Quiz App

Project Overview

The Quiz App is a full-stack online examination system built with an Angular front-end, a Spring Boot back-end, and a MySQL database. Administrators can view the results of all users and can create and view questions, while regular users can register, log in, take quizzes, and view their results. The application is designed to manage multiple-choice quizzes, score user submissions, and display results instantly. The goal of this project is to provide a robust quiz/exam portal with support for role-based access, authentication, quiz management, and result tracking.

Technologies Used

- **Angular** (TypeScript front-end framework) – used for building the single-page application (SPA) UI. The Angular CLI (command-line tool) is used to generate and serve the app.

  Note: Angular requires Node.js and npm to install and run the CLI angular.dev.

- **Spring Boot** (Java back-end framework) – used to build RESTful services. The Spring Boot `@SpringBootApplication` annotation auto-configures the application (e.g., setting up embedded Tomcat, component scanning)spring.io.
- **Spring Data JPA** – used by Spring Boot for data access. It provides a repository layer that simplifies CRUD operations on the database.
- **MySQL** – relational database for storing quiz data (questions, options, user accounts, etc.). The application connects to a MySQL instance (using the `quiz_server_db` schema) via JDBC.
- **MySQL Workbench** – an optional GUI tool for designing and querying the MySQL database.
- **Java 17+** – the project is compiled and run on Java 17 (as required by Spring Boot)spring.io.
- **Node.js & npm** – Node.js (with its npm package manager) is required to install Angular and other JavaScript dependencies angular.dev.
- **Maven** – build tools for the Java back-end. Maven can run Spring Boot with `./mvnw spring-boot:run`spring.io.)

Prerequisites and Installation

Before setting up the project, ensure the following prerequisites are installed on your system:

- **Java Development Kit (JDK) 17 or later** – required to compile and run the Spring Boot backend spring.io.
- **Node.js (LTS) and npm** – required to build and run the Angular frontend. Angular CLI requires Node.js (which includes npm) angular.dev.

- **Angular CLI** – install globally via npm:

  ```
  bash
  CopyEdit
  npm install -g @angular/cli
  ```

  This installs the Angular CLI command (`ng`), which is used to generate and serve the frontend angular.dev.

- **MySQL Server** – install and start MySQL. Create a database/schema named `quiz_server_db` (as required by the application). We can use MySQL Workbench or the command line to create the schema and user.
- **MySQL Workbench (optional)** – a GUI client for MySQL to manage the database.
- **Git** – to clone or download the project repository.

After installing prerequisites, clone the repository (or download its ZIP) to your local machine. The project consists of two main parts (backend and frontend) that you will configure and run separately.

Backend Setup (Spring Boot)

The Spring Boot backend provides RESTful APIs for authentication, quiz management, and results. It uses Spring Data JPA to interact with the MySQL database. Follow these steps to configure and run the backend:

1. **Configure the Database Connection:**
   In the `src/main/resources/application.properties` (or `application.yml`) file, set the MySQL connection properties. For example:

   ```
   properties
   CopyEdit
   spring.jpa.hibernate.ddl-auto=update
   spring.datasource.url=jdbc:mysql://localhost:3306/quiz_server_db
   spring.datasource.username=root
   spring.datasource.password=Sql@2025
   spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
   ```

   These properties tell Spring Boot how to connect to MySQLspring.io. Replace the `url`, `username`, and `password` with your MySQL details. The `spring.jpa.hibernate.ddl-auto=update` setting ensures that JPA will automatically create/update tables according to your entities.

2. **Build the Backend:**
   Navigate to the backend project root in a terminal. Run

   ```
   bash
   CopyEdit
   ./mvnw clean install
   ```

This compiles the code and builds the JAR file.

3. **Run the Backend:**
   Start the Spring Boot application. Use:

   ```
   bash
   CopyEdit
   ./mvnw spring-boot:run
   ```

   or run the JAR directly:

   ```
   bash
   CopyEdit
   java -jar target/*.jar
   ```

   This launches the embedded web server (default on port 8080). For example, Spring's Getting Started guide notes that running with Maven can be done via `./mvnw spring-boot:run`spring.io, and you should see Spring Boot logging output in your console. The backend will be accessible (e.g.) at `http://localhost:8080` for its API endpoints.

Frontend Setup (Angular)

The Angular frontend is a single-page application (SPA) that interacts with the Spring Boot REST APIs. To set up and run the Angular app:

1. **Navigate to the Angular project directory:**
   This is usually the folder containing `angular.json` and `package.json` (often named something like `quiz-app-frontend` or similar).
2. **Install Dependencies:**
   Run:

   ```
   bash
   CopyEdit
   npm install
   ```

   This installs all Angular and third-party libraries defined in `package.json`.

