Eötvös Loránd University

Faculty of Informatics

Dept. of Programming Languages and Compilers

Developing a Learning Platform with Personalized Quizzes and Study Tools

Supervisor:
Morse Gregory Reynolds
Lecturer

Author:
Nahid Muradli
Computer Science BSc

Budapest, 2025

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Topic Registration Form

Student's Data:

Student's Name: Muradli Nahid

Student's Neptun code: HJPIVF

Educational Information:

Training programme: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Morse Gregory Reynolds*

Supervisor's Home Institution: ***Department of Programming Languages and Compilers***

Address of Supervisor's Home Institution: ***1117, Budapest, Pázmány Péter sétány 1/C.***

Supervisor's Position and Degree: *Lecturer*

Thesis Title: Developing a Learning Platform with Personalized Quizzes and Study Tools

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

For my final thesis, I propose the development of a sophisticated web-based learning platform that combines AI-driven quiz generation with personalized study tools. The platform will allow users to input a topic, and AI will dynamically generate quizzes with relevant questions and answers. To achieve this, I will utilize the OpenAI GPT API for quiz content generation, ensuring I account for associated costs in the project budget.

The platform will store quizzes to enhance user experience, allowing quiz retrieval and review. This includes saving user-generated quizzes, results, and tracking progress. User data, including quiz performance and earned badges, will be securely stored in a database, enabling a personalized learning experience.

Sophisticated prompt engineering will be integral to the project. I will craft prompts that guide the AI in generating diverse quiz formats, such as multiple-choice and short-answer questions. Incorporating topic-specific keywords and contextual information will ensure that the AI produces relevant and engaging content.

The platform will include a multi-tiered user system where progress is tracked through a level-based system. Users will earn badges, titles, and achievements as they complete quizzes and challenges, enhancing interactivity and motivation. Additional features will include automated flashcard generation based on quiz data, daily and weekly challenges, access to AI-generated articles, and suggestions of trending topics to promote continuous learning.

The complexity of this project lies in integrating AI, user authentication, dynamic content generation, progress tracking, and various interactive features. Developing these components cohesively requires careful planning and the ability to handle complex user interactions and data flow, ensuring a smooth and functional user experience.

This project will demonstrate my ability to develop a complete, user-focused solution, managing both backend logic and user interface design while implementing AI-driven features. The platform is designed to be expandable, allowing for future feature additions and improvements.

Budapest, 2024. 10. 08.

Acknowledgements

First and foremost, I would like to express my heartfelt thanks to my family. Your constant support, encouragement, and understanding have been the foundation for getting through these challenges and staying focused on this journey. Your belief in me during the toughest moments means more than anything in the world to me.

To my friends, thank you for being there through all the ups and downs. Whether it was providing a feedback, or simply offering a break when I needed one. You made this experience less stressful and more enjoyable..

I would also like to sincerely thank my supervisor, Morse Gregory Reynolds, for all the support, constructive feedback, and the insights he has given. His support has helped me shape this thesis and pushed me to new ideas.

What a challenging but rewarding project! I am truly thankful to everyone.

Contents

Chapter 1

Introduction

QuizGG (Quiz "Good Game") is a platform that uses artificial intelligence (AI) and gamification to revolutionize conventional learning. It addresses some of the key flaws of conventional learning platforms, which provide no personalization. The platform demonstrates how modern architectures can maximize online learning experiences while sustaining user engagement.

The platform uses AI for generating (as well as grading) customized learning material (quizzes, articles) for diverse levels of knowledge, as well as preferences. Its motivational design involves progress tracking and rewards with interactive challenges to foster consistent participation.

It is built with a solid web technology stack. The application has a responsive interface, a strong backend, and a scalable cloud-based relational database for secure data management and efficient data retrieval.

A functional tool as well as a research platform, QuizGG showcases how modern technologies combined with systematic design can aid education. Its flexible architecture facilitates additional pedagogical as well as technical advances.

Chapter 2

User Documentation

This chapter is dedicated for QuizGG users: an AI-powered platform that offers custom quizzes, study tools, and gamified learning. This is where you can learn about the system requirements, installation procedure, initial configuration, and instructions for using the platform's main features. For students, educators, and all learners in general. Get started with QuizGG!

2.1 System Requirements

To use QuizGG, your system should meet the below minimum specifications:

Operating System: Microsoft Windows 10/11 [1], Apple macOS 12+ [2], or Linux [3] (Ubuntu 20.04+ [4] recommended).

Hardware: 4 Gigabytes (GB) Random Access Memory (RAM), 2 Gigabytes free disk space.

Software:

- Node.js (v18 or later) [5]
- A modern web browser (e.g., Google Chrome [6]) with JavaScript enabled

Internet Connection: Necessary to access the app, host a cloud-based database, generate AI quizzes, and external authentication (Google OAuth).

These are the requirements to make sure QuizGG runs smoothly on your device. You will need to create a Supabase [7] account to configure the database, as described in Section 2.2.3.

2.2 Installation and Setup

You can follow this section to set up QuizGG locally. Watch these steps carefully for a successful installation. Clone the repository, install dependencies, configure environment variables, set up a database and start the application.

2.2.1 Obtaining the Project

QuizGG can be acquired in two ways: cloning from GitHub [8] or downloading a ZIP (an archive file format for lossless data compression) [9] file from another source.

1. **Cloning from GitHub:**
   (a) Launch a terminal or command prompt.
   (b) Execute this command to clone the repository [10]:

```
git clone https://github.com/nahidmrdl/quizGG-thesis.git
```

   (c) Move to the project directory:

```
cd quizgg-thesis-main
```

2. **Using a ZIP File:**
   (a) Download the ZIP file from the provided source.
   (b) Extract the ZIP contents to a preferred location on your machine.
   (c) Open a terminal or command prompt and navigate to the extracted project directory:

```
cd path/to/extracted/quizgg-thesis-main
```

2.2.2 Installing Dependencies

QuizGG uses Node.js and npm (node package manager) to manage its dependencies. Install the required packages:

1. Ensure Node.js (v18 or later) is installed. Verify by running:

```
node --version
```

   If Node.js is not installed, download and install it from `https://nodejs.org/`

2. Install the project dependencies by running [11]:

```
npm install
```

This command installs all dependencies listed in the `package.json` file, including core libraries such as Next.js [12], Prisma [13], and React. [14] Other packages required for the functionality of QuizGG are also installed through this command.

2.2.3 Environment Variables Setup

To connect to a Supabase database, enable authentication, and use the DeepSeek API (Application Programming Interface) [15] for quiz generation, configure these environment variables:

1. **Create a Supabase Project**:

   (a) Sign up at `https://supabase.com/` and create a new project (e.g., "quizgg-project") with a secure database password.

   (b) After creation, go to `Settings > Database > Connection String` and copy the PostgreSQL URI (Uniform Resource Identifier):

```
postgresql://postgres:[YOUR-PASSWORD]@[PROJECT-REF].pooler
    .supabase.com:5432/postgres
```

   Replace `[YOUR-PASSWORD]` and `[PROJECT-REF]` with your credentials. See Supabase documentation [16] for details.

2. **Set Up .env (Environment file—configuration file File**:

   (a) Create a `.env` file in the project root with:

```
# Prisma database connection
DATABASE_URL="your_postgresql_supabase_db_url_here"
DIRECT_URL="your_postgresql_supabase_db_url_here"

# NextAuth configuration
NEXTAUTH_SECRET="your_nextauth_secret_here"
NEXTAUTH_URL="http://localhost:3000"

# Google OAuth credentials
```

```
GOOGLE_CLIENT_ID="your_google_client_id_here"
GOOGLE_CLIENT_SECRET="your_google_client_secret_here"

# DeepSeek API
DEEPSEEK_API_KEY="your_deepseek_api_key"
DEEPSEEK_API_BASE="https://api.deepseek.com/v1"

# Application URLs
NEXT_PUBLIC_API_URL="http://localhost:3000"
NEXT_PUBLIC_BASE_URL="http://localhost:3000"
```

(b) Replace placeholders:

- `DATABASE_URL` and `DIRECT_URL`: Use the Supabase PostgreSQL URI from Step 1.
- `NEXTAUTH_SECRET`: Generate a random string for NextAuth [17] with:

```
openssl rand -base64 32
```

- `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET`: Set up OAuth (Open Authorization) 2.0 credentials in Google Cloud Console (`https://console.cloud.google.com/`) [18, 19]. Set redirect URI to `http://localhost:3000/api/auth/callback/google`.
- `DEEPSEEK_API_KEY`: Obtain from `https://platform.deepseek.com/` account settings [20].

(c) Save the `.env` file.

3. **Initialize Database Schema**: Run:

```
npx prisma db push
```

This syncs the database with the Prisma schema [13] for QuizGG. Refer to Prisma documentation for advanced setup [21].

2.2.4 Running the Application

After completing the setup, you can start the QuizGG application and access it in your browser:

1. In the project root directory, start the development server by running [22]:

```
npm run dev
```

This command launches the Next.js development server

2. Open your web browser and navigate to:

```
http://localhost:3000
```

3. To secure QuizGG in production, enable HTTPS (HyperText Transfer Protocol Secure) by setting up an SSL (Secure Sockets Layer) certificate. [23]

4. You should see the QuizGG welcome page, as shown in Figure 2.1

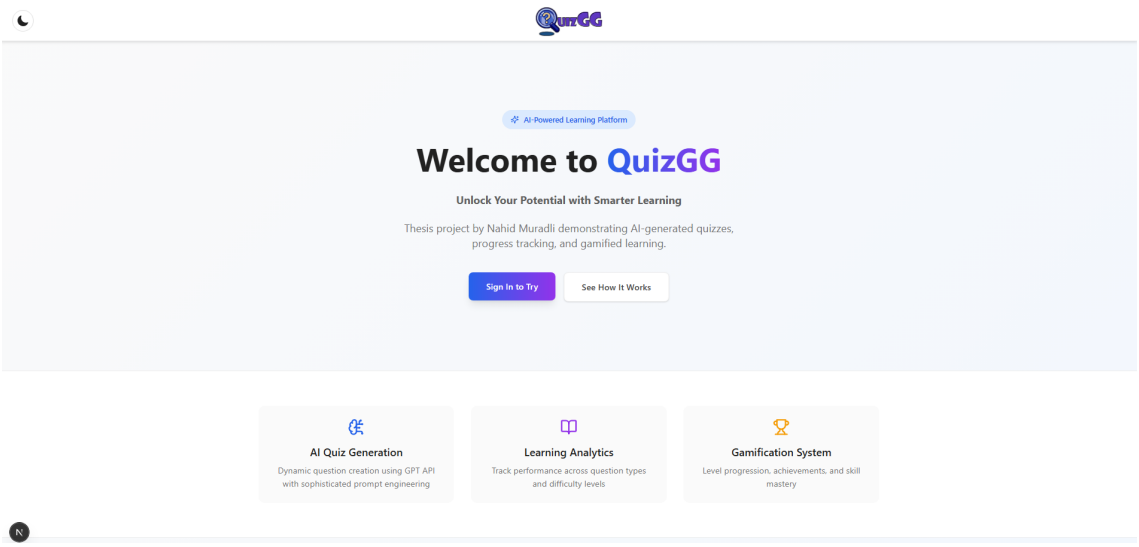


Figure 2.1: Welcome Page

2.3 Getting Started

This section explains how to create an account and helps you get familiar with the QuizGG dashboard - your central point of access to all features.

2.3.1 Signing Up and Logging In

To access QuizGG's features, you need to create an account or log in:

1. On the welcome page (Figure 2.1), click the **Sign In to Try** button
2. You will be redirected to the sign-in page, as shown in Figure 2.2
3. Choose one of the following options to log in:
   - **Google Sign-In**: Click the Google button to log in using your Google account
   - **Credentials**: Click **Sign Up** to create a new account. Enter your name, email, username, and password, then click **Sign Up**. If you already have an account, enter your username or email and password, then click **Log In**

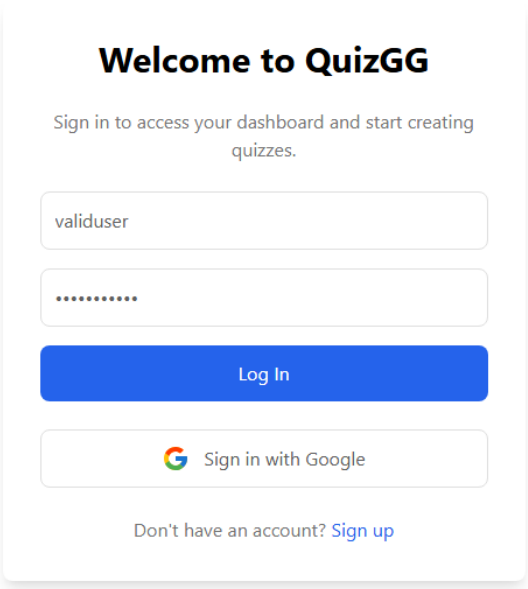


Figure 2.2: QuizGG Sign-In Page

4. After successful authentication, you will be redirected to the dashboard (see Section 2.3.2)

2.3.2 Navigating the Dashboard

The dashboard serves as the central hub for accessing all of QuizGG's features, provides users with a comprehensive overview of their learning progress and quick access to key functionalities, as shown in Figure 2.3:

Level Progress Bar: Visually tracks your current level, experience points (XP), and progression toward the next level.

Leaderboard: Shows top performers and your current ranking based on XP, coins, and correct answers.

Tools Grid: The primary navigation panel that provides immediate access to all core features:

- **Quiz Generator**: Create customized quizzes on demand
- **Quiz History**: Review previous quiz attempts & results
- **Stats & Insights**: Analyze learning patterns and performance metrics
- **Achievements**: Track earned badges & milestones
- **Study Tools**: Access flashcards & generated articles
- **Store**: Redeem earned coins for various items
- **Daily Challenge**: Offers a daily challenge to keep you engaged
- **Weekly Challenge**: Offers a weekly quest

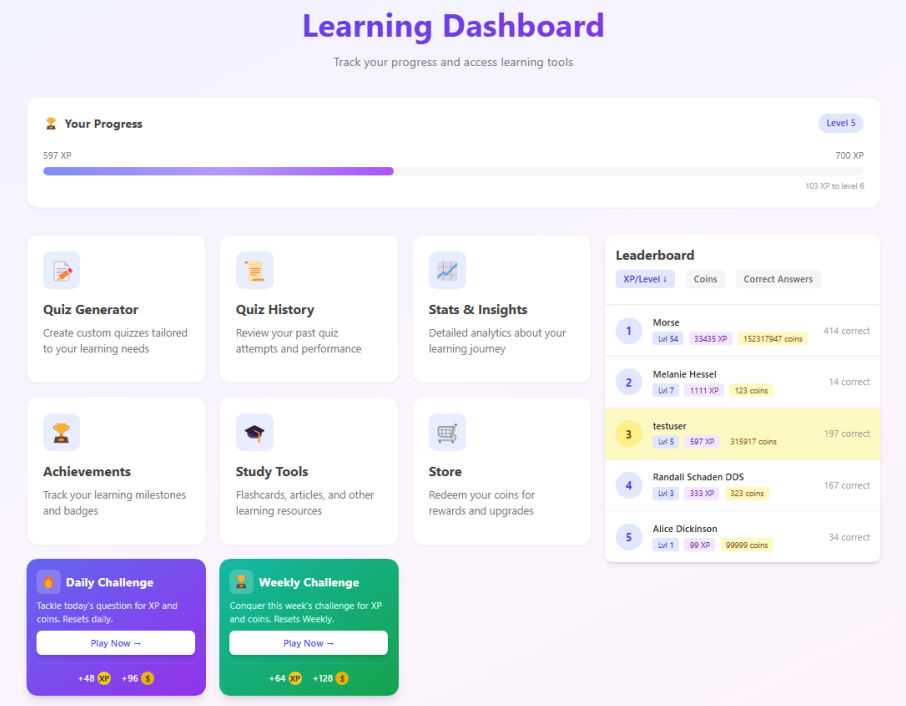


Figure 2.3: Dashboard Interface

2.4 Using QuizGG Features

QuizGG makes studying actually enjoyable by offering a variety of options. The platform keeps things fresh with daily challenges while letting you create custom quizzes on any topic, generate articles based on your interests, and even make flashcard sets. You will earn XP points & coins as you learn, compete with friends on the leaderboard, and watch your skills grow. You can see more about how to use it in this section.

2.4.1 Generating and Taking Quizzes

The quiz generation feature lets you build custom quizzes for any learning objective, whether you're studying for an exam or just learning about a new topic. Powered by AI, QuizGG generates questions from your input - you choose the topic, format, and difficulty. Open-ended questions are evaluated by AI, providing accurate feedback. Steps to generate and take a quiz:

1. From the dashboard, locate the **Tools Grid** and click **Quiz Generator** to access the quiz creation interface.

2. Complete the form (as shown in Figure 2.4) to define your quiz:

    - **Topic**: Specify a subject, such as "World History" or "Algebra." If you're unsure, click **Suggest a Topic** to receive suggestions based on popular topics (platform-wide).
    - **Quiz Format**: Choose from Multiple Choice (four answer options), True/False, or Open Ended (typed responses).
    - **Difficulty Level**: Select Easy, Medium, or Hard to match your skill level.
    - **Number of Questions**: Use the increment ("+") and decrement ("-") buttons to set the number of questions (1 to 20)

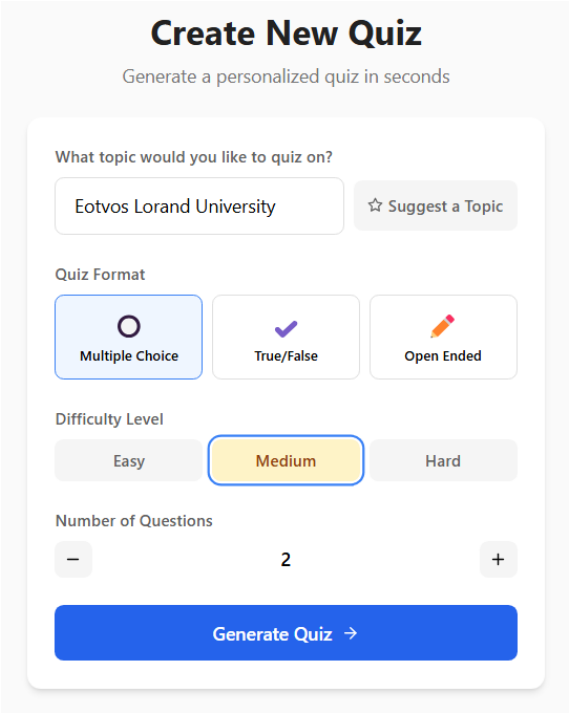


Figure 2.4: Quiz Generator Interface

3. Click **Generate Quiz** to start quiz creation. The system connects to the DeepSeek API to generate questions and redirects you to the quiz interface.

4. Answer each question based on its format:

   - As shown in Figure 2.5, for "Multiple Choice" and "True / False", select the correct option by clicking its radio button.

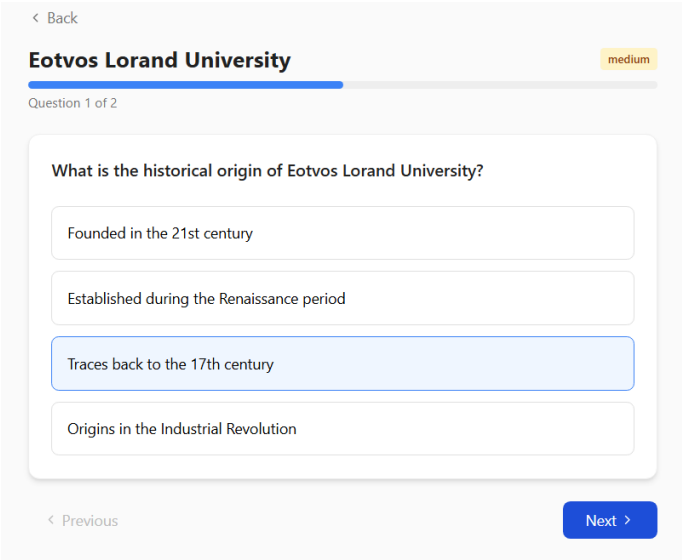


Figure 2.5: Multiple Choice Question Interface

- For Open Ended (see Figure 2.6), type your response in the provided text box. The AI will grade your answer according to strict but fair standards: it will provide a score based on accuracy, reasoning, and compliance to difficulty-specific rules.

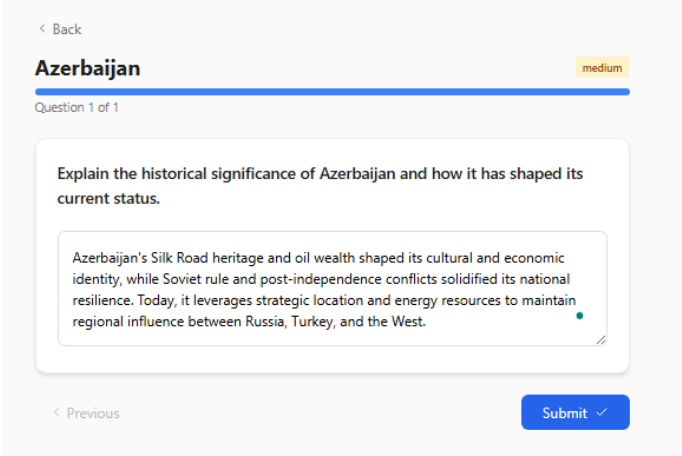


Figure 2.6: Open Ended Question Interface

5. After answering all questions, click **Submit** to view your results. Quiz submission is shown in Figure 2.7.

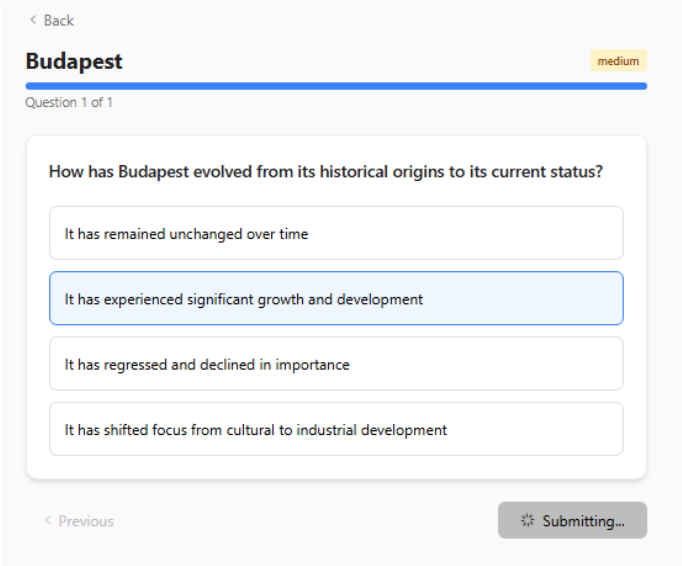


Figure 2.7: Quiz Submission Interface

6. The results page shows your score, correct answers, and feedback, including AI-generated sample answers for open-ended responses (see Figures 2.8 and 2.9) to help you improve.

