**American University Kyiv**

# CUSTOMIZABLE LLM-BASED QUIZ GENERATION SYSTEM FOR PROGRAMMING COURSES WITH AN INTERACTIVE ADMIN PANEL

# НАЛАШТОВУВАНА СИСТЕМА ГЕНЕРАЦІЇ ТЕСТІВ НА ОСНОВІ LLM ДЛЯ КУРСІВ З ПРОГРАМУВАННЯ З ІНТЕРАКТИВНОЮ ПАНЕЛЛЮ АДМІНІСТРАТОРА

by Mykhailo Korzh

Presented in Partial Fulfillment of the Requirements for the Degree
Master of Software Engineering

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

2025

# Abstract

This paper investigates the design and implementation of an intelligent e-learning system called AUK Quiz Generator that uses large language models (LLM) to automate the creation and delivery of tests in programming education.

The thesis provides an overview of the relevant theoretical background, existing solutions, and approaches to solving quiz generation problems. The study also presents the rationale for the chosen approach and a detailed architecture of the AUK Quiz Generator.

In modern education, interoperability with e-learning platforms is critical, with an emphasis on responsive design to ensure high-quality question and answer generation.

# Acknowledgments

I would like to express my heartfelt gratitude to Dr. Sergiy Tytenko, my thesis supervisor, for his invaluable support, guidance, and encouragement throughout the development of this project. His expertise and constructive feedback were instrumental in shaping the success of this work.

I am also deeply thankful to American University Kyiv for offering me the resources, knowledge, and opportunities that enabled me to explore this topic in depth as part of my academic journey. The supportive environment and rigorous education provided by the university have been pivotal in achieving my academic and professional goals.

# 1. Overview of LLM-Based Quiz Generation Systems for Programing Education

## 1.1. Introduction

Artificial intelligence (AI) will become a cornerstone of education, significantly changing the way students and teachers interact with learning materials and each other. One of AI's most potentially powerful applications is the creation of educational tools and quizzes specifically designed for different programming languages. These tools will improve learning outcomes by providing adaptive, personalized, and engaging content that meets the needs of each student [1,2].

Programming languages are often complex and require a deep understanding of both theoretical concepts and practical applications. To help students master these skills, AI-powered educational tools use large language models (LLM) and natural language processing (NLP) to create quizzes and exercises. These tools not only automate the labor-intensive process of quizzes creation but also provide real-time feedback, constantly adapting to the student's learning style and progress. This article examines the development and application of AI-based educational tools in programming education, focusing on quizzes creation, personalized learning, and the challenges and opportunities of AI in this area

## 1.2. The Role of AI in Programming Education

Teaching programming constantly poses a variety of and, most importantly, unique challenges associated with the huge variety of languages with different levels of both overall complexity and approaches to working with them. While AI hasn't revolutionized education yet, it has provided many opportunities to automate routine processes in various aspects of education. These advances will allow teachers to spend less time on routine work and more time on individualized student approaches.

Tools that use large language models (LLMs) such as GPT-3, Gemini, and GPT4o can process learning materials such as presentations, books, lecture notes, and other educational materials. This is particularly useful in both preparing a summary[3] of information and generating questions and test tasks based on the information provided. Natural Language Processing (NLP) plays a key role in the formation of these materials in modern LLMs. It is thanks to high-quality NLP with the ability to understand the context and take into account previous material that a modern LLM can provide high-quality and meaningful results. In addition, automatic quiz generation saves time, and the quizzes cover the lecture material well [4]. This is especially useful in programming, where constant practice and testing of knowledge are essential for developing working skills.

The ability to generate multiple-choice questions has the potential to be a key factor in making programming easier to learn. This can be achieved by being able to tailor questions to different levels of difficulty and specific requirements.

For example, in one study, ChatGPT demonstrated a “moderate positive correlation between the difficulty ratings assigned by ChatGPT-4 and the perceived difficulty ratings given by participants” indicating its ability to generate relevant questions for learners at different levels [5].

## 1.3. AI-Driven Quiz Generation

Using cues, the LLM (ChatGPT) can generate high-quality distractors, with research showing that 58.8% of the distractors generated were rated as high-quality [6]. This ensures that quizzes do not mislead students with meaningless or obviously incorrect options.

Modern multimodal models have opened new possibilities for automating the creation of quizzes for educational purposes [5,8], which allows for a significant reduction in teachers’ time. Also, as highlighted in several studies [5,7], Natural language processing (NLP)-based models can generate questions with different levels of difficulty, making them suitable for different learners with different levels of proficiency [7]. The ability of these models to understand and analyze text at a sufficient level provides a high degree of confidence that the questions generated will be relevant and have educational value. For example, a web application developed in the project "ePub-to-Quiz Conversion with Large Language Models"[7] allows users to upload EPUB documents and create interactive quizzes. This tool clearly demonstrates how AI can be integrated into educational institutions at the existing level of technology.

Another major benefit of NLP-based LLM in educational tools is its potential to personalize learning. By using NLP to process responses in real-time, LLMs can provide rapid feedback (in the form of hints, recommendations for review or further study), thereby improving comprehension. As noted in the study [2] LLMs such as GPT-4 have demonstrated satisfactory results in generating accurate multiple-choice quizzes.

Recent research [1,6,8,11] shows that by tuning the model and properly organizing the prompt, LLM can be configured to create complex and effective quizzes.

Potentially, quizzes can be adjusted in real time based on previous answers, thereby focusing on areas of knowledge where the student needs to deepen their knowledge. This approach to quizzes generation can improve and reduce the speed and quality of learning

## 1.4. Challenges and Limitations of AI in Programming Education

Despite the many benefits of AI-based educational tools, there are a few challenges and limitations when implementing them in programming education:

Quality control: While AI can automate quiz generation, there is always a risk of generating questions with errors or ambiguities. AI-generated quizzes require human review to ensure that the questions are technically accurate and relevant to the learner’s goals [4]. Additionally, because programming often requires a sophisticated understanding of algorithms

and data structures, it can be difficult for AI systems to create exercises that are both challenging and educationally useful [5].

Bias in AI models: AI models are trained on large data sets, and these data sets can carry inherent biases [9]. In the context of learning to code, this can lead to questions or explanations that favor certain languages, practices, or paradigms over others. This can limit the diversity of learning materials presented to students and potentially skew their understanding of programming best practices [4].

Technical limitations: AI-based educational tools, especially those that rely on LLM, are computationally expensive. Running these models can be expensive, especially in real-time applications where immediate feedback is needed. Additionally, while some models, like GPT-4, have extensive training, they still have limitations in terms of the amount of content they can process at one time, which can prevent them from creating comprehensive quizzes from long learning materials [4, 9].

Over-reliance on AI: There is a risk that students may become overly reliant on AI-generated feedback and tests, potentially reducing the role of critical thinking and self-assessment in the learning process. While AI tools provide valuable support, educators must ensure that students also develop the ability to critically assess their own work, rather than always relying on automated systems [4].

Lack of Reflective Judgment: A major problem with AIs is their limited ability to identify incorrect or irrelevant answer options. Many LLMs, such as GPT-4, tend to blindly follow instructions and may attempt to select an answer even if none of the options are correct. This limitation, known as a lack of "reflective judgment," limits the model's ability to handle complex questions. This problem is especially relevant in learning to code, where accurate interpretation and critical analysis are critical to both creating quizzes questions and assessing their correctness [10].

## 1.6. Future Directions for AI in Programming Education

As AI technology continues to advance, we can expect to see several interesting future applications in programming education:

Advanced Adaptive Learning: The next generation of AI-powered educational tools will benefit from further advances in adaptive learning algorithms. These systems should be able to develop a deeper, more nuanced understanding of individual learners' needs, allowing them to adjust the difficulty and focus of tests in real-time. With more advanced personalization, AI tools will be able to provide fully customizable learning paths that adapt not only to a learner's skill level but also to their preferred learning style, ensuring maximum engagement and effectiveness.

Support for more niche programming languages: While AI-powered tools currently focus on popular programming languages like Python, Java, and C++ [11, 13], future AI models will need to expand their capabilities to more specialized languages such as Rust, Haskell, and Go. This will provide students with more diverse learning opportunities and better preparation for different programming paradigms and technologies.

Improved feedback mechanisms: As AI models become more sophisticated, they will need to provide more detailed and meaningful feedback. Instead of simply marking an answer

as correct or incorrect, AI tools should be able to deeply analyze a student's code, offering feedback on performance optimization, code readability, and adherence to best practices. This type of feedback will be invaluable to students, especially those who aspire to become professional developers.

AI for Collaborative Learning: Future AI-powered tools could facilitate collaborative learning by allowing students to work on AI-assisted group projects. By analyzing group dynamics and individual contributions, AI could help identify areas where certain students need more support or where collaboration strategies could be improved. This would be especially useful in team-based programming environments, where effective collaboration is key to success.

Enabling multimodal learning: Future AI tools should support multimodal learning methods that combine text, audio, and visual learning materials. For example, AI should be able to generate quizzes based on video lessons, analyze students' code execution during real-time coding sessions, or provide real-time visualization of code performance.

## 1.7. Investigation of Prompt Design for Quiz Generation

The role of large language models (LLM) in creating educational content has expanded significantly, especially in the field of programming education. This section examines the research, testing, and refinement of a structured prompt for creating tests, highlighting its potential for addressing the specific needs of programming courses.

The goal of the research was to create a flexible, efficient, and scalable prompt that could handle a variety of educational scenarios. This included creating tests for different levels of difficulty, adapting to incomplete or insufficient materials, and providing output in a format suitable for integration into an online learning system.

