

Creating a Quiz Service

👉 Introduction to the Quiz Service

After successfully implementing our Question Service, we now need to build our second microservice: the **Quiz Service**. This service will act as the primary interface for client interactions and will communicate with the Question Service to fulfill its operations.

In a microservices architecture, this relationship exemplifies how services work together - clients interact directly with the Quiz Service, which then communicates with the Question Service when it needs question-related functionality.

➤ The Quiz Service will handle three main operations:

1. **Creating Quizzes** - Generating new quizzes for users
2. **Retrieving Quizzes** - Fetching quiz details by ID
3. **Calculating Scores** - Determining user scores for completed quizzes

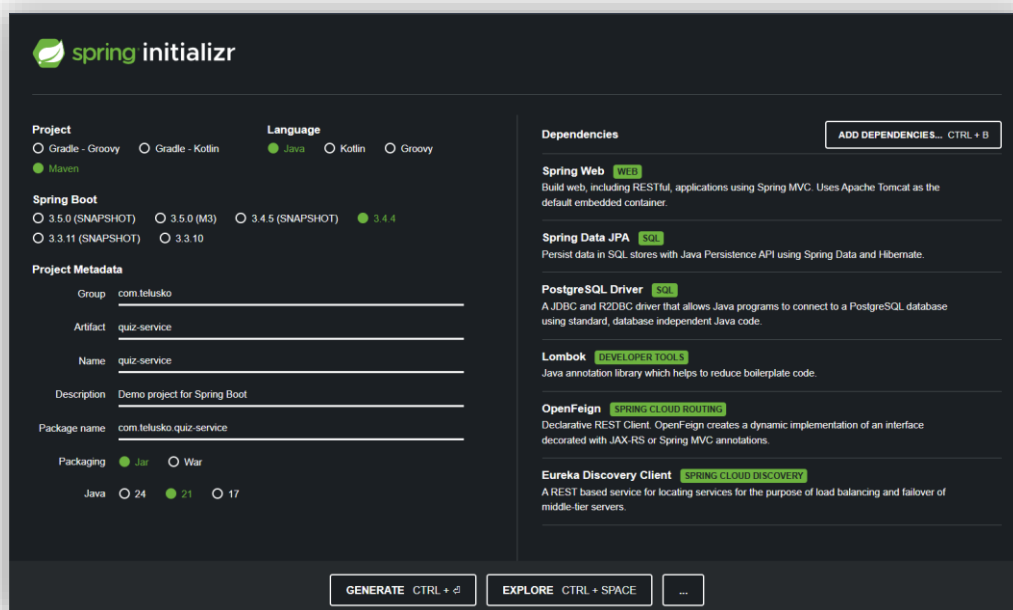
These operations represent the essential user-facing features of our quiz application.

➤ To create our Quiz Service:

Initialize a New Spring Boot Project. To initialize visit Spring Initializer (<https://start.spring.io/>) and name the project **quiz-service**.

➤ Add the following dependencies:

- Spring Web (for handling HTTP requests)
- Spring Data JPA (for database interactions)
- PostgreSQL Driver (for connecting to PostgreSQL database)
- Lombok (for reducing boilerplate code)
- OpenFeign (for service-to-service communication)
- Eureka Discovery Client (for service registration and discovery)



The screenshot shows the Spring Initializr web application interface. The 'Project' section has 'Maven' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '3.4.4' selected. The 'Project Metadata' section shows the following fields: Group (com.telusko), Artifact (quiz-service), Name (quiz-service), Description (Demo project for Spring Boot), Package name (com.telusko.quiz-service), Packaging (Jar), and Java version (21). The 'Dependencies' section lists the following dependencies: Spring Web, Spring Data JPA, PostgreSQL Driver, Lombok, OpenFeign, and Eureka Discovery Client. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and a menu icon.

➤ Download the generated project and open it in your preferred IDE

👉 Reusing Code from Monolith

One advantage of transitioning from a monolithic application is that we can reuse much of our existing code:

- Copy all files from the original **QuizApp** project to our new quiz-service project
- Include the `application.properties` file
- Modify the database name in `application.properties` to **quizdb**
- Create a new database with this name in PostgreSQL
- Since we now have a dedicated Question Service, we can remove:
 - `QuestionController`
 - `QuestionDao`
 - `QuestionService`
 - `Question`

