AUTOMATED QUIZ CREATION USING CHATGPT WITH INTEGRATION INTO THE CANVAS LMS

АВТОМАТИЗОВАНЕ СТВОРЕННЯ ТЕСТІВ ЗА ДОПОМОГОЮ CHATGPT З ІНТЕГРАЦІЄЮ В LMS CANVAS

by Lazebnyi Vitalii

Presented in Partial Fulfillment of the Requirements for the Degree
Master of Software Engineering
American University Kyiv
2025

APPROVED BY:
Sergiy Tytenko, Ph.D., Faculty Mentor

Acknowledgments

I express my deep gratitude to Dr. Sergiy Tytenko for his unwavering support and insightful guidance throughout my studies and the writing of this paper.

Furthermore, I am grateful to the American University Kyiv for providing me with the platform and opportunity to delve into this critical topic as my Capstone Project.

Summary

In today's rapidly evolving educational landscape, educators must create compelling and engaging assessments without expending excessive time and resources. Large Language Models (LLMs), such as OpenAI's GPT-4, have emerged as powerful tools to automate quiz generation, streamlining the assessment process. This article explores how LLMs can transform quiz creation, discusses the integration of AI-generated quizzes into Learning Management Systems (LMS), and addresses this technology's ethical considerations, limitations, and future directions. We provide practical guidelines for educators to effectively leverage LLMs in their teaching practices, supported by real-world examples and empirical data.

Abstract

The development of generative artificial intelligence (AI) and its various applications are still in the research and development phase. However, it is evident that the emergence of generative AI significantly impacts multiple industries, including education.

This study explores the potential applications of generative AI in education, such as personalized learning tools and AI-powered study resources.

The work highlights the importance of prompt engineering in facilitating communication between users and AI systems, which can lead to better educational outcomes. Acknowledges the challenges and risks of incorporating AI technologies into education, such as data privacy and security concerns.

Introduction

Creating compelling and engaging assessments is a significant challenge for educators. Traditional quiz creation is not only time-consuming but also often struggles to fully align with diverse learning objectives or adapt to the unique needs of various educational contexts. The manual process requires significant effort to ensure that assessments are pedagogically sound, cognitively diverse, and relevant to the course material. Furthermore, it can be resource-intensive, particularly for large-scale courses or institutions where assessments need frequent updates.

Large Language Models (LLMs), such as OpenAI's GPT-4, have emerged as powerful tools to address these challenges[1]. By leveraging advanced natural language processing (NLP) capabilities, these models can understand context, generate diverse question types, and adapt to various educational needs. Unlike traditional methods, LLMs can produce assessments that are not only accurate and contextually relevant but also tailored to specific cognitive levels, such as recall, application, or critical thinking. This enables educators to create quizzes that are both efficient and effective, significantly reducing the time and resources required for development.

The potential of LLMs extends beyond basic automation. These models are capable of generating sophisticated question formats, such as multiple-choice, short-answer, or scenario-based questions, which can challenge students at higher-order cognitive levels. For example, integrating Bloom's Taxonomy into quiz design through LLMs allows for assessments that encourage deeper learning, from basic knowledge recall to complex problem-solving and creativity. This adaptability ensures that quizzes are aligned with educational standards and learning outcomes, providing a meaningful evaluation of student progress.

When integrated with Learning Management Systems (LMS) such as Canvas, the advantages of AI-driven quiz generation become even more apparent. LMS platforms act as centralized hubs for managing educational content, student performance, and course logistics. Seamlessly integrating AI-generated quizzes into these systems streamlines the workflow for educators, allowing them to focus on instructional strategies rather than administrative tasks.

This work delves into the practical applications of LLMs in quiz generation, highlighting their potential to transform traditional assessment practices. This study

provides a comprehensive overview of how LLMs can revolutionize assessment creation, setting the stage for a more efficient and personalized learning experience.

In an era where technology is rapidly transforming education, the integration of LLMs into assessment practices represents a critical step toward modernizing pedagogy. This innovation has the potential to not only enhance efficiency but also improve the quality of teaching and learning outcomes, making education more accessible and impactful for students and educators alike.

Glossary of Terms

LLM (Large Language Model): An AI model trained on vast amounts of text data to understand and generate human-like language.

OCR (Optical Character Recognition): Technology that converts different types of documents into editable and searchable data.

QTI (Question and Test Interoperability): A standard format for exchanging quiz and test data between systems.

Bloom's Taxonomy: A framework for categorizing educational goals into levels of complexity and specificity.

API (Application Programming Interface): A set of protocols for building and integrating application software.

GDPR (General Data Protection Regulation): EU data protection and privacy regulations.

1. Transforming Quiz Generation with Large Language Models

1.1 Introduction

The advent of Large Language Models (LLMs) has catalyzed transformative innovations in educational assessment, particularly in automated quiz generation. These models can synthesize domain-specific knowledge and pedagogical principles, enabling the creation of intricate evaluative instruments tailored to diverse academic disciplines and cognitive levels. By leveraging their computational prowess and linguistic versatility, LLMs facilitate the generation of complex, multidimensional question sets that engage students in higher-order thinking and foster robust skill acquisition [4, 5].

In language learning, LLMs demonstrate a profound capacity to scaffold linguistic competency through the systematic design of assessment mechanisms that incorporate principles of second-language acquisition and multimedia learning theory [2, 3]. For instance, multiple-choice questions constructed by these models integrate contextualized vocabulary testing, promoting lexical retention and semantic precision. Furthermore, fill-in-the-blank exercises necessitate syntactic analysis and morphological accuracy, effectively reinforcing grammatical structures. Replete with interpretive and inferential queries, reading comprehension tasks align with Mayer's cognitive theory of multimedia learning, optimizing cognitive load management and enhancing textual interpretation skills [3]. The integration of these diverse question types exemplifies the pedagogical versatility of LLMs in addressing varying linguistic proficiencies and learning outcomes.

Within programming education, LLMs exhibit exceptional utility in crafting evaluative scenarios that bridge theoretical understanding and practical application. The models can generate sophisticated code-based tasks, such as completing syntactically intricate code snippets or diagnosing errors within algorithmic constructs. These exercises reinforce the acquisition of computational syntax and cultivate debugging insight, a critical competency in software development. Additionally, conceptual questions formulated by LLMs challenge students to articulate abstract programming paradigms, compare algorithmic efficiencies, and elucidate nuanced differences in data structures [4]. Such assessments underscore

the capacity of LLMs to operationalize complex domain-specific knowledge into pedagogically sound evaluations, fostering the development of computational fluency.

Applying LLMs in mathematics transcends conventional rote problem-solving, encompassing the formulation of multifaceted problem sets that integrate abstract reasoning, logical deduction, and applied mathematics. These models can generate tasks ranging from solving intricate differential equations to constructing formal proofs of mathematical theorems. Such problems compel students to engage in deep cognitive processing, aligning with Bloom's higher-order cognitive categories. Moreover, LLMs are adept at formulating application-based problems that contextualize mathematical concepts within real-world scenarios, augmenting their relevance and fostering the transference of theoretical knowledge to practical contexts [4].

In healthcare education, the deployment of LLMs for quiz generation exemplifies their adaptability in addressing complex, multidisciplinary learning objectives. By synthesizing clinical data and pedagogical frameworks, these models can construct case-based questions that simulate authentic diagnostic scenarios, requiring learners to integrate pathophysiological knowledge with clinical reasoning [5]. Additionally, treatment planning exercises, derived from algorithmic decision-making processes, assess students' aptitude for devising evidence-based medical interventions. Meanwhile, Ethical dilemma questions compel learners to critically reflect on professional responsibilities, moral frameworks, and regulatory standards [5]. This multidimensional approach to assessment, informed by systematic reviews of automated item generation in medical education, underscores the potential of LLMs to transform the evaluation of healthcare competencies.

In sum, utilizing LLMs for quiz generation represents a paradigm shift in educational assessment, integrating advanced computational capabilities, domain-specific expertise, and pedagogical theory. By generating sophisticated, multidimensional question sets, these models enhance the precision and adaptability of assessments and cultivate deep, transferable knowledge across diverse academic disciplines [2, 3, 4, 5].

The practical application of Large Language Models (LLMs) in academic environments has demonstrated their transformative potential to enhance both the efficiency of educational workflows and student learning outcomes. A notable case

study at Deakin University in Australia provides compelling empirical evidence supporting this claim. In this initiative, educators integrated OpenAI's GPT-4 into the assessment design process for introductory psychology courses, aiming to alleviate the significant time demands traditionally associated with quiz preparation. By leveraging GPT-4's advanced linguistic and contextual comprehension capabilities, the university succeeded in automating quiz generation while maintaining pedagogical alignment with course objectives [6].

The impact of this implementation was substantial, resulting in a quantifiable 30% reduction in the time required for quiz development. This time-saving benefit enabled educators to redirect their efforts toward interactive teaching methodologies, such as facilitating in-depth discussions, conducting active learning exercises, and providing personalized feedback. Furthermore, the increased frequency and diversity of assessments, made possible by the streamlined quiz creation process, contributed to a 15% improvement in student performance metrics. This improvement underscores the effectiveness of AI-generated quizzes in promoting more profound engagement with course material and reinforcing key psychological concepts [6].

By employing GPT-4, the university demonstrated the practical utility of integrating LLMs into higher education, particularly in contexts requiring frequent assessments. The results align with broader research advocating using AI in educational settings to enhance efficiency and student outcomes. These findings also emphasize the importance of continuously refining AI-generated content to ensure alignment with cognitive and pedagogical goals, as articulated in empirical studies on automated assessment tools [4, 5].

1.2 Optimizing Educational Resources for LLM-driven Quiz Generation

The quality and organization of input materials profoundly influence quiz generation's efficacy using large language models (LLMs). Ensuring that educational content is meticulously prepared in a machine-readable format is a foundational requirement for leveraging the capabilities of these advanced AI systems. The process involves rigorous digitization, strategic segmentation, and systematic organization, each critical to ensuring that the generated assessments align with pedagogical objectives and cognitive learning goals.

Digitization and Content Formatting. The digitization of teaching materials is essential in preparing content for LLM-based quiz generation. High-quality, machine-readable text minimizes inaccuracies and ensures that the LLM can process the material effectively. Advanced Optical Character Recognition (OCR) tools, such as Adobe Acrobat Pro or elDoc, play a pivotal role in converting scanned documents into editable and searchable digital text, thus reducing the risk of misinterpretation or data loss during processing [7]. However, the inherent limitations of automated OCR necessitate manual proofreading to correct errors, especially in technical materials containing specialized symbols, mathematical formulas, or diagrams. Standardizing file formats, such as plain text or Markdown, further facilitates seamless processing, ensuring compatibility with diverse AI platforms and enabling consistent analysis across datasets.

