Working with Feign

Now that we have our Question Service and Quiz Service registered with Eureka Server, we need to establish communication between them. Specifically, when creating a quiz in the Quiz Service, we need to retrieve questions from the Question Service.

👉 OpenFeign

OpenFeign is a declarative REST client for Java applications. It simplifies the process of making HTTP requests to other services by allowing developers to write interface declarations with annotations rather than writing HTTP client code.

- Eliminating boilerplate HTTP client code
- Seamless integration with service discovery tools like Eureka
- Declarative approach through simple interfaces and annotations
- Built-in support for load balancing and fault tolerance
- Automatic conversion between Java objects and HTTP requests/responses

👉 Implementing Feign Client in Quiz Service

Creating a Feign Interface

To enable communication with the Question Service, we need to create a Feign client interface:

```java
package com.telusko.quizservice.feign;

import com.telusko.quizservice.model.QuestionWrapper;
import com.telusko.quizservice.model.Response;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

// This annotation specifies which service we want to connect to
// The name must match exactly with the service registered in Eureka
@FeignClient("QUESTION-SERVICE")
public interface QuizInterface {
    // Declares the generate API from Question Service
    // The path and parameters must match the actual endpoint in Question Service
    @GetMapping("question/generate")
    public ResponseEntity<List<Integer>> getQuestionsForQuiz
            (@RequestParam String categoryName, @RequestParam Integer numQuestions );

    // Declares the getQuestions API from Question Service
    @PostMapping("question/getQuestions")
    public ResponseEntity<List<QuestionWrapper>> getQuestionsFromId(@RequestBody List<Integer> questionIds);

    // Declares the getScore API from Question Service
    @PostMapping("question/getScore")
    public ResponseEntity<Integer> getScore(@RequestBody List<Response> responses);

}
```

This interface serves as a contract for interacting with the Question Service. We declare the same method signatures as the endpoints in the Question Service, and OpenFeign handles the actual HTTP communication behind the scenes.

Modifying the Quiz Entity

When working with microservices, we need to adapt our entities to support distributed data. Instead of storing full Question objects in our Quiz entity, we'll store just the question IDs:

```java
package com.telusko.quizservice.model;

import jakarta.persistence.*;
import lombok.Data;

import java.util.List;

@Entity
@Data
public class Quiz {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String title;

    // @ElementCollection allows storing a collection of simple elements
    // This replaces the previous direct relationship with Question entities
    @ElementCollection
    private List<Integer> questionIds;

}
```

The @ElementCollection annotation tells JPA to store this collection of integers in a separate table that's linked to the Quiz table, rather than trying to establish entity relationships across different databases.

Implementing the Create Quiz Service

Now we can implement the Quiz Service to use our Feign client for retrieving questions:

```java
// Method in QuizService class
public ResponseEntity<String> createQuiz(String category, int numQ, String title) {
    // Call the Question Service through our Feign client
    // getBody() extracts the actual list from the ResponseEntity
    List<Integer> questions = quizInterface.getQuestionsForQuiz(category, numQ).getBody();

    // Create a new Quiz with the received question IDs
    Quiz quiz = new Quiz();
    quiz.setTitle(title);
    quiz.setQuestionIds(questions);
    quizDao.save(quiz);

    return new ResponseEntity<>("Success", HttpStatus.CREATED);
}
```

This method demonstrates the power of Feign - with a single line of code, we make an HTTP request to another service, without worrying about URLs, HTTP clients, or response parsing.

Enabling Feign Client

To activate Feign in our application, we need to add the @EnableFeignClients annotation to our main application class:

```java
package com.telusko.quizservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

// This annotation activates the OpenFeign mechanism
// It tells Spring to scan for @FeignClient interfaces
@SpringBootApplication
@EnableFeignClients
public class QuizServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuizServiceApplication.class, args);
    }

}
```

Configuring the Service

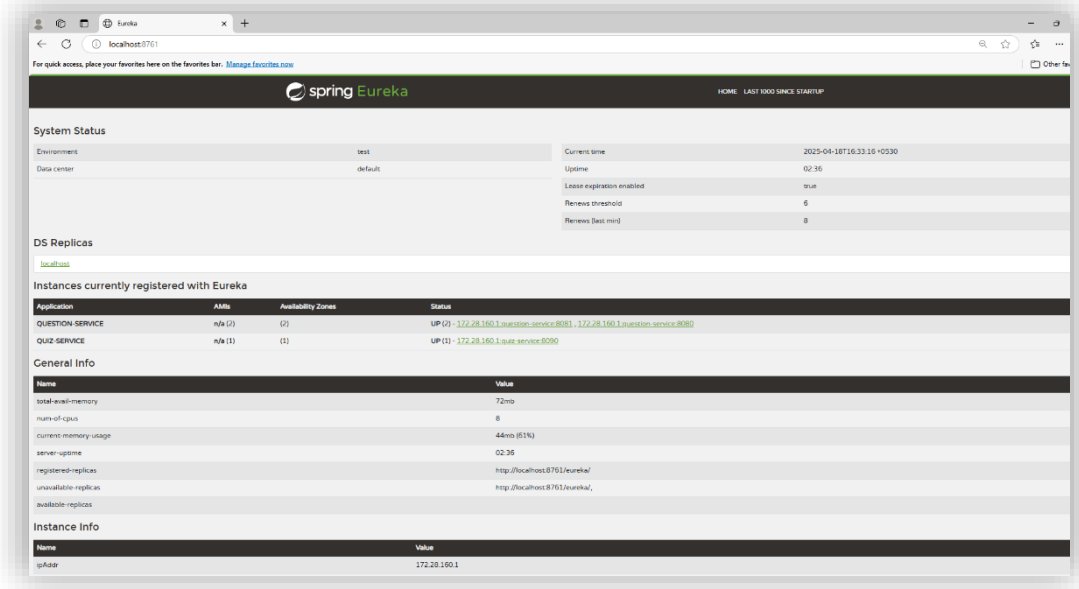
Finally, we configure our Quiz Service in the application.properties file:

```
spring.application.name=quiz-service
server.port=8090
```

- spring.application.name: Identifies our service to Eureka
- server.port: Sets the port to avoid conflicts with other services

Verifying Service Registration

After starting our Quiz Service, we can confirm it has registered with Eureka by checking the Eureka dashboard:



👉 Understanding the Flow

With Feign and Eureka working together, the process of creating a quiz now follows this flow:

- ➢ A client sends a request to the Quiz Service to create a quiz
- ➢ The Quiz Service uses OpenFeign to call the Question Service's generate API
- ➢ The Question Service returns a list of question IDs
- ➢ The Quiz Service stores these IDs in the database
- ➢ When the quiz needs to be displayed, the Quiz Service will again use Feign to fetch the actual questions