The following prompt was developed to meet these objectives:

```
    Generate a quiz with the following parameters:
    - Quiz Name: ${quizName || "Unnamed Quiz"}
    - Material: ${material || "No material provided."}
    - File Content: ${fileContent || "No file content provided."}
    - Multiple Choice: ${multipleChoice ? "Yes" : "No"}
    - Allow Supplemental Questions: ${allowSupplementalQuestions ? "Yes" : "No"}
    - Generate exactly between ${minQuestions} and ${maxQuestions} questions.
    - Each question must include ${minAnswerOptions}-${maxAnswerOptions} answer
options.

    ${
      allowSupplementalQuestions
        ? "If the provided material or file content is insufficient for generating the required
number of questions,
```

```
                supplement the quiz with questions on the same general topic."
      : "Do not generate questions not based on the provided materials."
  }

  Ensure that the final output contains at least ${minQuestions} questions, and strictly
adhere to the format:
  {
   "quizName": "Quiz Name",
   "questions": [
    {
     "question": "Question text",
     "options": ["Option 1", "Option 2", ...],
     "correctAnswer": "Correct Option",
     "isSupplemental": true // Optional, for supplemental questions
    },
    ...
   ]
  }
```

This prompt is an integral part of the React page code, specifically designed for generating quizzes based on user inputs from a web form. The parameters for the prompt, such as quiz name, material, file content, and various quiz settings, are dynamically populated based on the values provided by the user. The React component ensures that the prompt adapts to the user's choices while maintaining a structured format for integration with backend services and large language models.

### 1.7.1 Analysis of Key Parameters

Quiz Name

Purpose: Ensures that the quiz has a clear identifier for organizational and retrieval purposes.

Implementation:

```
Quiz Name: ${quizName || "Unnamed Quiz"}
```

If no name is provided, a default value ("Unnamed Quiz") is assigned to avoid ambiguity.

Material and File Content

Purpose: Provides the foundational context for question generation, ensuring the quiz remains thematically relevant.

Implementation:

```
Material: ${material || "No material provided."}
File Content: ${fileContent || "No file content provided."}
```

This dual-layered approach allows the inclusion of textual material and external files, such as lecture notes or textbooks, broadening the content scope.

Multiple Choice

Purpose: Specifies the format of the questions, determining whether single or multiple correct answers are expected.

Implementation:

```
Multiple Choice: ${multipleChoice ? "Yes" : "No"}
```

This enables the generation of diverse question formats, catering to different pedagogical needs.

Allow Supplemental Questions

Purpose: Addresses the challenge of insufficient material by enabling the generation of additional questions on related topics.

Implementation:

```
${
  allowSupplementalQuestions
    ? "If the provided material or file content is insufficient for generating the required number of questions, supplement the quiz with questions on the same general topic."
    : "Do not generate questions not based on the provided materials."
}
```

This parameter ensures that the quiz remains complete without compromising thematic relevance.

Question Count

Purpose: Controls the scope of the quiz by defining the minimum and maximum number of questions.

Implementation:

```
Generate exactly between ${minQuestions} and ${maxQuestions} questions.
```

Answer Options

> Purpose: Defines the number of answer choices per question, supporting varying difficulty levels.

Implementation:

```
Each question must include ${minAnswerOptions}-${maxAnswerOptions} answer options.
```

Output Format

> Purpose: Specifies the required JSON structure for seamless integration into learning management systems.

Implementation:

```
Ensure that the final output contains at least ${minQuestions} questions, and strictly adhere to the format:
{
  "quizName": "Quiz Name",
  "questions": [
    {
      "question": "Question text",
      "options": ["Option 1", "Option 2", ...],
      "correctAnswer": "Correct Option",
      "isSupplemental": true // Optional, for supplemental questions
    },
    ...
  ]
}
```

### 1.7.2 Challenges Identified

Despite the success of the prompt, several challenges were observed:

1. Contextual Accuracy: In some cases, the model generated questions that were only loosely connected to the material, requiring manual revision.

2. Complexity Calibration: While most questions were appropriately challenging, a minority were either too simple or overly complex, suggesting a need for improved tuning of difficulty parameters.
3. Bias in Supplemental Questions: When supplemental questions were generated, there was a slight tendency to overemphasize general programming topics, occasionally straying from the course-specific material.

Out study demonstrates the critical importance of well-structured prompts to exploit the full potential of LLM test creation. The developed prompt combines flexibility, adaptability, and accuracy sufficiently for use in AI-based educational tools.

## 1.8. Chapter 1 Conclusions

Integrating AI into programming education, especially through large language models (LLM) and natural language processing (NLP), is revolutionizing how students and teachers interact with learning materials. This article explores how AI-powered tools automate test generation, provide personalized learning experiences, and adapt to individual learners' needs. Despite challenges such as contextual accuracy and technical limitations, AI brings undeniable benefits including efficiency, scalability, and personalized feedback.

To summarize the above, we can highlight the requirements that will meet the modern needs of the educational process and the tasks associated with increasing the effectiveness of testing.

Firstly, materials management is a critical element, since the quality of educational texts directly determines the value of the final questions. It should be possible to upload and systematize materials, and then delete, rename or download them if necessary. This will allow for the control of materials and reduce the risk of using outdated data.

Secondly, the test settings should allow teachers to flexibly adapt quizzes to specific educational tasks: specify the number of questions, answer options. Such adaptation is necessary to meet different levels of learning complexity.

Finally, it is important to ensure test generation using NLP, which makes it possible to quickly and efficiently generate a significant volume of questions based on analyzed texts. It is important that the system automatically adds a set of distractors to each question, allowing for a more complete test of students' knowledge and avoiding simple guesses. Exporting quizzes in this format will allow for convenient integration with common LMS platforms and will simplify the automation of the educational process.

All these requirements - from flexible material management to intelligent question generation and export capabilities - reflect the desire to create a comprehensive solution that will effectively support the educational process and simplify the work of teachers.

# 2. Design and Implementation of the Customizable Quiz Generation System

The Customizable Quiz Generation System, hereafter referred to as the AUK Quiz Generator, was conceived to simplify the creation and administration of tests in educational settings, particularly in the context of programming courses. During the design phase, special attention was paid to automating quiz creation, supporting personalized learning, and integrating with learning management systems (LMS). This chapter outlines the primary objectives, use cases, and architectural decisions that informed both the conception and the actual implementation of the system.

## 2.1. Business Goals of the System

The main goal of the AUK Quiz Generator is to reduce the time teachers spend on preparing test assignments and improve the quality of assessment materials through the reasonable use of automatic generation technologies. In addition, the problem of the need to generate quizzes for different levels of educational programs should be solved.

Taking these goals into account, the following key business goals of the project were formulated, each of which reflects an important aspect of efficient and convenient automation of test creation:

1. Automating the Quiz creation process. The first major goal is to automate quiz creation. By adopting an automated approach, the system alleviates the burdens traditionally placed on instructors who devote substantial time to writing questions and carefully constructing distractors. Freed from the need to produce and manage quizzes by hand, instructors can focus on developing curricula and guiding students more effectively. Moreover, the quiz-creation process is driven by the materials that instructors upload – such as lecture summary, books or notes—ensuring that the resulting assessments remain aligned with the course content.
2. Supporting Personalized Learning.The system should provide tools for fine-tuning tests: teachers should be able to manually review and edit automatically generated questions. This will allow teachers to tailor generated questions to specific course objectives or the unique needs of a student group, and will contribute to improving the effectiveness of learning by more accurately matching tests to the actual needs of students;
3. Enhancing and Usability. It is necessary to provide a simple and intuitive interface that does not require deep technical skills. In addition, it should be possible to export tests in the QTI format, which is compatible with many LMSs (e.g. CANVAS). This compatibility will expand the potential user base of the system and ensure that existing infrastructures can be used for the maximum number of learning systems.

Several key use cases were developed to illustrate how the AUK Quiz Generator meets the real needs of teachers:

- The system allows the teacher to upload files  TXT with lectures or notes;
- The teacher sees a list of uploaded materials and can rename or delete them;
- Generate quizzes from selected.

In one foundational use case, instructors upload and manage materials by adding new files containing lectures or notes, then renaming or deleting those files if necessary. A second key use case relates to generating a quiz, where instructors select one or more previously uploaded documents and specify certain parameters, such as the number of questions and answer options required. The system communicates with the server, which processes the data and returns a fully generated quiz. Instructors can then edit the quiz by modifying the phrasing of questions, adding new items, and removing irrelevant ones. Finally, a fourth use case focuses on managing created quizzes, which includes renaming or deleting quizzes, as well as downloading them in QTI format.

The functional requirements outlined below serve as directives for how the system should function to meet the aforementioned goals and use cases. They guide the processes of designing, coding, and testing, while simultaneously clarifying how instructors will utilize the platform to create and manage quizzes.

The effectiveness of the AUK Quiz Generator depends heavily on the quality and completeness of the source materials from which the questions and answers are derived. Therefore, the system must offer a convenient and reliable file management mechanism.

Requirements for Material Management:

- Teachers must upload source materials in TXT format through an intuitive user interface;
- Display a summary list of uploaded files, including:
    - File names.
    - Metadata such as upload dates and original file names
- Allow renaming of files to ensure consistent naming conventions;
- Allow deletion of obsolete or unnecessary files;
- Provide download functionality for local storage;
- Securely store uploaded materials in a database for future use;
- Assign unique IDs to each uploaded file for tracking and identification during test setup.

To ensure that tests accurately match learning objectives, the system must allow users to customize the generation parameters. Teachers must choose which materials will serve as a source for generating questions.

Requirements for Quiz Configuration:

- Enable instructors to select specific materials using unique file IDs as the basis for question generation;
- Allow customization of quiz attributes, including:

- Target number of questions.
- Number of answer options per question (e.g., 3–5).
- Custom quiz naming.

After specifying the necessary configuration, the instructor initiates the automated generation of the quiz. During this phase, the system applies natural language processing (NLP) methods to analyze the selected texts and identify appropriate material for question creation.

Requirements for Quiz Generation:

- Leverage Natural Language Processing (NLP) techniques to extract question-relevant material from the selected files;
- Generate plausible but incorrect answers (distractors) for each question;
- Create quizzes in JSON format, including:
  - Question text.
  - Metadata such as creation date.
  - Link to a QTI-compatible archive for LMS integration.