1.3. Strategic Segmentation of Educational Materials

Effective segmentation of educational content enhances the relevance and specificity of quiz questions generated by LLMs. Grouping materials thematically ensures that the quizzes focus on coherent areas of knowledge, aligning with the structured progression of a curriculum. Furthermore, aligning segments with well-defined learning objectives ensures that the quizzes assess the intended competencies, promoting measurable educational outcomes [4]. Categorizing content by complexity levels is equally essential, as this enables the generation of quizzes tailored to diverse proficiency levels, thereby supporting differentiated instruction and scaffolding learners' progression from foundational to advanced concepts. This systematic segmentation ensures pedagogical precision and facilitates a more targeted evaluation of students' skills.[21]

Organization and Management of Digital Resources. The final stage in preparing educational materials for LLM-driven quiz generation involves organizing and managing content within a robust Content Management System (CMS). A CMS with advanced features, such as batch upload capabilities, significantly reduces the administrative burden of managing extensive instructional materials. Metadata tagging is indispensable, allowing educators to annotate content with relevant keywords, topics, and difficulty levels. This metadata not only improves the discoverability of materials but also enhances the quizzes' specificity, ensuring that they align closely with the desired learning outcomes. Furthermore, implementing version control mechanisms ensures that the content remains up-to-date, reflecting the latest revisions to course material and mitigating the risk of outdated or inaccurate assessments being produced [6].

By adhering to these rigorous practices, educators can optimize the preparation of educational materials for LLM-based quiz generation, ensuring that the resulting assessments are pedagogically robust, contextually accurate, and adaptable to diverse learning environments. This process exemplifies the synergy between meticulous instructional design and cutting-edge AI technology, underscoring the transformative potential of LLMs in educational assessment.

1.4. Designing Effective Prompts for LLMs

The efficacy of quiz generation using Large Language Models (LLMs) is contingent upon the precision and structure of the prompts provided. Prompt engineering—crafting directives that guide the AI's response—represents a critical interface between human educators and AI systems. Well-crafted prompts ensure that the generated outputs align with pedagogical objectives, are contextually relevant, and exhibit high coherence. Effective, prompt design necessitates adherence to foundational principles, the incorporation of cognitive frameworks such as Bloom's Taxonomy, and anticipating potential pitfalls to maximize the utility of LLMs.

Foundational Principles of Effective Prompting. Effective prompting is characterized by specificity, contextual clarity, and operational constraints. Specificity involves delineating explicit requirements, such as the number of questions, the targeted subject matter, and the intended difficulty level. This eliminates ambiguity and guides the LLM toward producing outputs that align with the desired outcomes. Contextual clarity is equally crucial; providing the model with background information ensures that it generates content grounded in the intended educational framework. For instance, prompts for introductory psychology might specify thematic areas like cognitive biases to narrow the model's focus. Additionally, operational constraints, such as word limits or response formats, refine the output, ensuring that it meets the logistical requirements of the assessment [7].

Example of a Structured Prompt. "Generate a quiz to check the knowledge level of students using this syllabus. There should be 40 questions. There are four possible options to answer for every question. For every question, there should be precisely one correct answer. Provide detailed explanations for each correct answer" (Figure 1). This prompt exemplifies how specificity, context, and constraints converge to produce pedagogically valuable and practically applicable outputs.

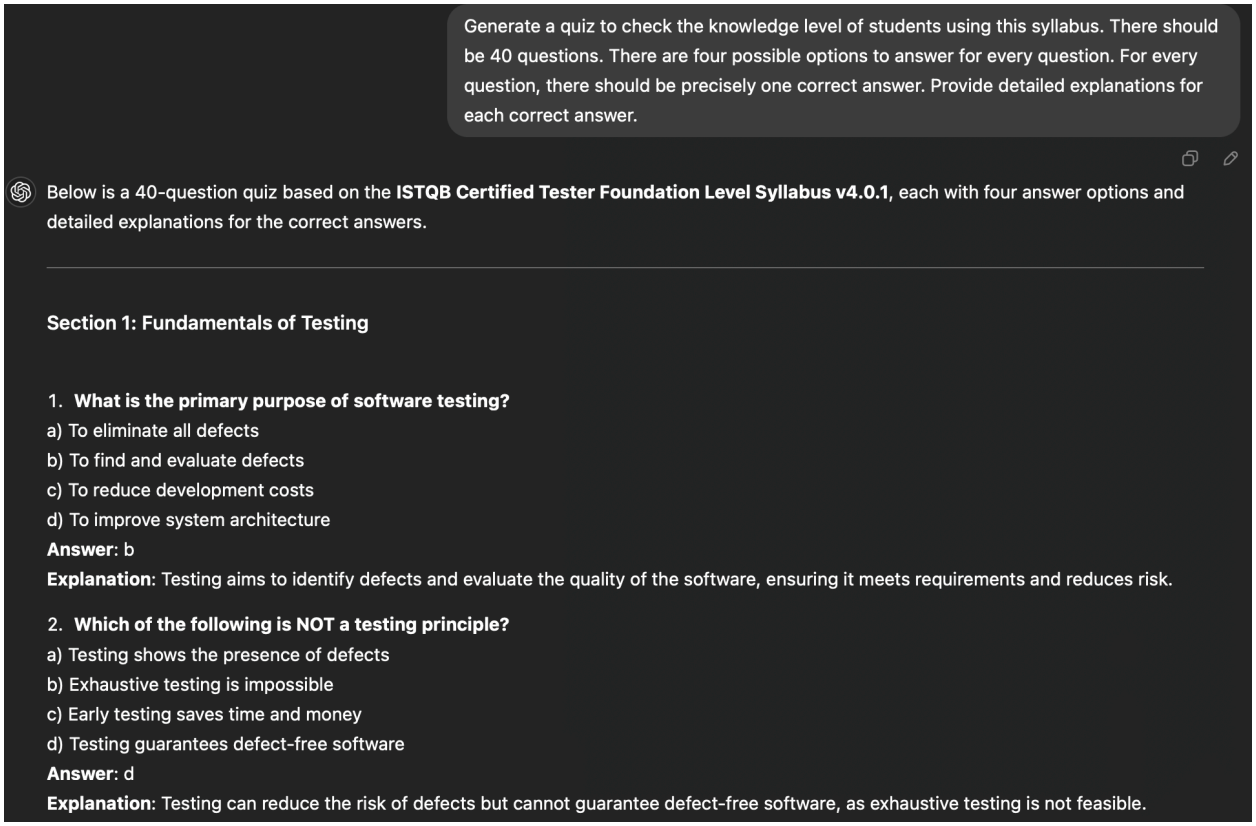


Figure 1. Quiz generated by ChatGPT in the user interface

1.5. Enhancing Prompt Design with Bloom's Taxonomy

Integrating Bloom's Taxonomy into prompt design ensures that the generated quizzes target a comprehensive range of cognitive levels, from foundational recall to advanced creative thinking[20]. Educators can systematically design assessments that address diverse learning outcomes by categorizing prompts according to the six hierarchical levels of the taxonomy [8]:

Remembering. Prompts at this level assess factual recall (e.g., "Questions should be constructed to solicit precise and comprehensive definitions of specified concepts.").

Understanding. For example: "Questions should be designed to assess the depth of comprehension of specified subject matter rigorously."

Applying. For example, "Questions should incorporate real-world scenarios to contextualize and evaluate the practical application of the given knowledge."

Analyzing. For example, “The questions should encompass the examination of relationships and the discernment of distinctions between various concepts.”

Evaluating. For example, “Construct questions that compel respondents to appraise and substantiate evaluative judgments by critiquing theoretical paradigms, methodological approaches, or empirical studies”).

Creating. For example, “Formulate questions that inspire respondents to generate innovative ideas or construct original solutions by synthesizing knowledge and applying it creatively”).

Incorporating this taxonomy into prompt design ensures cognitive diversity and aligns assessments with established pedagogical frameworks, enhancing educational validity.

Despite LLMs' potential, poorly designed prompts can yield suboptimal outputs. Ambiguity in prompts often leads to irrelevant or incoherent results. For instance, vague requests such as “Generate some questions on psychology” lack the precision to produce targeted assessments. To mitigate this, prompts must be detailed and explicit.

Similarly, overly complex language or excessive jargon may confuse the model, resulting in inaccurate outputs. Simplifying language while maintaining academic rigor ensures clarity. Finally, unrealistic expectations regarding the model’s capabilities, such as expecting domain-specific expertise beyond training, can lead to errors. Awareness of these limitations and iterative refinement of prompts are essential to achieving optimal results [7, 8].

By adhering to the principles of effective prompting, incorporating cognitive frameworks like Bloom’s Taxonomy, and proactively addressing potential challenges, educators can maximize the pedagogical utility of LLMs. Thoughtful, prompt engineering bridges the gap between human intent and AI capabilities, enabling the generation of assessments that are not only contextually relevant but also cognitively enriching.

1.6. Integration with Canvas Learning Management System (LMS)

Canvas LMS is widely recognized as one of the most robust and reliable Learning Management Systems available today. It is renowned for its widespread adoption, extensive feature set, and adaptability to various educational contexts. With millions of users globally, Canvas has established itself as a proven platform that is trusted by institutions ranging from primary and secondary schools to universities and corporate training environments. Its extensive documentation, regular updates, and strong support community make it a platform of choice for integrating advanced educational technologies, including AI-generated quizzes. This discussion focuses exclusively on integrating AI-generated quizzes into Canvas, emphasizing its advantages and exploring multiple implementation methods. Among these methods, the direct upload of quiz files in standardized formats emerges as the most efficient and educator-friendly approach.

The popularity and proven performance of Canvas are key reasons for its suitability in AI integration. Canvas offers a robust ecosystem that ensures scalability and seamless integration of advanced tools. The platform’s continuous development cycle ensures it remains at the forefront of technological innovation in education. Additionally, its flexible integration options, including APIs and file imports, make it particularly suited for educators seeking to leverage AI-generated content. Canvas supports standardized formats such as QTI, allowing educators to maintain quizzes' structural integrity while ensuring compatibility across systems. This commitment to open standards simplifies integrating external tools and content into the LMS, further highlighting its suitability for educational institutions.

Several methods can be employed to integrate AI-generated quizzes into Canvas LMS. One of the most practical and effective approaches involves uploading quiz files in QTI format directly into the system. This method requires minimal technical expertise and preserves the quiz's formatting, structure, and metadata. Educators can quickly generate quizzes using AI tools, export them as QTI files, and upload them to Canvas through a simple process. This approach is particularly advantageous for its accessibility, efficiency, and ability to support bulk uploads, making it highly scalable for educators managing large volumes of assessments.

The steps for file upload integration into Canvas are straightforward. First, the AI-generated quiz is exported in QTI format. The educator then logs into Canvas LMS and navigates to the course where the quiz will be implemented. Within the course settings, the educator selects the option to import course content and chooses the QTI .zip file as the content type. The file is uploaded, and the educator reviews and adjusts any settings to ensure alignment with course objectives. Once these steps are completed, the quiz is published and made available to students. This method ensures that the structure and content of the quiz remain intact and consistent with the original design.

