Title: *System Design Document for QuiZone*

Team Name: *QuiZone*

Project Description: *QuiZone is a quiz platform that allows users to create, take, and review quizzes with customizable features like question randomization, timers, and instant feedback.*

Team Members: *Karine Davtyan, Tejleen Kaur , Shaqayeq Salimy*

Date: *December 3rd, 2024*

Table of Contents

1. CRC Cards:

Class Name: Quiz	
Parent Class: none Subclasses:none	
Responsibilities: • Represents a quiz with a title, description, difficulty level and timer settings. • Manages the collection of Question objects. • Handles metadata like randomization settings and deadlines	Collaborators: • Question (contained within a quiz) • QuizController (handles operations on quizzes)

Class Name: Question	
Parent Class: none Subclasses:none	
Responsibilities: • Represents an individual quiz question. • Stores question text, multiple choice options and the correct answer.	Collaborators: • Quiz (part of a collection of questions within a quiz)

Class Name: QuizController	
Parent Class: none Subclasses:none	
Responsibilities: • Manages REST endpoints for quiz operations like creation, retrieval, submission, and archival • Interacts with the JSON file to save and retrieve quiz data	Collaborators: • Quiz (for handling quiz data) • Question (for managing question data within quizzes)

Class Name: QuizProjectApplication	
Parent Class: none Subclasses:none	
Responsibilities: • Initializes and runs the Spring Boot Application • Sets up application context and configuration	Collaborators: • QuizController (initialized for handling requests)

Class Name: form-creation.html	
Parent Class: none Subclasses:none	
Responsibilities: • Provides a web interface for creating quizzes with fields for title, description, difficulty and questions • Submits data to the backend to create or update quizzes	Collaborators: • QuizController (saves quiz data to backend) • styles.css (provides styling for the form).

Class Name: quiz-history.html
Parent Class: none Subclasses:none

Responsibilities:	Collaborators:
• Display the history of quizzes including details and completion status. • Provides functionality for deleting quiz records.	• QuizController (retrieves and deletes quiz history) • styles.css (styles the quiz history page)

Class Name: flashcards.html	
Parent Class: none Subclasses:none	
Responsibilities:	Collaborators:
• Displays quiz questions in a flashcard format for review • Supports interactive learning by allowing users to view answers.	• QuizController (fetches quiz data for display) • styles.css (styles the flashcard elements)

Class Name: styles.css

Parent Class: none Subclasses:none	
Responsibilities: • Defines the visual styling and layout for the web application's user interface • Manges styles for elements such as forms, buttons, text inputs, and flashcards • Ensures a consistent look and feel across all pages of the web application	Collaborators: • form-creation.html (applies styles to form elements and layout) • quiz-history.html (applies styles to historical records and display) • flashcards.html (applies styles to flashcard elements for interactive learning sessions) • certificate-template.html (styles the printable certificate of completion)

2. System Interaction with Environment

Overview:
The system is a web-based quiz application developed using Spring Boot. It integrates various software components and interacts with the hardware environment to provide a seamless user experience. The application is designed to be robust and platform-independent, capable of operating across different environments that meet specific technical requirements.

Dependencies:

- **Operating System:** The system is platform-independent and can run on any operating system that supports Java and Spring Boot, such as Linux, macOS, and Windows.
- **Java Virtual Machine (JVM):** A JVM compatible with Java 11 or higher is required to run the Spring Boot application. This ensures that all features and dependencies managed by Spring Boot are fully supported.
- **Web Browser:** The application is designed for modern web browsers that support HTML5, CSS3, and JavaScript. Recommended browsers include Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge.
- **Storage:** Data is managed using a local JSON file (`quizzes.json`) that serves as a lightweight database for storing and retrieving quizzes. This file needs to be accessible and writable by the application.