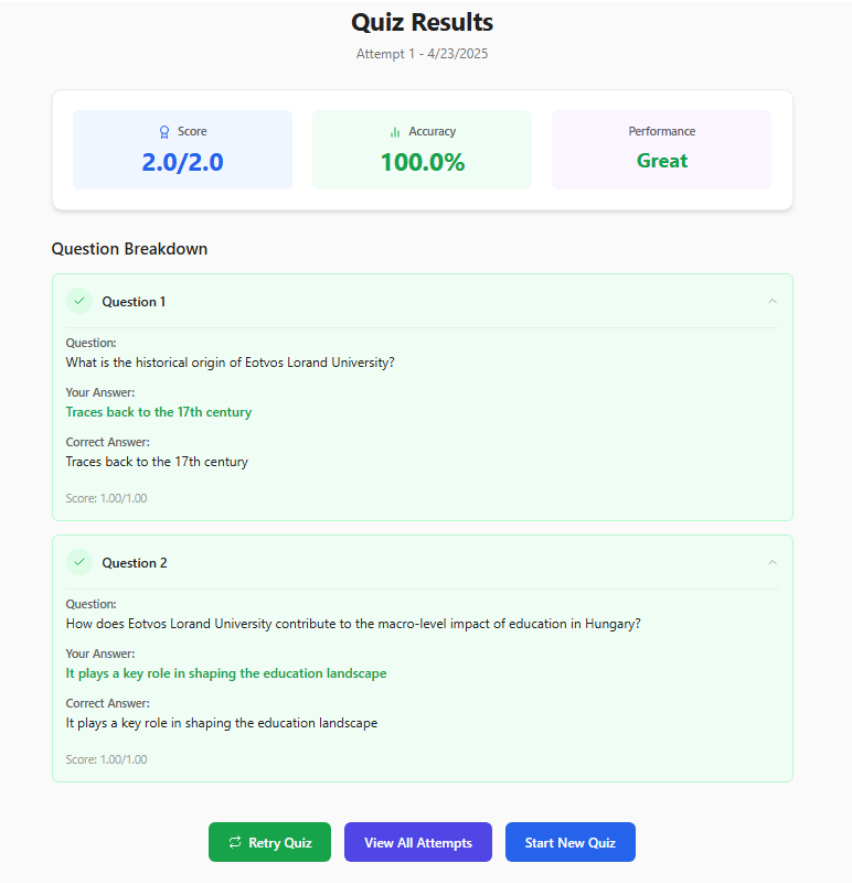


Figure 2.8: Multiple Choice Quiz Results

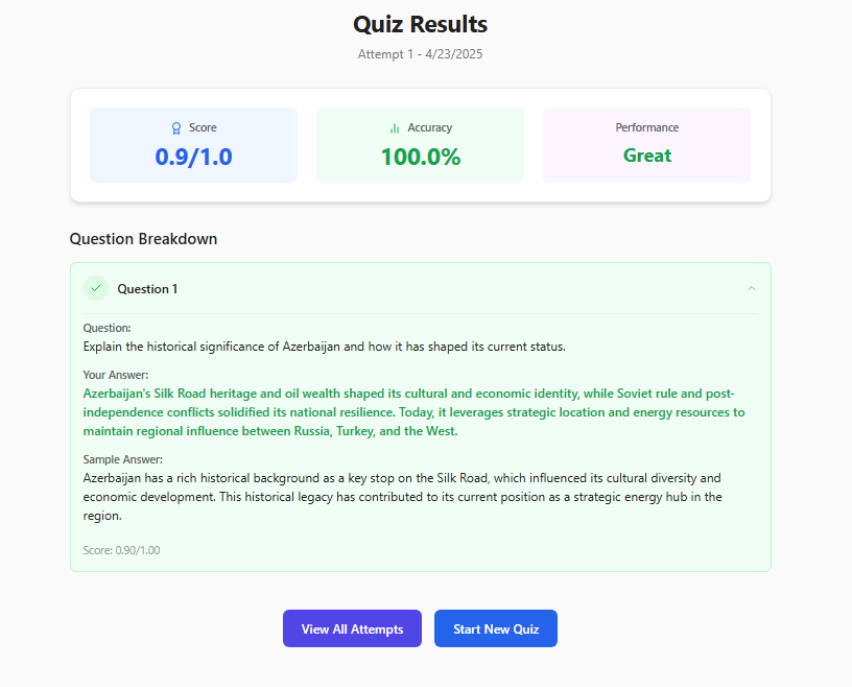


Figure 2.9: Open Ended Quiz Results with AI Grading

2.4.2 Reviewing Quiz History

You can track your quiz performance over time in the Quiz History page, which offers you insights into your progress and areas needed to improve. You can filter your history to select specific quizzes. This feature is great for reviewing past attempts and refining your study strategy. To use it:

1. From the dashboard, click **Quiz History** in the **Tools Grid** to access the quiz history page (shown in Figure 2.10).

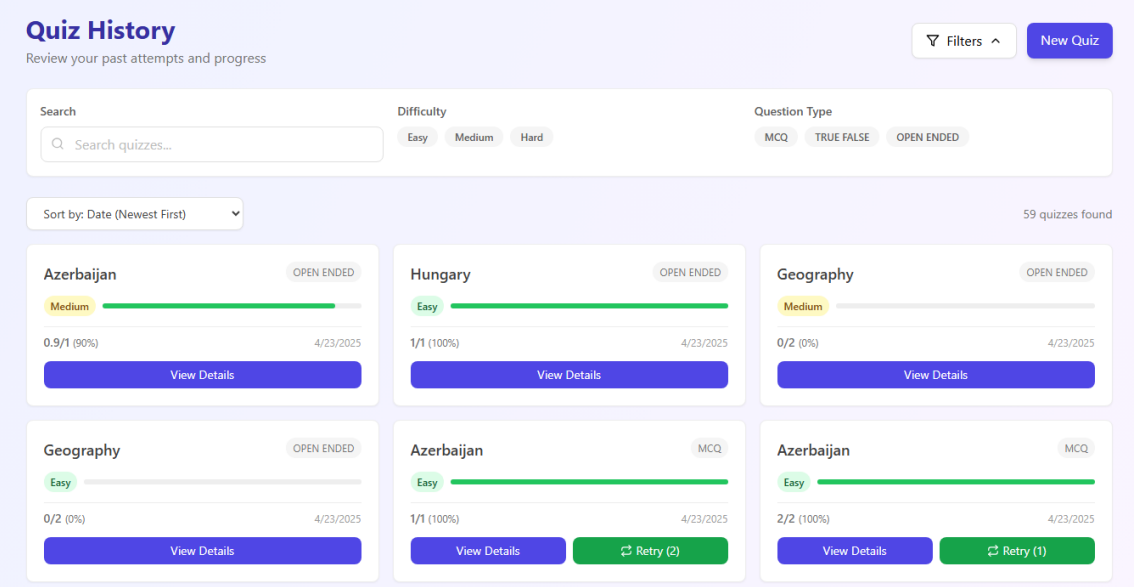


Figure 2.10: Quiz History Page

2. This page displays a list of all of your quizzes with details such as the question type, quiz topic, the date it was taken, the number of questions you answered, as well as your percentage score. Use the filter settings to narrow down quizzes based on topic, date, question type, or level of difficulty. Test yourself on previous quizzes by clicking the "Retry" button, as shown in Figure 2.11.

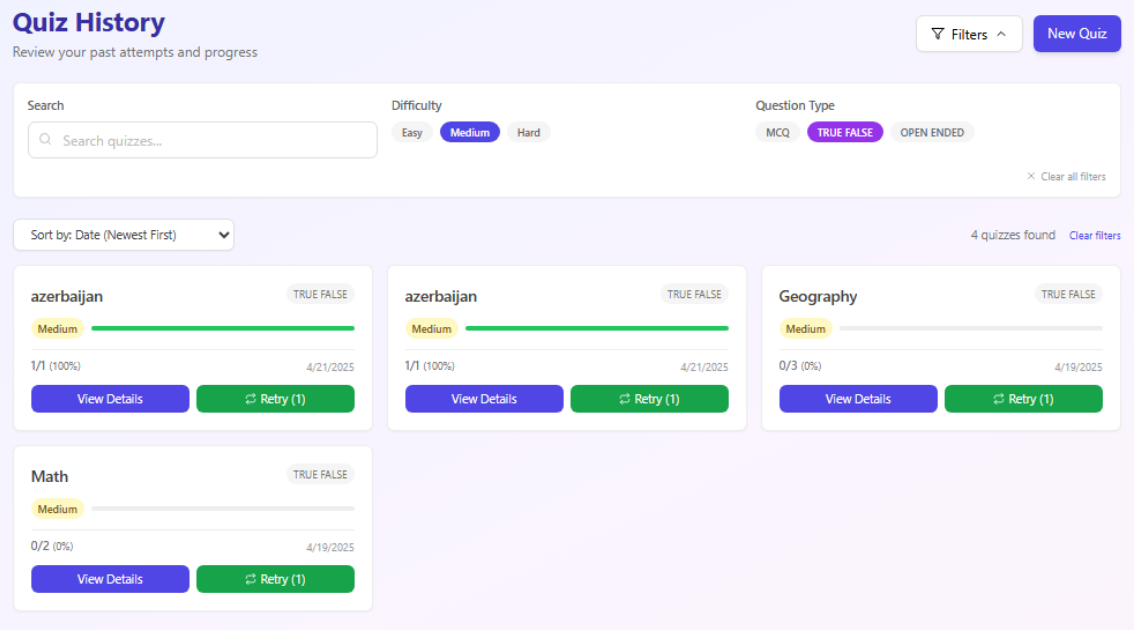


Figure 2.11: Filtered Quiz History Page

3. Click on a quiz entry to view a detailed breakdown, including each question, your answer, the correct answer, and sample answers from AI for open-ended questions, as demonstrated in Figure 2.12.

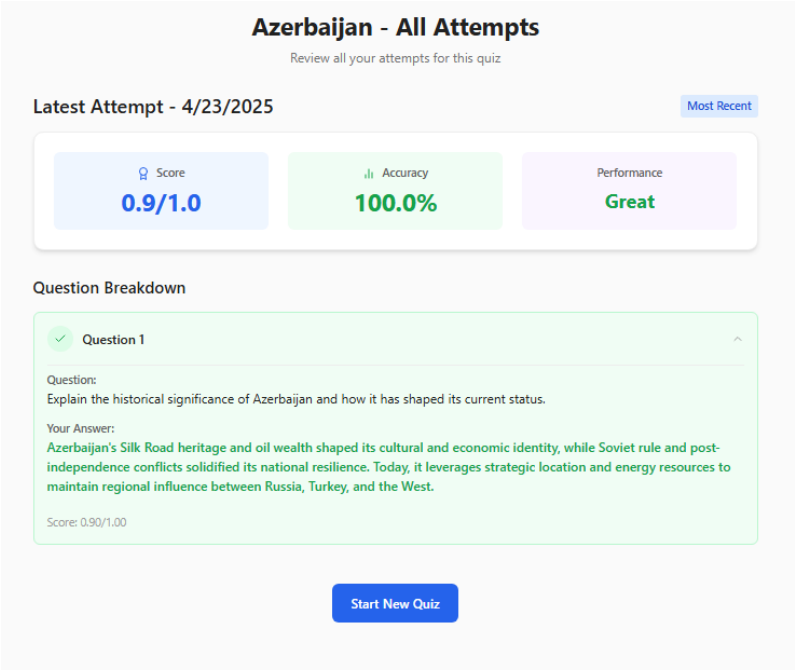


Figure 2.12: Detailed Quiz Results for a Completed Quiz

4. Use these insights to revisit weaker topics by generating new quizzes or creating flashcards for targeted review.

2.4.3 Using Study Tools (Articles and Flashcards)

QuizGG provides study tools to enhance your learning experience, such as custom flashcards and AI-generated articles. These tools support deeper understanding and quick review of key concepts.

Article Generation:

1. Navigate to **Study Tools** from the dashboard's **Tools Grid** to access the study tools page (shown in Figure 2.13).

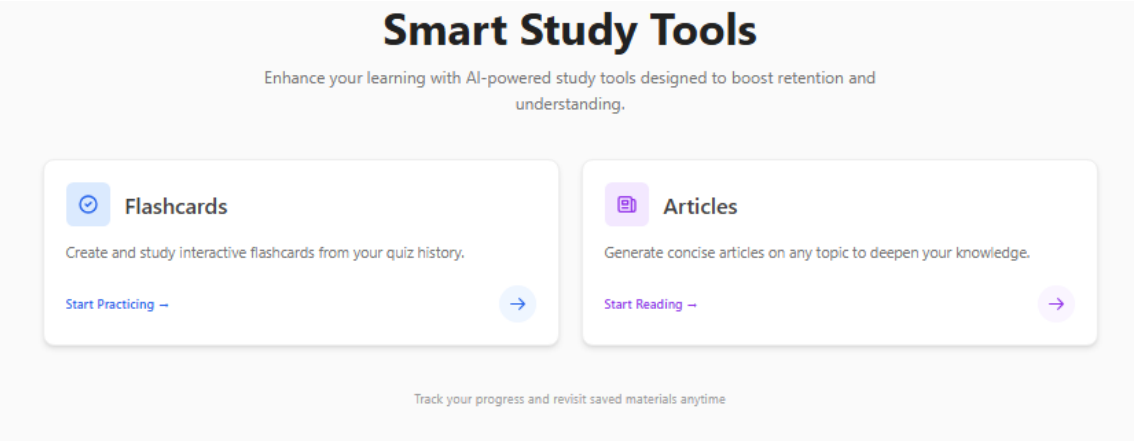


Figure 2.13: Study Tools Page

2. In the article generation section (check Figure 2.14), enter a topic (e.g., "Biology" or "Artificial Intelligence") in the input field.

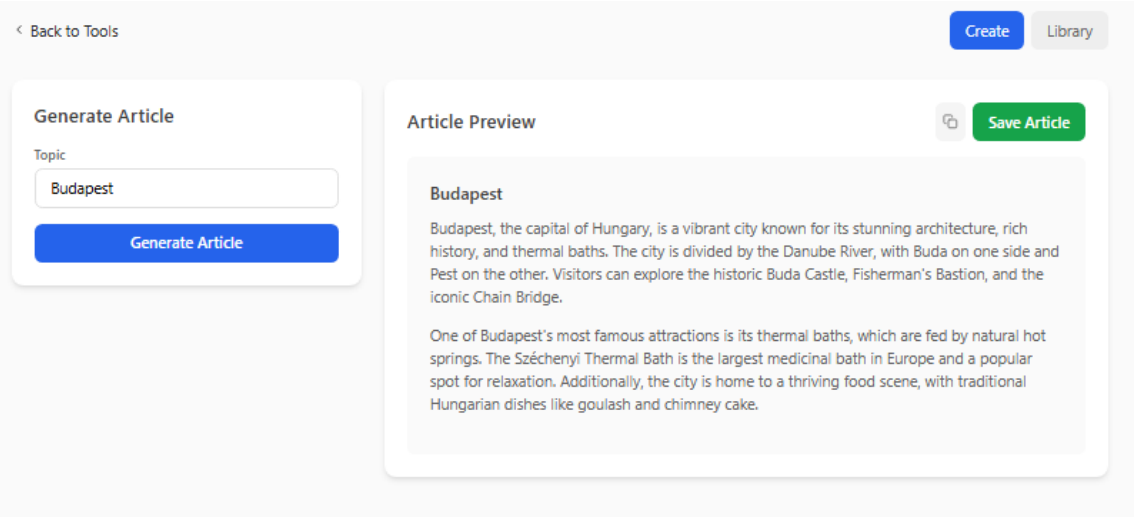


Figure 2.14: Article Generation Interface

3. Click **Generate Article** to create a concise article (100–140 words) using the DeepSeek API. The article provides a quick overview of the topic, ideal for building foundational knowledge.

4. Review the generated article and click **Save** to store it in your account for later reference or study.

Flashcards:

1. On the study tools page, locate the flashcard section (shown in Figure 2.15) and click **Start Practicing** to build a new deck.

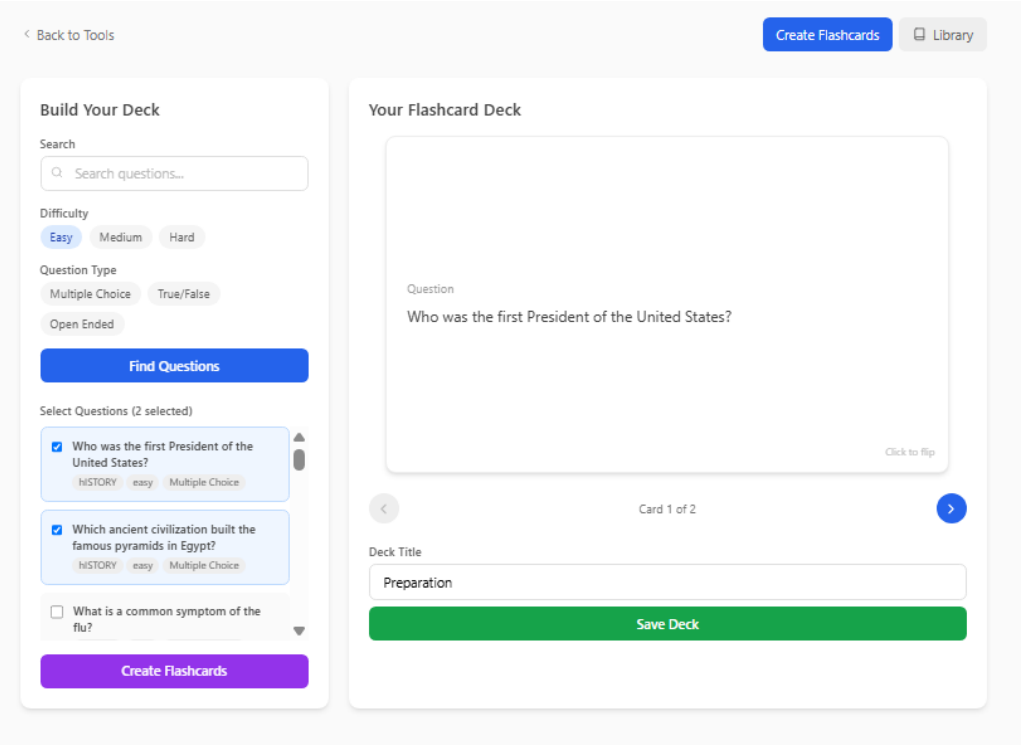


Figure 2.15: Flashcard Creation Interface

2. Filter and select questions from your quiz history to create your flashcard set:

   - **Filter Options**: Use the search bar to find specific questions, or filter by **Difficulty** (Easy, Medium, Hard) and **Question Type** (Multiple Choice, True/False, Open Ended) to narrow down the list.
   - **Select Questions**: Check the boxes next to the questions you want to include in your flashcard set. Each question's answer (from your quiz history) will automatically become the back of the card, with the question as the front.

3. Click **Create Flashcards** to generate the set. The system will create a deck where the front of each card is the question (e.g., "What is H2O?") and the back is the correct answer (e.g., "Water, a molecule composed of two hydrogen atoms and one oxygen atom.").

4. Review the flashcard set and click **Save Deck** to store it in your account for later reference or study.

2.4.4 Engaging with Daily Challenges

Daily Challenges offer a fun activity to maintain engagement. These matching questions test your ability to pair terms with definitions.

1. On the dashboard, click **Daily Challenge** in the **Tools Grid** to access the challenge page. See Figure 2.16.

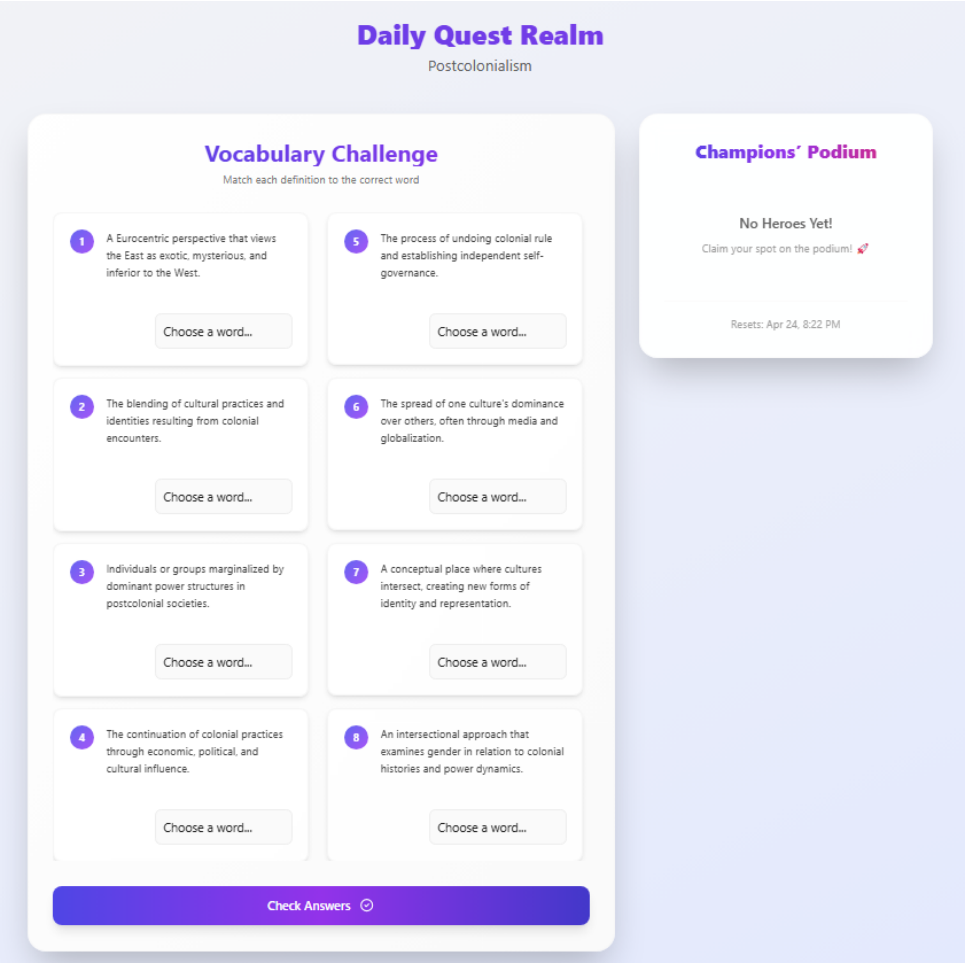


Figure 2.16: Daily Challenges Interface

2. The challenge presents 8 word-definition pairs. For each word, select the correct definition from a dropdown menu.

3. Review your choices and click **Submit**. You can only submit once per day, so make sure you do well!

4. After submission, the system displays your score and awards XP and coins for correct answers, contributing to your level and balance.

2.4.5 Managing Your Profile

The Profile Management (shown in Figure 2.17) feature allows you to personalize your QuizGG experience by updating your account details, customizing your appearance, and managing items purchased from the store. You can change your name and username, set titles and avatars, view owned badges as well as owned Mystery Boxes, open them, activate boosters, and review other owned items.

1. From the Navbar, click your avatar, then select **Profile** from the dropdown menu to access the profile management page.

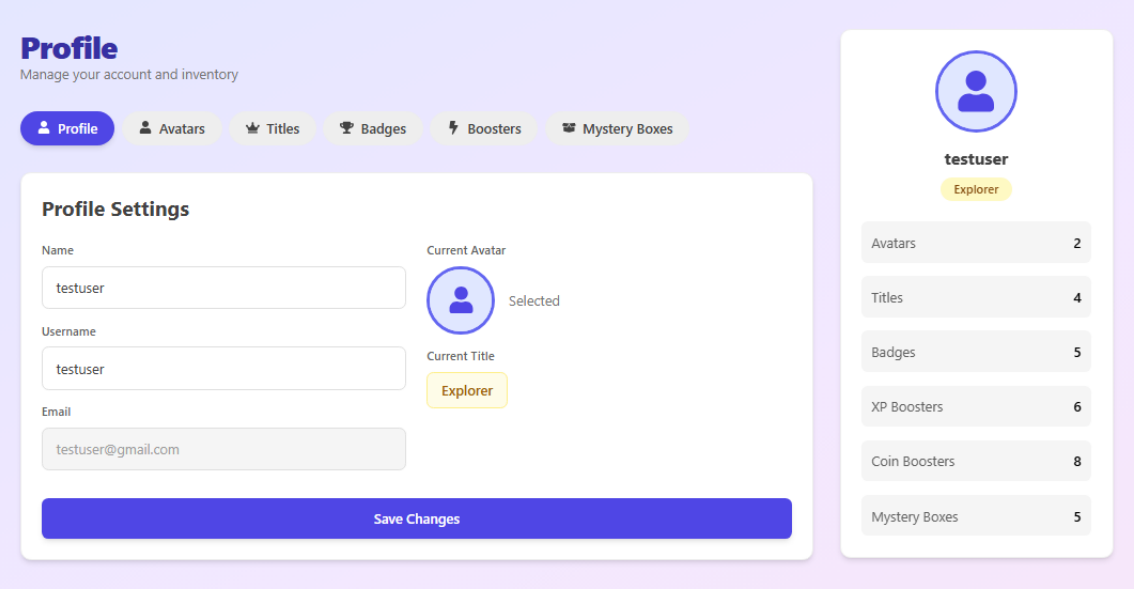


Figure 2.17: Profile Management Page

2. Update your account details:

   - **Name and Username**: Edit your display name (e.g., "Nahid Muradli") and username (e.g., "hjpivf") in the provided fields. Click **Save** to confirm changes, which will reflect across the platform, including the leaderboard.