The advantages of the file upload method are numerous. It requires minimal technical knowledge, making it accessible to educators with limited programming expertise. The process is efficient, allowing for the rapid integration of large numbers of quizzes. The method also ensures data integrity, maintaining the formatting of the original quiz, including question types, answer options, and metadata. Additionally, quizzes uploaded in this manner can be reused across different courses and academic terms, enhancing their scalability and long-term utility. This approach exemplifies how educators can efficiently leverage AI-generated content while minimizing the complexities associated with integration.

While file upload integration is the most practical and accessible method for incorporating AI-generated quizzes into Canvas LMS, an alternative approach involves utilizing the Canvas LMS API[19] to facilitate programmatic quiz creation. This method leverages the API's capability to automate quiz integration, allowing developers to authenticate and interact with Canvas endpoints using API keys. These keys enable the direct transfer of AI-generated quizzes into the system, providing a highly customizable and automated workflow. However, this approach is not without its limitations. Implementing API integration necessitates advanced programming expertise and a robust technical infrastructure, making it less viable for educators or institutions lacking such resources. Furthermore, API usage requires the active involvement of a Canvas administrator to generate, manage, and monitor API keys. These administrators often need to grant additional permissions to ensure the API can access the necessary endpoints for quiz integration. Such administrative support is not always readily available, particularly in institutions with limited IT staffing or stringent access controls. This dependency on administrative resources can introduce delays and operational complexities,

reducing the feasibility of API-based integration for some educational environments. Additionally, the time-intensive nature of developing and maintaining API-based workflows, particularly in response to updates or changes in the Canvas LMS, further underscores the challenges of this method compared to the simplicity and scalability of file uploads.

An alternative methodology for integrating AI-generated content into Canvas LMS involves the deployment of third-party automation platforms and external application programming interfaces. Notable examples include EduAppCenter, DesignPLUS by Cidi Labs, Make (formerly known as Integromat), and n8n. These platforms function as intermediary frameworks, enabling the orchestration of automated workflows to facilitate the synchronization and incorporation of AI-generated educational materials within Canvas's ecosystem. By leveraging the capabilities of such tools, institutions can automate repetitive operational processes, including the batch importation of quizzes, the configuration of course notifications, and the synchronization of content across multiple course modules. Despite their potential to streamline logistical workflows and enhance operational efficiency, these third-party platforms often exhibit inherent limitations in terms of compatibility and precision when juxtaposed with direct file upload methodologies or native API-based integrations. Furthermore, the utilization of these intermediary tools frequently necessitates intricate configuration and customization to align with the nuanced and heterogeneous requirements of specific institutional frameworks. This additional layer of operational complexity may inadvertently escalate implementation challenges, particularly in environments characterized by constrained technical support or divergent educational standards. Consequently, while these platforms offer a supplementary mechanism for integration, their applicability is often circumscribed by the technical and administrative infrastructure of the deploying organization.

Despite the availability of alternative methods, the direct upload of quiz files in QTI format remains the most advantageous approach for integrating AI-generated content into Canvas LMS. This method balances ease of use with technical precision, ensuring that educators can focus on their primary role of facilitating learning rather than navigating complex integration processes. The compatibility of Canvas with open standards such as QTI further underscores its suitability as a platform for deploying AI-generated quizzes. By simplifying the integration process and ensuring the integrity of educational content, file uploads

empower educators to enhance their teaching practices through the efficient and effective use of AI technologies.

In conclusion, Canvas LMS provides an ideal platform for integrating AI-generated quizzes due to its widespread adoption, robust development ecosystem, and flexible integration options. Among the various methods available, file uploads in QTI format offer the most practical and efficient solution. This approach minimizes technical barriers, preserves data integrity, and supports scalability, making it a highly effective strategy for educators seeking to leverage the capabilities of AI-generated content. This method exemplifies the transformative potential of integrating advanced technologies into modern educational frameworks by focusing on simplicity and compatibility.

1.9. Conclusion

Large Language Models (LLMs) represent a transformative advancement in education, offering unparalleled potential to streamline the creation of assessments and enhance learning outcomes across diverse contexts. By automating the generation of quizzes that are both diverse in format and contextually aligned with curricular objectives, these technologies allow educators to redirect their efforts toward more interactive and student-centered teaching practices. The time saved through automation can be reinvested in activities that foster deeper understanding and personalized support, addressing the unique needs of individual learners.

To fully harness the capabilities of LLMs, it is imperative to address key ethical and practical considerations. Ensuring compliance with data privacy standards, mitigating algorithmic bias, and maintaining rigorous human oversight are essential to safeguarding the integrity and inclusivity of AI-generated content. The integration of validation processes and feedback loops further enhances the reliability of these tools, ensuring that their outputs align with pedagogical goals and uphold educational equity.

Looking forward, the evolution of LLMs promises to further revolutionize education through advancements in adaptive learning systems and multimodal content delivery. The integration of real-time adjustments to quiz difficulty and personalized learning pathways offers new possibilities for tailoring education to the needs of individual students. Similarly, incorporating visual, auditory, and interactive elements into AI-generated assessments enhances engagement, making learning more dynamic and accessible. These developments underscore the potential of LLMs to optimize current practices and to pioneer new paradigms in teaching and learning.

In conclusion, the strategic implementation of LLMs in education has the potential to transform traditional pedagogical approaches, enriching the educational experience for both educators and learners. By addressing current challenges and embracing future innovations, institutions can leverage these technologies to create more effective, equitable, and engaging learning environments, setting the stage for a new era of educational excellence.

2. Quiz Generation Software System

2.1. Solution Overview

The Automated Quiz Creation Using ChatGPT with Integration into the Canvas LMS system provides an innovative approach to streamlining quiz creation for educational institutions. By leveraging the capabilities of OpenAI's GPT-4, the solution automates quiz generation directly from educator-provided study materials. This eliminates the traditionally time-intensive task of manual quiz creation, allowing educators to focus on improving students' learning experiences. By combining advanced language model capabilities with pedagogical frameworks such as Bloom's Taxonomy, the system ensures quizzes are contextually accurate and aligned with cognitive learning objectives.

The system Seamlessly integrates with the Canvas LMS and supports direct upload via QTI format. These integration methods offer flexibility, ensuring quizzes maintain structural integrity and ease of deployment into courses. The QTI format ensures that the quizzes are reusable, scalable, and compatible with other LMS platforms, enabling institutions to adapt the solution to various educational contexts.

The software stack relies on a Ruby on Rails-based API for efficient backend management, featuring endpoints for quiz CRUD operations, QTI exports, and Microsoft OAuth authentication for secure user access. Additionally, the system should use Redis and Sidekiq for background processing like quiz generation and file conversion, ensuring a high-performing, scalable workflow.

This solution prioritizes ethical and pedagogical considerations. To foster fairness and inclusivity, the quiz generation employs rigorous manual and automated reviews to mitigate biases in AI outputs. At the same time, robust validation processes ensure that generated quizzes align with educators' course objectives and institutional standards. Enhanced security protocols, such as anonymized data processing and encrypted storage, safeguard sensitive student and institutional information, building trust in the AI's deployment.

By integrating cutting-edge AI technology into established educational frameworks, this system brings significant efficiency gains to educational workflows. Automating repetitive tasks like quiz creation and grading allows educators to innovate their teaching methodologies and better support student

needs. This flexibility ensures educators and learners benefit from a streamlined, adaptable, and scalable solution that transforms traditional approaches to assessment in modern education.

2.2. Requirements

2.2.1 Business value

The Automated Quiz Creation System brings immense business value by addressing a critical pain point in educational workflows: the substantial time and effort involved in manually creating assessment materials. Educators, corporate trainers, and course administrators often spend hours designing quizzes that align with learning objectives, ensure cognitive variety, and engage diverse learners. With the integration of cutting-edge AI technologies, this product eliminates the manual burden from the process, enabling quick, accurate, and scalable generation of high-quality quizzes directly from study resources. This ensures educators can dedicate their efforts to higher-priority tasks, such as personalized teaching, student mentoring, and curriculum development.

By automating quiz generation, the solution can reduce the time required to produce quizzes by up to 70-80%, particularly for large-scale courses or institutions managing frequent assessments. Manually crafting quizzes often requires hours analyzing content, drafting questions, revising formats, and aligning difficulty with course goals. The system simplifies this process into seconds, allowing users to upload content and generate quizzes with diverse question types. These quizzes are tailored to specific difficulty levels and pedagogical frameworks, such as Bloom's Taxonomy, ensuring they meet cognitive and academic standards. This time savings is transformational for educators managing multiple classes or corporate trainers creating assessments across different teams.

Additionally, seamless integration with the Canvas LMS further enhances its value. The system generates quizzes and exports them directly in QTI format or uploads them automatically to LMS platforms. This eliminates redundant steps such as manually transferring and formatting quizzes, further streamlining the workflow and ensuring error-free deployment. With features like bulk uploads and reusable quiz templates, the system enables institutions to build robust, engaging assessment libraries in far less time than traditional methods.

The product doesn’t just save time—it also ensures content quality and consistency. Generating quizzes with the assistance of GPT-4 ensures straightforward, diverse, and bias-checked question sets that align with the uploaded study materials. This improves student engagement and reduces the

likelihood of human errors, ensuring fair and inclusive assessments. Furthermore, educators are provided with error-detection and manual editing capabilities, giving them complete control over the final output while significantly reducing the effort involved. From a business perspective, this ensures professional, scalable assessments without additional training or resources.

This system amplifies productivity, reduces operational inefficiencies, and delivers demonstrable ROI. By saving educators hours of manual work per quiz, reducing setup time for assessments, and promoting better teaching outcomes, the solution empowers institutions to adopt more innovative, AI-driven processes that modernize learning and assessment workflows in a rapidly evolving educational landscape.

2.2.2. User Authentication and Authorization with Microsoft OAuth Integration

Requirement:
Users must log in to the system using their Microsoft AUK accounts to ensure secure access and authentication.
How it works:

1. Upon accessing the system, users are prompted to log in via Microsoft OAuth.
2. Users are redirected to Microsoft's login page, where they provide their credentials.
3. After authentication, the system retrieves basic user profile information and tokens for secure access.

Note:
User accounts should be created automatically during the first login.
Non-AUK users shouldn't be able to log in and use the system.

2.2.3. Upload Study Materials

Requirement: Educators must be able to upload study materials (text files).
How it works:
The user uploads materials as text files to be able to use them for quiz creation.

The system stores materials as files.
Materials can have user-friendly names.
Users can also add some extra material as text without preupload it.

2.2.4. Generate Quizzes Based on Uploaded Materials

Requirement: Educators can generate quizzes using study material and customizable parameters.
How it works:
Users select the uploaded material from which the quiz will be generated.
Specify parameters for quiz generation, such as:

- Question number.
- The number of answers to a question
- Additional material as a text

The system displays a summary of the generated quiz and provides an option for manual editing or further refinement.