Quizzes are returned in JSON format. Example structure:


```
{
      "id": 1,
      "text": "Japan Quiz",
      "status": "done",
      "material": null,
      "questions_number": 10,
      "answers_number": 4,
      "created_at": "2025-01-09T11:16:20.787Z",
      "updated_at": "2025-01-09T14:05:22.967Z",
      "qti_package_url":
"http://localhost:3000/rails/active_storage/blobs/redirect/eyJfcmFp
bHMiOnsiZGF0YSI6MywicHVyIjoiYmxvYl9pZCJ9fQ==--
44742250a9d3ef1c1ce12125042e04682882233b/japan-quiz-qti.zip",
      "questions": [
            {
                  "id": 1,
                  "text": "What is the capital of Japan?",
                  "answers": [
                        {
                              "id": 1,
                              "text": "Tokyo",
                              "correct": true
```

```
                    },
                    {
                        "id": 4,
                        "text": "Hiroshima",
                        "correct": false
                    },
                    {
                        "id": 2,
                        "text": "Kyoto",
                        "correct": true
                    },
                    {
                        "id": 3,
                        "text": "Osaka",
                        "correct": true
                    },
                    {
                        "id": 46,
                        "text": "Kyoto2",
                        "correct": false
                    }
                ]
            }
        ],
        "materials": [
            {
                "id": 2,
                "name": "japan.txt",
                "filename": "japan.txt",
                "file_url":
"http://localhost:3000/rails/active_storage/blobs/redirect/eyJfcmFp
bHMiOnsiZGF0YSI6MSwicHVyIjoiYmxvYl9pZCJ9fQ==--
4a3cbdaf9385d6adfbbb5f035ad62b88be82c1b6/japan.txt"
            }
        ]
}
```


Since automatic generation alone cannot always cover the specialized nuances of a given course, the system should provide teachers with the ability to manually refine and improve the test content. Users should be able to edit quizzes.

Requirements for Quiz Editing:

- Provide users with the ability to view quizzes, all questions and answer options.
- Ability to edit the text of questions.
- Ability to edit the text of answers.
- Ability to add new questions.
- Ability to add new questions.
- Ability to delete questions.
- Ability to delete answer options.
- Determine the correctness of each answer option.

## 2.2. System Architecture

The AUK Quiz Generator is built on a modular architecture that separates the responsibilities of the user interface (frontend) from those of the server (backend). This approach simplifies system maintenance and upgrades [13], while also enabling the system to scale effectively as additional capabilities are introduced.

### 2.2.1. High-Level Overview

At its core, the AUK Quiz Generator adopts a three-tier architecture that separates the system into the presentation layer (frontend), the application layer (backend), and the data layer (database). This structure is grounded in well-established software engineering principles that emphasize separation of concerns and scalability.

Within this architectural model, the presentation layer is dedicated to user interaction. It receives input through a graphical interface and displays output to instructors who are generating or managing quizzes. By keeping the user interface distinct from the underlying logic, changes to the frontend—such as a new design aesthetic or an additional interface element - can be introduced with minimal impact on the rest of the system.

The application layer (backend) hosts the core business logic of the AUK Quiz Generator. It processes requests from the frontend, applies rules regarding quiz generation and editing, and communicates with the database to retrieve or store data. This layer is designed to be highly modular, ensuring that key functionalities like user authentication, file handling, and quiz generation can be extended or refined without necessitating wholesale rewrites of the server code.

Finally, the data layer is responsible for storing and retrieving all persistent information, including user details, uploaded materials, and metadata related to quizzes. By centralizing data management, the system achieves consistency and reliability, particularly when simultaneous operations—such as multiple instructors editing different quizzes—occur. Moreover, this layered approach facilitates horizontal scaling, as additional servers or databases can be integrated to handle increased workload or expanded user bases.

Data flow among these layers follows a clear request-response cycle: the frontend issues a request to the backend (e.g., to generate a quiz), the backend processes this request, consults

the database as needed, and then returns a response to the frontend. This design ensures that each layer can evolve independently, allowing, for instance, a switch to a different database management system or an update to more advanced user-interface frameworks, without compromising the overall system integrity.
The diagram in Figure 1 below shows the three-tier architecture of the AUK Quiz Generator, which divides the system into frontend, application and data layers.

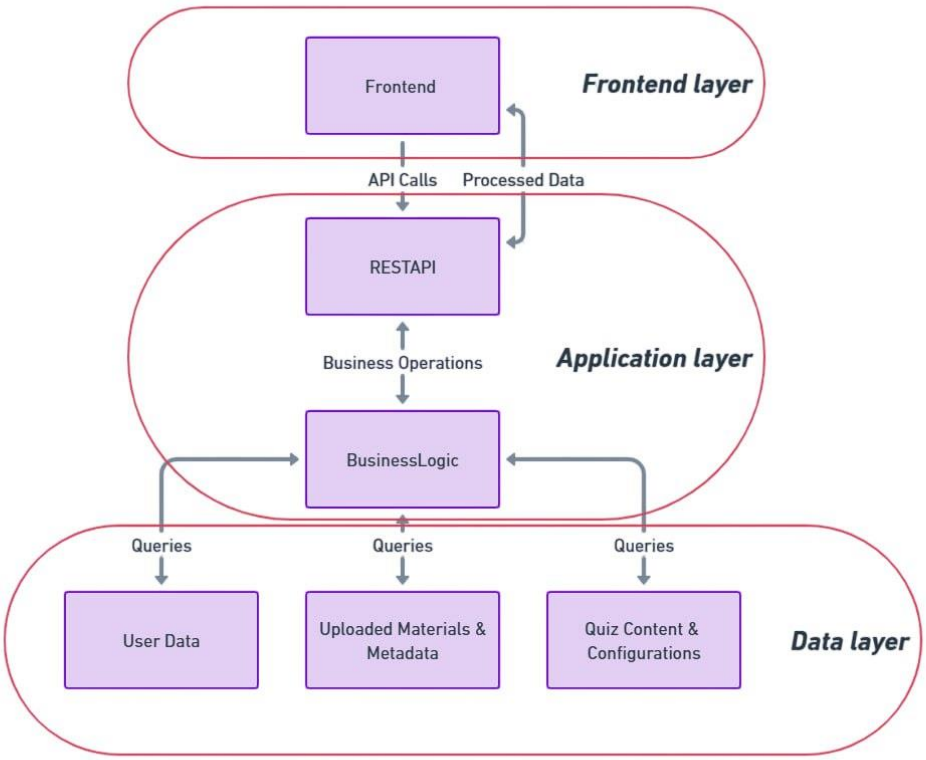


Figure 1: High-Level System Architecture

### 2.2.2. Frontend Architecture

The AUK Quiz Generator UI is built with React.js, which uses a component-based architecture that improves modularity and maintainability[14]. In this paradigm, each UI feature is encapsulated in a standalone component responsible for a separate part of the overall functionality. This approach follows the single responsibility principle, making the code more consistent and easier to test or refactor.

The foundational element of this architecture is the header component, which serves as the main navigation bar and relies on react-router-dom for client-side routing. By managing page transitions in the browser, the header component provides a seamless single-page

application (SPA) experience. Below the header are domain-specific components, each dedicated to a specific feature of the system.

The materials component provides an interface for uploading, renaming, or deleting instruction files. Users can easily interact with the system by selecting files TXT and tracking their status in a list. This not only simplifies the content management workflow, but also ensures that the source materials needed to generate questions are clearly organized and easily accessible.

In the QuizGenerator component, instructors define the parameters that will form the automatically generated tests. By specifying the desired test name, number of questions, number of answer options, or corresponding source materials, they create an accurate “task description” for the backend. Once the user submits these parameters, the component sends a request to the server, which responds with a newly generated test based on the provided data.

The QuizList component is responsible for displaying all previously generated tests. Instructors can rename tests for clarity, delete those that are no longer needed, or download them in QTI format for use in external LMS environments. The real-world nature of this list ensures that any new tests or updates (e.g. after editing) are immediately visible and accurately reflected in the user interface.

When further refinement is required, the QuizReviewModal component provides an interactive window for editing question text, adding or removing answer options, and setting or unsetting correct answers. By limiting editing tasks to the modal interface, the system preserves the user's broader workflow by allowing them to focus on one quiz at a time, but still offers detailed control over the content of each question.

The diagram in Figure 2 shows the frontend architecture of the AUK Quiz Generator. It highlights key React.js components, including the header component for navigation and routing, as well as domain-specific components such as the materials component, the QuizGenerator component, the QuizList component, and the QuizReviewModal component. These components communicate with the backend through a centralized REST API, enabling efficient communication with the database. This modular design ensures a seamless user experience and supports ease of maintenance and scalability.