3. Customize your profile appearance:
    - **Set Titles**: Choose from earned or purchased titles (e.g., *Brainiac* or *Explorer*). Click the **Titles** tab to view your available titles, select one by clicking it, then click **Save** in the profile section. The selected title will be displayed alongside your username.
    - **Set Avatars**: Click the **Avatars** tab to view your available avatars, select one by clicking it, then click **Save** in the profile section.
4. Manage owned items:
    - **View Mystery Boxes**: Check your inventory for Mystery Boxes purchased from the store. Each box is listed with its quantity. See Figure 2.18.

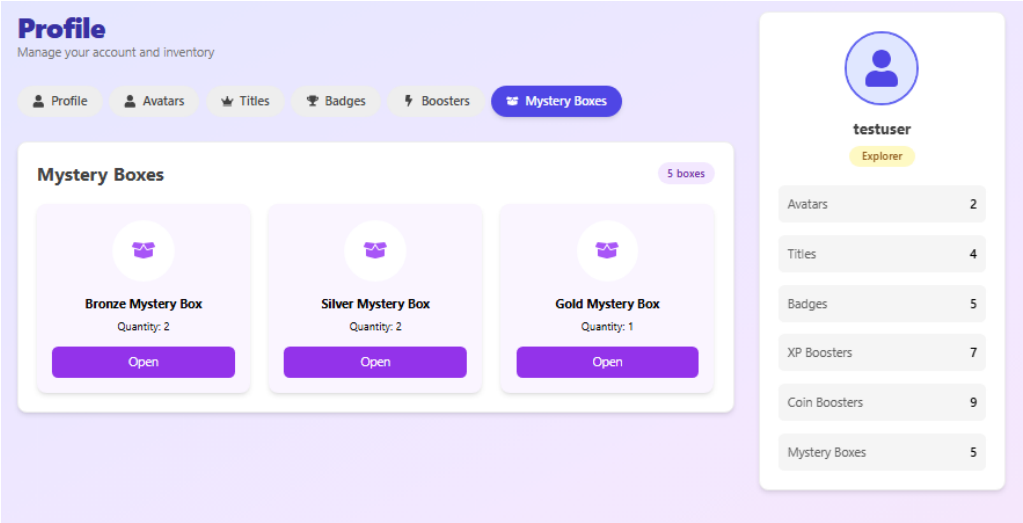


Figure 2.18: Mystery Box Inventory

- **Open Mystery Boxes**: Select a Mystery Box and click **Open** to reveal random rewards, such as Coin Boosters, XP Boosters, Ultra rare items, or other cosmetic items. You can see the box opening animation in Figure 2.19.

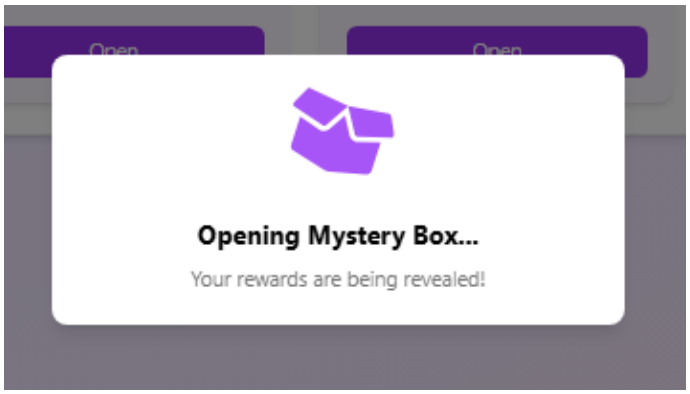


Figure 2.19: Opening a Mystery Box

- **View Rewards**: After opening, the rewards are displayed, showing what you've earned. These rewards (examples are shown in Figure 2.20) are added to your account immediately.

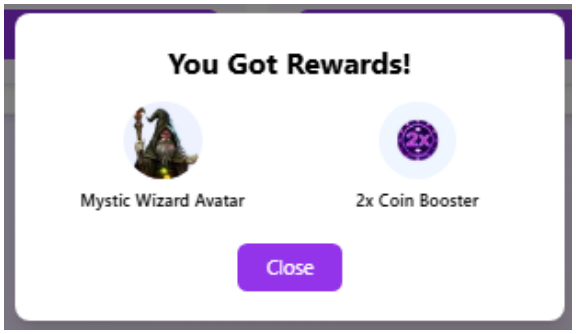


Figure 2.20: Rewards from a Mystery Box

- **Activate Boosters**: View available boosters (see Figure 2.21), like XP Boosters or Coin Boosters, which temporarily increase XP/Coin earned from quizzes. Click **Activate** to enable a booster, with its duration (e.g., 30 minutes) displayed.

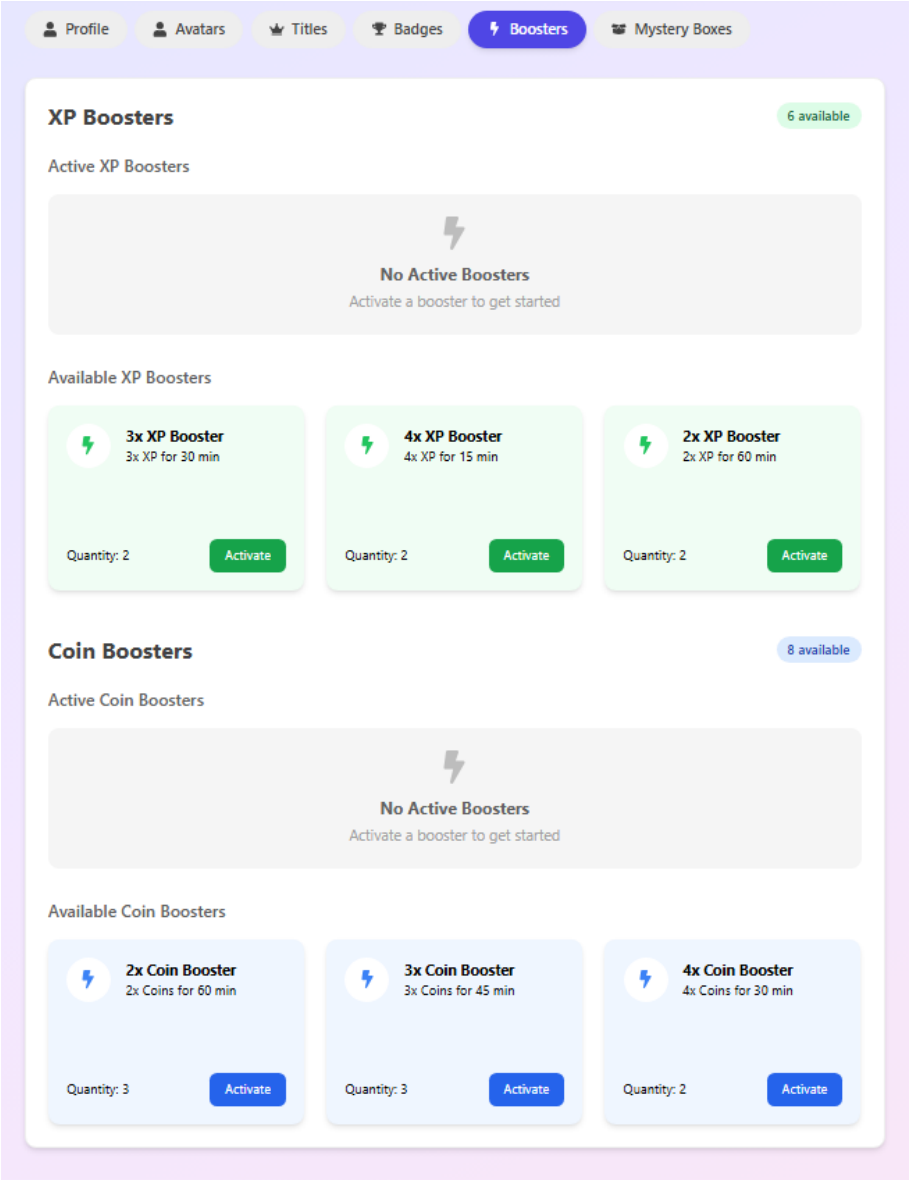


Figure 2.21: Booster Activation

- **See Owned Items**: Review all items in your inventory, including cosmetic items (e.g., avatars, titles) and consumables (e.g., boosters). Use or equip items as needed to enhance your experience.

5. Save any changes to your profile to ensure they take effect across QuizGG. Return to the dashboard to continue exploring other features.

2.4.6 Exploring the Store and Achievements

QuizGG's gamification features encourage learning by using rewards and recognition, making your study journey pleasurable and goal-driven.

Store:

1. Click **Store** in the dashboard's **Tools Grid** to visit the store page (can be seen in Figure 2.22)

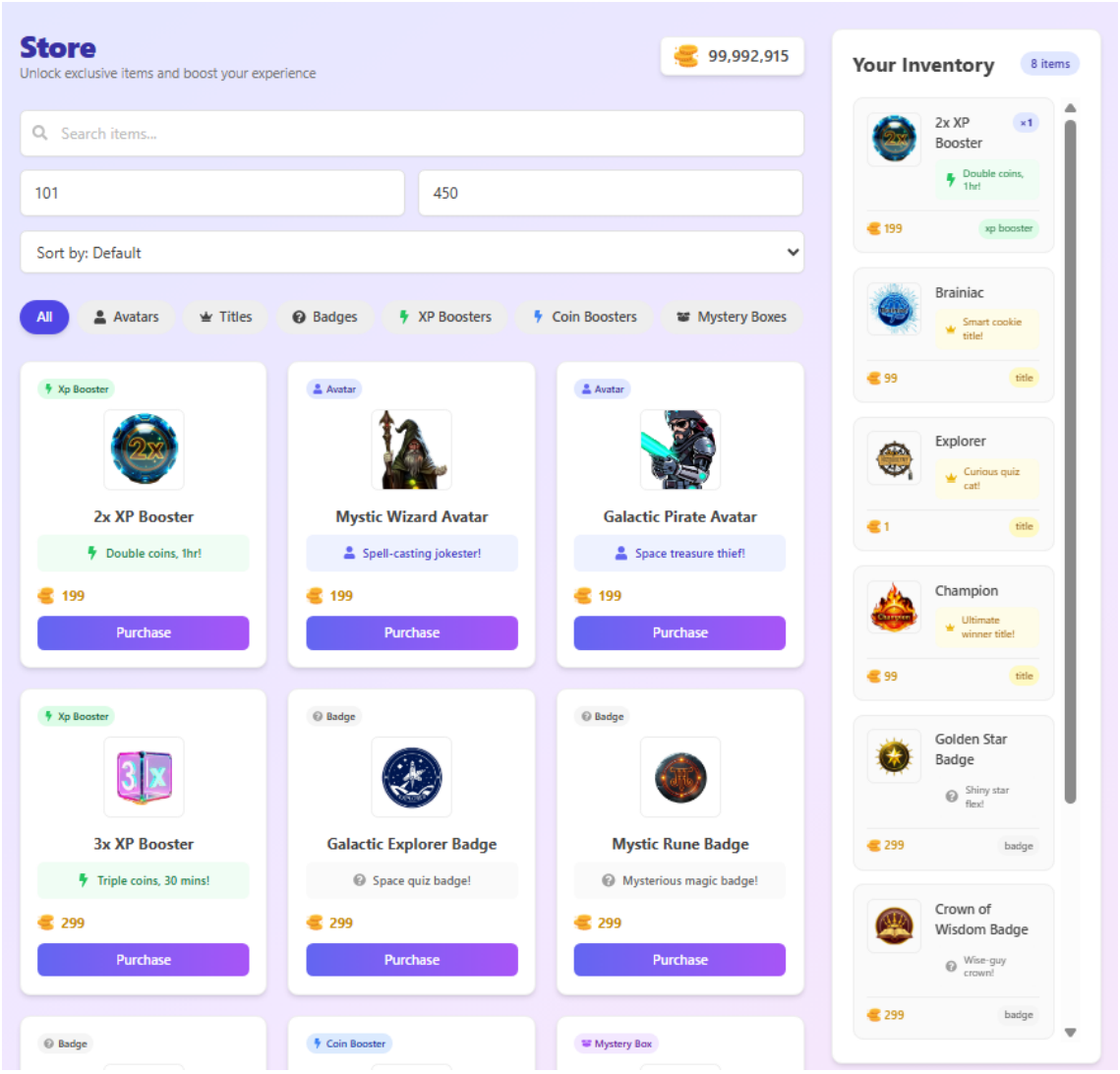


Figure 2.22: Store Page

2. Browse items like Mystery Boxes (random rewards) or Boosters. Each item lists its coin cost and benefits.

3. Check your coin balance, select an item, and click **Purchase** to add the item to your inventory, accessible via your Profile.

Achievements:

1. Navigate to **Achievements** from the **Tools Grid**. Achievements are shown in Figure 2.23.

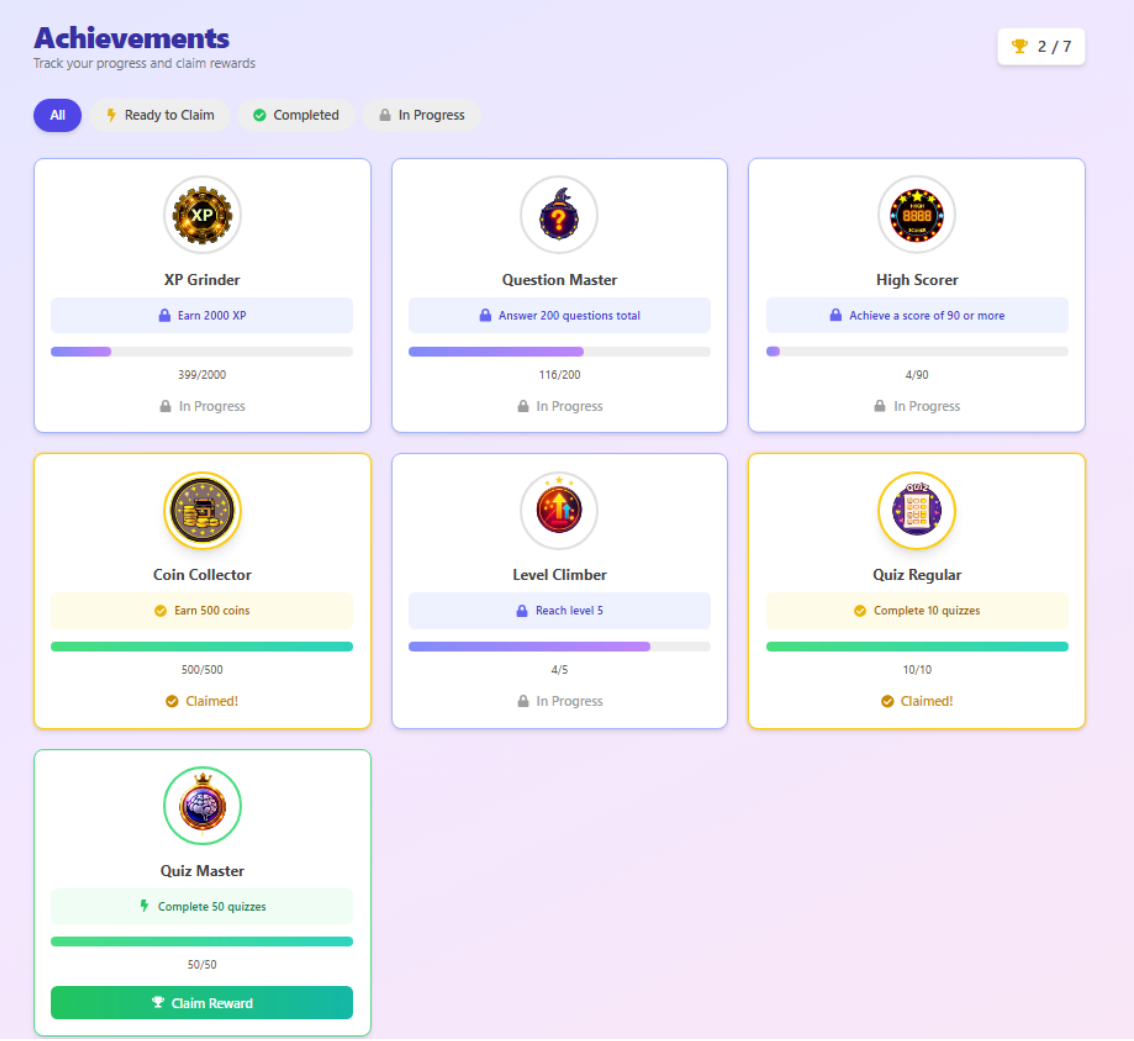


Figure 2.23: Achievements Page

2. View badges and milestones, such as “Complete 10 Quizzes” or “Earn 2000 XP.” Each achievement shows your progress and requirements.

3. Unlock badges by engaging with quizzes, challenges, and other activities. Unlocked badges will be displayed in your Profile to showcase your progress.

2.4.7 Tracking Progress and Analytics

The Stats & Insights feature offers a detailed view of your learning progress.

1. Click **Stats & Insights** in the **Tools Grid** to access the analytics page (shown in Figures 2.24 and 2.25).

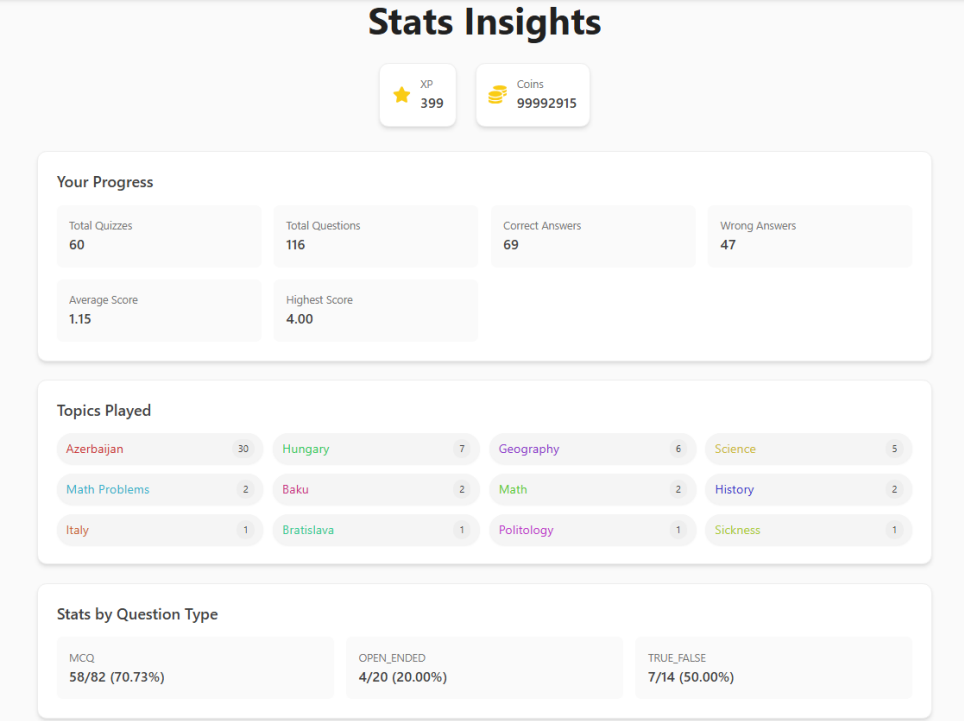


Figure 2.24: Stats & Insights Page (Part 1)

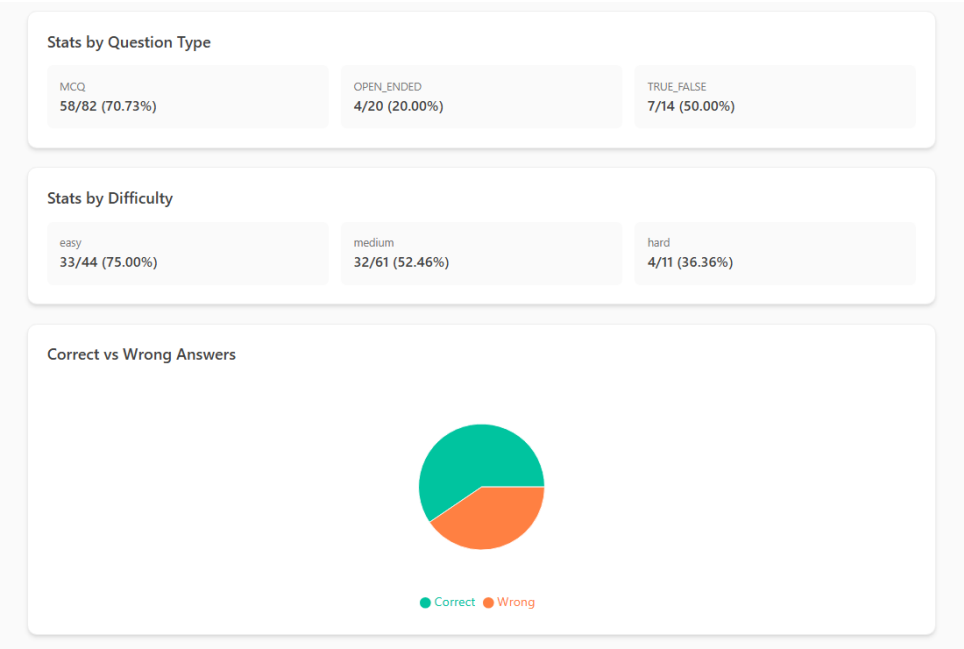


Figure 2.25: Stats & Insights Page (Part 2)

2. Review your progress metrics, including:
    - **XP and Coins**: Your current experience points and coin balance, displayed at the top.
    - **Total Quizzes**: Total quizzes completed.
    - **Total Questions**: Total questions answered across all quizzes.
    - **Correct/Wrong Answers**: Performance breakdown across questions.
    - **Average Score**: Average score across all quizzes.
    - **Highest Score**: Best quiz result.
    - **Topics Played**: Number of quizzes taken per topic (e.g., Biology, Hungary).
    - **Stats by Question Type**: Performance breakdown by question type (Multiple Choice Question (MCQ), Open Ended, True/False), showing correct answers and total attempts.
    - **Stats by Difficulty**: Performance breakdown by difficulty level (Easy, Medium, Hard), showing correct answers and total attempts.
3. Use visual charts, such as the pie chart for Correct vs Wrong Answers, to analyze your performance and target weaker areas with new quizzes or flashcards.

2.5 Troubleshooting

Here you can find about common issues you may face while using QuizGG and their detailed solutions to ensure a smooth experience. Each problem is described along with its root cause and step-by-step resolution, referencing setup steps from Section 2.2 where applicable.

2.5.1 Common Issues and Solutions

Some frequent problems and their solutions:

Database Connection Error:

- **Cause**: This occurs when the `DATABASE_URL` or `DIRECT_URL` in `.env` is incorrect or the Supabase database is unreachable, preventing QuizGG from accessing the remote PostgreSQL database.
- **Solution**:
  1. Open `.env` and verify `DATABASE_URL` and `DIRECT_URL` match the Supabase connection string (Section 2.2.3). Check for typos in the password or project reference.
  2. Ensure an active internet connection, as QuizGG communicates with Supabase servers.
  3. Log in to your Supabase dashboard to confirm the project is active. If paused, reactivate it or create a new project and update `.env`.

