q5_K_M (= 75%).

deepseek-r1-0528-qwen3-8b Q_4_K_M (= 50%).

ministral-3-14b-reasoning Q_4_K_M (= 20%).

- **Test Case 4 (Load/Stress Test):** Comparing 25 questions vs. 15 questions per prompt.

Result: Reducing the count to 15 prevented context overflow and improved response speed and improved response format correctness.

Test Suite 1: User Authentication (Accounts)

ID	Scenario Name	Description (Steps)	Expected Result	Status
TC-01	**Valid Signup**	Enter valid Name, Email, Password, Confirm Password.	Account created; Redirect to Verify account page	PASS

ID	Scenario Name	Description (Steps)	Expected Result	Status
TC-02	**Duplicate Email**	Try to sign up with an email that already exists.	Error message: "Email already taken."	FAIL
TC-03	**Password Mismatch**	Enter different passwords in "Password" and "Confirm Password".	Error message: "Invalid credentials."	PASS
TC-04	**Valid Login**	Enter correct Email and Password.	Success; Redirect to Home page.	PASS
TC-05	**Invalid Login**	Enter correct Email but wrong Password.	Error message: "Invalid credentials."	PASS
TC-06	**Valid Verify account**	Verify account using the code send to the Email used in creating the account	Success; Redirect to Log in page.	PASS
TC-07	**Invalid Verify account**	Enter incorrect code to verify account	Error message:” Invalid email verification token.”	PASS
TC-08	**Valid Rest password**	Open the link send to the user email and changing the password using a valid password	Success; redirect to log in page.	PASS
TC-09	**Invalid Rest password**	Opening the path of the Rest Password page without the valid rest Token page	Error message: “You don’t have access to this page.”	PASS

Test Suite 2: AI Quiz Generation (The Core Feature)

ID	Scenario Name	Description (Steps)	Expected Result	Status
TC-10	**Standard Quiz**	Enter Subject: "History", MCQ Count: "5". TF count:”5” Click Generate.	10 History questions appear.	PASS
TC-11	**Empty Subject**	Leave "Subject" blank. Click Generate.	Error message: "Subject is required."	PASS
TC-12	**Zero Questions**	Enter Subject: "Math", MCQ Count: "0" TF Count: ”0”.	Error message: "Minimum 1 question required."	PASS
TC-13	**Negative Questions**	Enter Subject: MCQ Count: "-1" TF Count:”-1”.	Error message: "Please enter a positive number."	PASS
TC-14	**Massive Count**	Enter Subject: "Math", MCQ Count: "100" TF Count: ”1000”.	System should limit it (e.g., "Max 20 questions") OR handle it without crashing.	PASS
TC-15	**Gibberish Subject**	Enter Subject: "asdfjk123". Click Generate.	AI returns a polite error or "No questions found for this topic."	FAIL
TC-16	**Non-English Input**	Enter Subject in Arabic (e.g., "تاريخ الأردن").	AI generates questions in Arabic (if supported) or English.	PASS
TC-17	**Large file size**	Enter a large subject file.	Error message: “File is too large”	PASS
TC-18	**Unprotected file extension**	Enter a wrong file extension	Error message: “File extension no supported”	PASS

Test Suite 3: Quiz Management & Modification

ID	Scenario Name	Description (Steps)	Expected Result	Status
TC-19	**Regenerate Question**	Click "Regenerate" on Question #3.	Question #3 changes to a new text; others stay the same.	PASS
TC-20	**Delete Question**	Click "Delete" on Question #2 (in a 10-question quiz).	Quiz now has 9 questions. Question #2 is gone.	PASS
TC-21	**View Saved Quiz**	Click on a quiz from the history list on the side bar.	The quiz opens with the exact same questions as before.	PASS
TC-22	**Delete Quiz**	Click "Delete" on a quiz in the option menu.	The quiz disappears from the list.	PASS
TC-23	**Regenerate Quiz**	Click “Regenerate Quiz” from the options menu	The quiz change to a new quiz.	PASS
TC-24	**Valid quiz submits**	Submit quiz after answering all question	The quiz displays the result of the quiz	FAIL
TC-25	**Invalid quiz submits**	Submit quiz after not answering all questions	Error message: “You have to answer all the questions before you submit your answers”	PASS

ID	Scenario Name	Description (Steps)	Expected Result	Status
TC-26	**Valid open shared quiz**	Open the shared quiz using the valid link	Quiz saved to user history	PASS
TC-27	**Invalid open shared quiz**	Open the shared quiz using invalid share link	Error message: “A problem occurred while sharing the quiz”	PASS