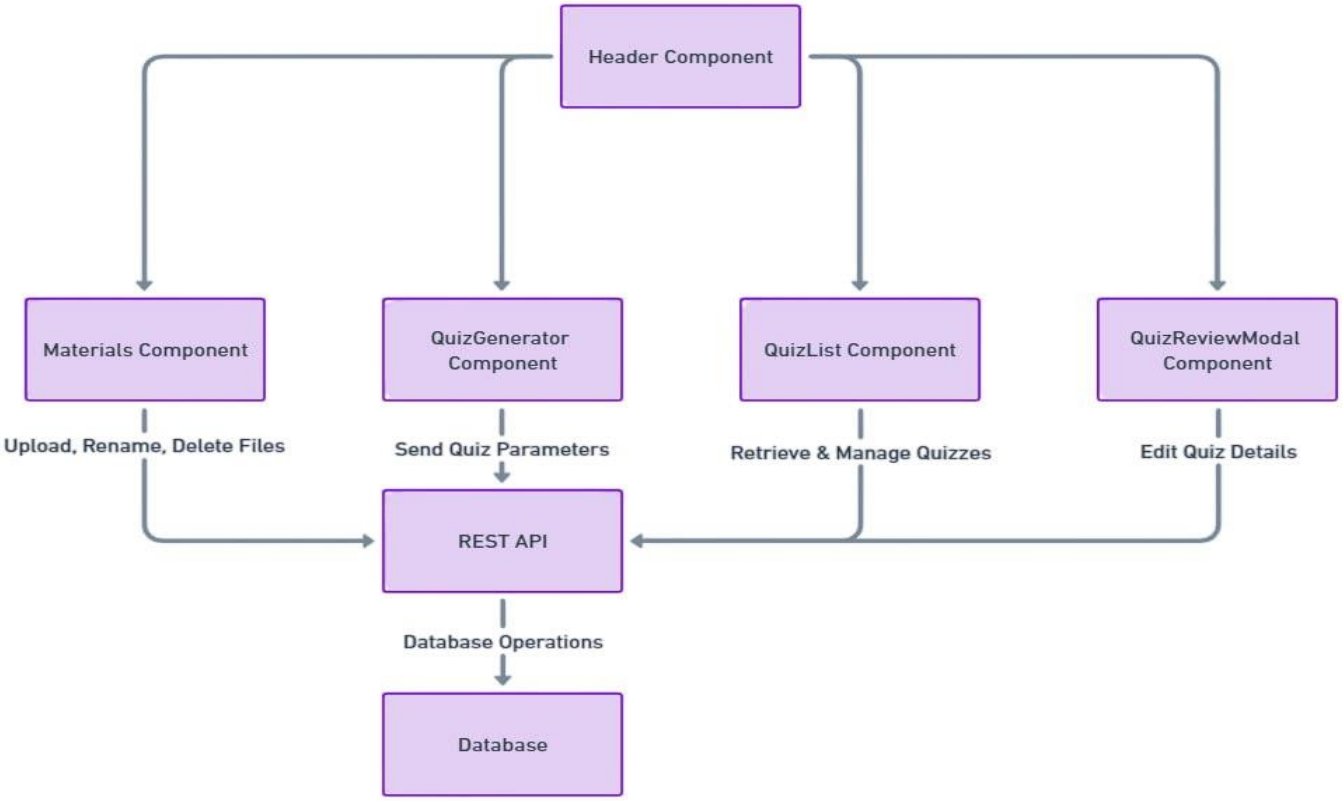


Figure 2: Frontend Architecture

### 2.2.3. Backend Architecture

The backend forms the core logic layer for the AUK Quiz Generator. Its primary objective is to process frontend requests, perform the necessary computations or lookups (often involving the database), and return well-structured responses in JSON format. By abstracting the system's business logic into this layer, the backend ensures that the frontend can remain focused on presentation without needing to handle intricate data manipulations or security measures.

A crucial part of the backend architecture is its REST API, which offers a series of endpoints for performing operations on materials and quizzes. For instance, the endpoint POST /materials accepts file uploads, validates the file format and metadata, and stores the relevant information in the database. Similarly, POST /quizzes triggers the quiz-generation workflow, which involves reading from instructor-selected materials, running Natural Language Processing (NLP) algorithms to extract potential questions, creating distractors, and composing

a final quiz object. This object includes all essential information, such as question text, answer choices, and metadata (e.g., timestamps, QTI links).

To achieve smooth and non-blocking operations, the backend relies on asynchronous programming through promises and async/await. This design ensures that the server remains responsive, even when performing resource-intensive tasks such as large file uploads or complex question-generation routines. Meanwhile, error handling is managed by returning meaningful status codes (e.g., 400 Bad Request for invalid input, 404 Not Found for missing resources) and descriptive messages that assist in both frontend debugging and deeper server-side troubleshooting.

In terms of security, the backend uses token-based authentication supported by Microsoft OAuth2, ensuring that only authorized instructors or administrators can perform sensitive operations such as uploading materials, creating tests, or modifying existing content. Additionally, if necessary, the system can be extended to implement role-based access[15] to differentiate between different levels of users, such as teaching assistants, course coordinators, or system administrators, and grant them permissions that match their specific responsibilities.

Finally, the backend interfaces with the database to store permanent records of materials, quizzes, and user activities. Through well-designed data-access methods, the backend achieves consistent and maintainable interactions with the database layer, reducing the likelihood of data corruption and enhancing system scalability. If user demand grows or if new features require more extensive data processing, the server can be scaled or upgraded to utilize more powerful hosting options, thus preserving performance and availability standards.

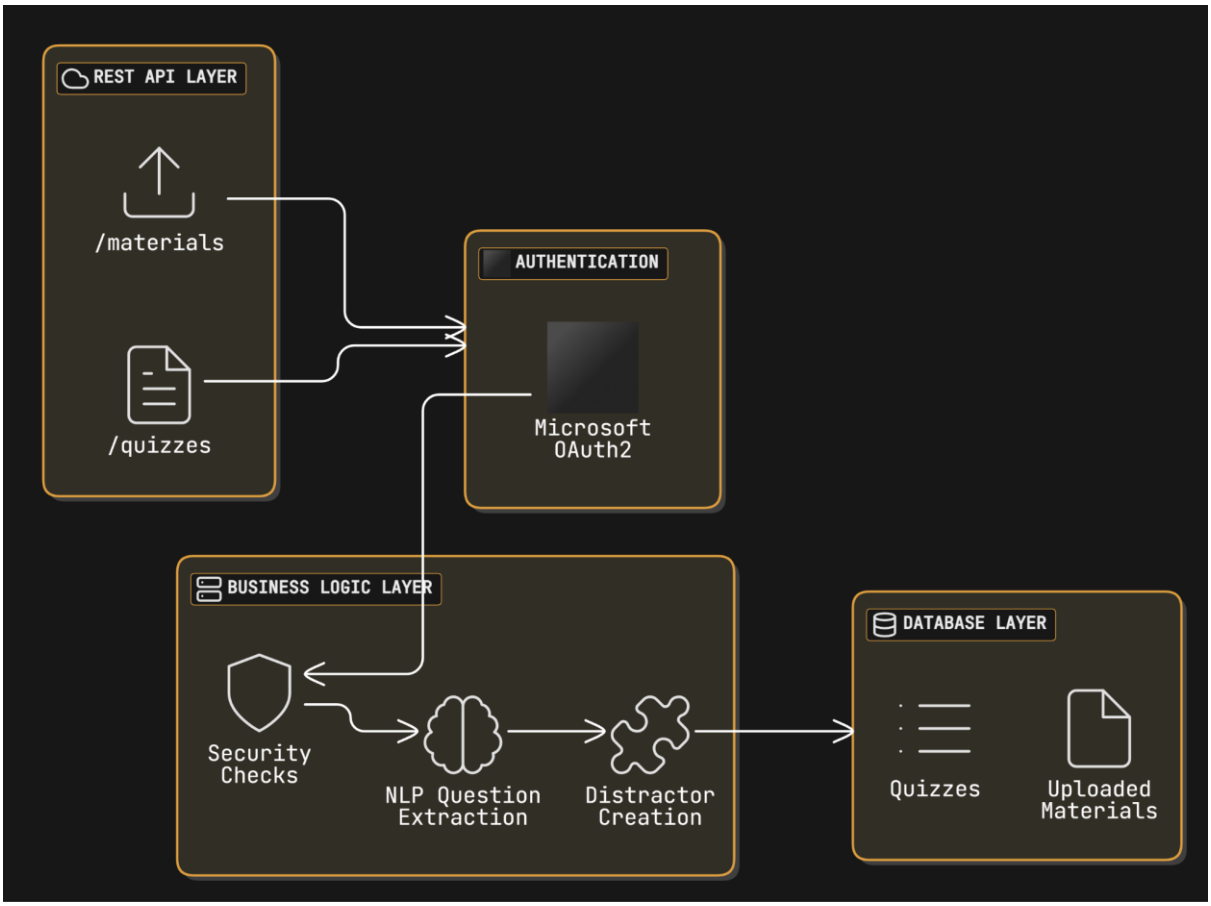

Figure 3: Backend Architecture of AUK Quiz Generator

## 2.3. Technology Stack and Justification

The technology stack for the Customizable Quiz Generation System was chosen to ensure scalability, maintainability, and ease of integration with both internal and external services. Given the core functionality of the system — automated quiz creation, content editing, and data processing — the technologies chosen are aimed at providing both robust server-side operations and a flexible customer experience.

### 2.3.1 Authentication and Security

Security is a critical pillar of the AUK Quiz Generator. To ensure that information remains secure throughout its lifecycle, the backend uses a token-based authentication mechanism using Microsoft OAuth2[16]. By using an established and widely adopted standard, the application benefits from robust session management functionality:

- Including secure token issuance,
- Automatic token expiration,
- Streamlined user consent flows.

This approach minimizes the need to store passwords directly, thereby reducing the vulnerability of the system to breaches and significantly increasing overall data security.

Under this scheme, each authorized user, primarily instructors and administrators, receives a token that verifies their credentials and the permissions associated with them[16]. The token-based design allows for stateless communication, in which the server validates requests by checking the token presented with each API call, without requiring user sessions to be maintained in memory at all times.

An important benefit of Microsoft OAuth2 is the ability to further enhance access control as the system supports role-based access control (RBAC)[15]. This functionality is particularly useful in academic contexts where users may assume different roles such as teaching assistants, course coordinators, guest lecturers, and full professors. Additionally, Microsoft OAuth2 enables multi-factor authentication (MFA) across the Microsoft ecosystem, providing another layer of verification to prevent unauthorized access even if tokens are stolen.

### 2.3.2 Integration with External APIs and Services

A central design choice in AUK Quiz Generator is to ensure seamless integration with a variety of third-party tools and services, including learning management systems (LMS) and advanced AI models. By adhering to RESTful principles and JSON-based data exchange, the system achieves broad compatibility with standard APIs and can flexibly incorporate new features over time. In an educational context, this approach ensures that AUK Quiz Generator can export and sync test data in QTI format across LMS platforms without requiring cumbersome manual processes.

The readability and ubiquity of JSON streamlines data analysis, logging, and troubleshooting, while RESTful endpoints enable scalable, stateless communication patterns.

Additionally, robust versioning and error handling methods in the API protect existing integrations from future changes. This modular and future-proof design helps ensure that as new analytics tools or authentication mechanisms emerge, AUK Quiz Generator remains flexible and adaptable.