Prisma Migration Fails:

- **Cause**: The `npx prisma db push` command fails if the Supabase database is inaccessible, the connection string is incorrect, or the database schema is corrupted.
- **Solution**:
  1. Verify `DATABASE_URL` and `DIRECT_URL` in `.env` match the Supabase connection string.
  2. Check your Supabase project status in the dashboard. Reactivate if paused.
  3. If the database is corrupted, navigate to `Settings > Database > Reset Database` in Supabase, then rerun:

```
npx prisma db push
```

4. Ensure the Prisma schema (`prisma/schema.prisma`) is correctly configured.

Quiz Generation Fails:

- **Cause**: This happens if the `DeepSeek_API_KEY` in `.env` is invalid, missing, or if your DeepSeek account lacks credits. It may also occur due to internet issues.
- **Solution**:
  1. Check `DEEPSEEK_API_KEY` in `.env` matches the key from `https://platform.deepseek.com/` (Section 2.2.3).
  2. Log in to DeepSeek to verify your account's credit balance. Add funds if needed.
  3. Confirm internet connectivity, as quiz generation requires API calls.
  4. Try generating a quiz with a different topic or fewer questions to test API functionality.

Google Sign-In Not Working:

- **Cause**: Incorrect `GOOGLE_CLIENT_ID` or `GOOGLE_CLIENT_SECRET` in `.env`, or a misconfigured OAuth redirect URI in Google Cloud Console.
- **Solution**:
  1. Verify `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET` in `.env` match Google Cloud Console credentials (Section 2.2.3).
  2. In `https://console.cloud.google.com/`, check the OAuth 2.0 Client ID and ensure the redirect URI is `http://localhost:3000/api/auth/callback/google`.
  3. Save changes and wait a few minutes, then retry signing in.
  4. Enable pop-ups for `localhost:3000` in your browser.

Application Not Loading:

- **Cause**: Port 3000 is in use by another process, preventing the Next.js server from starting.
- **Solution**:
  1. Check for processes on port 3000:

```
lsof -i :3000
```

  Terminate using `kill -9 <PID>`. PID stands for Process Identifier.

2. Alternatively, use a different port:

```
PORT=3001 npm run dev
```

   Access at `http://localhost:3001`.

3. Rerun `npm install` to ensure dependencies are correctly installed.

- For further details on changing the server port refer to the official Next.js documentation [24].

2.5.2 Support

Contact the Developer: Email `hjpivf@inf.elte.hu` with the following details for personalized assistance:

- Steps leading to the issue
- Error messages received
- Your operating system and browser details

Chapter 3

Developer documentation

This chapter outlines the technical aspects of QuizGG. QuizGG uses Next.js, React, Prisma ORM, a relational cloud-based database (Supabase PostgreSQL), NextAuth, and DeepSeek API (deepseek-chat) for developing a modular, user-oriented learning tool.

3.1 Design

This section outlines the QuizGG design principles. User experience, system scalability, and data integrity are the fundamental design principles for an efficient learning experience. User interface wireframes, system architecture, data model, and UML diagrams are covered by the subsections under this section and they provide a full overview of the platform's design.

3.1.1 User Interface Wireframes

The user interface (UI) design of QuizGG is focused on usability and engagement. Wireframes for the most significant pages were created initially to provide a clear structure. Below are wireframes for five main pages: Welcome Page, Dashboard, Quiz Generator, Quiz History, and Store.

Initial wireframes were created using Excalidraw [25].

The Welcome Page introduces users to the platform with a minimalistic layout consisting of a header, a short description, and sign-up/login buttons. Its design makes sure newcomers can understand the platform's purpose quickly and how to use it.

Figure 3.1: Wireframe of the Welcome Page created using Excalidraw [25]

The Dashboard provides an overview of user progress and access to features. It includes a progress bar, a leaderboard, and a tool grid for navigation.

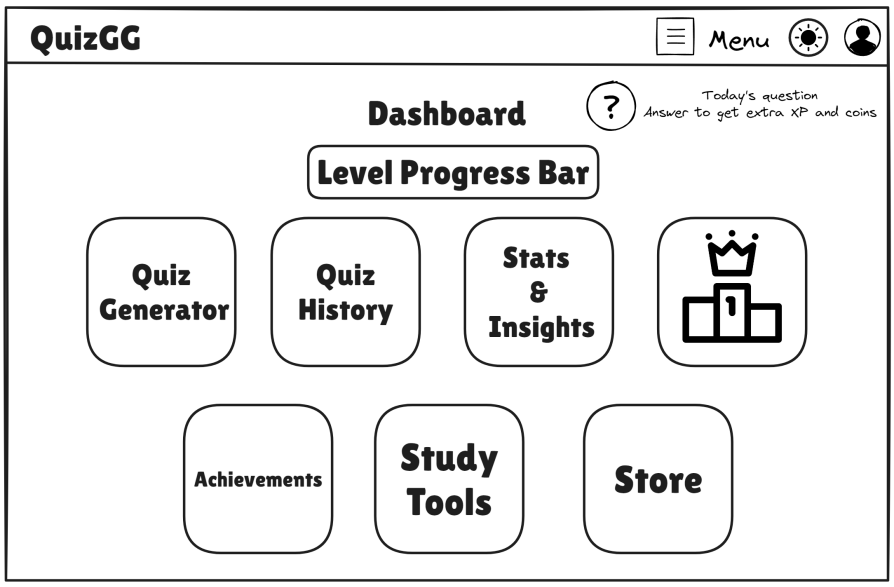


Figure 3.2: Wireframe of the Dashboard created using Excalidraw [25]

The Quiz Generator page allows users to create quizzes by specifying parameters like topic, question type, and difficulty.

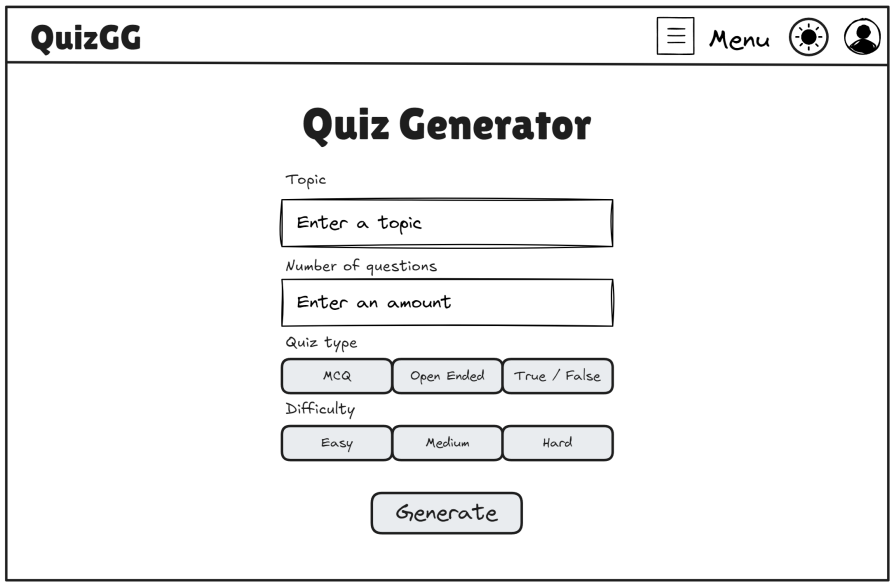


Figure 3.3: Wireframe of the Quiz Generator Page created using Excalidraw [25]

The Quiz History page lets users review past attempts by offering a list of quizzes, key details (topic, date, score), and filtering options. The design prioritizes clarity and enables users to analyze performance easily.

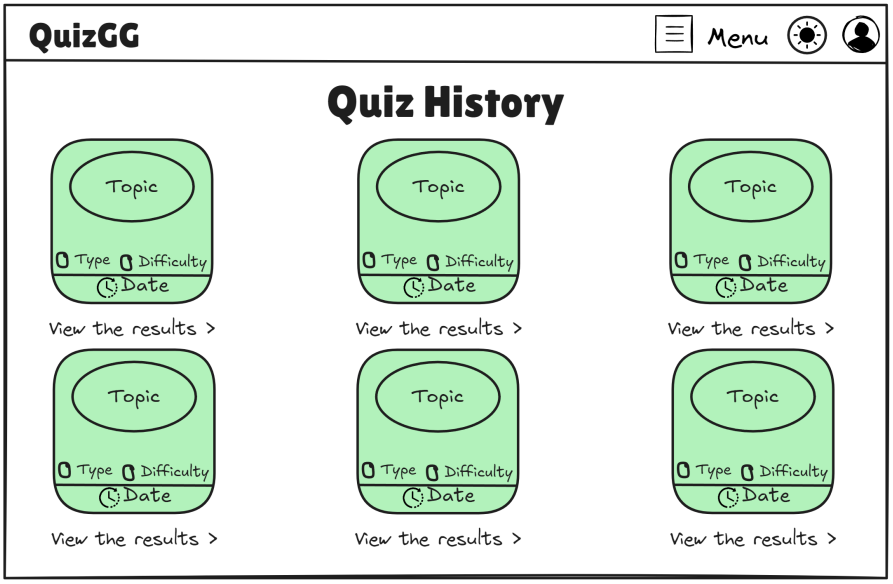


Figure 3.4: Wireframe of the Quiz History Page created using Excalidraw [25]

The Store facilitates the expenditure of coins (transactions) on products (e.g., boosters or mystery boxes). The wireframe includes a filter icon for browsing options, a grid of the products including their names, prices, and buy buttons, and a coin balance display in the top right.

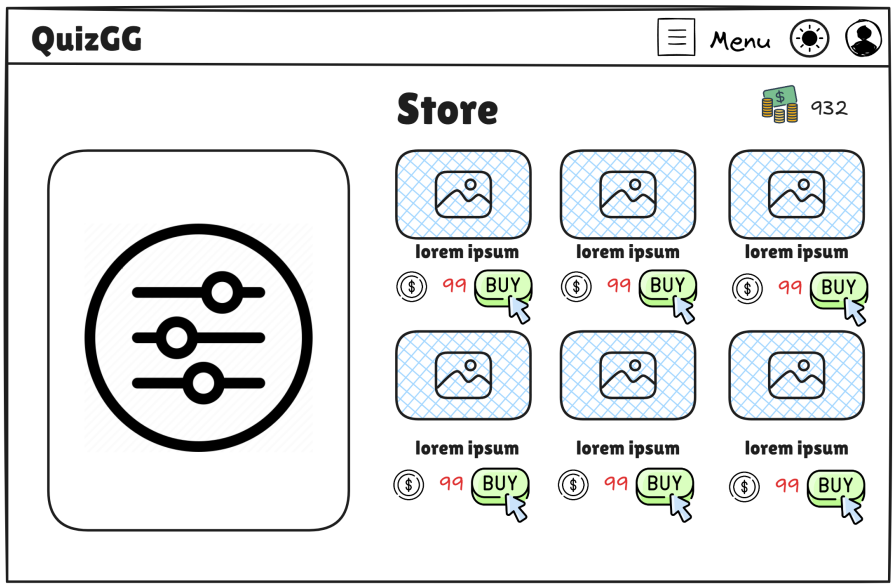


Figure 3.5: Wireframe of the Store Page created using Excalidraw [25]

These wireframes have user-centered design principles applied to minimize cognitive load and ensure consistency. Grid-based layouts maintain alignment, and interactive elements are placed for accessibility.

3.1.2 System Architecture

The system architecture of QuizGG is designed for scalability, modularity, and efficient data flow, which supports user interactions, AI-generated content, and data management. It follows a layered approach, where the concerns are divided into components that interact seamlessly. Figure 3.6 illustrates the system architecture and shows the main components and how they interact.

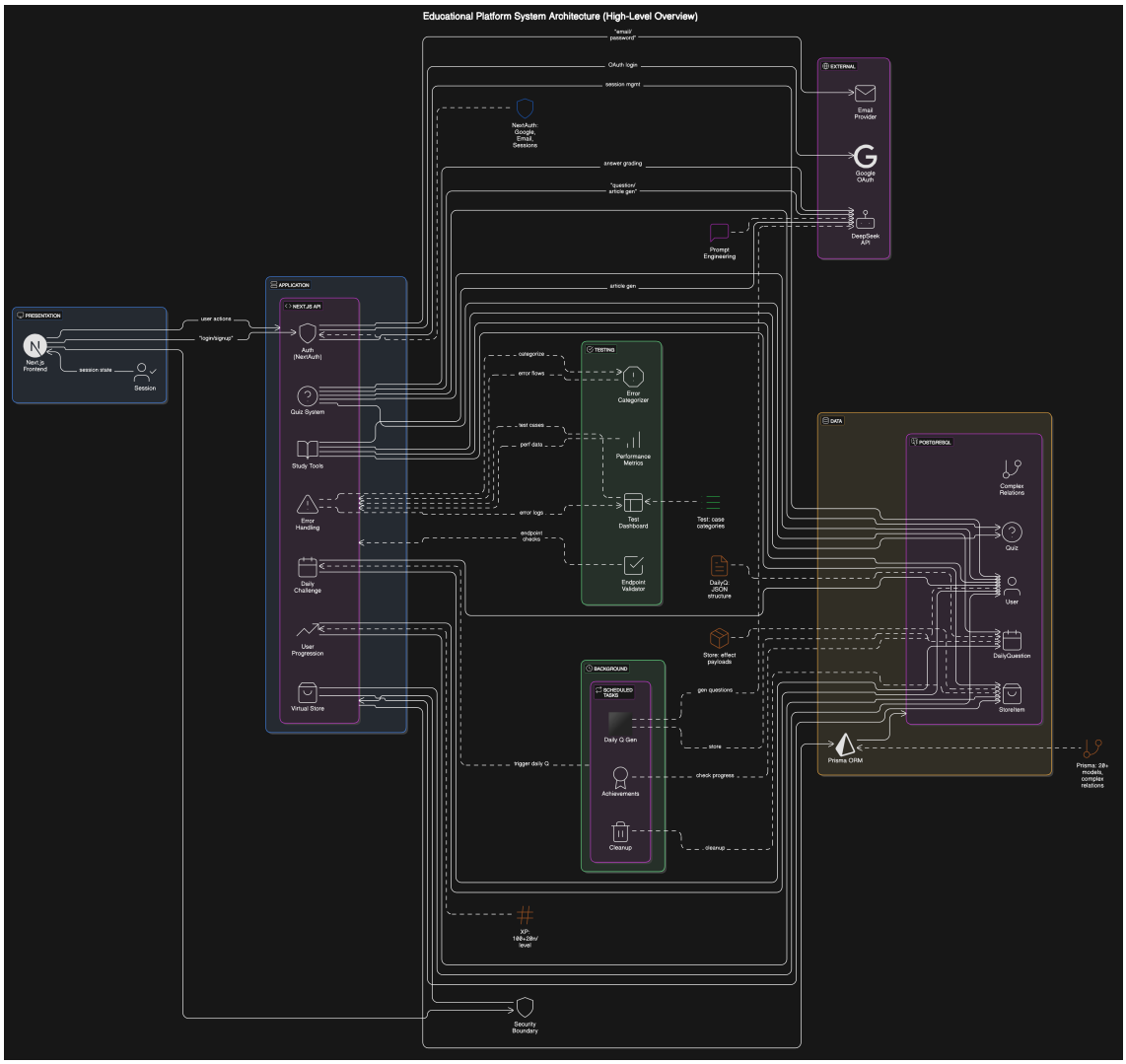


Figure 3.6: System Architecture of QuizGG generated with Eraser [26]

The three layers of architecture are the presentation, application, and data layers. The presentation layer manages the user interface, renders the pages, and captures the inputs. It provides responsiveness and accessibility on any device.

The application layer encapsulates the logic for things such as quiz generation and progress updates. It coordinates interaction with presentation & data layers, provides

gamification, and integrates external services for AI-based content generation. Its modular design supports future feature additions.

The data layer stores persistent data for consistency and efficient access. It is scalable and can handle growing user data without any performance issues. The layers interact through designated interfaces. These ensure independence and maintainability.

External services that provide AI-generated content are accessed securely.

3.1.3 Data Model

QuizGG's data model shows how all platform data connects and works together. It handles user info, quiz content, and game features in an organized way. You can see the Entity-Relationship Diagram (ERD) in Figure 3.7, which maps out all the main parts and how they relate to each other. See Figure 3.1 to get familiar with Database Schema.

Data Persistence

QuizGG stores all user data in a cloud database for safekeeping. This includes profiles, quiz history, and collected items. For session information and display preferences that require immediate access, we utilize secured (with encryption) local storage. The system automatically records all significant user actions—such as quiz completions and purchases—in real-time to maintain data accuracy and integrity.

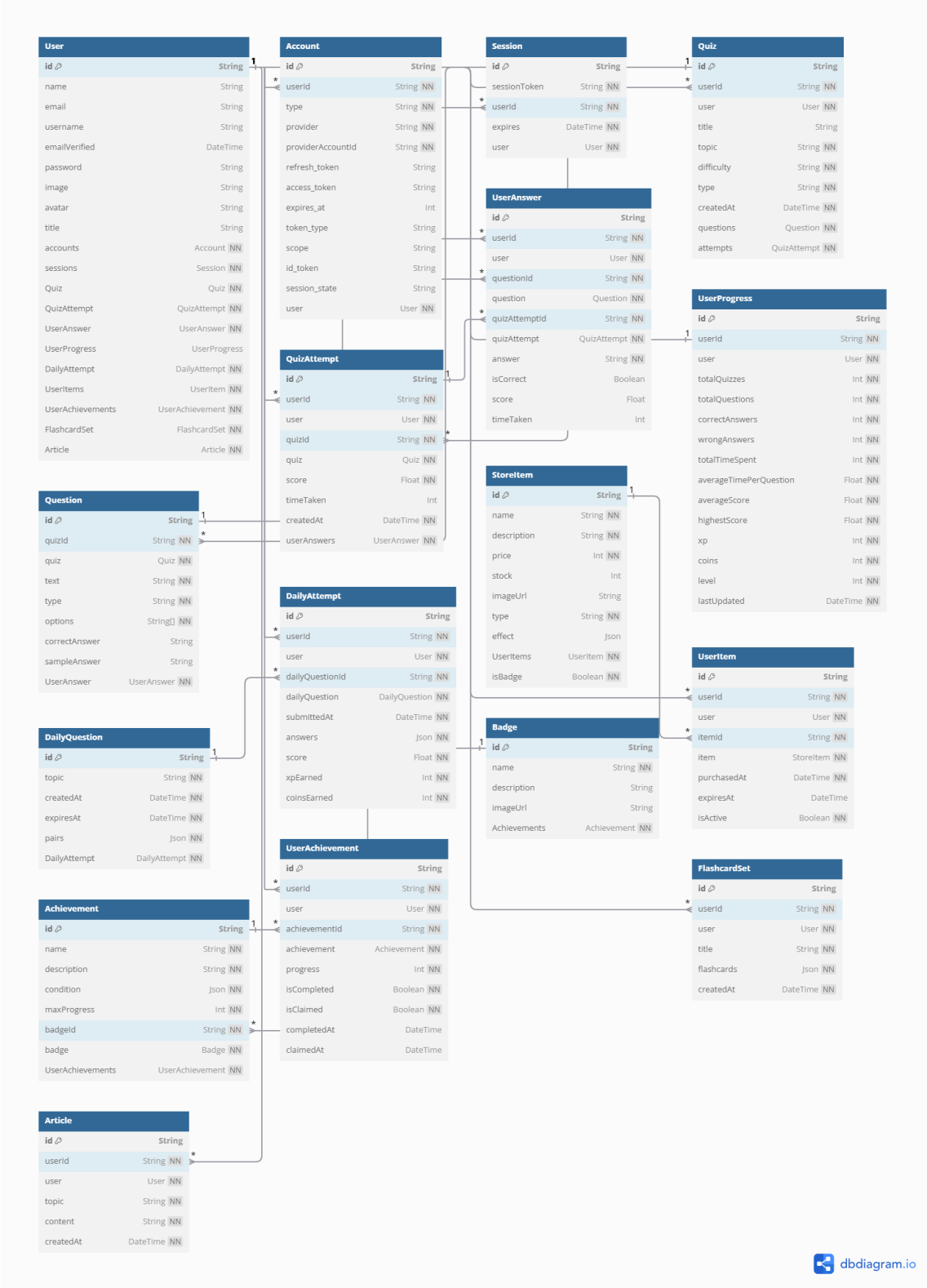


Figure 3.7: Entity-Relationship Diagram (ERD) of QuizGG created with DBML Diagram [27]

Table 3.1: Database Schema Overview

Component	Description
Authentication	
User	Central identity entity storing *name, email, username, avatar, title*. Links to authentication, quizzes, gamification, and learning resources.
Account	Manages authentication credentials with *provider, access token, refresh token*. Has a 1:1 relationship with `User`.
Session	Active login sessions recording *sessionToken, expiration*. Maintains secure user access.
Quizzing	
Quiz	Assessment content with *title, topic, difficulty, type*. Contains `Question` children and tracks `QuizAttempt` records.
Question	Quiz components storing *text, type (e.g., MCQ, true/-false), options, correctAnswer, sampleAnswer*. Links to user responses via `UserAnswer`.
QuizAttempt	Performance records with *score, timeTaken, createdAt*. Joins `User` and `Quiz` with attempt metadata.
UserAnswer	Individual question responses tracking *answer, isCorrect, score, timeTaken*. Connects `User`, `Question`, and `QuizAttempt`.
Gamification	
UserProgress	Gamification analytics including *totalQuizzes, totalQuestions, correctAnswers, averageScore, highestScore, xp, coins*. Tracks user performance metrics.
StoreItem	Marketplace inventory with *name, price, type, effect*. Includes both consumables and badges (*isBadge* flag).
UserItem	Ownership records tracking *purchasedAt, expiresAt, isActive*. Links `User` and `StoreItem`.
Achievement	Progress-based goals with *condition, maxProgress*. Associated with `Badge` rewards.
Badge	Visual achievements with *name, description, imageUrl*. Awarded via `UserAchievement` completion.
UserAchievement	Progress tracker (*progress, isCompleted, claimedAt*). Junction between `User` and `Achievement`.
Learning Resources	
FlashcardSet	Study aids containing *title, flashcards (JSON - JavaScript Object Notation)*. User-created learning resources.
Article	AI-generated content with *topic, content*. Supplemental learning material.
Daily/Weekly Challenges	
DailyQuestion	Time-limited challenges storing *topic, pairs (JSON)*. Basis for daily attempts.
DailyAttempt	Challenge submissions recording *answers, score, xpEarned, coinsEarned*. Tracks daily engagement.

Data Model Overview

The platform's data model includes these main parts, organized to support all key functions. The database relationships work efficiently: Users can have many Quizzes and QuizAttempts, and each Quiz can have multiple Questions. UserProgress and UserAchievement keep track of gaming elements, while UserItem and StoreItem handle the store functions.

3.1.4 Unified Modeling Language (UML) Diagrams

UML diagrams show QuizGG's workflows, mapping out user actions and system processes. This part presents a use case diagram and quiz flow diagram.

Use Case Diagram

Figure 3.8 shows system interactions with Guest Users, Authenticated Users, and Admins through key use cases.