3. **Serve the Angular Application:**
   After dependencies are installed, start the development server:

   ```
   bash
   CopyEdit
   ng serve --open
   ```

   The `ng serve` command compiles and launches the app locally angular.dev. Adding `--open` (or `-o`) will automatically open `http://localhost:4200/` in your web browser angular.dev. Now the Angular app is running and will reload automatically as you make changes.

4. **Angular CLI:**
   (If you haven't installed it globally yet, you can install Angular CLI with `npm install -g @angular/cli`angular.dev.) The CLI provides commands like `ng serve`, `ng build`, and `ng generate` to help develop the Angular app.

Once the Angular server is running, the front-end will communicate with the back-end APIs (typically at `http://localhost:8080` by default). Make sure that the back-end is running before interacting with the front-end to avoid API errors.

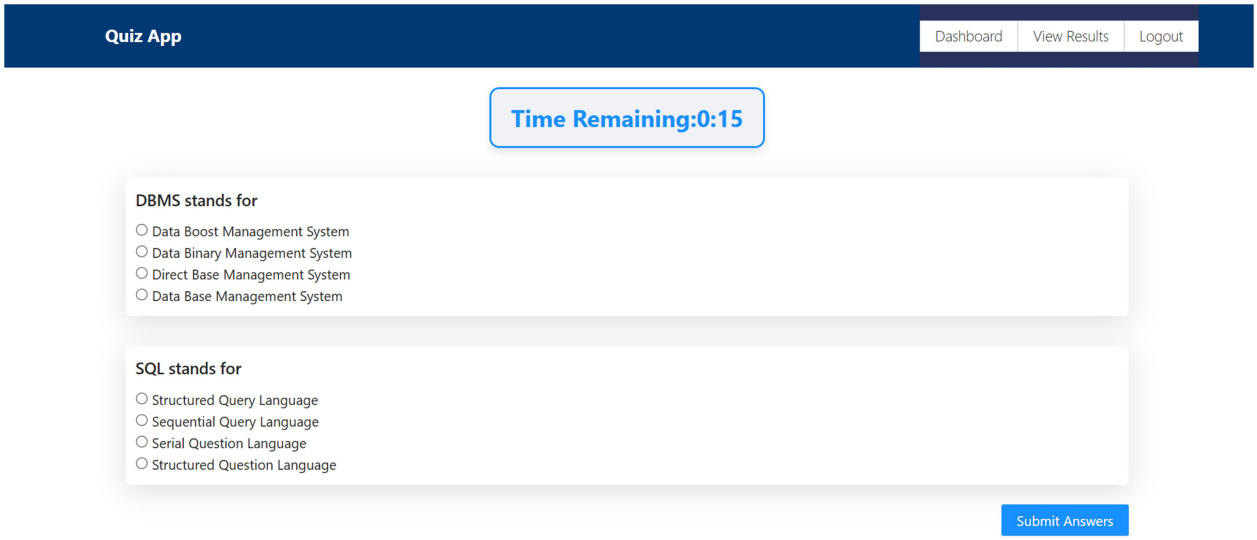
User Roles

This application uses role-based access control (RBAC) with two main roles: **Admin** and **User**. Each role has specific permissions:

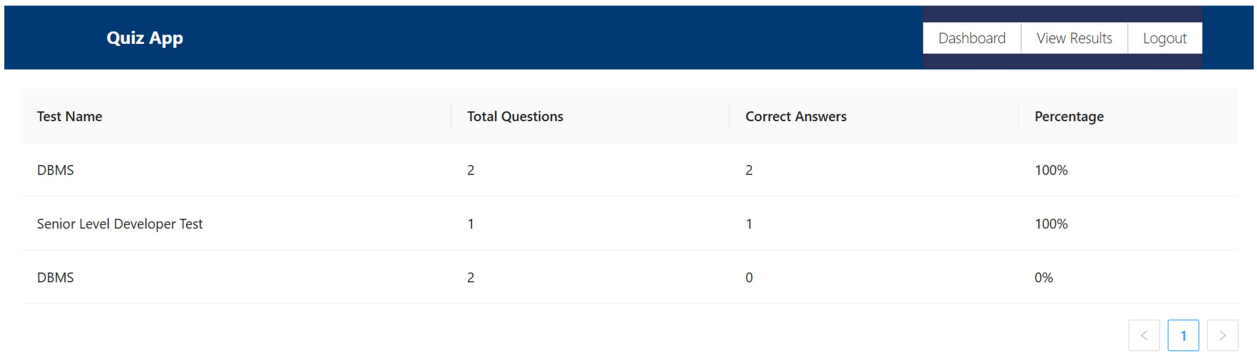
- **Admin:**
  - *Quiz Management:* Can create quizzes and questions. Admins manage all quiz content and categories.
  - *View User Results:* can view the results of all users.
  - *Full Access:* Essentially has full privileges over the system (e.g., viewing all user results, statistics, etc.).
- **User:**
  - *Authentication:* Can register a new account and log in to the system.
  - *Quiz Participation:* Can view available quizzes, take quizzes, and submit answers.
  - *Result Viewing:* Can view their own quiz scores and results after submission.
  - *Limited Access:* Cannot create or modify quizzes or other users; only has access to their own data and the actions necessary to take quizzes.