### 2.3.3 Large Language Model: 4o-mini

An integral part of the AUK Quiz Generator functionality is the Large Language Model (LLM), which underpins automated question generation, distractor generation, and content summarization. In modern educational software, the choice of LLM can significantly impact the scalability of the system, cost structure, and overall efficiency. The 4o-mini model was adopted here after evaluating several GPT-based solutions, primarily due to its balance between expanded token capacity, cost effectiveness, and responsive performance.

**Comparative Overview of GPT-based Models**

The table below summarizes the key features and tradeoffs of the four main GPT-based models considered in the selection process. Each model differs in token capacity, cost, and performance profile[17].

| Model | Token Capacity | Cost & Performance | Ideal Use Cases |
|---|---|---|---|
| gpt-3.5-turbo-0301 | 4K tokens per request | $2.00 / 1M tokens | Moderate scenarios with smaller source material |
| gpt-3.5-turbo-16k-0613 | 16K tokens per request | $4.00 / 1M tokens | When detailed review questions from extensive lecture materials are required |
| gpt-4o | 4K tokens per request | $1.25 / 1M input tokens | Very complex sets of questions or highly specialized topics |
| gpt-4o-mini | 8K tokens per request | $0.075 / 1M input tokens | Medium file size courses; repetitive test creation tasks on a tight |

| | | | |
|---|---|---|---|
| | | | budget |

Table 1.1. Comparative Review of GPT-Based Models

Based on the characteristics listed in the table, gpt-4o-mini is a particularly suitable candidate for the AUK Quiz Generator, meeting the requirements of token capacity, cost containment, and response speed.

Key Advantages of 4o-mini:

1. As shown in the table, 4o-mini supports up to 8,000 tokens in a single request. This feature is critical for coding-focused courses or topics that require parsing detailed lecture notes, providing coherent context and richer test items than those generated by models limited to ~4K tokens.
2. Frequent test generation tasks can be prohibitively expensive if the model has high operational overhead (e.g. gpt-4o). In contrast, the balanced price of 4o-mini makes large-scale deployment possible for cost-sensitive educational institutions.
3. Compared to more resource-intensive models (such as the gpt-4o), the 4o-mini delivers faster response times. This low latency simplifies the feedback loop for instructors, who can review and improve tests in short development cycles or live training sessions.
4. Because the system relies on RESTful endpoints and JSON-based data exchange, 4o-mini integration requires minimal custom development. This design further facilitates future migrations: if an institution later adopts a different GPT-based model, such as gpt-3.5-turbo-16k-0613, the adaptation would involve adjusting the query parameters rather than rebuilding the entire architecture.

### 2.3.4 Frontend

AUK Quiz Generator uses React as its primary front-end framework. Developed by Meta (formerly Facebook), React has become one of the most widely used JavaScript libraries for building dynamic single-page applications[18]. React was chosen for several key reasons:

1. Component-Based Architecture. The main advantage of React is its component-oriented structure[19], in which individual functions are encapsulated in separate modules responsible for specific functionality (e.g. test creation forms, file upload widgets, question editing dialogs). This approach not only increases the readability of the code, but also simplifies debugging and facilitates parallel development. If a certain component (e.g. a test preview panel) requires modification or optimization, developers can focus on this isolated module without risking unintended side effects in the rest of the application.
2. Virtual DOM for Performance Optimization. React's reliance on the Virtual DOM provides a performance advantage by selectively updating only those UI elements that

have changed, rather than re-rendering the entire page. This "diffing" mechanism reduces unnecessary computation, resulting in smoother transitions and faster interactivity, even under complex conditions such as large test data sets or multiple concurrent edits.

3. Robust Ecosystem and Development Tooling. The vast React ecosystem includes a wide range of third-party libraries and official extensions, enabling a highly customizable development experience. Key tools and libraries frequently used in AUK Quiz Generator include:
    a. React Router[20]. Handles client-side routing, enabling a single-page application (SPA) structure where different pages (e.g. Materials, Quiz Generator, Quiz List) can be viewed without a full page reload;
    b. State management solutions (Redux[21] or Context API): support efficient state handling between components, ensuring consistent data flows when managing test parameters, user sessions, or complex editing workflows;
    c. Forms and validation libraries: simplify the creation and validation of user data, thereby reducing boilerplate code and minimizing potential errors in data collection.

4. Community Support and Evolutionary Stability. One of the most significant strengths of React is its active and global developer community. This robust user base translates to:
    - Frequent Updates. The core library benefits from continual enhancements and bug fixes, ensuring alignment with modern web standards;
    - Extensive Knowledge Base. Comprehensive documentation, online tutorials, and an abundance of open-source plugins make it faster for new contributors to familiarize themselves with best practices;
    - Longevity and Maintenance. React's widespread adoption by major companies implies a continued investment in its core functionality, reducing the risk of obsolescence.

    For AUK Quiz Generator, using such a well-established ecosystem ensures a sustainable foundation that reduces the long-term maintenance burden.

5. Integration with External Services and Modularity. In addition to its internal benefits, React's structure facilitates easy integration with external services[22]. By adopting modular design patterns, developers can incorporate third-party APIs (such as Microsoft OAuth2 for authentication or RESTful endpoints for test generation) without reworking the entire codebase. This extends to advanced features such as:
    a. Localization and i18n[23]: When expanding the system to multiple languages or regions.
    b. Accessibility Improvements (A11y[24]): Ensure WCAG compliance with specialized React components.

As AUK Quiz Generator evolves, the flexible architecture of React plugins supports adding or removing these services with minimal disruption to existing modules. This design is critical when integrating new features.

## 2.4. System Context Diagram

The Figure 4 below illustrates the AUK Quiz Generator system and its interactions with external entities.

At the center is the primary software component - AUK Quiz Generator - while the surrounding elements represent the actors and services with which the system exchanges data.

Instructors. Teachers are the primary users of the AUK Quiz Generator. They initiate critical workflows such as uploading teaching materials, setting test parameters (e.g. number of questions, difficulty level), and editing automatically generated questions to ensure pedagogical accuracy.

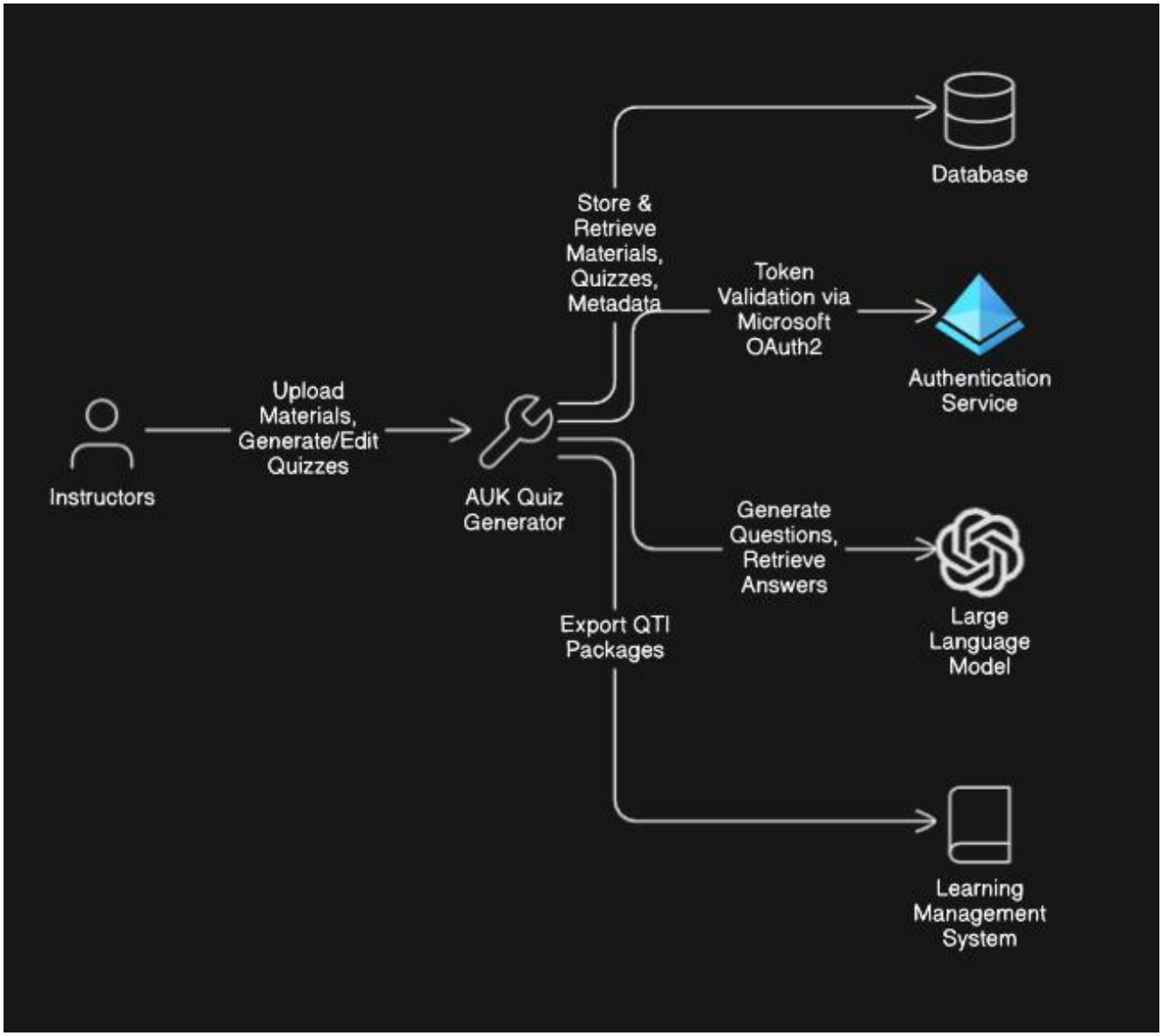


Figure 4: System Context Diagram of AUK Quiz Generator

Key interactions:

- Upload materials. Teachers can provide detailed notes or summary documents in TXT format. The system then analyzes these documents, extracting information to generate questions;

- Create and edit tests. After specifying test settings such as number of questions or answer options, teachers can retrieve an initial test draft from the system and refine it by adding, removing, or changing questions and distractors.