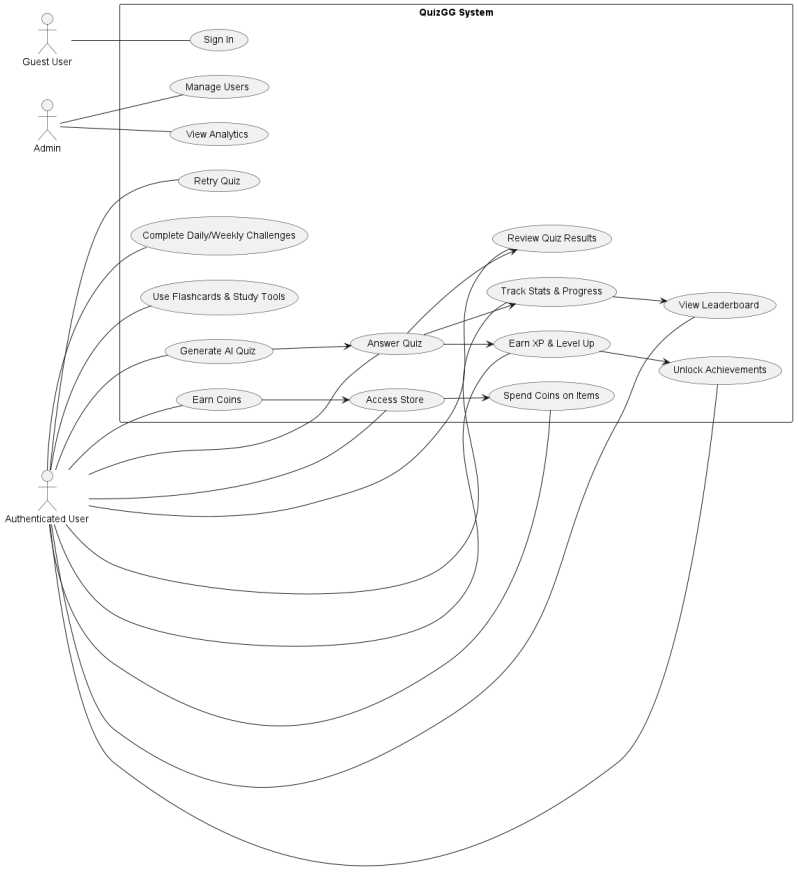


Figure 3.8: Use Case Diagram for QuizGG generated with PlantUML [28]

The diagram includes the following use cases shown in Figure 3.2.

Table 3.2: System Features Overview

Feature	Description
Sign In	Guest Users and Admins authenticate to access features.
Generate AI Quiz	Authenticated Users create quizzes, with the system generating questions based on specified parameters.
Answer Quiz	Authenticated Users take quizzes, submitting their responses for evaluation.
Review Quiz Results	Users view past attempts and feedback to assess their performance.
Track Stats & Progress	Users analyze performance metrics, such as scores and completion rates.
View Leaderboard	Users compare rankings, fostering competition and motivation.
Unlock Achievements	The system awards badges based on user activity and milestones.
Earn XP & Level Up	Users gain experience points, progressing through levels as they engage with the platform.
Earn Coins	Users earn coins through activities like quiz completion and challenges.
Spend Coins on Items	Users spend coins in the store to purchase items like boosters or badges.
Access Store	Users browse and purchase items to enhance their learning experience.
Use Flashcards & Study Tools	Users create and use study resources like flashcards for learning.
Complete Daily/Weekly Challenges	Users participate in time-limited challenges to earn rewards.
Retry Quiz	Users can retake quizzes to improve their scores and understanding.
Manage Users	Admins oversee user accounts, ensuring proper access and system integrity.
View Analytics	Admins access system performance data and user activity metrics.

Quiz Flow Diagram

The quiz flow diagram, Figure 3.9, shows how quizzes are created and submitted as a series of actions between users and the system.

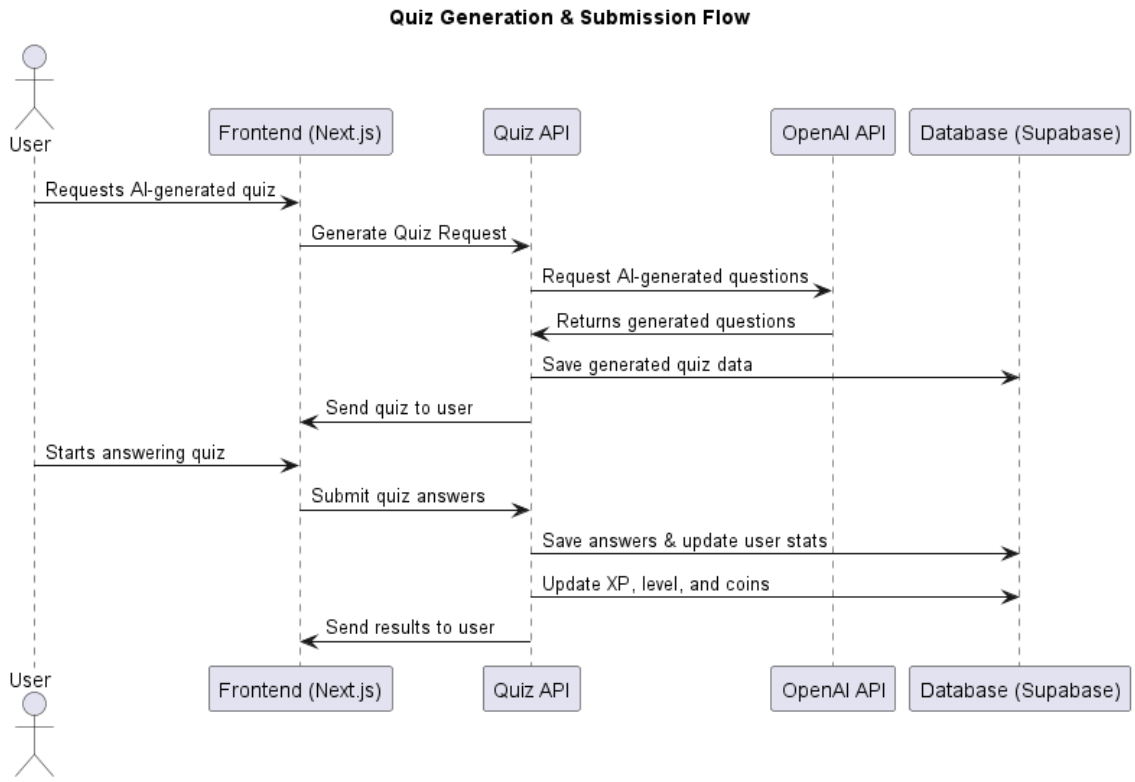


Figure 3.9: Quiz Flow Diagram for QuizGG generated with PlantUML [28]

Users start by requesting for a quiz and picking options like topic, question type, and difficulty level. The system then creates questions using AI, saves the quiz, and sends it to the user. After answering the questions, users submit their answers. The system checks these answers, updates the user's stats with new XP points and coins, and shows results including scores and example answers.

These UML diagrams give a clear picture of how QuizGG works, making sure user interactions run smoothly and match what the platform aims to do.

3.2 Implementation

This section covers QuizGG's technical implementation details, including the technology stack, backend systems, frontend development, database connections, AI integration, automated CRON jobs, and areas for future improvement. The platform uses current web technologies to create an interactive education system focused on quiz generation and learning experiences.

3.2.1 Tech Stack

QuizGG uses modern JavaScript technology with a focus on performance, scalability, and developer experience.

- **Framework**:
  - Next.js (15.0.2) - React framework for SSR (Server-Side Rendering), routing, and API routes (custom REST API - Representational State Transfer API [29])
- **Core Languages**:
  - TypeScript (5.6.3) - Type-safe JavaScript superset [30]
  - JavaScript (ECMAScript 6, known as ES6) - Core programming language [31]
- **Frontend Libraries**:
  - React (with TSX) - JavaScript library for building user interfaces with TypeScript and JSX [14].
  - Framer Motion (12.4.10) - Animation library [32]
  - Tailwind CSS (Cascading Style Sheets) (3.4.17) - Utility-first CSS framework [33]

XML and W3C Standard

TSX [34] stands for TypeScript XML and JSX [35] stands for JavaScript XML (Extensible Markup Language) [36], an open standard for structured data markup defined by the W3C [37].

- **Backend & Database**:
  - Prisma (5.21.1) - ORM (Object-Relational Mapping) for database access
  - NextAuth.js (4.24.10) - Authentication framework
  - Axios (1.7.7) - HTTP (HyperText Transfer Protocol) client adhering to the HTTP 1.1 specification [38]
  - Node.js Crypto - Built-in security utilities
- **AI Integration**:
  - DeepSeek API - deepseek-chat integration
- **DevOps & Tooling**:
  - Node-cron (3.0.3) - Job scheduler [39]
  - Concurrently - Run multiple commands
  - ts-node - TypeScript execution
- **Data Visualization**:
  - Chart.js (4.4.8) - Charting library [40]
  - React-ChartJS-2 (5.3.0) - React wrapper [41]
- **Security**:
  - Bcrypt (5.1.1) - Password hashing [42]
  - JWT - Token authentication [43]

3.2.2 Backend Implementation

The backend runs on a custom REST API implemented with Next.js API routes that handle authentication, database interactions, and AI-powered quiz generation. Key components include Prisma for database setup, NextAuth.js for authentication, and various API routes for core functions.

The database connection uses Prisma as shown in the code from `lib/db.ts` in Code 3.1.

```
import { PrismaClient } from "@prisma/client";
import "server-only";

declare global {
  var cachedPrisma: PrismaClient;
}

export let prisma: PrismaClient;

if (!process.env.DATABASE_URL) {
    throw new Error("DATABASE_URL environment variable is missing.");
  }

if (process.env.NODE_ENV === "production") {
    prisma = new PrismaClient({
      datasources: {
        db: {
          url: process.env.DATABASE_URL,
        },
      },
    });
  } else {
    if (!global.cachedPrisma) {
      global.cachedPrisma = new PrismaClient({
        datasources: {
          db: {
            url: process.env.DATABASE_URL,
          },
        },
      });
    }
    prisma = global.cachedPrisma;
  }
```

Code 3.1: Prisma Client Setup in lib/db.ts

This implementation keeps a single Prisma client instance during development (using a global variable to prevent multiple instances during hot reloading) and creates a new instance in production. It connects to Supabase PostgreSQL through the `DATABASE_URL` environment variable.

Authentication works through NextAuth.js, set up in `lib/nextauth.ts`, with JWT-based [43] sessions following the JSON Web Token standard. The configuration includes environment variable checks, type extensions for the session and JWT, and a Prisma adapter to integrate with the database. The code highlights provider configuration for Google OAuth (Open Authorization) and credentials-based login, along with essential session handling shown in Code 3.2.

```
export const authOptions: NextAuthOptions = {
  adapter: PrismaAdapter(prisma),
  session: {
    strategy: "jwt",
  },
  secret: process.env.NEXTAUTH_SECRET,
  pages: {
    signIn: "/auth/signin",
    error: "/auth/error",
  },
  callbacks: {
```

```
    jwt: async ({ token, user }) => {
      if (user) {
        token.id = user.id;
        token.username = user.username;
      }
      return token;
    },
    session: ({ session, token }) => {
      if (token) {
        session.user.id = token.id;
        session.user.username = token.username;
      }
      return session;
    },
  },
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      authorization: {
        params: {
          prompt: "consent",
          access_type: "offline",
          response_type: "code",
        },
      },
      profile: (profile) => {
        const username = profile.email?.split("@")[0] || "user";
        return {
          id: profile.sub,
          name: profile.name,
          email: profile.email,
          username,
          image: profile.picture,
        };
      },
    }),
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        login: { label: "Username or Email", type: "text" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        const user = await prisma.user.findFirst({
          where: {
            OR: [
              { email: credentials.login },
              { username: credentials.login },
            ],
          },
        });
        if (!user || !user.password || !(await bcrypt.compare(credentials.password, user.password))) {
          throw new Error("Invalid credentials");
        }
        return { id: user.id, name: user.name, email: user.email, username: user.username, image: user.image };
      },
    }),
  ],
};
```

Code 3.2: NextAuth.js Provider Configuration in lib/nextauth.ts

The setup relies on Prisma to handle user data storage. It supports Google OAuth, where a custom profile mapping creates a username from the user's email or name. For credentials-based login, bcrypt verifies passwords. Sessions use JWT and include

the user's ID, username, and access token.

A core API endpoint, `/api/quiz/generate`, builds quizzes and uses the DeepSeek API (deepseek-chat) to craft questions (see Section 3.2.5 for more). The backend also features algorithms for search, sorting [44], and weighted selection [45], with a Fisher-Yates shuffle [46] coded in the `lib/algorithms` directory. For instance, `lib/algorithms/search.ts` uses the KMP (Knuth-Morris-Pratt) [47] algorithm for substring search. See the Code 3.3.

```
export function computeLPSArray(pattern: string): number[] {
    const lps = new Array(pattern.length).fill(0);
    let len = 0;
    let i = 1;

    while (i < pattern.length) {
      if (pattern[i] === pattern[len]) {
        len++;
        lps[i] = len;
        i++;
      } else {
        if (len !== 0) {
          len = lps[len - 1];
        } else {
          lps[i] = 0;
          i++;
        }
      }
    }
    return lps;
  }

  export function KMPSearch(text: string, pattern: string): boolean {
    if (pattern === "") return true;
    const textLower = text.toLowerCase();
    const patternLower = pattern.toLowerCase();

    const lps = computeLPSArray(patternLower);
    let i = 0;
    let j = 0;

    while (i < textLower.length) {
      if (patternLower[j] === textLower[i]) {
        i++;
        j++;
      }
      if (j === patternLower.length) {
        return true;
      } else if (i < textLower.length && patternLower[j] !== textLower[i]) {
        if (j !== 0) {
          j = lps[j - 1];
        } else {
          i++;
        }
      }
    }
    return false;
  }
```

Code 3.3: KMP Search in lib/algorithms/search.ts

The algorithm powers efficient searches on pages such as the store, articles, and flashcards, with a time complexity of $O(n + m)$, where n represents the text length

and m denotes the pattern length.

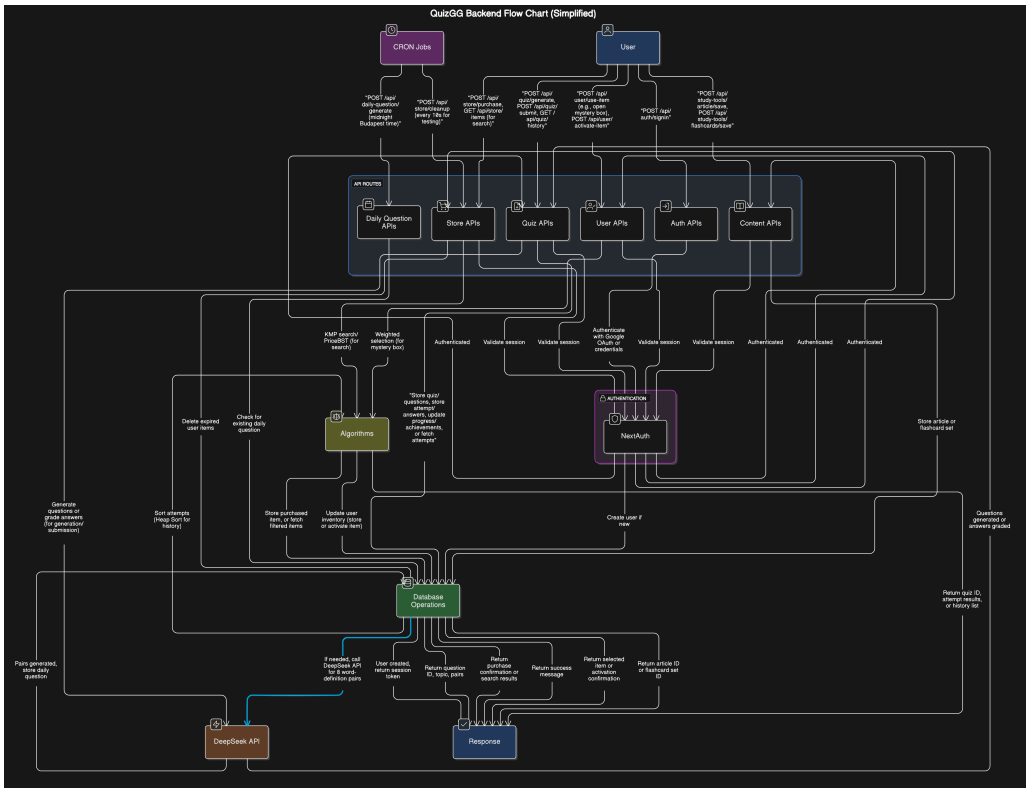


Figure 3.10: Backend Flow Diagram for QuizGG generated with Eraser [26]

The QuizGG Backend Flow Chart (shown in Figure 3.10) illustrates how the app's main components interact. User requests flow through API groups: Auth APIs (`/api/auth/`) handle logins, Quiz APIs (`/api/quiz/`) manage quiz features, Daily Question APIs (`/api/daily-question/`) support daily challenges, Store APIs (`/api/store/`) process purchases, User APIs (`/api/user/`) deal with inventory actions, and content APIs (`/api/study-tools/`) enable article and flashcard creation. NextAuth.js oversees authentication, managing sign-ins and session checks, while Prisma stores user data. For quizzes, the DeepSeek API (deepseek-chat) generates and grades questions, with Prisma saving data and Heap Sort arranging history. CRON jobs create daily questions at midnight and remove expired store items every 10 seconds (for testing).

3.2.3 Frontend Implementation

This part explains the frontend of a learning platform, diving into its layout, navigation, and user experience.

Overview

The frontend creates an engaging space for users to interact with educational content using:

- React with Next.js for routing and rendering.
- Tailwind CSS for styling.
- Framer Motion for animations.
- Axios [48] for API requests.
- NextAuth.js for authentication.

Project Structure

The frontend is organized for scalability:

- `app/`: Pages for core features
- `components/`: Reusable UI elements and feature-specific components.
- `lib/`: Utilities and algorithms.
- `scripts/`:Automation scripts for development and build processes.

Routing & Navigation

The app uses Next.js's app router to render pages with HTML5-compliant (HyperText Markup Language 5) markup [49].

- `app/page.tsx` maps to the root route (`/`).
- Dynamic routes like `app/quiz/[quizId]/page.tsx` for specific content.
- `layout.tsx` provides a shared `Navbar` across all pages, as shown in Code 3.4.

```
1  import "./globals.css";
2  import ThemeProviderWrapper from "@/components/ThemeProviderWrapper";
3  import SessionProviderWrapper from "@/components/SessionProviderWrapper";
4  import Navbar from "@/components/Navbar";
5  import { Toaster } from "react-hot-toast";
6  import  { AvatarProvider } from "@/components/AvatarContext";
7
8  export default function RootLayout({ children }: { children: React.ReactNode }) {
9    return (
10     <html lang="en" suppressHydrationWarning>
```

```
      <body className="bg-background text-foreground">
        <SessionProviderWrapper>
            <ThemeProviderWrapper>
              <AvatarProvider>
                <Navbar />
                {children}
                <Toaster position="top-center" reverseOrder={false} />
              </AvatarProvider>
            </ThemeProviderWrapper>
        </SessionProviderWrapper>
      </body>
    </html>
  );
}
```

Code 3.4: Shared Navbar in app/layout.tsx

State Management

The app uses React hooks and Context:

- `useState` and `useEffect` for local state
- Theme state is managed with `ThemeProviderWrapper` for dark mode.

Authentication Handling

Authentication is handled with NextAuth.js:

- `useSession` hook checks user login state.
- Protected routes secure user-specific pages.
- The interface displays a spinner (shown in 3.11) while verifying sessions, sending unauthenticated users to `/auth/signin`.

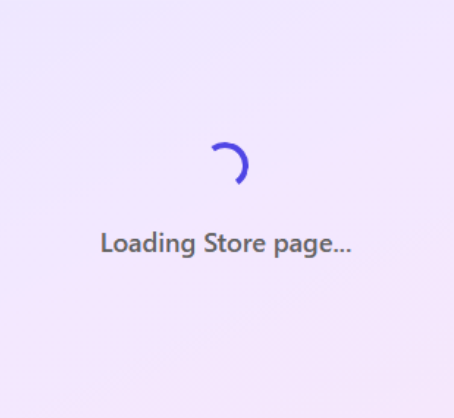


Figure 3.11: Loading Spinner

User Interface (UI) and User Experience (UX) Enhancements

The app's UI is responsive and engaging:

- Tailwind CSS uses card-based layouts for content and uses CSS specifications [50].
- Framer Motion adds animations.
- Loading spinners and toasts (`react-hot-toast` [51]) provide feedback.
- Dark mode is supported via `ThemeProviderWrapper`. See the Figure 3.12.

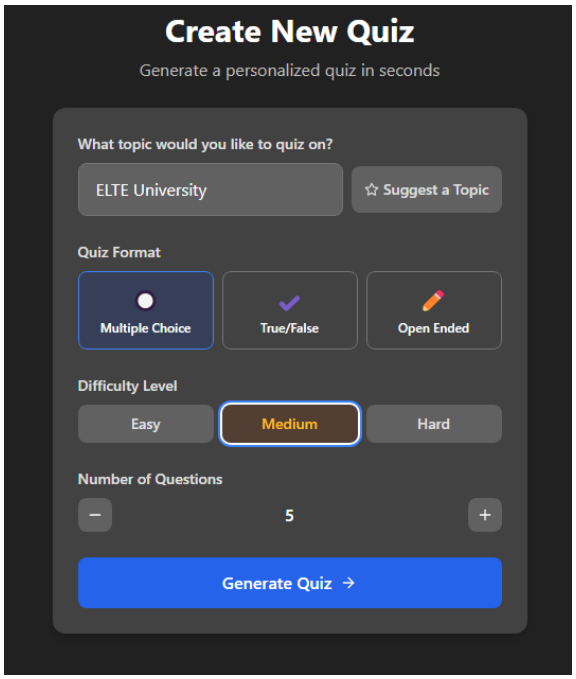


Figure 3.12: Quiz Generator Dark Mode

3.2.4 Database

This section covers the database configuration for the learning platform.

Schema Overview

The database schema, specified in `schema.prisma`, runs on PostgreSQL and employs Prisma's `relationMode = "prisma"` for virtual foreign keys optimized for cloud environments. Key generators include:

- `prisma-client-js`: Generates the Prisma client for TypeScript. [52]

- `prisma-dbml-generator`: Creates a DBML (Database Markup Language) file for schema visualization. [53]

The schema encompasses models for users, quizzes, quiz attempts, and progress records, with relationships designed for efficiency. Below, a code snippet 3.5 illustrates the `User` model along with its relations.

```
model User {
  id                String             @id @default(cuid())
  name              String?
  email             String?            @unique
  username          String?            @unique
  emailVerified     DateTime?
  password          String?
  image             String?
  avatar            String?
  title             String?
  accounts          Account[]
  sessions          Session[]
  Quiz              Quiz[]
  QuizAttempt       QuizAttempt[]
  UserAnswer        UserAnswer[]
  UserProgress      UserProgress?
  DailyAttempt      DailyAttempt[]
  UserItems         UserItem[]
  UserAchievements  UserAchievement[]
  FlashcardSet      FlashcardSet[]
  Article           Article[]

  @@index([username])
}
```

Code 3.5: User Model in schema.prisma

Relationships and Constraints

Relationships are designed for data integrity and performance:

- `onDelete: Cascade`: Ensures dependent records (e.g., user items) are deleted when a user or store item is removed, preventing orphaned data.

- Unique constraints: `@@unique([quizAttemptId, questionId])` in the `UserAnswer` model prevents duplicate answers for the same question within a quiz attempt.
- JSON fields: Used for flexible data (e.g., `StoreItem.effect` and `DailyQuestion.pairs`), balancing structure and adaptability.

The `UserItem` model tracks user-owned items from the store. See the Code 3.6.

```
model UserItem {
id          String    @id @default(cuid())
userId      String
user        User      @relation(fields: [userId], references: [id],
    onDelete: Cascade)
itemId      String
item        StoreItem @relation(fields: [itemId], references: [id],
    onDelete: Cascade)
purchasedAt DateTime  @default(now())
expiresAt   DateTime?
isActive    Boolean   @default(true)

@@index([userId])
@@index([itemId])
}
```

Code 3.6: UserItem Model in schema.prisma

Indexes on `userId` and `itemId` accelerate queries for user-specific inventory searches, while cascading deletes preserve data integrity.