Assumptions:

- **JSON File Location and Accessibility:** The system assumes that the `quizzes.json` file is located in the application's root directory and has appropriate permissions set to allow read and write operations.
- **Server Port Configuration:** By default, the application is configured to run on port 8080. It can be reconfigured as needed within the application settings.
- **JavaScript Support:** The application's interactive and dynamic features require that the user's browser has JavaScript enabled. This is essential for handling form submissions, quizzes, and other interactive elements dynamically.

Security Considerations:

- **Data Handling and Privacy:** Care should be taken to ensure that personal data stored in quizzes or collected from users is handled securely, adhering to applicable data protection laws.
- **Input Validation:** All user inputs are validated both on the client-side and server-side to prevent common vulnerabilities such as SQL injection, cross-site scripting (XSS), and other malicious attacks.

3. System Architecture

The architecture of the system is based on the **Model-View-Controller (MVC)** pattern. The components are organized as follows:

1. Model:

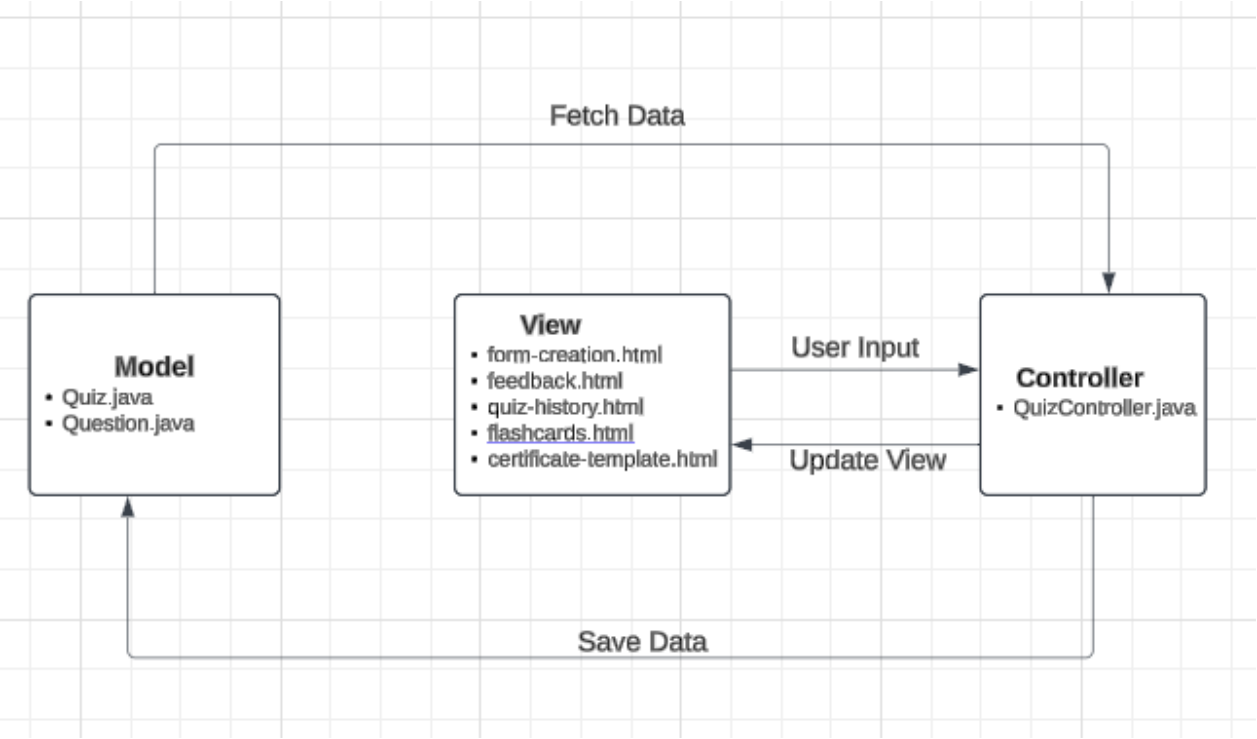
- **Quiz.java** and **Question.java**: Serve as the data models for storing quiz details and individual question data.
- **Data storage**: Quizzes and their details are stored in `quizzes.json`, which acts as a lightweight database.