Database. All uploaded files, quiz metadata, and other important data are stored in the database. The AUK Quiz Generator communicates with the database to save new materials and retrieve the information required to generate questions.

Authentication Service (Microsoft OAuth2). This service handles user authentication and authorization. The AUK Quiz Generator sends token verification requests to the service to ensure that only authorized users can access and operate the system.

Large Language Model. The AUK Quiz Generator delegates the creation of questions, answers, and distractors to an external LLM service. Upon receiving text segments extracted from uploaded materials, the LLM returns a list of potential quiz questions and multiple-choice options, which instructors may then refine.

Learning Management System (LMS). Once a quiz is generated and approved, instructors can export it in QTI format. This package is then imported into an LMS (e.g., CANVAS), allowing students to take the quizzes and enabling automated result collection.
The labeled arrows on the diagram highlight the main data flows and actions:

- "Upload Materials, Generate/Edit Quizzes" shows the communication between instructors and the system.
- "Store & Retrieve Materials, Quizzes, Metadata" indicates how the system interacts with the database.
- "Token Validation via Microsoft OAuth2" underscores user authorization steps.
- "Generate Questions, Retrieve Answers" reveals how the system engages the language model for automated quiz generation.
- "Export QTI Packages" represents the export of quiz data in an LMS-compatible format.

Overall, this context diagram demonstrates who interacts with the AUK Quiz Generator, how those interactions occur, and for what purpose. It also highlights the key external services and actors required to deliver a fully functional system in an educational environment, providing a broad view of the project's scope and structure.

## 2.5. Component Diagram

Figure 5 shows the modular architecture of the AUK Quiz Generator, highlighting how the application interface interacts with the service layer (AppServices), which in turn interacts with the REST API and the underlying database.

The figure shows a schematic structure of the AUK Quiz Generator application, displaying the main components of the frontend, the service layer, as well as interaction with the REST API and the database. Each of the rectangles symbolizes a certain group of functionality or service, and the directional arrows illustrate method calls or data transfer between them.

The Header component serves as the main navigation bar at the top of the application. It provides instructors with quick access to various functions, such as downloading materials, creating new tests, or viewing existing ones, by directing them to the appropriate modules of the user interface.

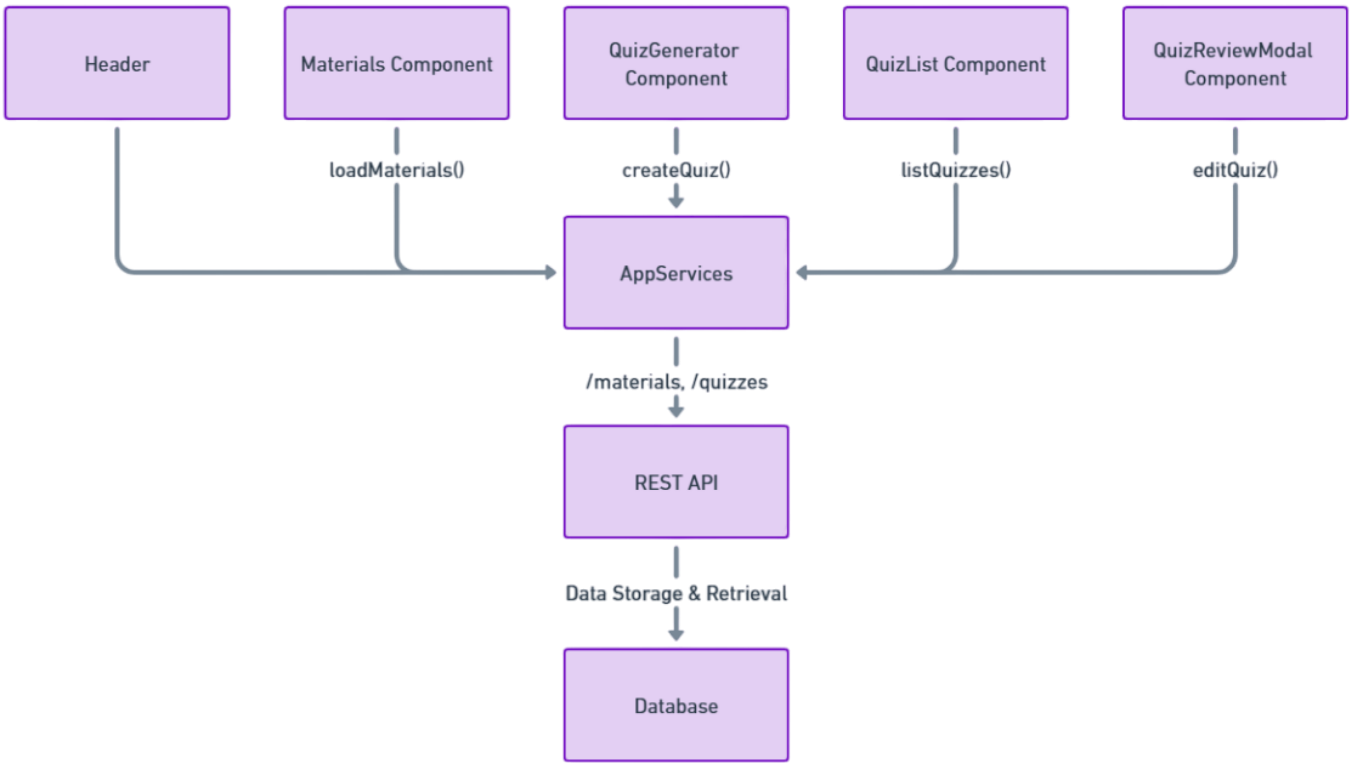


Figure 5: Component Diagram of AUK Quiz Generator

Functions:

1. Display Navigation Links. Renders top-level menu items or icons (e.g., “Materials,” “Quiz List”).
2. Route Management. Uses client-side routing (e.g., React Router) to switch between different pages.

The Materials component manages the uploading and presentation of educational resources. It acts as a central repository for course-related files (e.g. lecture text, notes), allowing instructors to organize and track their materials within the app.

Functions:

1. File Upload. Allows instructors to select TXT files from their local system and upload them to the AUK Quiz Generator.
2. Listing Uploaded Files. Displays a list of all previously uploaded materials.
3. Load and Refresh Materials. Calls a service-layer method to retrieve updated file data from the backend, ensuring instructors see the latest file state.
4. Basic File Operations. Include renaming, deleting, or downloading existing files.

The QuizGenerator component is the main interface for creating new quizzes. It collects instructor data, such as the number of questions or answer options, and converts it into a valid quiz generation request, which is then processed by the system.

Functions:

1. Parameter Input. Offers fields for specifying parameters for the quiz (e.g. quiz name, number of questions, number of distractors, selected materials).
2. Create Quiz Request. Calls a method on the service layer to initiate quiz generation via the REST API, which the LLM can use to formulate questions.
3. Error Handling. Displays validation alerts or warnings if parameters are invalid or if the backend returns an error response.

The QuizList component provides instructors with an overview of all previously created quizzes. It allows them to view, select, or manage existing assessments without having to recreate them from scratch.

Functions:

1. Retrieve Quiz Data. Calls a service layer method to retrieve the current list of quizzes stored in the system.
2. Display Quiz Summaries.
3. Selection and Navigation. Allows instructors to select a quiz for further actions such as editing or previewing by redirecting them to the appropriate UI components.
4. Maintenance Actions.May offer functions to rename, archive, or delete quizzes if available through the service layer.

QuizReviewModal is a pop-up interface that allows instructors to refine the content of a quiz. It is typically displayed when an instructor wants to edit the questions or answers of a newly created or previously created quiz.

Features:

1. Edit Question Text. Allows you to directly edit the wording of a question, giving instructors fine-grained control over wording and difficulty.
2. Configure Correct and Incorrect Answers.Allows instructors to mark correct answers, remove distractors, or add new distractors to improve question validity.
3. Add/Remove Questions. Offers the ability to introduce entirely new questions or remove outdated ones, ensuring that the quiz remains aligned with the learning objectives.

AppServices acts as a business logic layer between the frontend React components and the REST API. It unifies data handling, error handling, and security checks, allowing UI components to stay focused on the presentation.

The REST API is an internal gateway to manage test creation, content uploads, user requests, and any interaction with external services such as the Large Language Model (LLM). It processes incoming data and interacts with the database to perform read/write operations.

The database is a persistent storage layer that contains test metadata (titles, timestamps), question and answer data, and links to uploaded course materials. Its design ensures data consistency and performance by supporting simultaneous reads and writes from multiple instructors.

This Component Diagram clearly demonstrates how roles and functionality are separated in the application and how each module interacts with adjacent ones. This approach to architecture simplifies code maintenance and adding new functions, since most of the logic (AppServices and REST API) is separated from the interface components, and the frontend structure itself is clearly divided into functional blocks.

## 2.6. Customization Features of Quiz Generation

In the process of automated test generation, it is essential to provide users, especially teachers, with flexible customization mechanisms that allow them to tailor the resulting questions and answer options to the specific learning objectives and level of the audience. AUK Quiz Generator has a number of such features that simplify the process of personalization and ensure high quality of test questions.

### 2.6.1. Selecting sources and creating content

One of the key options is selecting source materials TXT from which key points will be extracted to create questions. The teacher can mark several files corresponding to different lectures or sections of the course, thereby creating a more extensive subject base. As a result, the system creates a task based on the most relevant fragments of the text, which ensures thematic compliance of the test and increases the value of knowledge testing.

### 2.6.2. Flexible parameter settings

When starting generation, the teacher defines a number of parameters:

- Number of questions
  Allows you to adjust the length of the test depending on the depth of the material being studied and the time available for completion.
- Number of answer options
  Important for choosing a multiple choice format (for example, from three to five options). The more options, the more difficult the task, but this also contributes to a deeper testing effect.

### 2.6.3. Manual adjustment and expansion

Although AUK Quiz Generator automatically generates questions and answer options, the teacher reserves the right to:

- Edit the text of the question or answers, eliminating possible inaccuracies or stylistic imperfections.
- Add your own questions based on empirical observations, complex practical examples or historical facts not reflected in the uploaded materials.
- Delete duplicate or incorrect items, as well as questions that, in the teacher's opinion, do not correspond to the educational objectives.