Key Database Operations

Prisma enables efficient querying. One key operation involves fetching a random quiz topic suggestion to help users choose a quiz topic as demonstrated in Code 3.7.

```
const topics = await prisma.quiz.findMany({
select: { topic: true },
distinct: ['topic'],
});

const shuffledTopics = topics.sort(() => Math.random() - 0.5);
```

```
7  const result = shuffledTopics.slice(0, 1);
8
9  if (!result || result.length === 0) {
10 return NextResponse.json(
11 { error: "No topics available" },
12 { status: 404 }
13 );
14 }
```

Code 3.7: Topic Suggestion Query in app/api/quiz/topic-suggestion/route.ts

The query applies Prisma's `findMany` with `distinct` to retrieve unique quiz topics and pulls only the `topic` field to optimize performance. After retrieval, a random shuffle reorders the topics, and the first one serves as a suggestion, so users can explore new quiz content with minimal database overhead.

Database Visualization

Figure 3.7 shows an Entity-Relationship Diagram (ERD) that outlines the schema's structure and reveals connections among models.

3.2.5 AI Integration

Prompt Architecture & Cognitive Engineering

The prompt engineering[54] approach combines structured frameworks with dynamic constraints. Using Bloom's Taxonomy [55] ensures cognitive progression, while Depth of Knowledge criteria maintain appropriate challenge levels. Interdisciplinary synthesis is enforced through explicit matrix combinations. See Figure 3.13

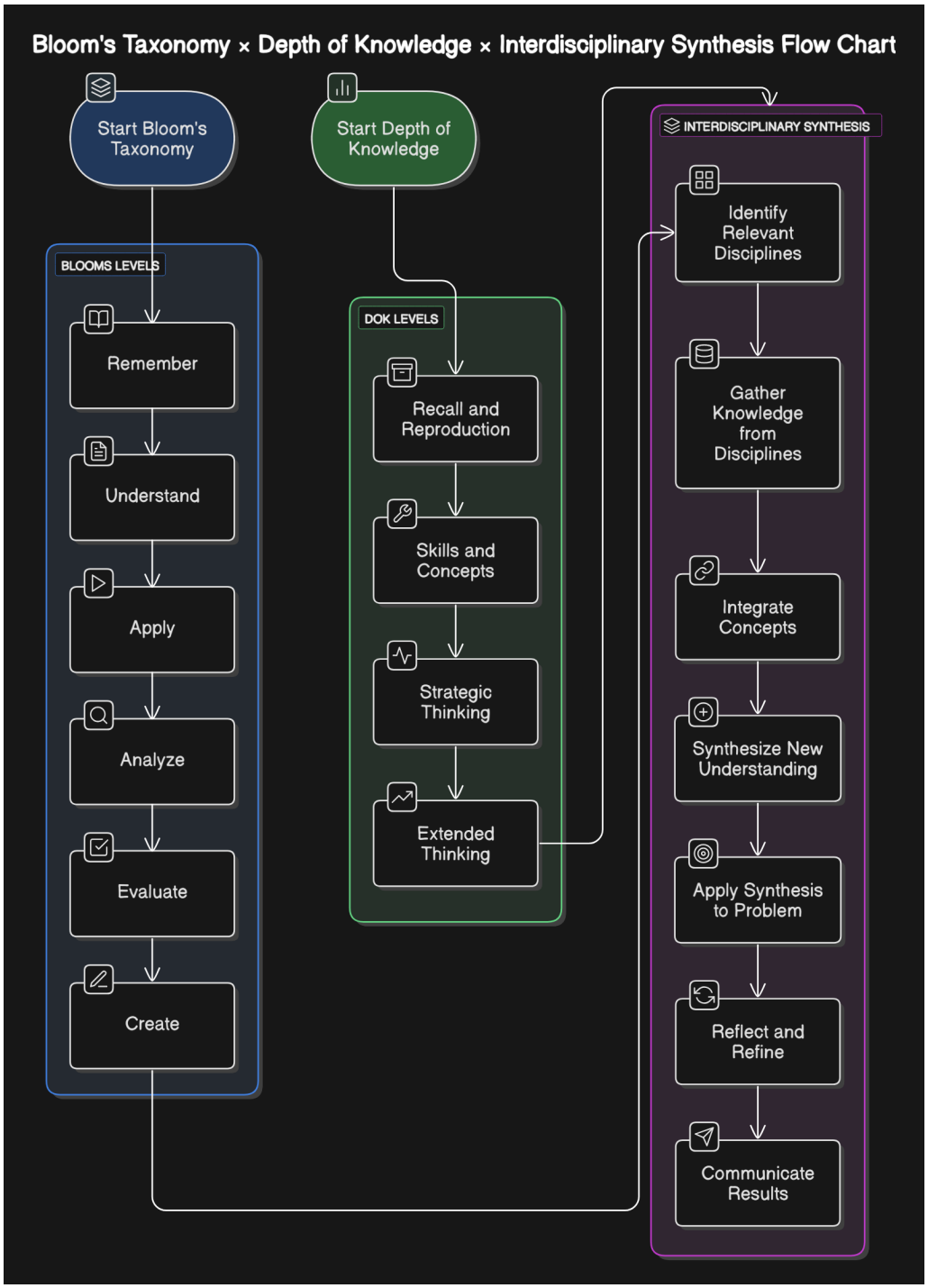


Figure 3.13: Quiz Generation Logic generated with Eraser [26]

```
### COGNITIVE FRAMEWORK (Bloom's Taxonomy x Depth of Knowledge x
    Interdisciplinary Synthesis)
2. **Understand/Apply (Medium)**:
   - Focus: Contextual comprehension or practical application of
     concepts across related subtopics.
   - Cognitive Demand: Basic analysis or transfer of knowledge to
     familiar scenarios.
```

```
5     - Interdisciplinary Angle: Link two disciplines (e.g., history
          and economics for Azerbaijan's oil boom).
6     - Example: "Explain how Azerbaijan's oil exports influence its
          economic policies."
```

Code 3.8: Cognitive Framework Excerpt

Question Design Matrix

```
1 ### QUESTION DESIGN MATRIX (Must cover at least 5 of 10 dimensions,
      ensuring maximal diversity)
2 1. **Conceptual**: Abstract theories, models, or frameworks (e.g.,
    economic diversification models for Azerbaijan).
3 2. **Factual**: Concrete data, historical events, or verifiable
    knowledge (e.g., key dates in Azerbaijan's history).
4 3. **Procedural**: Processes, methodologies, or operational
    techniques (e.g., oil extraction processes in the Caspian Sea).
5 4. **Metacognitive**: Self-reflection, critical thinking strategies
    , or learning processes (e.g., evaluating one's understanding of
     Azerbaijan's cultural identity).
6 5. **Applied**: Real-world applications or practical implications (
    e.g., Azerbaijan's role in global energy markets).
7 6. **Comparative**: Cross-context or cross-disciplinary comparisons
     (e.g., comparing Azerbaijan's and Norway's oil economies).
8 7. **Perspectival**: Diverse viewpoints, stakeholder perspectives,
    or ethical debates (e.g., environmentalist vs. industrialist
    views on Azerbaijan's oil industry).
9 8. **Speculative**: Hypothetical scenarios, future projections, or
    "what-if" analyses (e.g., Azerbaijan's economy in a post-oil
    world).
10 9. **Historical**: Temporal evolution or historical causation (e.g
    ., impact of Soviet-era policies on modern Azerbaijan).
11 10. **Emergent**: Cutting-edge trends, innovations, or disruptive
    changes (e.g., Azerbaijan's adoption of renewable energy
    technologies).
```

Code 3.9: Question Design Dimensions

Difficulty Level Criteria

```
#### Medium
- Focus: Contextual understanding or application of concepts, requiring basic connections (e.g., Azerbaijan's historical events, cultural influences).
- Cognitive Alignment: Understand/Apply, blending Factual, Procedural, or Applied dimensions.
- MCQ: Options demand moderate knowledge; distractors include nuanced errors or related concepts (e.g., mistaking a cultural practice for a neighboring country).
- TRUE_FALSE: Statements require contextual knowledge or minor nuances (e.g., specific economic trends).
- OPEN_ENDED: Brief explanations (1-2 sentences) with basic reasoning.
- Example (TRUE_FALSE): {"text": "Azerbaijan's economy heavily relies on oil exports.", "type": "TRUE_FALSE", "correctAnswer": "True", "sampleAnswer": "Azerbaijan's economy is significantly driven by oil and gas exports, particularly from the Caspian Sea region."}
```

Code 3.10: Difficulty Level Examples

Question Design Rules

```
### QUESTION DESIGN RULES
- **Maximal Diversity**: Questions MUST span distinct subtopics within "${topic}" (e.g., for Azerbaijan: geography, history, culture, economy, politics, environment, international relations, technology). Use a combinatorial approach to pair subtopics with matrix dimensions (e.g., culture + speculative, economy + comparative). Avoid any overlap in facts, concepts, or subtopics across questions or difficulty levels.
- **Dynamic Difficulty Progression**: Align questions with the specified difficulty (${difficulty}) using the cognitive framework. Hard questions MUST demand deeper reasoning, cross-disciplinary synthesis, or novel insights, explicitly avoiding rephrased lower-difficulty content. At least one Hard question MUST involve a speculative, evaluative, or creative task.
- **Perspective Rotation**: Systematically cycle through at least 5 matrix dimensions and 2 cognitive levels across questions to
```

```
   prevent thematic clustering. Use a randomized selection
   algorithm (if applicable) to ensure unpredictability in
   perspective combinations.
- **MCQ Specifics**: Include 4 options: 1 correct, 1 common
   misconception, 1 partial truth, 1 plausible distractor. Options
   MUST reflect the question's matrix dimension and cognitive level
   , ensuring plausibility and relevance. For Hard, distractors
   MUST require fine-grained knowledge or reasoning to dismiss.
- **Question Stem Variation**: Employ diverse stems to engage
   different cognitive processes (e.g., "Assess," "Justify," "
   Predict," "Synthesize," "Critique"). At least one stem MUST
   provoke lateral thinking (e.g., "What if Azerbaijan never
   discovered oil?").
- **Novelty Requirement**: At least two questions MUST adopt an
   unexpected angle, leveraging emergent trends, interdisciplinary
   connections, or counterintuitive perspectives (e.g., Azerbaijan'
   s potential in AI-driven agriculture).
- **Uniqueness Enforcement**: Each question MUST test a unique
   combination of subtopic, matrix dimension, and cognitive level.
   Implement a checkpoint to verify no two questions share the same
    factual base, conceptual focus, or analytical approach.
```

Code 3.11: Question Design Rules

Structured Output Priming

The JSON [56] schema acts as the final engineering boundary, so the output remains machine-readable and retains all planned cognitive and interdisciplinary traits in the generated content as demonstrated in Code 3.12.

```
### FINAL INSTRUCTION
- Return **only** a valid JSON array with exactly ${amount}
   questions
- Example output:
[
  {
    "text": "Question text",
    "type": "${type}",
    "options": ["...", "..."],  // Only for MCQ
    "correctAnswer": "...",     // For MCQ/TRUE_FALSE
    "sampleAnswer": "..."      // Required for all types
```

```
  }
]
```

Code 3.12: Strict JSON Output Requirement

For grading open-ended questions, `app/api/quiz/submit/route.ts` uses AI with a strict grading prompt. You can see the prompt in Code 3.13.

```
const prompt = `
You are a professional quiz grader tasked with evaluating open-ended student responses with precision and academic rigor. Your goal is to assign a granular score between 0.00 and 1.00 (to two decimal places, e.g., 0.53) based on factual accuracy, reasoning quality, completeness, and alignment with the question's difficulty. Follow these rules strictly to ensure consistency and fairness.
## Grading Philosophy
- **Accuracy Is First**: Incorrect facts lead to significant deductions, scaled by their severity and the question's difficulty.
- **Reasoning Is Critical**: Logical coherence and depth are required, especially for medium and hard questions.
- **Completeness Matters**: Answers must address all parts of the question, with deductions for omissions.
- **Difficulty Scaling**:
  - **Easy**: Focus on exact recall of basic facts.
  - **Medium**: Require correct facts plus concise reasoning.
  - **Hard**: Demand depth, multi-step logic, and nuanced insight.
- **Math/Science Responses**:
  - Accept equivalent forms (e.g., 0.25 = 1/4 = 25%).
  - Deduct -0.10 for incorrect or missing units unless trivially implied.
- **Spelling/Grammar**:
  - Ignore up to 2 typos if meaning is clear.
  - Deduct -0.05 per additional typo, up to -0.20 total.

## Scoring Methodology
1. **Start at 1.00** and apply deductions based on the following:
   - **Factual Inaccuracies**:
     - Major error (e.g., wrong core concept): -0.20 to -0.30 per error, depending on severity.
```

```
22      - Minor error (e.g., incorrect detail): -0.10 to -0.15 per
            error.
23      - Maximum deduction for inaccuracies: -0.50.
24    - **Reasoning Issues**:
25      - Weak or illogical reasoning: -0.10 to -0.20, depending on
            extent.
26      - Missing reasoning (for medium/hard): -0.15 to -0.25.
27    - **Completeness**:
28      - Missing a key component: -0.10 to -0.20 per omission.
29      - Overly brief (e.g., <2 sentences for hard): -0.10 to -0.15.
30    - **Irrelevance**:
31      - Partially relevant but off-topic: -0.30 to -0.50.
32      - Completely irrelevant: Score = 0.00.
33    - **Overly Verbose**:
34      - Unnecessary elaboration without adding value: -0.05 to
            -0.10.
35  2. **Difficulty-Specific Adjustments**:
36    - **Easy**:
37      - Expect 1-5 word answers matching a known fact.
38      - Exact match or synonym: 1.00.
39      - Any error or vagueness: 0.00 (no partial credit).
40    - **Medium**:
41      - Expect 1-2 sentences with a key fact + explanation.
42      - Fully correct + logical: 0.90-1.00.
43      - Core idea present but flawed: 0.60-0.89.
44      - Major errors or weak reasoning: 0.30-0.59.
45    - **Hard**:
46      - Expect 2-4+ sentences with layered reasoning or analysis.
47      - Exceptional depth and precision: 0.90-1.00.
48      - Strong but missing nuance: 0.70-0.89.
49      - Partial relevance or errors: 0.30-0.69.
50  3. **Calibration**:
51    - Compare the response to the ideal answer (sampleAnswer) to
          gauge completeness and accuracy.
52    - Normalize scores to avoid over- or under-scoring by
          considering the topic's complexity and typical student
          performance (e.g., a hard question on quantum physics should
          score similarly to one on geopolitics if errors are
          equivalent).
53    - If the calculated score feels misaligned (e.g., too high for a
          flawed answer), adjust by up to +-0.05 to reflect fairness.
```

Code 3.13: Core Structure of Quiz Grading Prompt

Daily questions and articles are generated with a similar but less complicated approach.

3.2.6 Technical Implementation of AI Components

Question Generation Temperature Settings

The temperature [57] parameter governs the degree of randomness in AI-generated content. For quiz question generation, we use a moderate temperature of 0.7 to achieve an optimal balance between creative variation and structural consistency. This configuration produces questions with diverse phrasing and approaches while maintaining adherence to the specified content requirements. Check Code 3.14.

```
const response = await openai.chat.completions.create({
model: "deepseek-chat",
messages: [{ role: "user", content: prompt }],
temperature: 0.7,
});
```

Code 3.14: Temperature Configuration for Question Generation

Temperature Effects:

- **Lower values (e.g., 0.2-0.5)**: More predictable, repetitive questions with minimal diversity
- **Higher values (e.g., 0.8-1.2)**: Produce more creative but potentially less reliable questions with possible deviations from requirements

Answer Evaluation Temperature Settings

For response grading, we implement a temperature of 0.1 to maximize scoring consistency. This setting ensures minimal fluctuation in evaluation outcomes and provides reliable assessment of open-ended answers according to predefined criterias. Check Code 3.15.

```
const response = await openai.chat.completions.create({
model: "deepseek-chat",
messages: [{ role: "user", content: prompt }],
```

```
4 temperature: 0.1,
5 });
```

Code 3.15: Temperature Configuration for Answer Evaluation

Temperature Effects:

- **Lower values (approaching 0)**: Same evaluations for similar responses, which maximizes grading consistency
- **Higher values (e.g., >0.3)**: Scoring variability where equivalent answers might receive different grades

Token Usage and Constraints for Quiz Operations

Quiz operations involve generation (`app/api/quiz/generate/route.ts`) and submission (`app/api/quiz/submit/route.ts`) using the DeepSeek API (`deepseek-chat`). Token usage—text units (words, punctuation) processed by the model—affects cost and performance.

Quiz Generation: The generation prompt creates 1–20 questions, using ∼2,500–2,550 input tokens (text units sent to the API; fixed prompt: ∼2,450–2,500 tokens, variables: ∼20–30 tokens for `topic`, `difficulty`, `type`, `amount`). Output (JSON array) varies by question type: MCQ (∼100–200 tokens each), TRUE_FALSE (∼40–80), OPEN_ENDED (∼100–250), totaling ∼800–5,000 tokens for 20 questions. With `max_tokens: 5000` (maximum output tokens allowed), the total (∼3,300–7,550 tokens) fits within the 64K-token context window (maximum tokens, input plus output, per call; 64,000 tokens).

Quiz Submission: Grading OPEN_ENDED questions uses ∼1,500–1,900 input tokens per call (fixed: ∼1,200, variables: ∼300–700). Output (JSON score) is ∼4–5 output tokens (text units generated by the API) per question, totaling ∼100 tokens for 20 questions across 20 calls. With `max_tokens: 4000`, total usage is ∼30,100–38,100 tokens, each call fitting the 64K context window.

Character Limit: OPEN_ENDED answers are capped at 800 characters (∼180–200 tokens, limiting input length) to manage input size. Additionally, Quiz Generation topic is capped at 70 characters.

Max Tokens Configuration: Generation uses `max_tokens: 5000`, submission uses `max_tokens: 500`, but actual outputs are smaller (generation: ∼800–5,000, submission: ∼5 per call). Reducing to `max_tokens: 100` for submission is advised.

Context Window: The 64K-token context window of `deepseek-chat` exceeds per-call usage (generation: ∼3,300–7,550, submission: ∼1,504–1,905 tokens), ensuring no truncation. Each submission call is independent.

Context Window Note

Input tokens: The number of text units (words, punctuation, etc.) sent to the API, including the system prompt and user message.

Output tokens: The number of text units generated by the API in response.

Max tokens: The maximum number of output tokens the API is allowed to generate per request.

Context window: The combined limit of input and output tokens a single API call can handle. For `deepseek-chat`, this is 64,000 tokens.

Variables: The dynamic parts of the input, like user-selected `topic`, `difficulty`, `type`, and `amount`.

JSON array: The structured data format returned by the API, typically containing the generated quiz questions or scores.

Shortcomings and Potential Improvements

Quiz generation, with one API call (∼3,300–7,550 tokens), could optimize the prompt by trimming redundant instructions, potentially saving ∼200–300 input tokens. Quiz submission makes 20 API calls for 20 OPEN_ENDED questions, sending the fixed prompt (∼1,200 tokens) each time, totaling ∼30,100–38,100 tokens, which increases costs and latency. Batching all questions into one call could reduce usage to ∼7,300–14,500 tokens, cutting redundancy and improving efficiency.

3.2.7 CRON Jobs

QuizGG uses `node-cron` to automate recurring tasks for efficient system maintenance and content updates. The store cleanup job in `scripts/cron.ts` removes expired `UserItem` records. See CRON job in Code 3.16.

```
1 import cron from "node-cron";
2 import axios from "axios";
3
4 // run every 10 seconds
5 cron.schedule("*/10 * * * * *", async () => {
```

```
try {
await axios.post("http://localhost:3000/api/store/cleanup");
console.log("Expired items cleaned up successfully");
} catch (error) {
console.error("Failed to run cleanup cron job:", error);
}
});
```

Code 3.16: Store Cleanup CRON Job in scripts/cron.ts

This job runs every 10 seconds (for testing), invoking `/api/store/cleanup`, which uses Prisma's `deleteMany` to remove `UserItem` records where `expiresAt` is past the current time, and keeps the user inventory up-to-date.

As shown in Code 3.17, store cleanup route in `app/api/store/cleanup/route.ts` handles the deletion.

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/db";

export async function POST(req: Request) {
try {
const now = new Date();
await prisma.userItem.deleteMany({
where: { expiresAt: { lte: now } },
});
return NextResponse.json({ success: true, message: "Expired items
    cleaned up" });
} catch (error) {
console.error("Failed to clean up expired items:", error);
return NextResponse.json({ error: "Failed to clean up expired items
    " }, { status: 500 });
}
}
```

Code 3.17: Store Cleanup Route in app/api/store/cleanup/route.ts

Daily/Weekly question generation in `scripts/cronForDailyQuestion.ts` schedules new question creation with environment-specific logic. See Code 3.18.