2.2.5. Edit generate Quiz

Requirement: Educators can view and edit quizzes (change title, questions and answers, add new items and remove present)

2.2.6. Export Quiz in QTI Format

Requirement: The system must allow quiz export in QTI (Question and Test Interoperability) format for Canvas LMS.
How it works:
The system generates a .zip file in QTI format containing formatted questions and metadata.
The user can download the QTI file.
Downloaded files can be uploaded into Canvas LMS.

2.2.7. Multi-Language Support

Requirement: Support multiple languages to generate quiz content.
How it works:
Users can upload materials in any of the widespread human languages (at least a few million natives) and get quizzes in these languages.

2.2.8. Administrative Features: System status Monitoring

Requirement: Administrators must check system health and background job statuses.

How it works:

The system provides a health check endpoint to check availability.

2.3. ChatGPT pricing

gpt-4o

$2.50 / 1M input tokens
$1.25 / 1M cached** input tokens
$10.00 / 1M output tokens

gpt-4o-mini

$0.150 / 1M input tokens
$0.075 / 1M cached** input tokens
$0.600 / 1M output tokens

gpt-3.5-turbo-0125

$0.50 / 1M tokens
$1.50 / 1M tokens[22]

The most cost-efficient model is gpt-4o-mini.

2.4. System Architecture

2.4.1. General Overview

The system is built on a modular architecture, separating concerns between the frontend, backend, and database to ensure scalability, maintainability, and efficiency. Database schema is shown on the Figure 2.

Frontend Responsibilities. The frontend layer represents the system's user interface (UI). It provides an interactive, user-friendly platform for educators, administrators, and other stakeholders to access the system's features. Designed using modern frameworks like React.js or Vue.js, the front end communicates with the backend through a secure RESTful API.

Backend Responsibilities. The backend layer acts as the core of the system. Built using Ruby on Rails (or another server framework), it handles all the business logic, process coordination, and communication between the front end, GPT-4 APIs, and the database. The backend provides a seamless API-first interface, enabling secure and efficient interaction between the front end and other components. Rails API application is organized using a classical MVC design pattern.

Database Responsibilities. The database layer is the data storage and retrieval foundation. Using a relational database like PostgreSQL, this layer stores persistent data for quizzes, user information, uploaded files, background jobs, and system analytics.

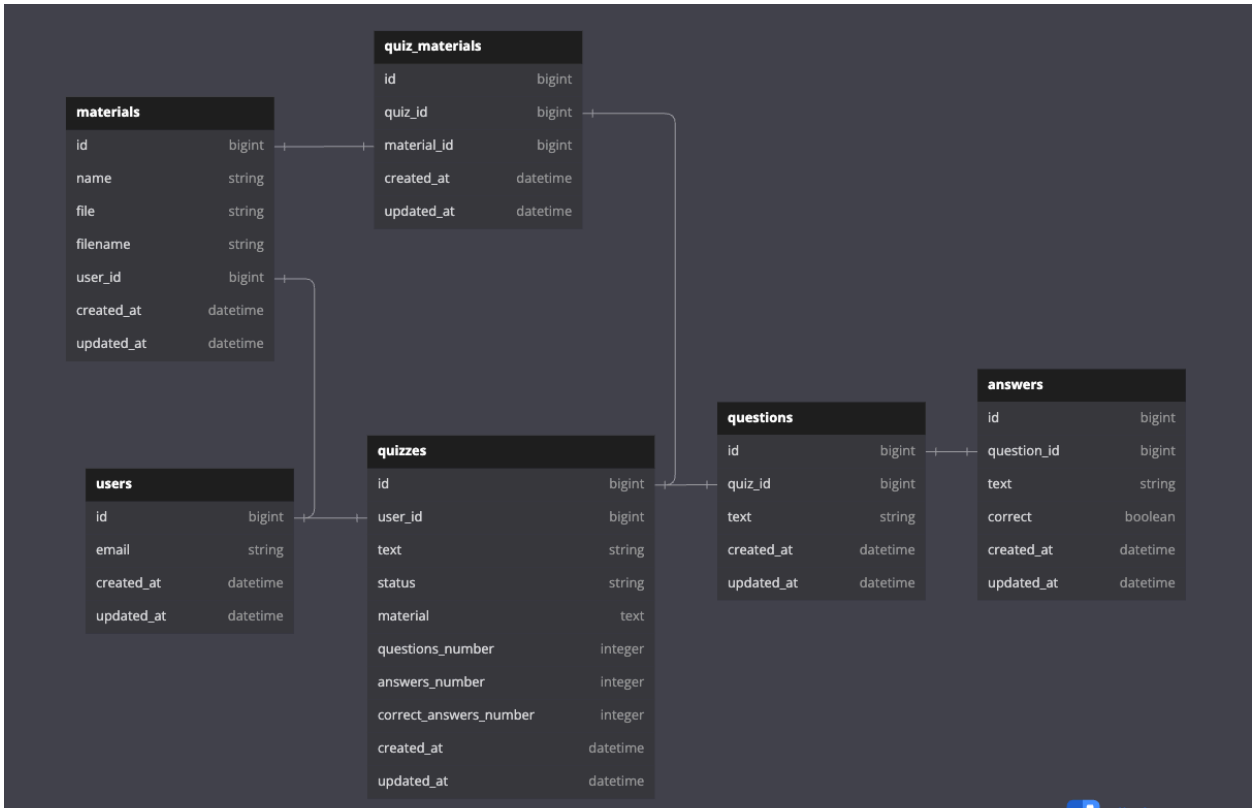


Figure 2. System's database schema

2.4.2. Controllers and their Responsibilities

In a web application, controllers act as intermediaries between the client (front-end) and the back-end services or database. They play a crucial role in the Model-View-Controller (MVC) design pattern by managing data flow through the application. (all the system Controllers are displayed at Figure 3)

The controllers handle incoming HTTP requests, interact with models to fetch or modify data, and construct and send appropriate HTTP responses back to the client. This separation of concerns ensures that the application remains modular, maintainable, and easy to test. The application becomes more organized, scalable, and secure by encapsulating the core logic for specific features into separate controllers.

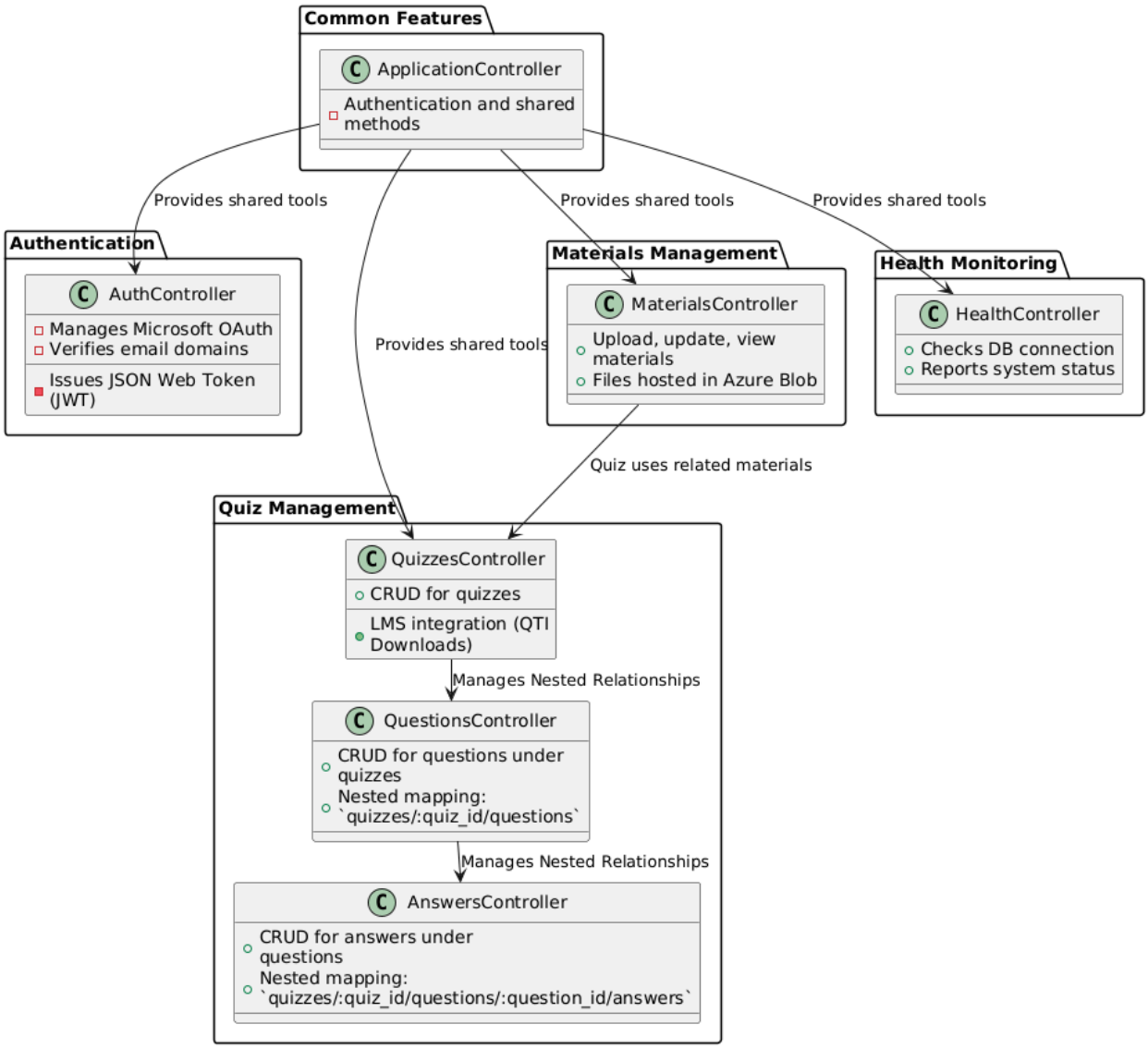


Figure 3. Controllers interconnection

AuthController

1) handles authentication and authorization via microsoft oauth.
2) manages oauth workflow, including generating an authorization url and handling the callback to retrieve tokens
3) verifies user emails, restricting access to those meeting specific criteria (e.g., valid domains), and generates jwt tokens for secure session management

QuizzesController

1) manages the creation, retrieval, updating, exporting, and deletion of quizzes
2) handles resourceful actions such as listing a user's quizzes (index), creating new quizzes by associating materials (create), and exporting quizzes as qti packages for lms integration (download_qti)
3) ensures that only the owner of a quiz can perform actions (e.g., update, delete, or view)

QuestionsController

1) handles operations related to questions within a specific quiz
2) enables users to retrieve all questions for a quiz (index), view a specific question (show), create a new question or update an existing question (create, update), and delete a question from a quiz (destroy)
3) ensures questions are always linked to a valid quiz owned by the current user

AnswersController

1) manages answer data for specific questions: lists all answers for a particular question (index), view, create, update, or delete an answer (show, create, update, destroy)
2) makes nested routing (/quizzes/:quiz_id/questions/:question_id/answers) to enforce relationships between answers, questions, and quizzes
3) validates business logic, such as ensuring that answers are tied to valid questions and quizzes