6.2 White-box

Test Suite 1: Backend Security & Authentication Logic

ID	Unit / Function	Logic to Verify	How to Test (Developer Action)	Status
TC-WB-01	GenerateVerificationToken()	Verify that the email verification token is unique for every user and associated *only* with their email.	**Debug:** Set a breakpoint in the Registration Controller. Register two users back-to-back. Verify the tokens are completely different strings.	**Pass**
TC-WB-02	ValidateResetToken()	Verify logic rejects a password reset token if it is expired (> 24 hours) or has been tampered with.	**Code Test:** Manually change a valid token string in the database (e.g., change the last letter). Try to use it to reset a password. The backend must throw an "Invalid Token" exception.	**Pass**
TC-WB-03	[Authorized] Routes	Verify that API endpoints (like DeleteQuiz) check for a valid Session/JWT on the server side, not just the client	**Network Test:** Send a DELETE request to /api/quiz/delete/5 using Postman *without* a login header. The server must return 401 Unauthorized.	**Pass**

ID	Unit / Function	Logic to Verify	How to Test (Developer Action)	Status
		side.		
TC-WB-04	SMTP Configuration	Verify that the email sender uses secure credentials from the configuration file, not hardcoded passwords.	**Code Review:** specific Check appsettings.json to ensure sensitive email passwords are read from environment variables/config, not written in the code.	**Fail**

Test Suite 2: AI Integration & File Processing Logic

ID	Unit / Function	Logic to Verify	How to Test (Developer Action)	Status
TC-WB-05	FileParserService	Verify the code extracts actual text from uploaded files (PDF/Docx) and rejects binary/image data.	**Log Check:** Upload a PDF file. Add Console.WriteLine(extractedText) in the backend. Verify the console prints the *readable paragraphs* from the PDF.	**Pass**
TC-WB-06	BuildAIPrompt()	Verify the string sent to the AI correctly combines: **Subject** + **File Content** + **MCQ Count** + **TF Count**.	**Debug:** Upload a file and ask for "3 MCQs". Inspect the final string sent to the AI API. It should look like: *"Context: [File Text]... Generate 3 MCQs..."*	**Pass**
TC-WB-07	JSON_Sanitization	Logic: AI models often return code wrapped in Markdown (e.g., json ...). The parser must strip these	**Unit Test:** Create a string variable: json {"question": "test"} . Pass it to ParseResponse(). It should succeed and not throw a JSON format error.	**Pass**

		backticks.		
TC-WB-08	AI_Error_Handler	Verify the try/catch block correctly handles timeouts or "Rate Limit" errors from the AI provider.	**Simulation:** Temporarily disconnect the internet or use an invalid API Key. Trigger a quiz generation. The code should catch the error and return a friendly message, not crash.	Pass

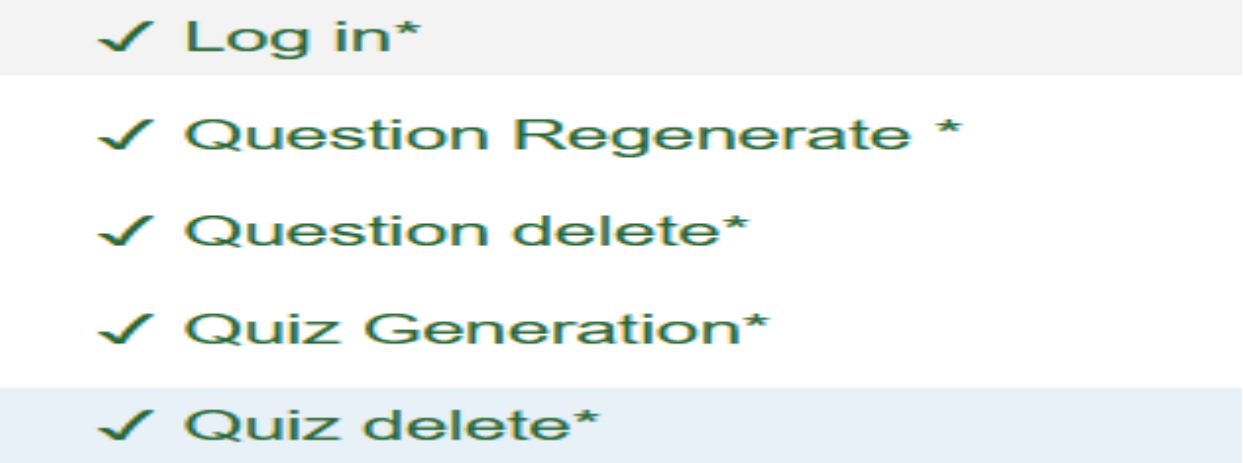
6.3 Testing automation

1.The automation tools used:

Selenium IDE

2. Decide what test cases to automate:

A. API Latency Monitoring: Measuring the time between an API call and the initial model response.



CHAPTER 7: CONCLUSION AND RESULTS

7.1 Summary of accomplished project

The project successfully delivers a fully operational client–server architecture that enables users to upload files and receive automatically generated questions based on their content. The system is supported by a database that utilizes stored procedures to ensure efficient and secure data handling. The overall architecture is designed to provide high usability, strong modularity, and scalability, allowing for future expansion and integration.