```
const nextRun = DateTime.now()
.setZone(CONFIG.PROD.TIMEZONE)
.plus({ days: 1, seconds: 30 })
.set(CONFIG.PROD.RUN_AT)
```

```
5 .toUTC();
6
7 const cronExpression = ${nextRun.second} ${nextRun.minute} ${
    nextRun.hour} * * *;
8
9 cron.schedule(cronExpression, async () => {
10 if (process.env.NODE_ENV === "production") {
11 logger.info(${CONFIG.PROD.LOG_PREFIX} Midnight generation starting
    ...);
12 await generateDailyQuestion();
13 }
14 });
```

Code 3.18: Daily Question CRON Job in scripts/cronForDailyQuestion.ts

This job runs at midnight in Budapest time (`Europe/Budapest`) in production, or 35 seconds after startup in test mode. It calls `/api/daily-question/generate` to create 8 word-definition pairs for a daily challenge, ensuring fresh daily content.

3.2.8 Implementation Shortcomings, Areas for Improvement, and Opportunities for Future Work

QuizGG has a solid foundation but requires improvements to enhance security, usability, and engagement:

- **Email Verification Not Implemented**: The `User` model includes an `emailVerified` field, but lacks functionality to send verification emails or enforce verification before accessing features like quizzes. This could allow unverified accounts (Security risks)
- **Password Reset Functionality Missing**: There's no mechanism for users to reset forgotten passwords, despite the `User` model's `password` field.
- **Multiplayer Realtime Matchmaking Not Supported**: Realtime multiplayer quiz challenges are missing. Adding this (e.g., via WebSockets in `app/quiz/matchmaking`) would foster competitive learning and boost user engagement.

- **Lack of Rate Limiting on API Endpoints**: Endpoints like `/api/quiz/submit` lack rate limiting, risking abuse (e.g., excessive submissions). This could overload the server and needs addressing for scalability.
- **CRON Job Scheduling Limitations**: The store cleanup job in `scripts/cron.ts` runs every 10 seconds, which is excessive for production and could strain server resources.
- **Future Opportunities for Gamification**: Expanding gamification with features like daily streaks for quiz completion could increase user motivation and retention.
- **Social Sharing Features Not Implemented**: The platform lacks functionality to share quiz results, achievements, or custom quizzes on social media platforms. Adding this would promote user interaction, foster community engagement, and attract new users through social visibility.

Addressing these will strengthen QuizGG's security, performance, and engagement.

3.3 Testing

This section describes the testing methods and tools used to validate QuizGG's functionality, performance, and reliability. Main testing methods:

- Manual testing for core features,
- API testing for backend endpoints,
- Database query testing for data operations,
- Network payload analysis for request/response validation,
- Performance testing for system efficiency.

Custom dashboards were created to perform API and database testing.

Access Note

To access the Database and API testing dashboards, register a test account with these credentials:

Username: `testuser` (username is optional)

Email: `testuser123@example.com` (email must match)

3.3.1 Manual Testing

Manual testing was conducted to validate core functionalities, security, and usability of QuizGG. Tests were performed by simulating user interactions and verifying expected outcomes. Below you can see key manual test cases, their steps, expected results, and references to images showing the test and result combined. Each image captures both the action taken and the outcome.

3.3.2 Manual Test Cases for QuizGG with Screenshots

Below are the manual test cases for QuizGG, each documented with relevant details and screenshots.

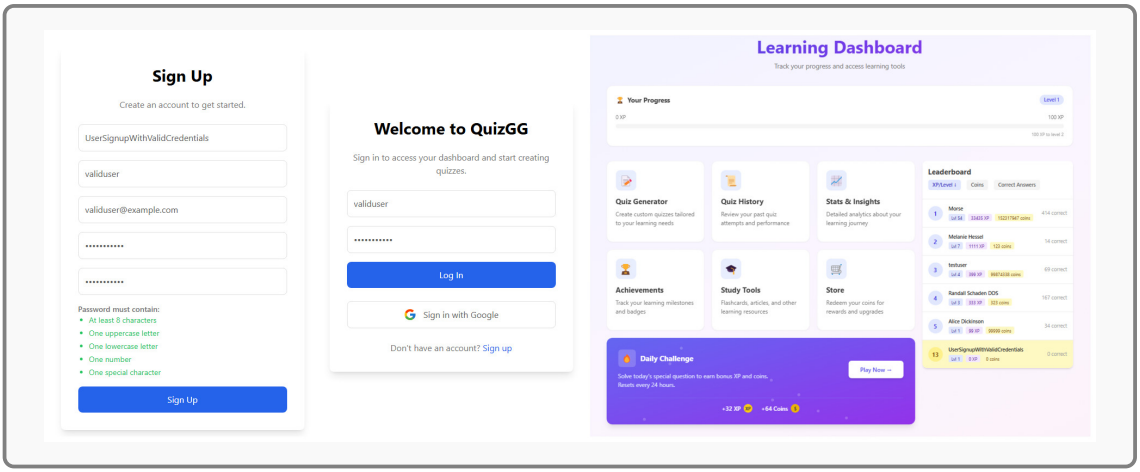
Test 1: Verify User Signup and Login Flow with Valid Credentials

Category: Functionality

Description: Ensure users can sign up with valid credentials, get redirected to login, and successfully log in.

Test Step: Navigate to the signup page, enter a valid name, email, username, and password, then click "Signup". Upon redirection, enter the same credentials on the login page and click "Login".

Expected Result: User is successfully signed up, redirected to the login page, logs in successfully, and is then redirected to the dashboard.



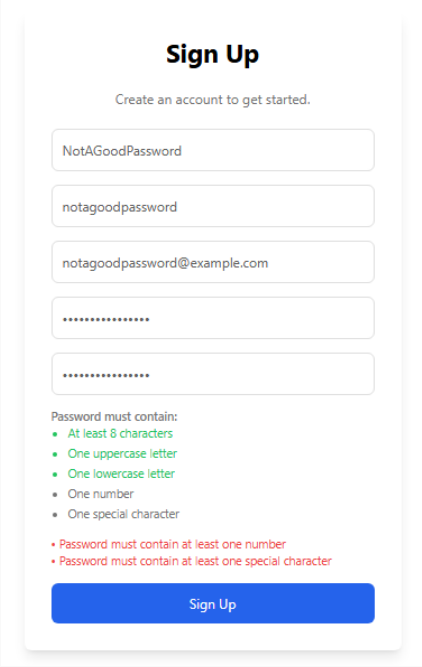
Test 2: Verify Password Security During Signup

Category: Security

Description: Ensure the signup process enforces password security rules.

Test Step: Attempt to sign up with a password shorter than 8 characters.

Expected Result: Signup fails with an error message indicating password requirements.



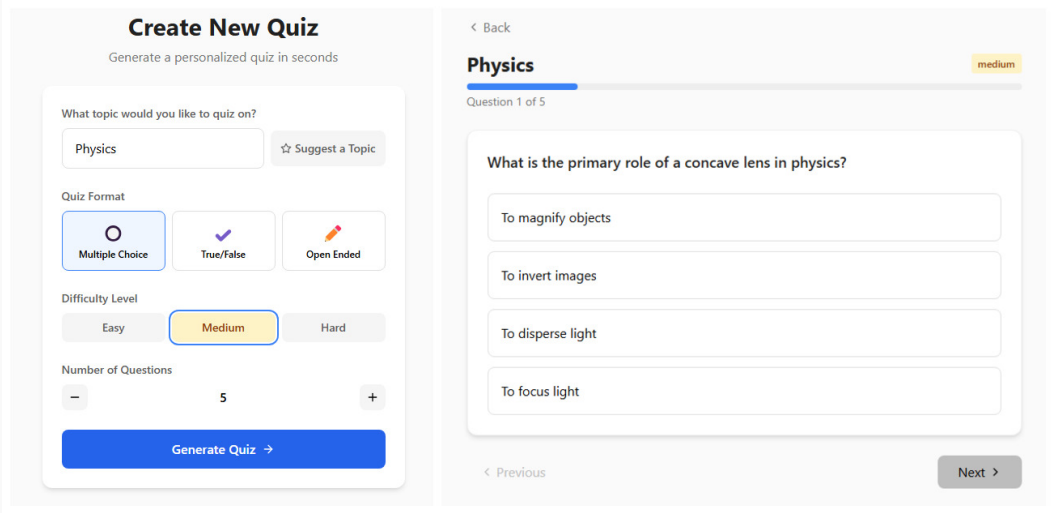
Test 3: Verify Quiz Generation with Valid Parameters

Category: Functionality

Description: Ensure quizzes can be generated with valid input parameters.

Test Step: On the quiz creation page, select a topic (e.g., "Physics"), difficulty (e.g., "Medium"), question type (e.g., "MCQ"), and amount (e.g., 5), then click "Generate Quiz".

Expected Result: A quiz with 5 medium-difficulty MCQ questions on Physics is generated.



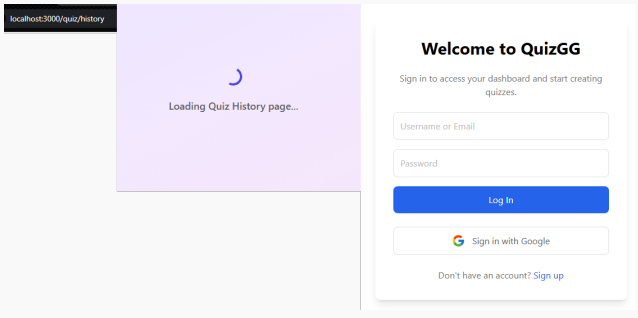
Test 4: Prevent Unauthorized Access to Protected Routes

Category: Security

Description: Verify that protected API routes are inaccessible without authentication.

Test Step: Log out, then attempt to access `/api/user/progress` directly.

Expected Result: Access is denied with a 401 error and redirect to login.



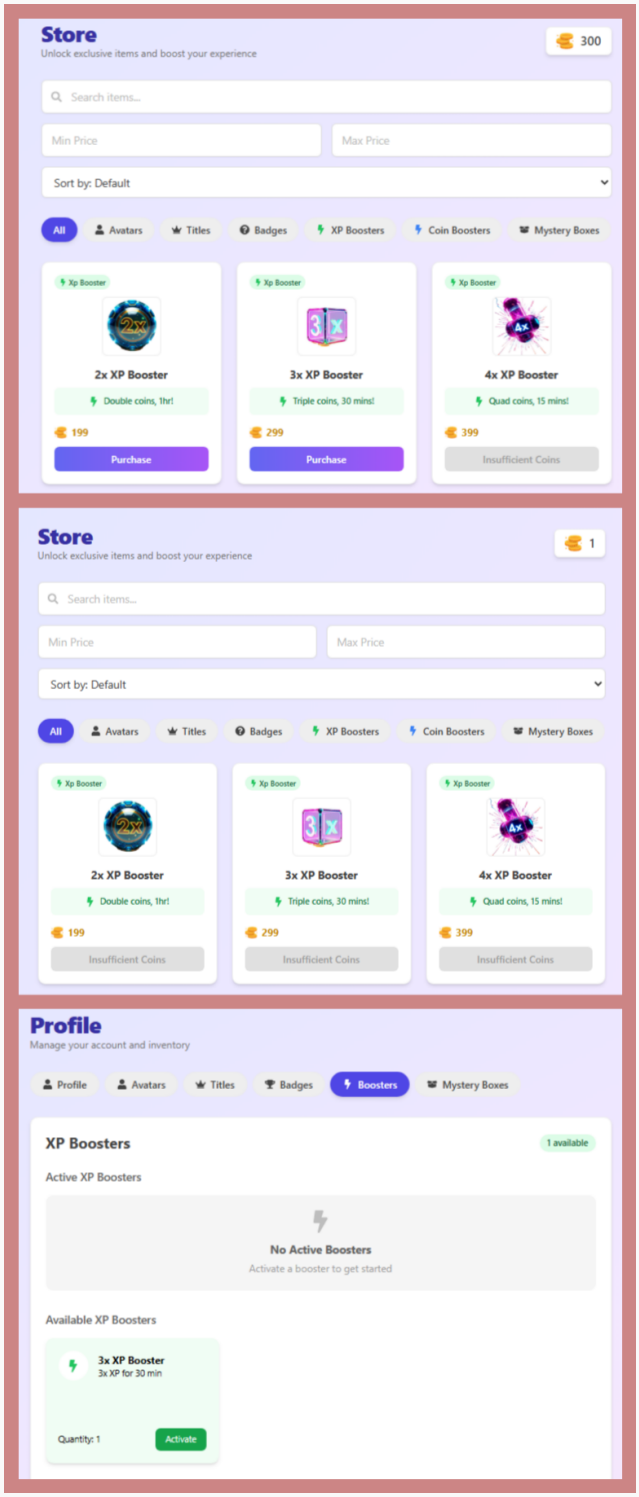
Test 5: Verify Store Item Purchase and Inventory Update

Category: Functionality

Description: Confirm that users can purchase items from the store.

Test Step: Select an item with sufficient coins and click "Purchase".

Expected Result: Item is purchased, coins deducted, and inventory updated.



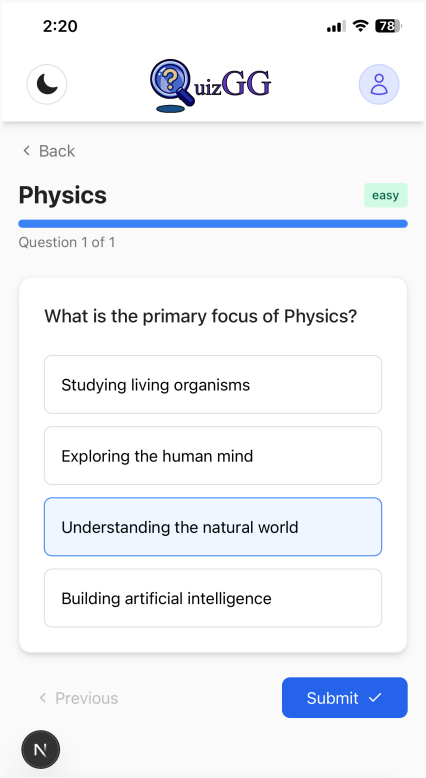
Test 6: Check Responsiveness of Quiz Interface on Mobile

Category: Usability

Description: Ensure the quiz interface is responsive on mobile devices.

Test Step: Open quiz page on mobile and interact with questions.

Expected Result: Interface adapts to mobile screen with all elements functional.



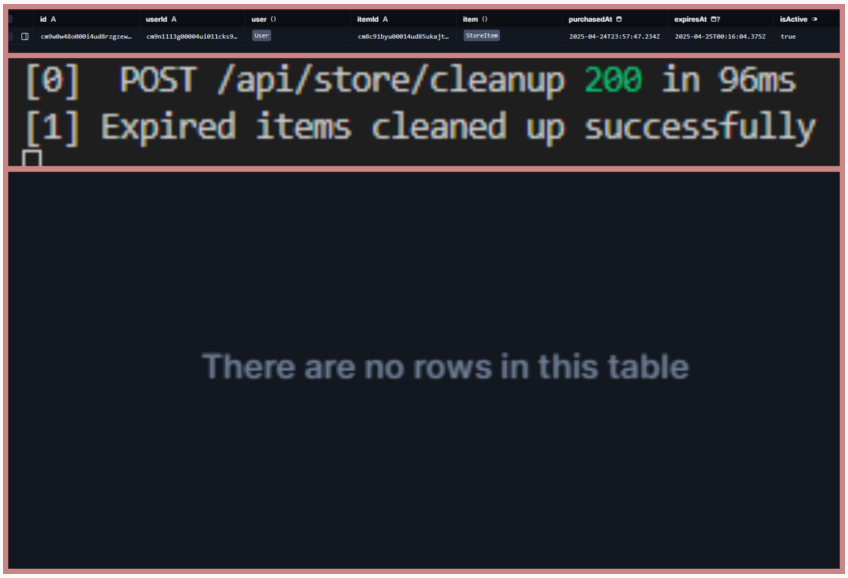
Test 7: Verify CRON Job for Store Cleanup

Category: Functionality

Description: Ensure CRON job removes expired items from inventories.

Test Step: Set `expiresAt` to past date and wait for CRON job.

Expected Result: Expired item is removed from inventory.



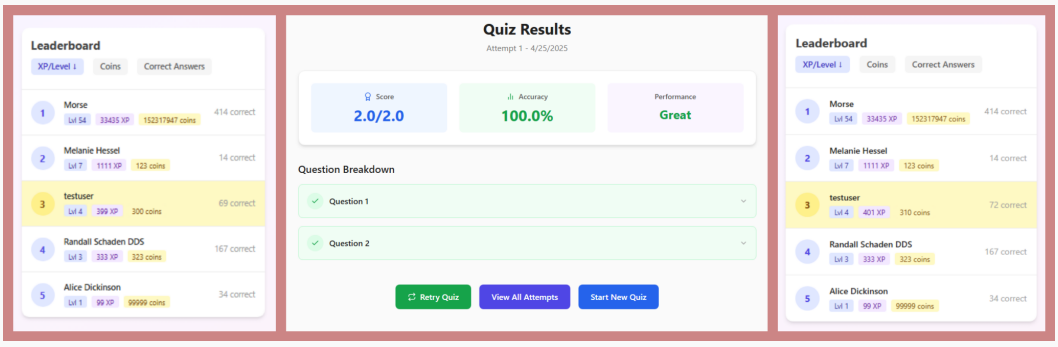
Test 8: Verify XP and Coin Updates After Quiz Completion

Category: Functionality

Description: Ensure XP and coins are correctly awarded after quiz.

Test Step: Complete a quiz and check the user's progress.

Expected Result: XP and coins are awarded based on results.



3.3.3 API Testing

API testing was conducted using a custom-built API Testing Dashboard, inspired by tools like Postman [58], to validate the functionality and performance of QuizGG's backend endpoints. The dashboard allowed for interactive testing of endpoints and provided insights into response times and success rates.

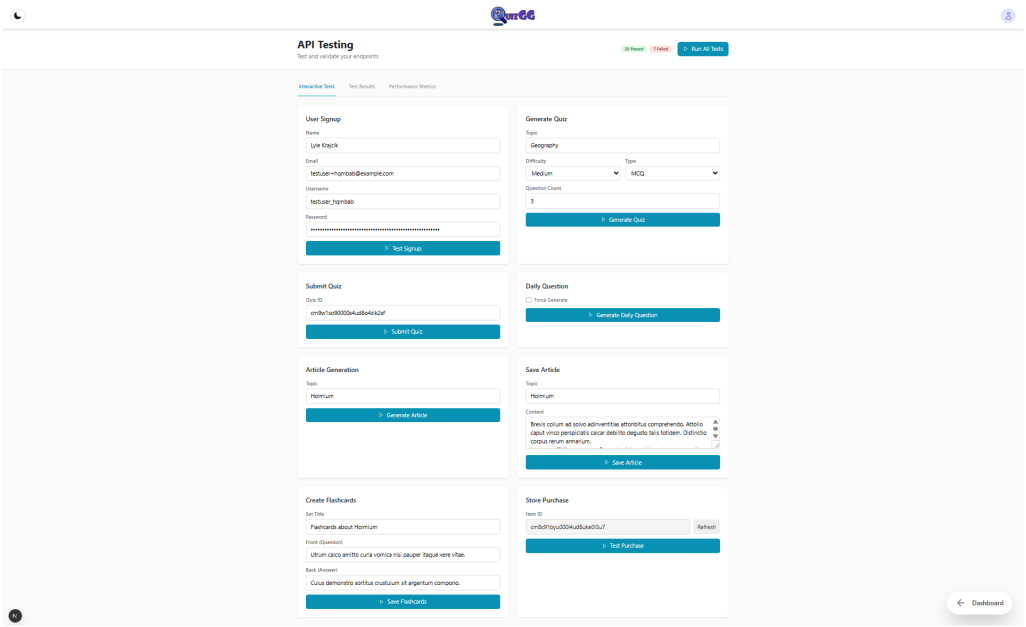


Figure 3.14: Interactive Tests Tab of the API Testing Dashboard

As demonstrated in Figure 3.14, Interactive Tests tab offers some API endpoint tests. "Run All Tests" button allows execution of all tests at once.

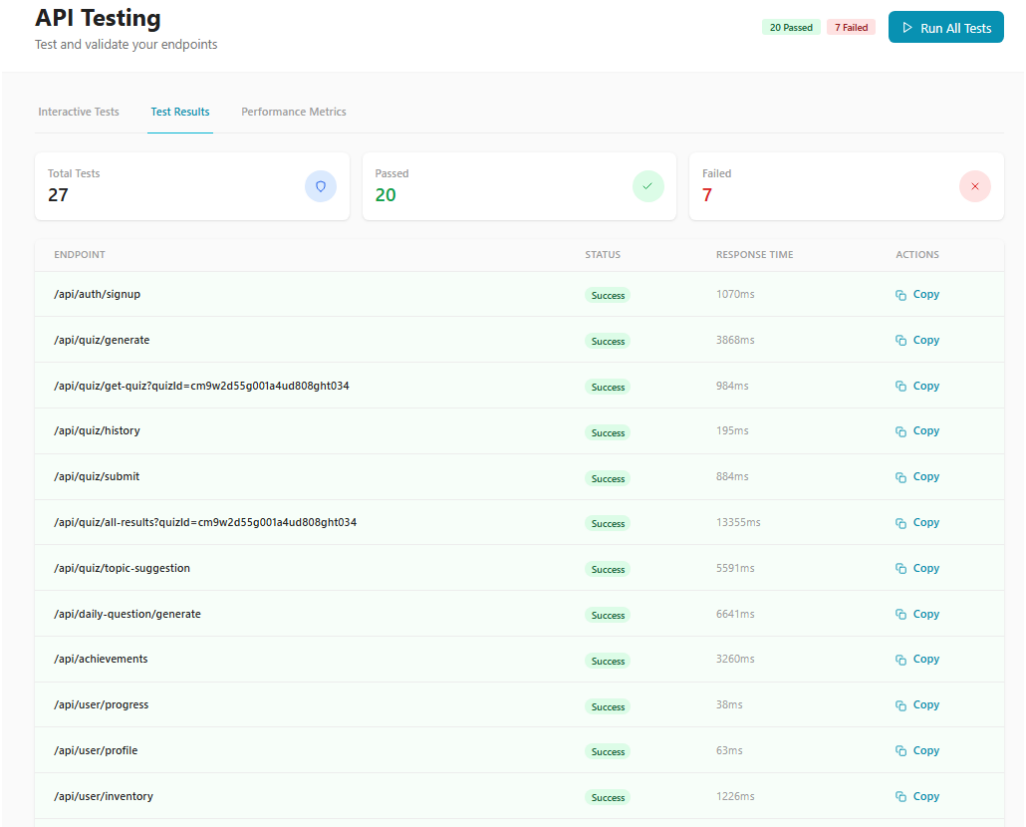


Figure 3.15: Test Results Tab of the API Testing Dashboard

The Test Results tab provides details on the 7 failed tests out of 27, with edge

cases. These failures shows validation errors, such as missing fields, and not-found issues. Check Figure 3.15.

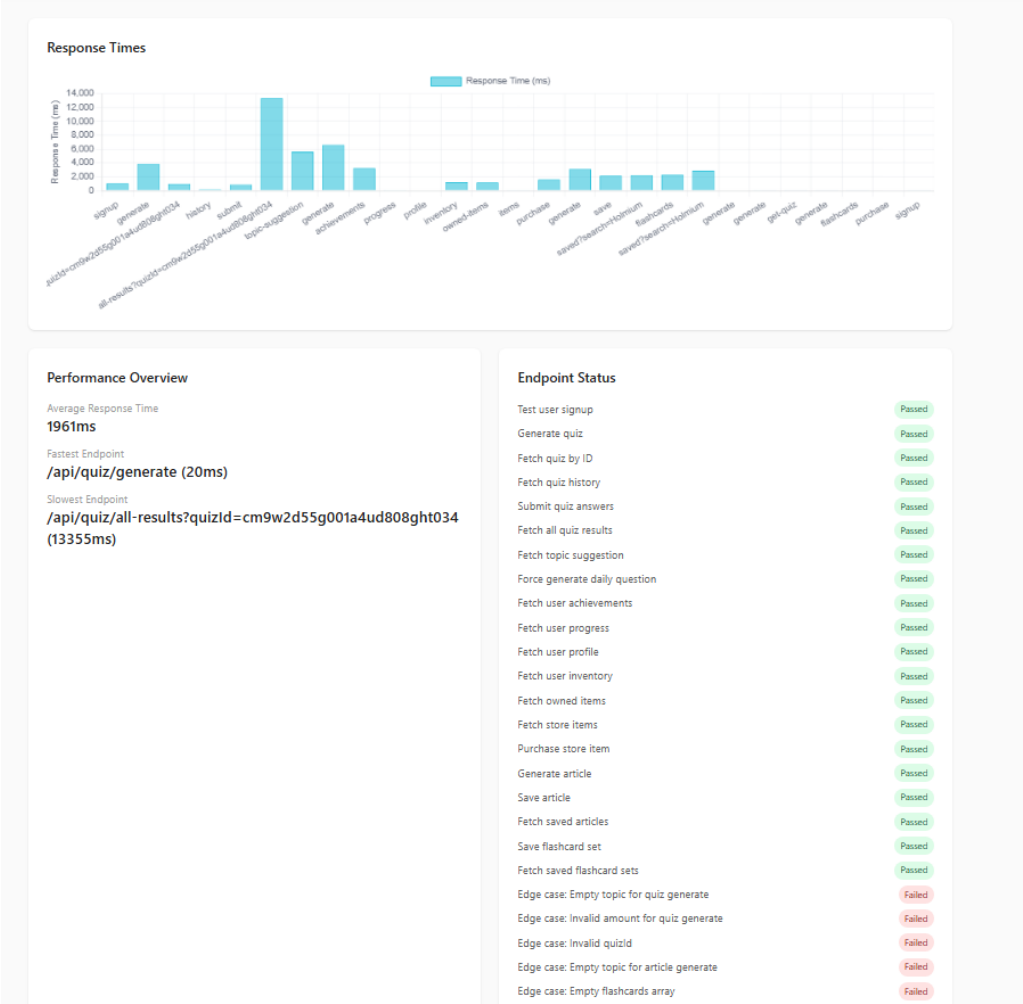


Figure 3.16: Performance Metrics Tab of the API Testing Dashboard

The Performance Metrics tab, shown in Figure 3.16, includes a histogram of response times. These figures highlighted areas for improvement, such as caching for endpoints with longer response times.

3.3.4 Database Query Testing

Database queries were tested with a custom Database Query Tester to check how well Prisma worked with the Supabase PostgreSQL database. This tool let us run queries, analyze the results, and measure key stats like execution speed, success rates, and the number of rows returned.

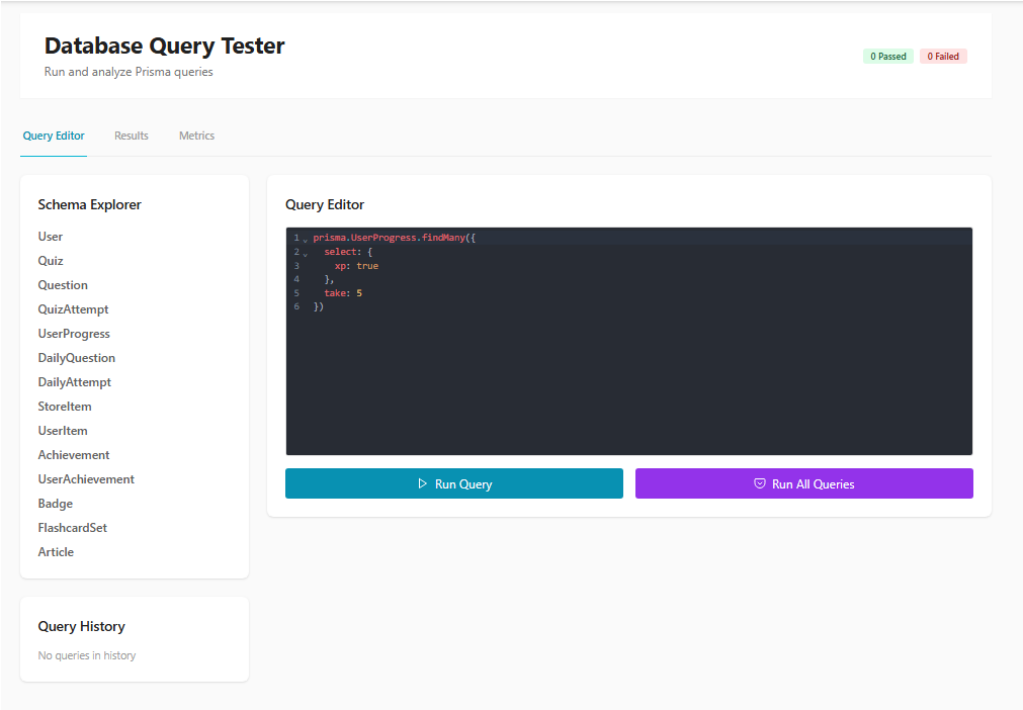


Figure 3.17: Query Editor Tab of the Database Query Tester

The Query Editor tab (see Figure 3.17) includes a schema explorer listing tables like `User`, `Quiz`, and `UserProgress`. A sample query, `prisma.userProgress.findMany()`, retrieves fields like `xp`. Queries execute via the "Run Query" button, while "Run All Queries" enables batch testing.

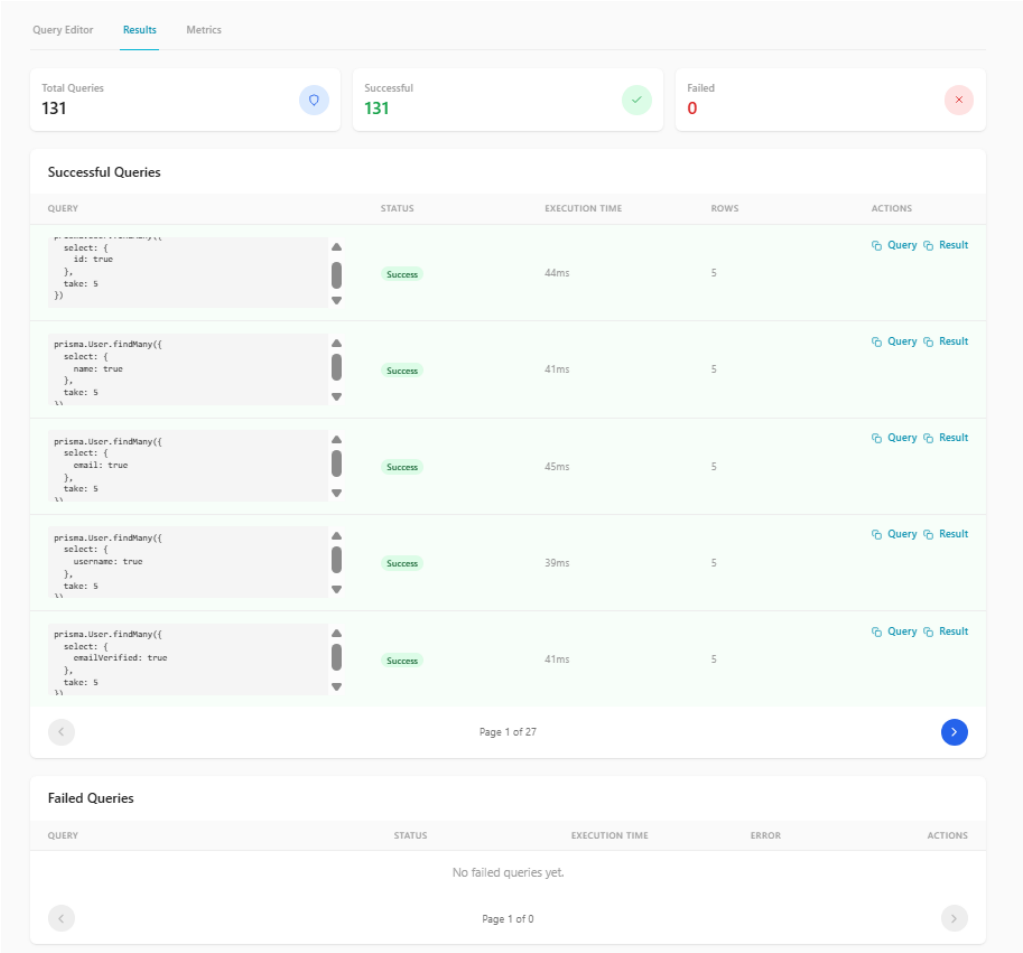


Figure 3.18: Results Tab of the Database Query Tester

The Results tab (see Figure 3.19) shows 131 successful queries. Each query returned 5 records. The execution times ranged from 30ms to 100ms. This confirms the database performs consistently. All test operations retrieved the required data from the specified tables.

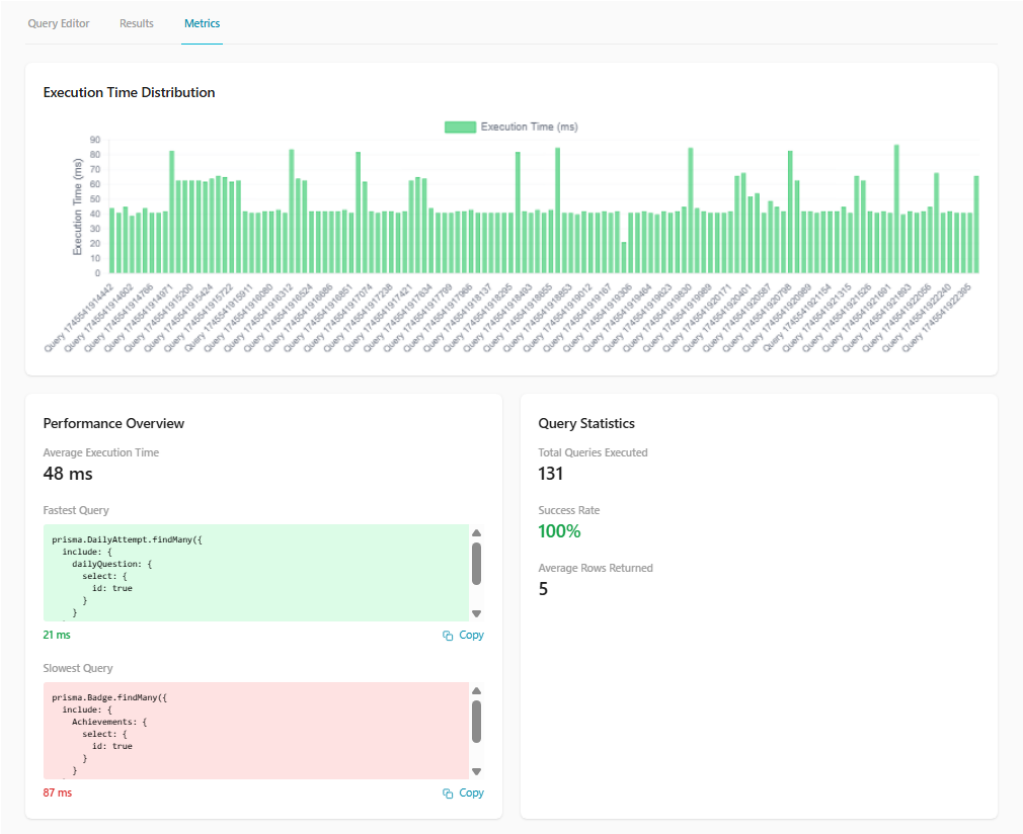


Figure 3.19: Metrics Tab of the Database Query Tester

The Metrics tab displays an execution time histogram, with most queries between 20ms and 80ms. The average execution time is 48ms, with the fastest query at 21ms (`prisma.DailyAttempt.findMany()`) and the slowest at 87ms (`prisma.Badge.findMany()`). These metrics helped optimize query performance by indexing frequently accessed tables.

3.3.5 Network Payload Analysis

Network payload analysis was conducted using Chrome DevTools [59] to inspect the request and response payloads of QuizGG's API calls. This helped validate the structure and content of data exchanged between the frontend and backend.

As shown in Figure 3.20, the request payload for `/api/quiz/generate` specifies the quiz parameters: a "Azerbaijan" topic, "easy" difficulty, and 2 questions, "MCQ" type. This aligns with QuizGG's quiz generation feature. And it responses with "quizID" parameter. See Figure 3.21

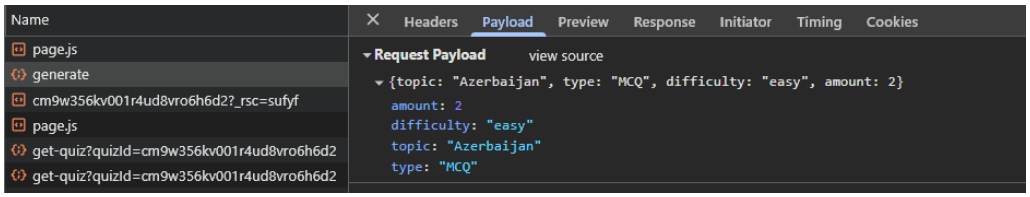


Figure 3.20: Payload for the `/api/quiz/generate` endpoint, captured from the network tab

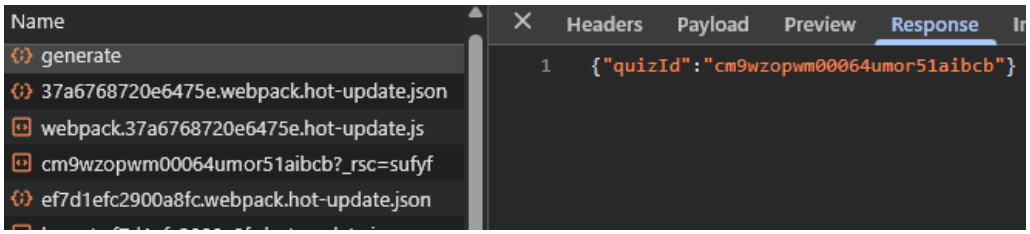


Figure 3.21: Response for /api/quiz/generate

The response from `/api/quiz/results` includes the quiz results with a `quizId`, a list of `questions` (e.g., "What is the capital of Azerbaijan?"), and the user's `score` (3/3). The frontend uses this data to display the quiz questions and results to the user. Check the details in Figure 3.22.

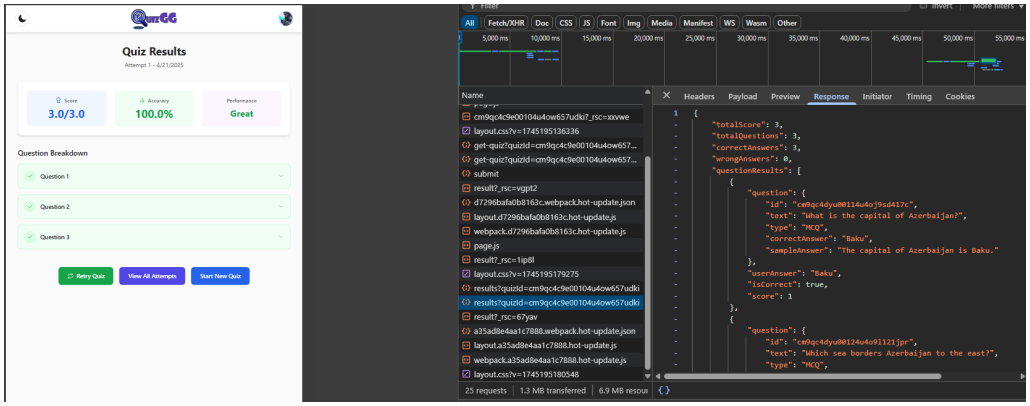


Figure 3.22: Response for /api/quiz/results

3.3.6 Performance Testing