MaterialsController

1) handles actions related to materials (study resources) uploaded by users: lists all materials owned by the current user (index), enables users to upload, update, or remove materials (create, update, destroy), returns specific material details, such as its metadata or attached file (show)
2) enforces user ownership for all material operations and processes file attachments when uploading

ApplicationController

1) acts as a base class for all other controllers in the application

2) implements shared logic for authentication (authenticate_user!) and current user retrieval (current_user)
3) handles exceptions like expired or invalid json web tokens (jwts) and ensures secure, consistent request handling

HealthController

1) provides a lightweight endpoint (up) for health checks
2) confirms the database connection status and returns a simple json response indicating whether the backend system is operational
3) useful for infrastructure monitoring and ensuring the system's uptime

2.4.3. API part / RESTful Endpoints

Application routes and endpoints are demonstrated on Figure 4. Application routes interconnection and in Table 1. Application routes

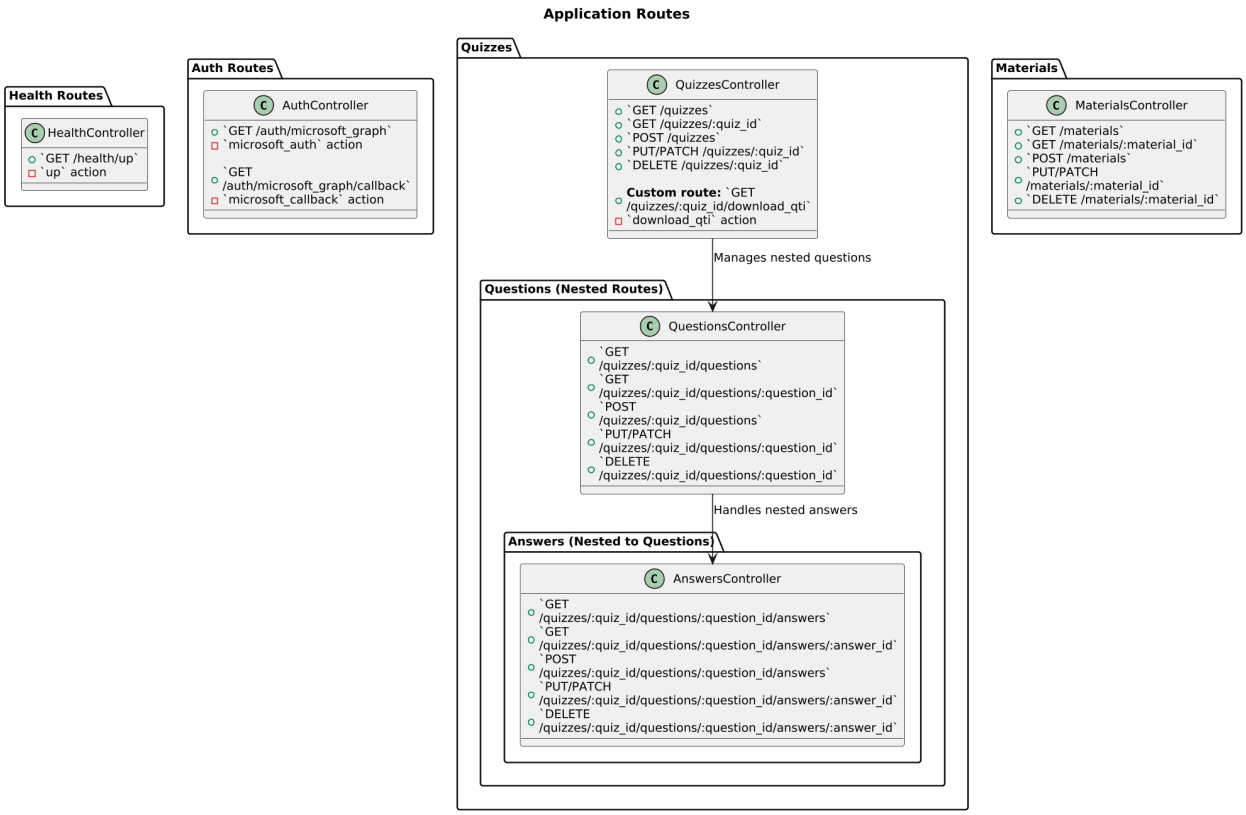


Figure 4. Application routes interconnection

Table 1. Application routes

Prefix	Verb	URI Pattern	Controller#Action
health_up	GET	/health/up(.:format)	health#up
auth_microsoft_graph	GET	/auth/microsoft_graph(.:format)	auth#microsoft_auth
auth_microsoft_graph_callback	GET	/auth/microsoft_graph/callback(.:format)	auth#microsoft_callback
download_qti_quiz	GET	/quizzes/:id/download_qti(.:format)	quizzes#download_qti
quiz_question_answers	GET	/quizzes/:quiz_id/questions/:question_id/answers(.:format)	answers#index

quiz_question_answers	POST	/quizzes/:quiz_id/questions/:question_id/answers(.:format)	answers#create
quiz_question_answer	GET	/quizzes/:quiz_id/questions/:question_id/answers/:id(.:format)	answers#show
quiz_question_answer	PATCH	/quizzes/:quiz_id/questions/:question_id/answers/:id(.:format)	answers#update
quiz_question_answer	PUT	/quizzes/:quiz_id/questions/:question_id/answers/:id(.:format)	answers#update
quiz_question_answer	DELETE	/quizzes/:quiz_id/questions/:question_id/answers/:id(.:format)	answers#destroy
quiz_questions	GET	/quizzes/:quiz_id/questions(.:format)	questions#index
quiz_questions	POST	/quizzes/:quiz_id/questions(.:format)	questions#create
quiz_question	GET	/quizzes/:quiz_id/questions/:id(.:format)	questions#show
quiz_question	PATCH	/quizzes/:quiz_id/questions/:id(.:format)	questions#update
quiz_question	PUT	/quizzes/:quiz_id/questions/:id(.:format)	questions#update
quiz_question	DELETE	/quizzes/:quiz_id/questions/:id(.:format)	questions#destroy
quizzes	GET	/quizzes(.:format)	quizzes#index
quizzes	POST	/quizzes(.:format)	quizzes#create
quiz	GET	/quizzes/:id(.:format)	quizzes#show
quiz	PATCH	/quizzes/:id(.:format)	quizzes#update
quiz	PUT	/quizzes/:id(.:format)	quizzes#update
quiz	DELETE	/quizzes/:id(.:format)	quizzes#destroy
materials	GET	/materials(.:format)	materials#index
materials	POST	/materials(.:format)	materials#create
material	GET	/materials/:id(.:format)	materials#show
material	PATCH	/materials/:id(.:format)	materials#update
material	PUT	/materials/:id(.:format)	materials#update
material	DELETE	/materials/:id(.:format)	materials#destroy

2.4.4. Jobs

In the project, jobs are crucial in handling background processes, ensuring that resource-intensive or time-consuming tasks do not block the main application thread. The system achieves flexibility and efficiency by utilizing *ActiveJob* as the

framework for encapsulating these tasks and *Sidekiq* as the processing engine. *ActiveJob* serves as the central abstraction layer for background jobs, allowing tasks to run asynchronously and improving the application's overall responsiveness. All jobs inherit from the *ApplicationJob* class, which is a shared foundation for defining retries, error-handling mechanisms, or other common behaviors. For instance, it allows the project to discard jobs gracefully if the underlying data becomes unavailable, such as when records are deleted. Retry configurations for database-related issues are also possible, ensuring that transient errors do not halt the processing pipeline.

At the core of the job system is *Sidekiq*. *Sidekiq*, the job processing engine, works with Redis to manage job queues and concurrency. Configurations for *Sidekiq*, defined in files like *sidekiq.yml* and *queue.yml*, ensure that job processing remains efficient and scalable. These files govern how workers handle different queues, such as polling intervals, thread counts, and concurrency settings. The concurrency level is particularly significant in production environments, where the system may need to simultaneously process a high volume of tasks. Developers can fine-tune this concurrency using the *JOB_CONCURRENCY* environment variable, which defines the number of processes and threads workers use. Moreover, these configurations are consistent across development, test, and production environments, allowing developers to test job behaviors predictably and minimizing discrepancies between local and production environments.

One of the key jobs in the project is the *GenerateQuizJob*. This job is central to the application's functionality as it handles the dynamic creation of quizzes based on user inputs and uploaded materials. The *GenerateQuizJob* scans the database for quizzes marked as new, signaling that they are ready to be processed. Each quiz is then updated to a processing status to indicate that they are currently being handled and to prevent duplicate work. During the job’s execution, material provided by the user—whether text fields or uploaded files—is prepared into a single cohesive string. This data, combined with the quiz's configuration, such as the number of questions and answers, is passed to the *OpenaiService*. This custom service interacts with an external system like *OpenAI* to generate the required quiz questions and answers. The external system responds with a structured dataset containing the quiz content, which is parsed and stored in the database.

Once the data is retrieved, *GenerateQuizJob* ensures that the old quiz questions and answers are cleared before adding the freshly generated ones. Each

question and its associated answers are stored in the database, reflecting the structure provided by the AI generation service. The next step involves creating a *QTI* package—a specialized file format for quizzes—using a custom service called QtiGenerationService. If an old *QTI* package exists for the quiz, it is replaced with the newly created one, ensuring that the output matches the latest updates. Finally, the quiz's status is marked as done, and a log message is printed to confirm that the processing has been completed. This step-by-step workflow encapsulates a complex process into a structured job, making it easy to manage and debug while keeping the main application thread free from heavy lifting.

Job execution is automated through scheduling. By integrating *Sidekiq* Scheduler with cron-like configurations, *GenerateQuizJob* is set to run every minute, ensuring new quizzes are processed in near real-time. The cron schedule, defined in sidekiq.yml, triggers the specific job class (*GenerateQuizJob*) with precision. Combined with the high polling frequency of Sidekiq workers, this keeps the system responsive to new requests. The broader queue configuration, specified in queue.yml, ensures smooth job execution by defining polling intervals and batch sizes for retrieving tasks. This setup supports parallel processing of multiple jobs and ensures timely execution without overwhelming system resources.

This background job architecture offers several key benefits. It offloads time-consuming processes like quiz generation into the background, allowing the user-facing application to remain responsive. The system is fault-tolerant, as retries and error-handling mechanisms are built into the *ActiveJob* architecture, ensuring that transient or recoverable issues don’t block jobs indefinitely. Moreover, configuring concurrency levels and job queue priorities guarantees that the system can scale with higher workloads during peak periods. Automation through cron schedules keeps repetitive tasks like quiz generation consistent without manual intervention, further streamlining operations.

2.4.5. Services

Services in this application are specialized classes that encapsulate specific business logic, keeping the codebase clean and well-organized. They are designed

to perform complex operations that often interact with external systems, manage reusable logic, or handle auxiliary tasks that don't belong directly in controllers or models as it is shown on the Figure 5. By isolating these responsibilities into dedicated service classes, the application promotes maintainability, scalability, and single-responsibility principles, making the codebase easier to test and extend.