7.2 Future Work

- **Limitations:** Due to hardware constraints, the system is currently limited to deploying only small to medium-sized models, These limitations restrict the use of larger, more capable models that could further improve performance and response quality.

- **Expansion:**

Future work aims to enhance the system's functionality, usability, and scalability. Planned features include a personal user dashboard that provides learning analytics such as quiz generation statistics, quiz type distribution, and average correctness rates.

The system will also be extended to support additional question formats, including open-ended questions, and to allow users to upload quizzes and receive structured PDF outputs for improved study and revision.

Further improvements include developing a mobile-friendly interface and enabling the generation of downloadable, print-friendly learning cards.

On an institutional level, the system may be integrated into university and school e-learning platforms. From a technical perspective, scalability can be improved through multi-GPU server deployment to support higher user loads and larger models, alongside replacing LLaMA 3.1 8B with a more advanced large language model to improve response quality. Additionally, virtual exam halls with time and access controls can be introduced to support formal assessment scenarios.

REFERENCES

- International Telecommunication Union (ITU). (2025). *Global internet penetration data*. https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx
- World Bank. (2025). *Digital development: Global connectivity gaps*. https://www.worldbank.org/en/topic/digitaldevelopment
- GSMA. (2025). *Mobile economy report: Mobile internet trends*. https://www.gsma.com/mobileeconomy/
- Statista. (2025). *Digital population worldwide: Internet user forecasts*. https://www.statista.com/statistics/617136/digital-population-worldwide/
- Kinsta. (n.d.). *How to install React*. https://kinsta.com/knowledgebase/install-react/
- Microsoft. (2025). *Visual Studio 2022 system requirements*. https://learn.microsoft.com/en-us/visualstudio/releases/2022/system-requirements
- openai.com (2025). *introducing-gpt-oss*. https://openai.com/index/introducing-gpt-oss
- OpenAI. (2025). *Introducing GPT-5*. https://openai.com
- Meta AI. (2024). *Llama 3 models*. https://ai.meta.com/llama
- Google DeepMind. (2024). *Gemini 1.5 announcement*. https://deepmind.google
- Anthropic. (2024). *Claude 3 family*. https://www.anthropic.com
- Yin, W., et al. (2024). *A survey on multimodal large language models*. https://arxiv.org/abs/2401.13601
- Lewis, P., et al. (2020). *Retrieval-augmented generation for knowledge-intensive NLP*. https://arxiv.org/abs/2005.11401
- Hu, E., et al. (2022). *LoRA: Low-rank adaptation of large language models*. https://arxiv.org/abs/2106.09685
- Questgen AI. (2025). *AI-powered question generation platform*. https://www.questgen.ai
- Quizbot AI. (2025). *Automated quiz creation tool*. https://quizbot.ai
- Shahid, O., Hussain, S., & Shoaib, M. (2021). *VidVersityQG: Automated video-based question generation for education*. https://arxiv.org/abs/2112.01229
- Quizaic. (2025). *A generative AI case study*. Medium. https://medium.com/google-cloud/quizaic-a-generative-ai-case-study-190b02baa8df
- ChatGPT. (2025). *OpenAI language model responses and related documentation*. OpenAI.
- Observations. (2025). *Quizlet, Quizziz, Kahoot, QuestionPro, and ProProfs platforms*.
- Alsmadi, I., & Almarashdeh, I. (2023). Artificial intelligence applications in education: A review of current trends and future perspectives. *Education and Information Technologies, 28*(4), 4513–4532. https://doi.org/10.1007/s10639-022-11298-4
- Chen, Y., & Wang, X. (2022). Enhancing learning engagement through AI-assisted content creation. *Computers & Education, 181*, 104453. https://doi.org/10.1016/j.compedu.2022.104453
- Kovačević, A., & Li, J. (2023). The role of generative AI in personalized education. *British Journal of Educational Technology, 54*(5), 1290–1307. https://doi.org/10.1111/bjet.13315
- Zhang, M., & Lee, D. (2021). Artificial intelligence for multimedia-based education: Opportunities and challenges. *Interactive Learning Environments, 29*(7), 1132–1149. https://doi.org/10.1080/10494820.2019.1703012