Creating a Question Service part 1

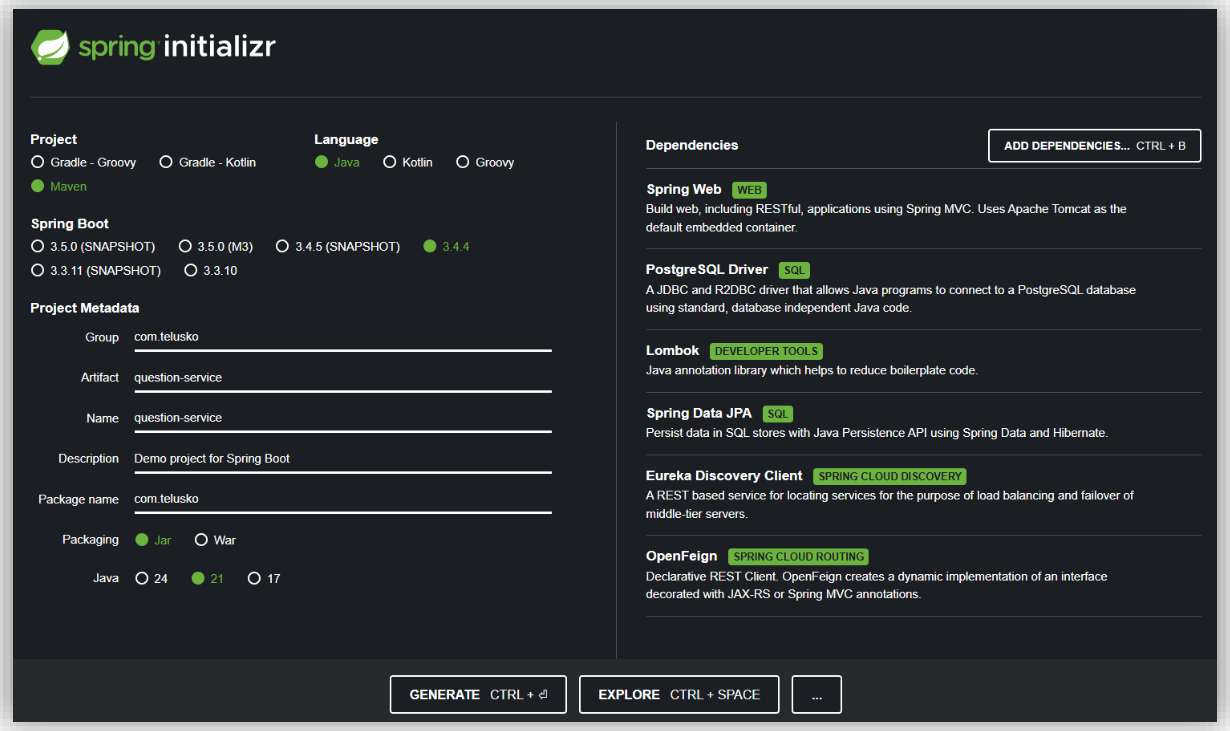
👉 Setting Up the Question Service

In our journey from a monolithic application to microservices, we'll start by creating our first microservice: the **Question Service**. This service will handle all question-related functionality independently.

➢ Creating a New Spring Boot Project

To create our Question Service:

1. Go to Spring Initializer (https://start.spring.io/)
2. Set up a new project with the name **question-service**
3. Add the following dependencies:
   - Spring Web (for REST endpoints)
   - PostgreSQL Driver (for database connectivity)
   - Lombok (to reduce boilerplate code)
   - Spring Data JPA (for database operations)
   - OpenFeign (for service-to-service communication)
   - Eureka Discovery Client (for service registration and discovery)
4. Generate the project and open it in your IDE



Project
Gradle - Groovy   Gradle - Kotlin   [✓ Maven]

Language
[✓ Java]   Kotlin   Groovy

Spring Boot
3.5.0 (SNAPSHOT)   3.5.0 (M3)   3.4.5 (SNAPSHOT)   [✓ 3.4.4]
3.3.11 (SNAPSHOT)   3.3.10

Project Metadata

Group	com.telusko
Artifact	question-service
Name	question-service
Description	Demo project for Spring Boot
Package name	com.telusko
Packaging	[✓ Jar] War
Java	24 [✓ 21] 17

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

PostgreSQL Driver SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Eureka Discovery Client SPRING CLOUD DISCOVERY
A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

OpenFeign SPRING CLOUD ROUTING
Declarative REST Client. OpenFeign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.

👉 Migrating Code from Monolith to Microservice

Since we already have our QuizApp monolithic application, we can reuse some of its code:

- ➢ Copy relevant files from the QuizApp project to our new question-service project
- ➢ Delete all Quiz-related components:
  - QuizController
  - QuizDao
  - Quiz entity
  - QuizService
- ➢ Copy the application.properties file from QuizApp to configure our database connection

👉 Making the Question Service Independent

To make our Question Service function independently, we need to add new functionality to the QuestionController. This will allow other services (like the Quiz Service) to interact with it.

➢ Adding New Endpoints to QuestionController

We need to implement three key functionalities:

1. **Generate Quiz Questions**

   - In the monolithic application, the Quiz component accessed the question database directly
   - Now, the Quiz Service will need to request questions from our Question Service
   - We'll add a method to generate a set of questions for a new quiz

2. **Get Questions by Quiz ID**

   - The Quiz Service will store quiz information but not question details
   - When needed, Quiz Service will request questions using the quiz ID
   - We'll add a method to retrieve all questions associated with a specific quiz

3. **Calculate Score**

   - Score calculation requires comparing user answers with correct answers
   - Since Question Service has all question data including correct answers, it's responsible for score calculation
   - We'll add a method to calculate the score based on user responses

👉 Summary of Required Endpoints

Our Question Service needs these three essential endpoints:

1. **generate** - Creates and returns a set of questions for a new quiz
2. **getQuestions** - Retrieves questions associated with a specific quiz ID
3. **getScore** - Calculates and returns a score based on user responses

By implementing these endpoints, our Question Service will be fully independent and capable of supporting the Quiz Service through API calls rather than direct database access.