2. View:

- **HTML files**: `form-creation.html`, `quiz-history.html`, `feedback.html`, `flashcards.html`, and `certificate-template.html` constitute the frontend, allowing users to interact with the system through forms, view quiz history, receive feedback on quiz submissions, study with flashcards, and view completion certificates.
- **JavaScript and CSS**: JavaScript embedded in the HTML files fetches data from the backend and updates the user interface accordingly. The `styles.css` file standardizes the presentation across these pages.

3. Controller:

- **QuizController.java**: Manages all backend logic, handling API endpoints for operations such as creating, retrieving, and submitting quizzes, as well as archiving quiz attempts. This controller interacts with the `quizzes.json` file to fetch and store quiz data.



4. Error Handling Strategy

Description:
The application is designed to ensure a stable user experience by proactively managing and mitigating errors. Below is a detailed description of how various error types are systematically handled:

1. Invalid User Input:

- **Frontend Validation:** JavaScript implemented in the HTML forms ensures that all required fields are filled out before submission. This prevents incomplete data from being processed.
- **Backend Validation:** The `QuizController` performs additional validations to check the JSON structure of incoming data. It ensures that mandatory fields such as quiz title and questions are present and correctly formatted.

2. File I/O Errors:

- **Data Accessibility Checks:** When attempting to save data to `quizzes.json`, the application catches `IOException` to handle scenarios where the file may not be accessible (e.g., file permissions issues or the file being locked by another process).
- **Error Logging and User Notification:** If the file is not writable or cannot be read, the application logs an error and displays a user-friendly message to inform the user of the issue.

3. Server Errors:

- **Error Logging:** The Spring Boot server is configured to log all server-level exceptions. This helps in diagnosing failures during startup or runtime.
- **Fallback Error Page:** Users are redirected to a fallback error page in the event of server failures, maintaining a professional user experience even when unexpected issues occur.

4. Network/External System Failures:

- **Handling Fetch Requests:** In cases where the application is deployed online and relies on external systems or network connections, JavaScript code handles failed fetch requests by displaying user-friendly messages, such as "Error loading feedback data." This informs users there has been a network issue without exposing them to technical details.
- **Robustness in Data Handling:** The application implements retries for network requests where appropriate, enhancing reliability in unstable network conditions.

5. Security and Data Integrity:

- **Sanitization:** User inputs are sanitized both on the client and server-side to prevent cross-site scripting (XSS) and injection attacks.
- **Secure Data Transactions:** HTTPS is used for all data transactions to ensure that data integrity and privacy are maintained.

Error Messages: For each type of error, the system provides feedback to users, such as:

- “Failed to save the quiz. Please try again.”
- “Error loading feedback data.”

5. Summary

The system design of QuiZone leverages the Model-View-Controller (MVC) pattern to provide a robust, scalable, and maintainable framework for quiz management. This architecture facilitates clear separation of concerns, making the system easy to manage and extend.

Key Features and Strengths:

- **Data Management:** Quizzes and their associated data are currently stored in a `quizzes.json` file, serving as a lightweight database. This simplifies data operations while ensuring that the system remains lightweight and responsive.
- **Error Handling:** Comprehensive error management strategies are implemented to anticipate and address user errors, file access issues, and server exceptions efficiently. When errors occur, the system not only provides clear, user-friendly feedback but also logs detailed information for developers, aiding in quick resolution and system improvements.
- **User Interaction:** Through its MVC architecture, the system efficiently manages dynamic user interactions, data processing, and rendering, resulting in a seamless user experience.

With the successful implementation of this sprint, QuiZone has reached a significant milestone, completing the foundational structure and core functionalities outlined in the project's initial scope. The system is robust, extendable, and ready to accommodate future enhancements, which will further solidify its position as a versatile educational tool. This sprint not only marks the completion of the planned architecture but also sets the stage for continuous improvement and innovation in the future.