# 3. Deployment and User Interface of the AUK Quiz Generation System

## 3.1. System Interface Overview

The main dashboard of the AUK Quiz Generator acts as the central hub where users can view, manage, and organize their quizzes. Key features include:

- Review and Edit: Opens the test in a modal window for further customization.
- Rename: Allows you to rename the test for better organization.
- Delete: Removes the test from the system.
- Download: Exports the test in QTI format for integration with external LMS platforms.
- Status indicators: Green markers indicate completed tests, and yellow marks indicate tests that have just been created or are in the process of being created.

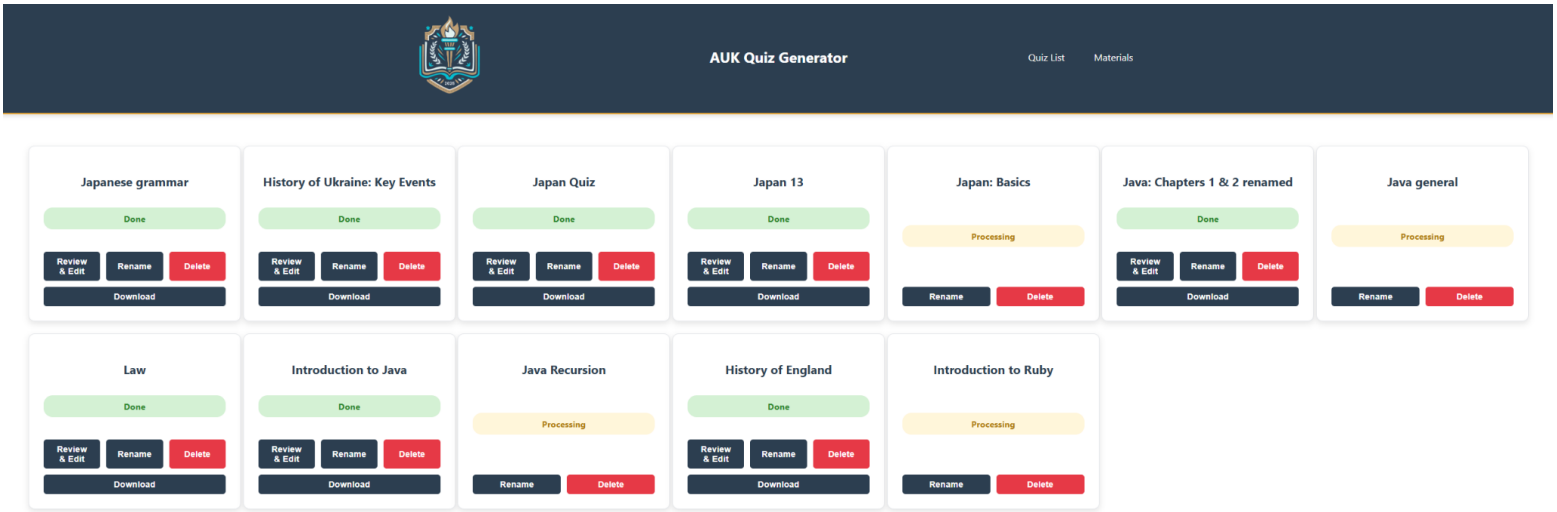


Figure 6: AUK Quiz Generator dashboard

The Materials section is shown in Figure 7. It provides users with tools to upload, organize, and manage source files used to generate tests. Key features include:

- Rename files
- Delete material
- Upload material
- Enter a test name.
- Select materials as source text.
- Specify the number of questions and answer options.
- Create a test by sending the configuration to the backend.

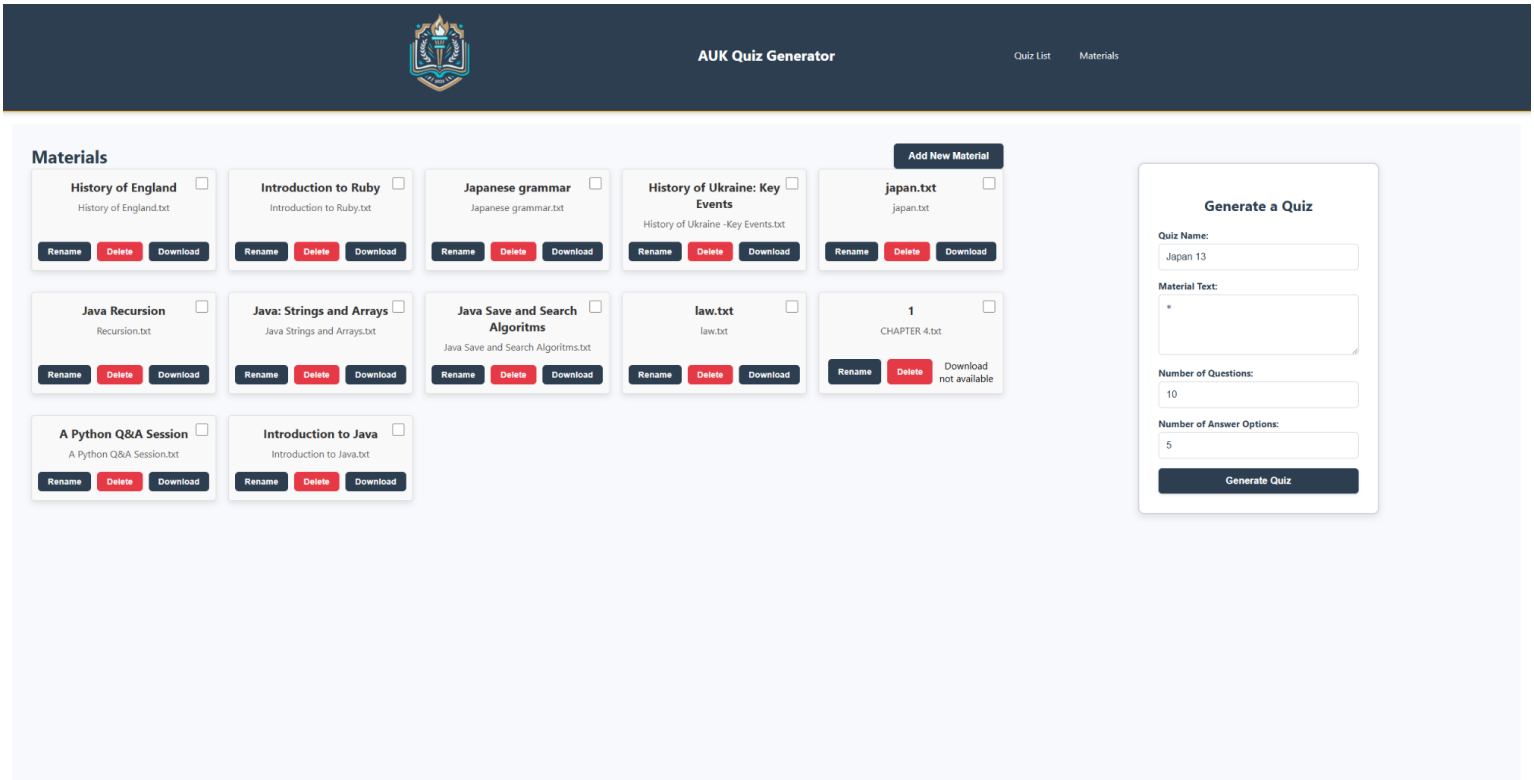


Figure 7: AUK Quiz Generator materials section

The Quiz Review Modal is shown in Figure 8. The Review & Edit feature opens a modal window that provides a focused interface for editing test content. Key features include:

- Edit question text.
- Add or remove answer options.
- Set or clear correct answers.
- Add a new question
- Delete a question.

This interactive editing tool ensures that tests meet specific teacher requirements and are aligned with educational goals.