Core Functionalities

- **Authentication (Register & Login):** Users must register an account with a username/email and password. The backend authenticates users (typically using Spring Security or custom logic) and establishes a session or token upon successful login. After logging in, the user's role (Admin or User) is determined and stored for access control.
- **Quiz Management (Admin only):** Administrators can create quizzes, add questions on quizzes. This includes creating new quizzes and adding questions. In Spring Boot, this is typically done via controller endpoints that accept HTTP POST requests. The backend persists quizzes in MySQL using Spring Data JPA (for example, a `QuizRepository` interface extends `JpaRepository` for database operations).
- **Quiz Participation (User):** Regular users can browse available quizzes and start a quiz session. During a quiz, all questions are displayed at a time with total time. The Angular front-end iterates over the questions and their options, rendering them with radio buttons. For example, the template may use an `*ngFor` to list each question and its options, binding each option to a selection model codeproject.com. The user selects answers and navigates through the quiz.



- **Result Viewing:** Once a user completes a quiz, the backend calculates the score and returns the results. The front-end then displays a result summary. This typically includes the total score, total question, correct answer and optionally the correct answers or explanations. For example, a result view might highlight correct vs. incorrect responses and show a final score.



This result interface provides immediate feedback. The application also store the result in the database so that users can review past attempts.

How the Frontend Communicates with Backend

The front-end and back-end communicate over HTTP using RESTful APIs. The Spring Boot backend exposes various endpoints (e.g. `/api/auth/login`, `/api/quizzes`, `/api/quizzes/{id}/take`, etc.) via annotated controllers. For instance, a Spring `@RestController` can use `@GetMapping`, `@PostMapping`, etc. to define an endpoint that returns JSON dataspring.io. As the Spring guide notes, a class annotated with `@RestController` "is

ready for use by Spring MVC to handle web requests… returning data rather than a view" spring.io.

On the Angular side, the `HttpClient` service is used to send HTTP requests to these endpoints. The official Angular documentation states that most front-end apps need to communicate with a server and that Angular provides an `HttpClient` API for this purpose angular.dev. In practice, the Angular code will inject `HttpClient` and make calls like `http.post('http://localhost:8080/api/quizzes', quizData)` or `http.get('http://localhost:8080/api/quizzes')`. Responses (usually JSON) are received as JavaScript objects which the Angular app then uses to update the UI or application state.

For example, when a user logs in, the Angular app might call a backend endpoint with credentials; the backend authenticates and returns a JWT or session token. When an admin creates a quiz, the Angular app sends a POST request with the quiz data in JSON format to a Spring Boot endpoint, which then saves it to MySQL via JPA. This HTTP-based communication (using REST principles) cleanly separates the front-end and back-end logic.

Project Directory Structure Overview

The project separates front-end and back-end code into distinct directories. A typical structure might look like:

```
bash
CopyEdit
quiz-app-backend/          # Spring Boot backend project
  ├── src/main/java/...    # Java source code (controllers, services,
entities)
  ├── src/main/resources/
  │     └── application.properties  # DB and app configuration
  └── pom.xml (or build.gradle)   # Build configuration for Java
quiz-app-frontend/         # Angular front-end project
  ├── src/app/...          # Angular components, services, models
  ├── src/assets/...       # Static assets (images, styles)
  ├── angular.json         # Angular CLI configuration
  ├── package.json         # npm and Angular dependencies
  └── tsconfig.json        # TypeScript configuration
README.md               # Project documentation (this file)
```