Performance testing was conducted using data from the API Testing Dashboard and Database Query Tester to evaluate the efficiency of QuizGG's backend and database operations. Key metrics included API response times and query execution times. These identified areas for optimization such as caching for slower API endpoints and indexing for database queries.

3.3.7 Testing Results and Analysis

The testing process provided comprehensive insights into QuizGG's functionality, performance, and reliability. Below is a summary of the results from each testing phase:

- **Manual Testing**: Confirmed core functionalities like signup, quiz generation, navigation, and XP/coin updates work as expected.
- **API Testing**: 20/27 tests passed, 7 failed due to edge cases (e.g., invalid quizId, empty topic). Average response time of 1961ms, with performance metrics indicating areas for optimization.
- **Database Query Testing**: 131/131 queries succeeded, with an average execution time of 48ms and a 100% success rate, indicating great database performance.
- **Network Payload Analysis**: All payloads were correctly structured, with no data integrity issues. The `/api/quiz/generate` request and response validated proper quiz generation and submission.
- **Performance Testing**: Identified areas for optimization in API endpoints and database queries.

Chapter 4

Conclusion

QuizGG transforms online education by integrating gamification and artificial intelligence (AI) to overcome the obstacles of traditional learning platforms, most notably the lack of personalization, with the help of AI-driven features such as custom-generated quizzes, automatic open-ended response grading, and study materials such as flashcards and articles. These dynamically adapt to unique user needs. Gamified aspects such as progress monitoring, leaderboards, achievements, and a virtual shop motivate learners on the basis of rewards as well as competition.

The application's solid tech stack of Next.js, React, Prisma, and a cloud based relational database supports scalability, responsiveness, and secure data management. Its extensibility supports the addition of multiplayer matchmaking, improved authentication with email verification, and deeper analytics. Various tests verified the user flows, API functionality, and database operations. Future work includes social sharing features for collaboration and competition, the addition of a rate limiter for API calls, password resetting feature, as well as better CRON job scheduling for performance.

QuizGG exists as a potent educational tool and a platform for exploring AI-driven educational solutions. Its extensibility allows for further development, putting QuizGG on a path of rising interest by diverse learners.

Bibliography

[1] Microsoft Corporation. *Windows Dev Center Documentation*. Redmond, WA, USA, 2025. URL: https://docs.microsoft.com/en-us/windows/.

[2] Apple Inc. *macOS Developer Documentation*. Cupertino, CA, USA, 2025. URL: https://developer.apple.com/documentation/.

[3] The Linux Kernel Organization. *The Linux Kernel Documentation*. United States, 2025. URL: https://www.kernel.org/doc/html/latest/.

[4] Canonical Ltd. *Ubuntu Official Documentation*. London, UK, 2025. URL: https://help.ubuntu.com/.

[5] Node.js Foundation. *Node.js Documentation*. San Francisco, CA, USA, 2023. URL: https://nodejs.org/.

[6] The Chromium Projects. *Chromium Developer Documentation*. United States, 2025. URL: https://www.chromium.org/developers/.

[7] Supabase. *Supabase: Open-Source Firebase Alternative*. Singapore, 2023. URL: https://supabase.com/.

[8] GitHub, Inc. *GitHub: Platform for Version Control and Collaborative Development*. San Francisco, CA, USA, 2023. URL: https://github.com/.

[9] Wikipedia contributors. *ZIP (file format)*. 2025. URL: https://en.wikipedia.org/wiki/ZIP_(file_format).

[10] GitHub. *GitHub Documentation*. Open Source Community, 2025. URL: https://docs.github.com/en.

[11] npm, Inc. *npm Documentation*. United States, 2025. URL: https://docs.npmjs.com/.

[12] Vercel, Inc. *Next.js Documentation*. San Francisco, CA, USA, 2023. URL: https://nextjs.org/docs.

[13] Prisma. *Prisma: Next-Generation ORM for Node.js and TypeScript*. Berlin, Germany, 2023. URL: https://www.prisma.io/docs.

[14] Meta AI. *React: A JavaScript Library for Building User Interfaces*. Menlo Park, CA, USA, 2023. URL: https://react.dev/.

[15] DeepSeek. *DeepSeek API Documentation*. Beijing, China, 2023. URL: https://api-docs.deepseek.com/.

[16] Supabase. *Supabase Documentation*. Singapore, 2025. URL: https://supabase.com/docs.

[17] NextAuth.js. *NextAuth.js: Authentication for Next.js*. Open Source Community, 2023. URL: https://next-auth.js.org/.

[18] Google LLC. *Google Cloud Console Documentation*. Mountain View, CA, USA, 2023. URL: https://console.cloud.google.com/.

[19] Google LLC. *Using OAuth 2.0 to Access Google APIs*. Mountain View, CA, USA, 2025. URL: https://developers.google.com/identity/protocols/oauth2.

[20] DeepSeek. *DeepSeek API Usage Guide*. Beijing, China, 2025. URL: https://platform.deepseek.com/docs/api-guide.

[21] Prisma. *Prisma Documentation*. Berlin, Germany, 2025. URL: https://www.prisma.io/docs.

[22] Vercel. *Next.js: Getting Started with Local Development*. San Francisco, CA, USA, 2025. URL: https://nextjs.org/docs/getting-started.

[23] Internet Engineering Task Force (IETF). *Transport Layer Security (TLS) Protocol Version 1.2*. 2008. URL: https://tools.ietf.org/html/rfc5246.

[24] Vercel. *Next.js Documentation*. San Francisco, CA, USA, 2025. URL: https://nextjs.org/docs.

[25] Excalidraw. *Excalidraw: Collaborative Whiteboard for Diagramming*. Open Source Community, 2023. URL: https://excalidraw.com/.

[26] Eraser. *Eraser: Collaborative Diagramming Platform*. San Francisco, CA, USA, 2023. URL: https://www.eraser.io/.

[27] DBML Diagram. *DBML: Database Markup Language for Schema Visualization*. Open Source Community, 2023. URL: https://dbml.dbdiagram.io/.

[28] PlantUML. *PlantUML: Open-Source Tool for UML Diagrams*. Open Source Community, 2023. URL: https://plantuml.com/.

[29] Roy T. Fielding. *Chapter 5: Representational State Transfer (REST)*. Irvine, CA, USA, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[30] Microsoft Corporation. *TypeScript: JavaScript with Syntax for Types*. Redmond, WA, USA, 2023. URL: https://www.typescriptlang.org/.

[31] Ecma International. *ECMAScript 2024 Language Specification*. Geneva, Switzerland, 2024. URL: https://262.ecma-international.org/15.0/.

[32] Framer. *Framer Motion: Animation Library for React*. Amsterdam, Netherlands, 2023. URL: https://www.framer.com/motion/.

[33] Tailwind Labs. *Tailwind CSS: Utility-First CSS Framework*. United States, 2023. URL: https://tailwindcss.com/.

[34] Microsoft Corporation. *TypeScript with JSX: TSX*. Redmond, WA, USA, 2023. URL: https://www.typescriptlang.org/docs/handbook/jsx.html.

[35] Meta Platforms, Inc. *Introducing JSX*. Menlo Park, CA, USA, 2023. URL: https://reactjs.org/docs/introducing-jsx.html.

[36] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Cambridge, MA, USA, 2008. URL: https://www.w3.org/TR/xml/.

[37] W3C. *World Wide Web Consortium (W3C)*. Cambridge, MA, USA, 2023. URL: https://www.w3.org/.

[38] Fielding, R. and Gettys, J. and Mogul, J. and Frystyk, H. and Masinter, L. and Leach, P. and Berners-Lee, T. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: https://www.rfc-editor.org/rfc/rfc2616.

[39] Lucas Merencia. *node-cron: Cron-Like Job Scheduler for Node.js*. 2023. URL: https://github.com/kelektiv/node-cron.

[40] Chart.js. *Chart.js: Open-Source Charting Library*. Open Source Community, 2023. URL: https://www.chartjs.org/.

[41] Chart.js Contributors. *React Chart.js 2: React Wrapper for Chart.js*. 2023. URL: https://github.com/reactchartjs/react-chartjs-2.

[42] Niels Provos and David Mazières. *Bcrypt: Adaptive Blowfish-Based Hashing for Passwords*. 2023. URL: https://github.com/kelektiv/node.bcrypt.js.

[43] Jones, M. and Bradley, J. and Sakimura, N. *JSON Web Token (JWT)*. 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.

[44] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford. *Introduction to Algorithms, Chapter 15: Sorting Algorithms*. Cambridge, MA, USA, 2022. URL: https://mitpress.mit.edu/books/introduction-algorithms-fourth-edition.

[45] How Dev. *What is the Weighted Random Selection Algorithm*. United States, 2025. URL: https://how.dev/answers/what-is-the-weighted-random-selection-algorithm.

[46] Wikipedia Contributors. *Fisher–Yates Shuffle*. San Francisco, CA, USA, 2025. URL: https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle.

[47] Knuth, Donald E. and Morris, James H. and Pratt, Vaughan R. *Fast Pattern Matching in Strings*. Philadelphia, PA, USA, 1977. URL: https://epubs.siam.org/doi/10.1137/0206024.

[48] Axios. *Axios: Promise-Based HTTP Client for Node.js and the Browser*. Open Source Community, 2023. URL: https://axios-http.com/.

[49] W3C. *HTML5 Specification*. Cambridge, MA, USA, 2025. URL: https://dev.w3.org/html5/spec-LC/.

[50] W3C. *Cascading Style Sheets (CSS) Specifications*. Cambridge, MA, USA, 2025. URL: https://www.w3.org/Style/CSS/specs.en.html.

[51] React Hot Toast. *React Hot Toast: Notifications for React*. Open Source Community, 2023. URL: https://react-hot-toast.com/.

[52] Prisma. *Prisma Client: Type-Safe Database Access for Node.js and TypeScript*. Berlin, Germany, 2023. URL: https://www.prisma.io/client.

[53] Prisma. *Prisma DBML Generator: Convert Prisma Schema to DBML Format*. 2023. URL: https://github.com/notiz-dev/prisma-dbml-generator.

[54] Liu, Kai and Wang, Lei and Chen, Jundong and Wang, Xiao and Zhang, Yuansen and Wang, Peng and Huang, Yang. *Prompt Injection Attacks and Defenses in LLM-Integrated Applications*. Ithaca, NY, USA, 2023. URL: `https://arxiv.org/abs/2310.12815`.

[55] Bloom, Benjamin S. and Engelhart, Max D. and Furst, Edward J. and Hill, Walter H. and Krathwohl, David R. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*. 1956.

[56] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. United States, 2017. URL: `https://datatracker.ietf.org/doc/html/rfc8259`.

[57] OpenAI. *OpenAI API Documentation: Temperature Parameter*. San Francisco, CA, USA, 2025. URL: `https://platform.openai.com/docs/api-reference/completions/create#temperature`.

[58] Postman, Inc. *Postman: API Testing and Development Platform*. San Francisco, CA, USA, 2023. URL: `https://www.postman.com`.

[59] Google LLC. *Chrome DevTools: Tools for Web Developers*. Mountain View, CA, USA, 2023. URL: `https://developer.chrome.com/docs/devtools/`.

List of Figures

List of Tables

List of Codes