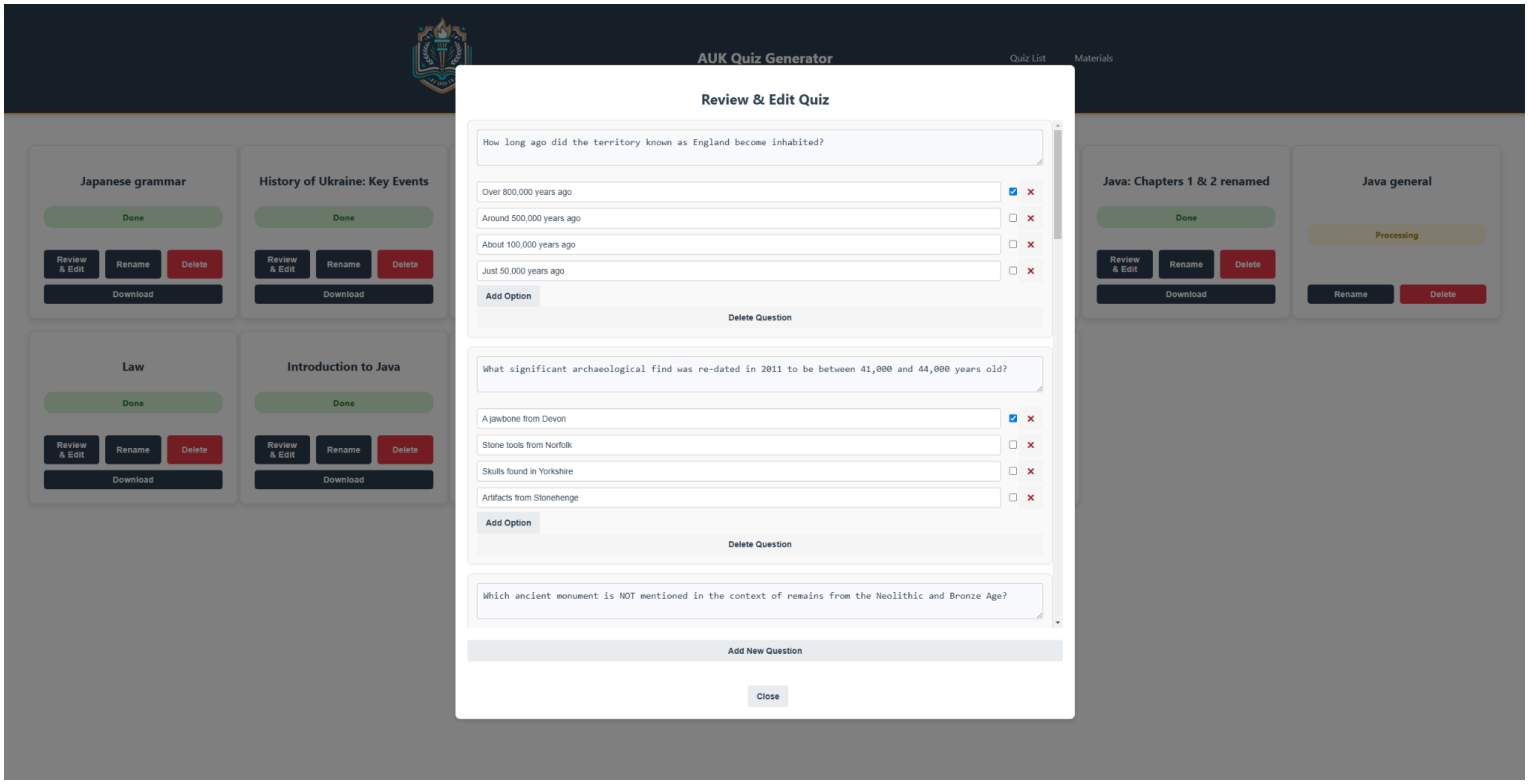


Figure 8: AUK Quiz Generator Quiz Review Modal

## 3.2. Local Deployment Guide for the AUK Quiz Generator

To deploy the AUK Quiz Generator locally, ensure the following tools are installed:

1. Node.js: Download and install the LTS version from the official website[25].
2. Git: Confirm Git is installed on your system. If not, download it from the official Git website[26].
3. Visual Studio Code (optional but recommended): Install this IDE from the official website[27].

### Cloning the Repository

1. Open a terminal or use the integrated terminal in Visual Studio Code.
2. Execute the following command to clone the project repository:

```
git clone https://github.com/MadHatter013/AUK-Quiz-Generator.git
```

3. Navigate to the project directory:

```
cd AUK-Quiz-Generator
```

4. Run the following command in the terminal within the project directory to install all required dependencies:

```
npm install
```

5. This command will download and set up all necessary libraries and modules defined in the package.json file.

To launch the development environment, execute:

```
npm start
```

The development server will start, and the terminal will display the URL of the running application, typically http://localhost:3000. Open this URL in a web browser to access the AUK Quiz Generator.

# 6. Conclusions

This thesis is a significant contribution to the development of modern educational technologies, demonstrating the potential of using artificial intelligence to automate routine tasks and personalize the educational process. AUK Quiz Generator was developed as a quiz generation tool that can be integrated into modern curricula.

It is important to note that the system structure, built on a three-tier architecture (frontend, backend, data layer), meets modern requirements for scalability and modularity. Using React.js as a frontend allows you to create an adaptive and flexible user interface, and the REST API provides effective integration with external services and platforms.

The key feature of AUK Quiz Generator is the use of natural language processing (NLP) methods for automatic generation of questions and answer options. The system offers teachers tools for customizing test generation parameters and their subsequent editing. This approach allows you to take into account specific educational tasks and requirements.

Integration with learning management systems (LMS) via quizzes expands the functionality of the system, making it compatible with existing platforms. This opens up opportunities for its implementation in various educational institutions.

Using such a system will reduce the workload of teachers by automating the process of creating quizzes, and will allow them to focus on more important aspects of the educational process, such as working with students and developing teaching methods. In addition, the ability to personalize the approach to learning provided by AUK Quiz Generator can significantly improve the quality of the educational process.

# References:


1. Hasan, A. S. M. M. E., Shahnoor, M. A., Tasneem, K. B., & Sumaiya, S. (2024). Automatic question & answer generation using generative Large Language Model (LLM). BRAC University Institutional Repository. URL: https://dspace.bracu.ac.bd:8443/xmlui/handle/10361/22833

2. Sreekanth, D. (2023). AI-based quiz generation: The role of LLMs in digital education. Kennesaw State University Digital Commons. URL:https://digitalcommons.kennesaw.edu/cday/2023fall/Undergraduate_Research/7/

3. Korzh, M., & Tytenko, S. (2024). AI in the Construction of Educational Tools and Quizzes for Different Programming Languages. *Modern Engineering and Innovative Technologies*, Issue No. 35, Part 1, October 2024. DOI: 10.30890/2567-5273.2024-35-00-042. URL: https://www.moderntechno.de/index.php/meit/issue/view/meit35-01/meit35-01

4. Meißner, Niklas; Speth, Sandro; Kieslinger, Julian; Becker, Steffen (2024): EvalQuiz – LLM-based Automated Generation of Self-Assessment Quizzes in Software Engineering Education. Software Engineering im Unterricht der Hochschulen 2024. DOI: 10.18420/seuh2024_04. Bonn: Gesellschaft für Informatik e.V.. PISSN: 1617-5468. ISBN: 978-3-88579-255-0. pp. 53-64. Linz, Österreich. 29. Februar - 1.März 2024. URL: https://dl.gi.de/items/1561864d-a5b9-42d2-821d-41baf80eb630/

5. Lopez, C., Morrison, M., & Deacon, M. (2024). Language models for generating programming questions with varying difficulty levels. European Public & Social Innovation Review, 9, 1–19. https://doi.org/10.31637/epsir-2024-760

6. Bitew, S. K., Deleu, J., Develder, C., & Demeester, T. (2023). Distractor generation for multiple-choice questions with predictive prompting and large language models. *arXiv*. URL: https://arxiv.org/abs/2307.16338

7. Ersoy, B. I. (2024). ePub-to-Quiz Conversion with Large Language Models. URL: https://www.cl.uzh.ch/dam/jcr:5c3787db-bc0f-42a6-82ab-92fbc435585e/masterarbeit.final.pdf

8. 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. Proceedings of the AAAI Conference on Artificial Intelligence, 38(21), 23084-23091. https://doi.org/10.1609/aaai.v38i21.30353

9. Pagano, T. P., Loureiro, R. B., Lisboa, F. V. N., Peixoto, R. M., Guimarães, G. A. S., Cruz, G. O. R., Araujo, M. M., Santos, L. L., Cruz, M. A. S., Oliveira, E. L. S., Winkler, I., & Nascimento, E. G. S. (2023). Bias and Unfairness in Machine Learning Models: A Systematic Review on Datasets, Tools, Fairness Metrics, and Identification and Mitigation Methods. URL: https://www.mdpi.com/2504-2289/7/1/15

10. Góral, G., Wiśnios, E., Sankowski, P., & Budzianowski, P. (2024).When All Options Are Wrong: Evaluating Large Language Model Robustness with Incorrect Multiple-Choice Options. *arXiv*. URL: https://arxiv.org/abs/2409.00113

11. Frankford, E., Höhn, I., Sauerwein, C., & Breu, R. (2024). A Survey Study on the State of the Art of Programming Exercise Generation using Large Language Models. *arXiv*. URL: https://arxiv.org/abs/2405.20183

12. Diehl, P., Nader, N., Brandt, S., & Kaiser, H. (2024). Evaluating AI-generated code for C++, Fortran, Go, Java, Julia, Matlab, Python, R, and Rust. arXiv. URL:https://arxiv.org/abs/2405.13101

13. Monetti, F. M., Lundström, A., & Maffei, A. (2024). Barriers to Adopting Design for Assembly in Modular Product Architecture: Development of a Conceptual Model Through Content Analysis. *arXiv*. URL: https://arxiv.org/abs/2411.17768

14. Mari, E., & Eila, J. (2003). The impact of maintainability on component-based software systems. *2003 Proceedings 29th Euromicro Conference*. URL: https://ieeexplore.ieee.org/abstract/document/1231563

15. Microsoft Learn. (2024). Implement role-based access control. Article. URL: https://learn.microsoft.com/en-us/entra/identity-platform/howto-implement-rbac-for-apps

16. Microsoft Security. (n.d.). What is OAuth? *Microsoft Entra ID*. URL: https://www.microsoft.com/en-us/security/business/security-101/what-is-oauth#:~:text=Microsoft%20Entra%20ID%20(formerly%20Azure,modern%20identity%20capabilities%20into%20apps.

17. OpenAI. (n.d.). Pricing. *OpenAI API*. URL: https://openai.com/api/pricing/

18. State of React. (2023). Usage Statistics for React. *State of React 2023*. URL: https://2023.stateofreact.com/en-US/usage/?utm_source=chatgpt.com

19. Serverspace. (n.d.). Review of the React JS Framework: Advantages, Disadvantages, and Use Cases. *Serverspace Blog*. URL: https://serverspace.io/about/blog/review-of-the-react-js-framework-advantages-disadvantages-and-use-cases/?utm_source=chatgpt.com

20. React Router. (n.d.). React Router Documentation. *React Router*. URL: https://reactrouter.com/home

21. Redux. (n.d.). Redux Documentation. *Redux*. URL: https://redux.js.org/

22. Ani, A. (n.d.). Mastering API Requests in ReactJS: Examples, Explanations, and Use Cases. *Dev.to*. URL: https://dev.to/anii1429/mastering-api-requests-in-reactjs-examples-explanations-and-use-cases-4hkg?utm_source=chatgpt.com

23. i18next. (n.d.). Internationalization Framework for JavaScript. *i18next Documentation*. URL: https://www.i18next.com/

24. The A11Y Project. (n.d.). Resources to Make Accessibility Easier. *The A11Y Project*. URL: https://www.a11yproject.com/

25. Node.js. (n.d.). Node.js Official Website. *Node.js*. URL: https://nodejs.org/en/

26. Git. (n.d.). Git Official Website. *Git*. URL: https://git-scm.com/

27. Visual Studio Code. (n.d.). Visual Studio Code Official Website. *Microsoft*. URL: https://code.visualstudio.com/