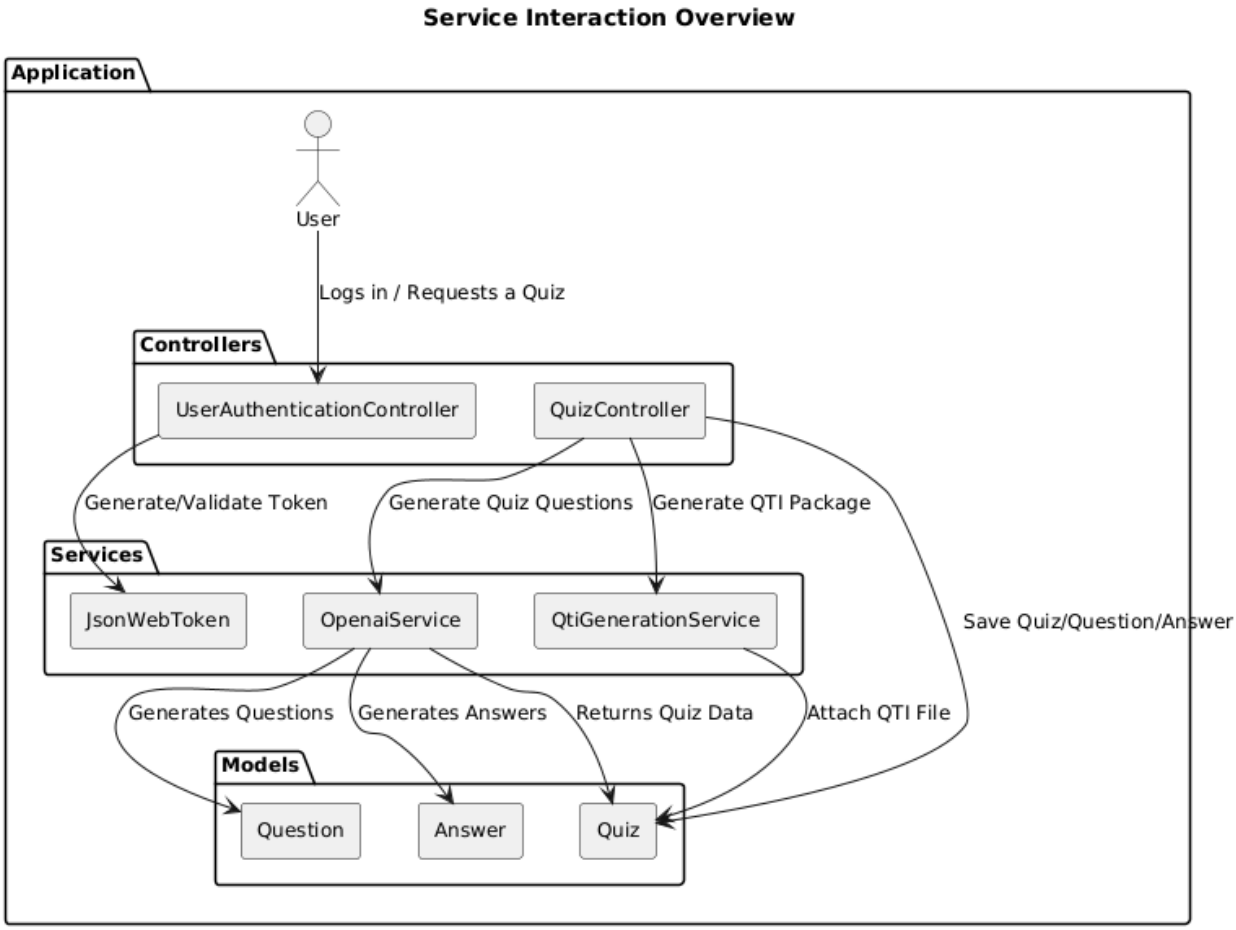


Figure 5. Service-user interactions

One key example is the *JsonWebToken* service, which provides tools for encoding and decoding *JSON Web Tokens* (*JWTs*). *JWTs* are commonly used for secure authentication, and this service ensures that tokens are handled efficiently and securely within the application. The service uses a secret key derived from the Rails application's secret_key_base to sign and verify the validity of tokens. The encoding method takes a payload and optionally sets an expiration time, defaulting to four hours. It then uses the *JWT* gem to generate the token, which is returned as

a string. On the other hand, decoding works by validating the provided token and parsing its payload into a *HashWithIndifferentAccess*, allowing convenient access to its contents regardless of key notation. The design of this service keeps all JWT-related logic centralized, improving security by reducing the risk of token-handling errors and making it easy to adapt if the app's authentication mechanism evolves.

Another example is the *OpenaiService*, which handles communication with the external OpenAI API. This service is responsible for constructing requests, sending them to OpenAI's GPT model, and processing the responses. For the application's quiz generation feature, the service accepts user-provided materials and quiz configuration parameters—such as the number of questions, answers per question, and correct answers—and formats them into a pre-defined AI prompt. It uses the *HTTParty* gem to send *HTTP POST* requests to the *OpenAI API* with the relevant headers and *JSON* payload. Upon receiving the response, the service parses it and extracts the generated quiz data, which is passed back to the parts of the application that process quizzes further. The OpenaiService encapsulates all complexity related to interacting with OpenAI, including API endpoints, headers, request format, and error handling. This makes it easier to manage third-party dependencies and reduces duplicative logic elsewhere in the application.

The QtiGenerationService is another significant example of a service class in this application. Its primary responsibility is to generate and attach *QTI* (*Question and Test Interoperability*) packages for *quizzes*. *QTI* is a widely supported file format in e-learning platforms, and this service ensures that quizzes in the application comply with these standards. The service begins by creating the necessary QTI files, such as imsmanifest.xml and quiz.xml, in a temporary directory. These files are generated dynamically based on the quiz's content, including questions, their answers, and metadata that describes the quiz structure. Once the files are prepared, the service packages them into a *ZIP* file using Ruby's Zip library, ensuring that all required files are compressed. Afterward, the *ZIP* file is attached to the quiz model via ActiveStorage, allowing the quiz to be easily exported or shared in a platform-compliant format. The service also includes proper error handling, logging issues during file generation or zipping, and ensuring that exceptions are raised if something goes wrong. The application maintains a clear separation of concerns by isolating the *QTI* generation process

into its service. It ensures that quiz generation remains modular and easy to extend in the future.

These services highlight the application's architectural focus on modularity and scalability. By delegating complex or reusable logic to services, the application reduces code duplication and keeps controllers and models focused on their core responsibilities. This design approach makes the app easier to maintain, especially as it grows in complexity since individual services can be tested and modified independently. Additionally, isolating external dependencies like *OpenAI* or *QTI* file generation into their services ensures that the rest of the application is shielded from API changes or specific implementation details of those systems. Overall, services in this project are carefully designed tools that enhance the application's robustness, maintainability, and reusability.

2.4.6. Models

In this application, models are fundamental components that represent and correspond to entities in the database. They bridge the gap between the application's logic and database, allowing developers to interact with stored data using the ActiveRecord ORM (Object Relational Mapping). Each model typically maps to a table in the database, with rows acting as instances of the model and columns corresponding to class attributes. Models play a crucial role by encapsulating business logic, enforcing validations, and defining the relationships between different entities, making working with the data layer easier.

The base of all models in the application is the *ApplicationRecord*, which inherits from *ActiveRecord::Base*. This is an abstract class where shared behavior for all models can be defined. Specific models inherit from *ApplicationRecord* to gain direct access to powerful *ActiveRecord* features such as querying, validations, and callbacks.

For instance, the *Quiz* model represents a quiz entity associated with multiple *Question* and *Material* entities. A *Quiz* belongs to a *User*, with many *Questions* and associated *Answers* for each question. (Figure 6) Furthermore, it integrates with *ActiveStorage* to manage file attachments like the *QTI (Question and Test Interoperability)* package, which enables quizzes to be exported in a platform-compatible format. The model enforces domain-specific constraints, such

as limiting the number of questions or ensuring that the correct answers do not exceed the total number. These validations safeguard the consistency and reliability of the database against invalid data.

The *Question* model, on the other hand, represents individual questions within a quiz. Each *Question* belongs to a *Quiz* and has many potential answers represented by the *Answer* model. The *Answer* model stores information about possible responses to a question, including whether the answer is correct. This hierarchical structure, where quizzes contain questions and questions contain answers, reflects the logical organization of quiz content.

Materials, represented by the *Material* model, are resources that belong to a *User* and can be associated with multiple *Quizzes* via the *QuizMaterial* join model. The *Material* model uses *ActiveStorage* to manage file attachments, such as PDFs or documents and includes custom validation logic to ensure files are attached, correctly sized, and valid. It also provides helper methods to retrieve URLs and paths for attached files, allowing seamless integration between the application logic and the storage system.

The *User* model manages information about the system's users. A user can own multiple *quizzes* and *materials*, and the model establishes these relationships using has_many associations. Additionally, it ensures that associated records, like *quizzes* and *materials*, are deleted when the *User* is destroyed, promoting data consistency.

The *QuizMaterial* join model serves as a linking entity between *Quizzes* and *Materials*. It allows many-to-many relationships, where multiple *quizzes* can use the same material, and a quiz can reference numerous materials. This intermediate model simplifies working with complex relationships and ensures proper database integrity.

Overall, the models in this application define a well-organized and structured data layer. By using *ActiveRecord* associations, validations, and callbacks, these models ensure that relationships between entities like users, quizzes, questions, and materials are both intuitive and efficient. They enable developers to perform complex operations on interconnected data with minimal effort while abstracting much of the underlying *SQL* logic.

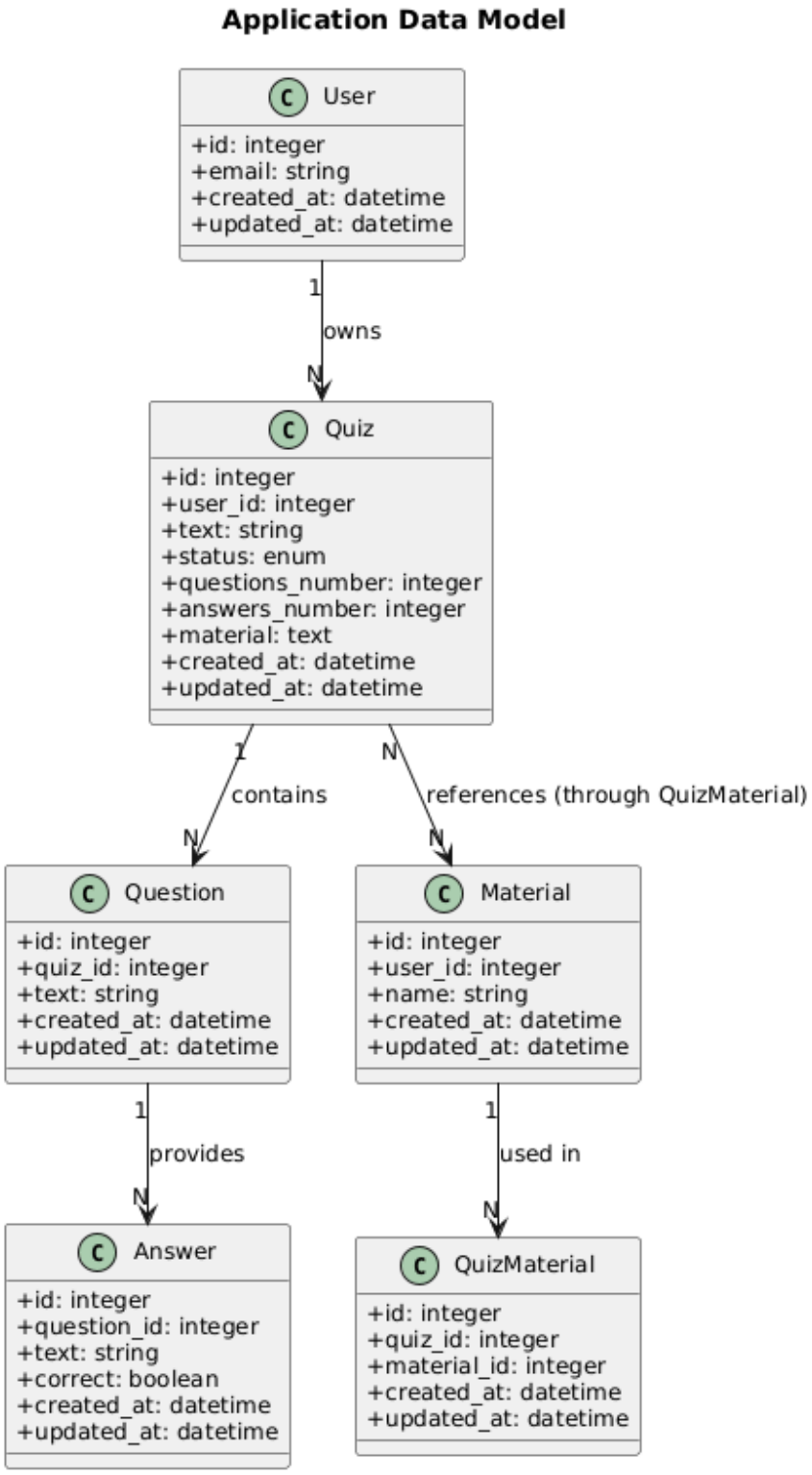


Figure 6. Application data model

3. Deployment

This is a detailed step-by-step guide on installing and running the Quiz Generator Application on Ubuntu 25.04.

First, ensure your system is up to date:

```
sudo apt update && sudo apt upgrade -y
```

Install the necessary system packages, including utilities, libraries, and tools for Ruby, Rails, PostgreSQL, and Node.js.

```
sudo apt install -y build-essential curl zlib1g-dev
libssl-dev libreadline-dev libyaml-dev libsqlite3-dev
sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev
software-properties-common libffi-dev git nodejs yarn
libpq-dev
```

Install Mise Version Manager

```
curl -fsSL https://get.mise.app | bash
echo 'eval "$(mise init)"' >> ~/.bashrc
source ~/.bashrc
```

Install Ruby Using Mise

```
mise install 3.4.1
mise global 3.4.1
ruby -v
gem install bundler -v 2.5.23
```

Install the PostgreSQL database, which is required for this application.

```
sudo apt install -y postgresql postgresql-contrib
```

Start and enable PostgreSQL to ensure it runs automatically on boot:

```
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

Create a Database User

```
   sudo -i -u postgres
   createuser --interactive --pwprompt
   exit
```

Clone code

```
git clone <repository-url>
cd <repository-name>
```

Install the required Ruby gems using Bundler:

```
bundle install
```

Configure Environment Variables

```
nano .env
```

Example content for `.env`:

```
MICROSOFT_CLIENT_ID=<your-client-id>
MICROSOFT_CLIENT_SECRET=<your-client-secret>
OPENAI_API_KEY=<your-api-key>
```

Update the database configuration in *config/database.yml* with credentials for the PostgreSQL user you created earlier:

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: 5
  username: <your-postgres-user>
  password: <your-postgres-password>
  host: localhost

development:
  <<: *default
  database: quiz_generator_development

test:
  <<: *default
  database: quiz_generator_test

production:
  <<: *default
  database: quiz_generator_production
```

Run the following to initialize the database:

```
rails db:create
rails db:migrate
```

Install Redis for Sidekiq

```
sudo apt install -y redis
```

```
sudo systemctl enable redis
sudo systemctl start redis
```

Start the Application to access the application in your browser at *http://localhost:3000*.

```
rails server
```

Start Sidekiq for Background Jobs (separate tab from server)

```
bundle exec sidekiq
```

4. Import of quiz to Canvas

Here is the detailed instruction how user can import Quiz into Canval LMS using QTI file generated by the System.

Step 1. Prepare the QTI File

Step 2. Log In to Canvas LMS

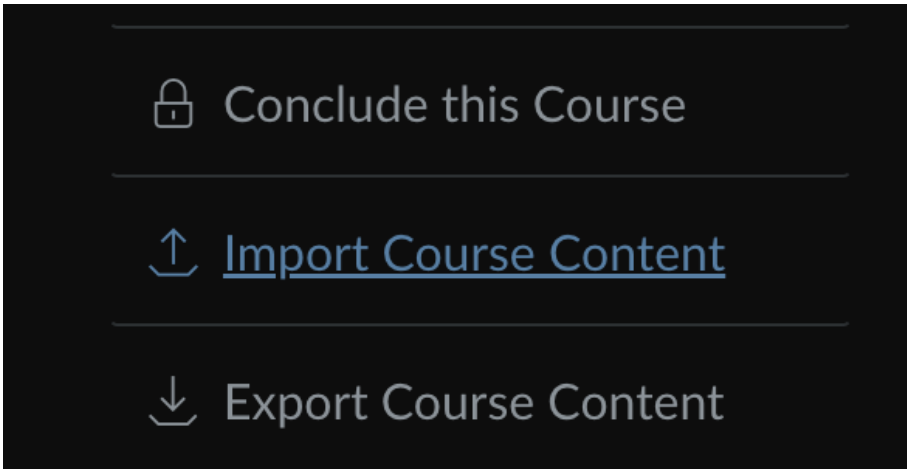
1. Open your web browser and navigate to your institution's Canvas LMS URL.
2. Log in with your credentials.

Step 3. Navigate to the Course

1. From the dashboard or "Courses" menu, select the course where you want to import the quiz.
2. Ensure you have instructor privileges for this course to access import settings.

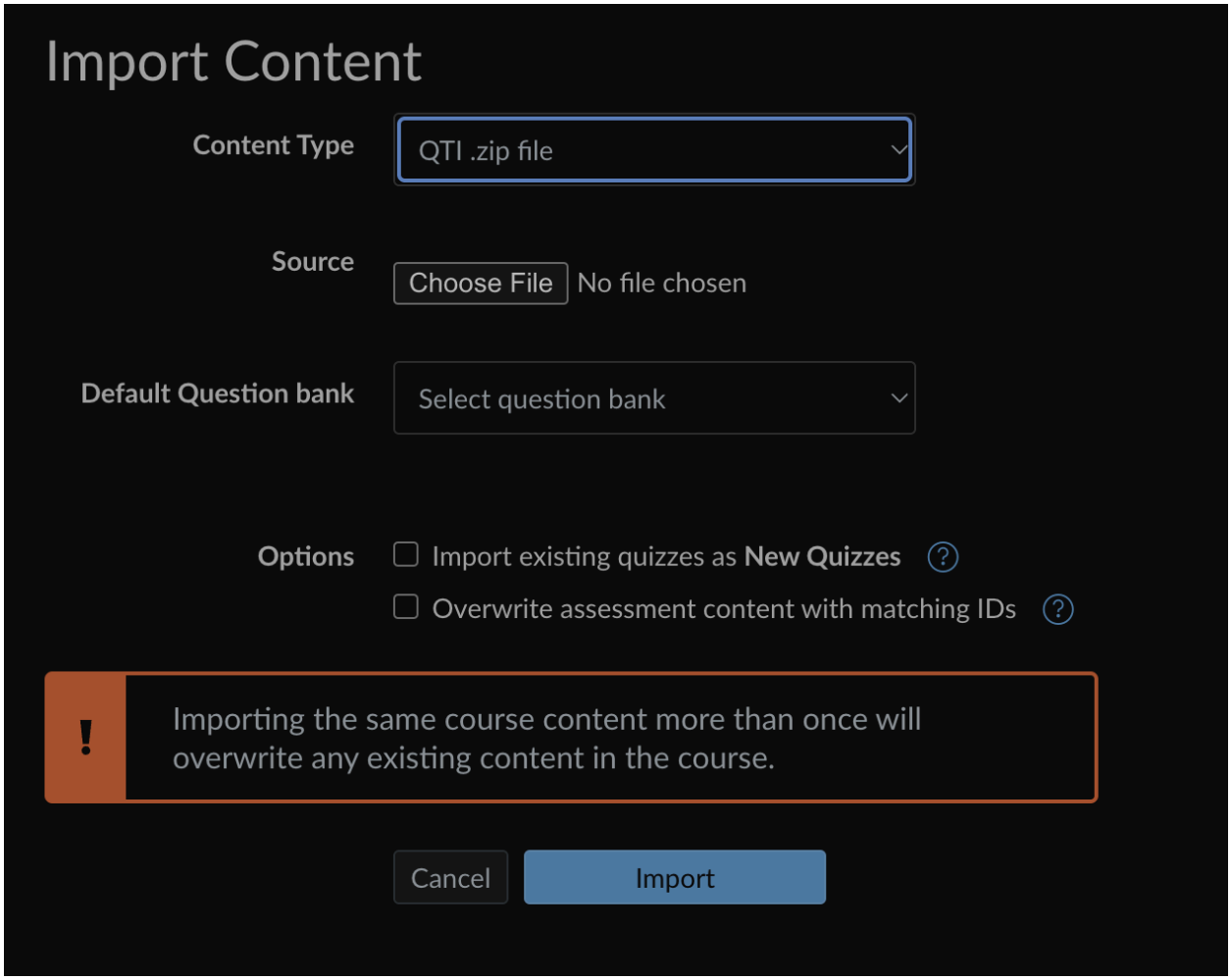
Step 4. Access the Import Tool

1. click Settings (usually found at the bottom) in the course navigation menu.
2. click the Import Course Content button on the Settings page on the right.



Step 5. Choose Content Type

1. In the “Import Content” screen, find the dropdown labeled Content Type.
2. Select QTI .zip file from the dropdown menu.



Step 6. Upload the QTI File

1. Click the Choose File button under “Source.”
2. Browse to the location of your QTI .zip file on your computer and select it.

Step 7. Choose Quiz Options

1. If you want the questions to remain as a question bank without creating a quiz, choose Create Question Bank and provide a name for the bank.
2. Leave this unchecked if you want the quiz to be directly available for students.

Step 8. Adjust Additional Settings (Optional)

1. Use the All Content or Select Specific Content options to control what is imported. For QTI files, “All Content” is typical.

2. Adjust quiz settings (like time limits or availability) after the import, as these are not always perfectly transferred.

Step 9. Start the Import

1. Click Import to start the upload process.

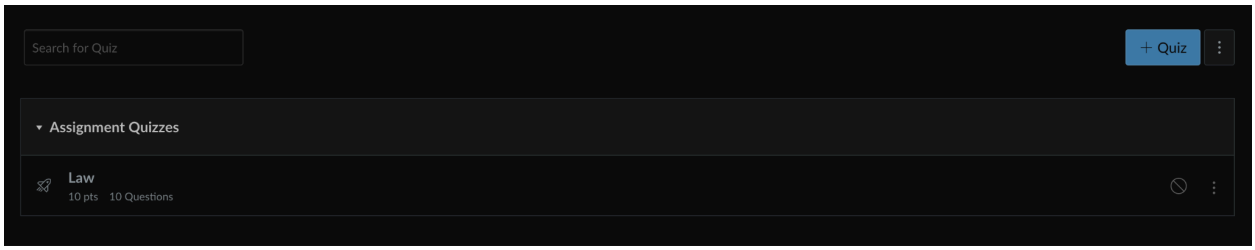
2. Wait for the upload to complete. This may take a few moments, depending on the file size.

Step 10. Verify the Import

1. Once the import finishes, you’ll see a green “Completed” status in the Current Jobs section.

2. Navigate to the Quizzes section in your course to verify that the quiz has been successfully imported.

3. Open the quiz to review questions, answers, and settings to ensure everything looks correct.



Step 11. Edit and Publish the Quiz (Optional)

1. If needed, edit the quiz by clicking on its name and selecting Edit.

2. Adjust quiz settings, such as time limits, availability, or instructions for students.

3. When satisfied, click Publish to make the quiz available to students.

5. Limitations

The Quiz Generator Application demonstrates excellent potential as a time-saving tool for educators and trainers, but there are clear areas where its functionality can be improved. The quality of the quizzes generated depends entirely on the content of the uploaded study materials. The application performs admirably when the input materials are well-structured and written, producing questions relevant to the provided topics. However, in cases where the input lacks structure or is overly complex, the system tends to struggle. Some generated questions may be overly simplistic, lack grammatical precision, or even fail to capture the deeper context of the study material. This indicates that while the application provides good baseline functionality, there is room for improvement in how thoroughly it understands and processes uploaded content.

If user will upload to system too short materials ChatGPT will start haluzinate and will create pretty much random questions for random topics. Also user should understand that ChatGPT bad in differentiation of dialects inside of language or even similar languages. This can cause student's confusion and can be negotiated, again, with high-quality materials on input.

The application does little to enrich, analyze, or contextualize the input beyond basic keyword or concept extraction. For example, if a study document includes nuances, implied ideas, or complex relationships between concepts, the application might miss these, leading to shallow or disconnected questions. In addition, the phrasing of the questions can sometimes be awkward or ambiguous, confusing for users attempting to answer. This highlights a need for better natural language processing (NLP) or contextual understanding capabilities to enhance the quality and clarity of quiz content.

The export functionality is helpful, allowing quizzes to be saved in QTI format, commonly used in academic settings. However, this limits the system to users who can work with QTI-compliant platforms such as Canvas. Educators using alternative LMS platforms may need export formats like DOX, PDF or SCORM for compatibility. The lack of broader integration options can reduce the system's flexibility and appeal to a wider audience.

From a user experience perspective, the ability to edit, refine, or customize the generated quizzes is limited. Once a quiz is generated, educators may need to manually adjust questions to align them better with the desired difficulty level or

focus. Adding features like quiz customization or filtering based on topics or preferred question types would significantly improve the system's usability for educators. A guided interface for editing questions, selecting key issues, or excluding irrelevant information would make it more user-friendly.

Relying on external integrations like Microsoft OAuth and potentially OpenAI's API (if used for quiz generation) introduces some risks. Downtime, rate limits, or configuration errors in these services could lead to disruptions in the system's key functionalities, such as login or quiz generation. Robust fallback mechanisms or alternative options for these integrations would make the system more reliable.

Overall, while the Quiz Generator Application solves a vital use case—automating quiz creation—it feels slightly underdeveloped in areas that could make it exceptional. Improving the quality and contextual understanding of generated questions by leveraging better NLP models and AI could significantly enhance the system's appeal. Additionally, supporting multiple export formats, improving background task handling, and introducing more robust editing and customization capabilities would take the application to the next level. With thorough refinement, this application could become a standout tool for educators and organizations looking to streamline their quiz creation processes.

6. Conclusions

This study systematically addressed the complexities associated with automating quiz creation in contemporary educational environments by leveraging the capabilities of Large Language Models (LLMs), specifically OpenAI's GPT-4. The project aimed to analyze prevailing methodologies, establish critical functional requirements, and develop a robust system architecture capable of integrating seamlessly with existing Learning Management Systems (LMS) like Canvas. These objectives were achieved through a scientifically grounded approach, showcasing the practical and transformative potential of AI-driven tools in educational workflows.

The research thoroughly examined current approaches to quiz generation, demonstrating the remarkable efficiency and adaptability of LLMs in producing diverse, contextually accurate, and pedagogically relevant assessments. The proposed solution utilized GPT-4 to generate quizzes that align with specific learning objectives, covering a wide range of cognitive demands from recall-based questions to higher-order analytical and creative tasks.

Key functional requirements for the automated quiz generation system were identified and implemented. These included secure user authentication using Microsoft OAuth, multi-language support to accommodate diverse linguistic needs, customizable parameters for quiz generation (e.g., question types, difficulty levels, and response options), and compatibility with standardized file formats such as QTI. By addressing these requirements, the system ensures high usability, flexibility, and adherence to educational standards, making it suitable for various institutional and pedagogical contexts.

A pivotal aspect of this work was the introduction of advanced prompt engineering techniques. These techniques were systematically designed to optimize the interaction between the LLM and the educational material, ensuring that generated quizzes are both relevant and aligned with frameworks such as Bloom's Taxonomy. This focus on structured and purposeful input design enhances the precision and relevance of the output.

The developed system architecture emphasized modularity, scalability, and maintainability. Built on the Ruby on Rails (RoR) framework, the backend employed a Model-View-Controller (MVC) design pattern to ensure logical separation of concerns and robust application structure. The architecture integrated

seamlessly with external APIs, including OpenAI's GPT-4 for quiz generation and Canvas LMS for content deployment. By leveraging Redis and Sidekiq, the system efficiently handled background processes such as quiz generation, file conversion, and API requests, ensuring high performance and scalability.

Features such as real-time editing, automated QTI exports, and system health monitoring further enhanced the system's reliability and utility.

In conclusion, this study not only demonstrated the significant value of LLMs in automating assessment creation but also contributed to the broader understanding of integrating AI into educational frameworks. The developed Ruby on Rails-based system, with its advanced functionality and robust architecture, represents a practical, scalable solution to modern educational challenges. By addressing technical, pedagogical, and usability considerations, this work highlights the potential for AI-driven tools to revolutionize traditional educational methodologies, creating more efficient, personalized, and impactful learning experiences.

References

1. OpenAI. (2020). GPT-3: Language Models are Few-Shot Learners. https://arxiv.org/abs/2005.14165
2. Brown, H. D. (2007). Principles of Language Learning and Teaching (5th ed.). Pearson Education.
3. Mayer, R. E. (2009). Multimedia Learning (2nd ed.). Cambridge University Press.
4. Mitkov, R., Ha, L. A., & Karamanis, N. (2006). A Computer-Aided Environment for Generating Multiple-Choice Test Items. Natural Language Engineering, 12(2), 177–194. https://doi.org/10.1017/S1351324906004177
5. Zeng, J., Wang, Z., & Zhang, W. (2020). Evaluation of Automated Item Generation Tools in Medical Education: A Systematic Review. Medical Education, 54(11), 1047–1055. https://doi.org/10.1111/medu.14281
6. Popenici, S., & Kerr, S. (2017). Exploring the impact of artificial intelligence on teaching and learning in higher education. Research and Practice in Technology Enhanced Learning, 12(1), 22. https://doi.org/10.1186/s41039-017-0062-8
7. Sung, L.-C., Lin, Y.-C., & Chen, M. C. (2007). An Automatic Quiz Generation System for English Text. Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007), 196–197. https://doi.org/10.1109/ICALT.2007.56
8. Anderson, L. W., & Krathwohl, D. R. (Eds.). (2001). A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. Longman.
9. Haladyna, T. M. (2004). Developing and Validating Multiple-Choice Test Items (3rd ed.). Lawrence Erlbaum Associates.
10. Clugston, R. W. (2005). Journey into Literature. McGraw-Hill.
11. Stearns, P. N. (2013). The Industrial Revolution in World History (4th ed.). Westview Press.
12. European Union. (2016). General Data Protection Regulation. https://gdpr.eu/
13. U.S. Department of Education. (2020). Family Educational Rights and Privacy Act (FERPA). https://www2.ed.gov/policy/gen/guid/fpco/ferpa/index.html
14. Federal Trade Commission. (1998). Children's Online Privacy Protection Act (COPPA). https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa

15. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A Survey on Bias and Fairness in Machine Learning. ACM Computing Surveys, 54(6), 1–35. https://doi.org/10.1145/3457607
16. Johnson, D. W., & Johnson, R. T. (2009). An Educational Psychology Success Story: Social Interdependence Theory and Cooperative Learning. Educational Researcher, 38(5), 365–379. https://doi.org/10.3102/0013189X09339057
17. World Wide Web Consortium (W3C). (2018). Web Content Accessibility Guidelines (WCAG) 2.1. https://www.w3.org/TR/WCAG21/
18. Brusilovsky, P., & Millán, E. (2007). User Models for Adaptive Hypermedia and Adaptive Educational Systems. In The Adaptive Web (pp. 3–53). Springer. https://doi.org/10.1007/978-3-540-72079-9_1
19. Canvas LMS API Documentation https://canvas.instructure.com/doc/api/
20. Elkins, S., Kochmar, E., Cheung, J. C. K., & Serban, I. (2024). How Teachers Can Use Large Language Models and Bloom's Taxonomy to Create Educational Quizzes. arXiv. https://arxiv.org/abs/2401.05914
21. Castro-Alonso, J. C., de Koning, B. B., Fiorella, L., & Paas, F. (2021). Five Strategies for Optimizing Instructional Materials: Instructor- and Learner-Managed Cognitive Load. Educational Psychology Review, 33(4), 1379–1407. https://doi.org/10.1007/s10648-021-09606-9
22. OpenAI Pricing Page https://openai.com/api/pricing/